Algorithm Syllabus Details GATE, UGC-NE

unacademy

1. Asymptotic Notation
2. Time & Space Complexity
3. Divide & Conquer, सारे के सारे Algorithms और Heap Tree
4. Greedy Methods (Job Sequencing, Knapsack, Optimal. Merge Pattern, Huffman encoding, Dijkstra's, Prisms, Kruskal, MST)
5. Graph Traversal (DFS, BFS, etc.)
6. Dynamic Programming (All pair shortest path, Multistage graph, Optimal Binary Search Tree, TSP, 0/1 knapsack, LCS, Matrix chain multiplication, Sum of subset)
7. Hashing
8. NP, NPC, HP, NPH

unacademy

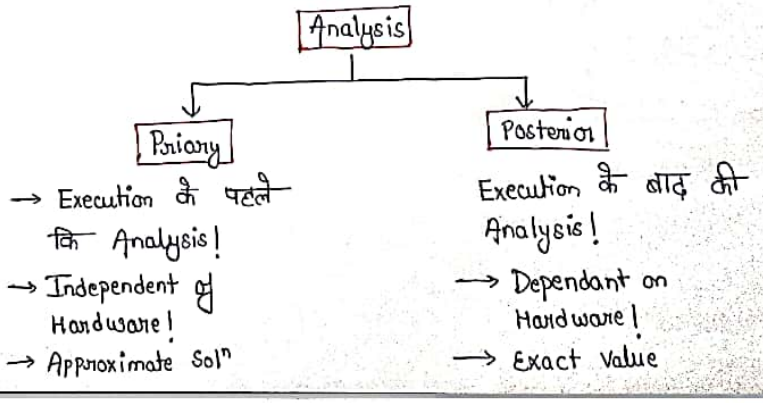
Q. क्या है ये Algorithm?

→ Finite set of steps to solve a problem is called algorithm.

→ अपने को Algorithm का Analysis करना है, कैसे करे

Analysis : Time complexity & Space Complexity!

→ Analysis करने से अपने को ये idea milta है कि konsa Algorithm kaha suitable है!



unacademy

Asymptotic Notation:

→ Used to describe running time of an algorithm showing the order of growth of function.

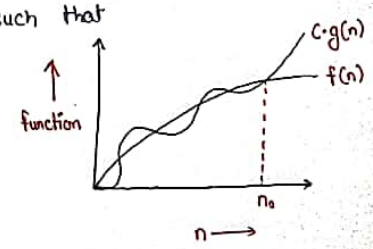
1. Big oh notation (O) : (upper bound)

f(n) is said to be the asymptotic upper bound of function g(n), If there a positive constant c & no such that

$$f(n) \leq c \cdot g(n)$$

where, $c > 0$

$n > n_0 \geq 0$

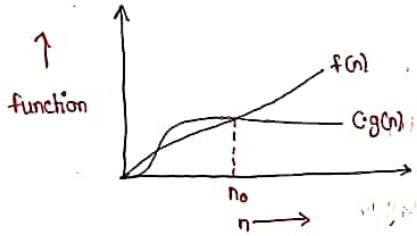


(2.) Big Omega (Ω) : (lower Bound)

$f(n)$ is said to be the asymptotic lower bound of function $g(n)$. If there exist a positive constant c & n_0 such that

$$f(n) \geq c \cdot g(n)$$

where, $c > 0$
$n > n_0 \geq 0$



(4.) Small oh notation : (o)

$$f(n) = o(g(n))$$

$$\text{iff} \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

(5.) Small omega notation : (ω)

$$f(n) = \omega(g(n))$$

$$\text{iff} \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$$

	Reflexive	Symmetric	Transitive
(1.) Big (O): $f(n) \leq c \cdot g(n) \Rightarrow a \leq b$	✓	✗	✓
(2.) Big Omega (Ω): $f(n) \geq c \cdot g(n) \Rightarrow a \geq b$	✓	✗	✓
(3.) Theta (Θ): $c_1 g(n) \leq f(n) \leq c_2 g(n)$ $\Rightarrow f(n) = g(n)$ $\Rightarrow a = b$	✓	✓	✓
(4.) Small (o): $f(n) < c\,g(n)$ $\Rightarrow a < b$	✗	✗	✓
(5.) Small (ω): $f(n) > c \cdot g(n)$ $\Rightarrow a > b$	✗	✗	✓

unacademy

Comparison of various Time complexities :

$$O(c) < O(\log\log n) < O(\log n) < O(n^{1/2}) < O(n) < O(n\log n) < O(n^2) < O(n^3) < O(n^k) < O(2^n) < O(n^n) < O(2^{2^n})$$

unacademy

(1.) $f_1(n) = n^2 \log n$

$f_2(n) = n (\log_n n)^{10}$

$n = 2$

$f_1(2) = 2^2 \log_2 2 = 4$

$f_2(2) = 2 (\log_2 2)^{10} = 2$

$n = 10^3$

$f_1(10^3) = 10^6 \log_{10} 10^3 = 3 \times 10^6$

$f_2(10^3) = 10^3 (\log_{10} 10^3)^{10} = 10^3 \times 3^{10}$

3×10^6 $\quad 3^{10} \times 10^3$

10^6 $\quad 3^9 \times 10^3$

$10^3 <$ $\quad 3^9$

$n = 10^{10}$

$f_1(10^{10}) = 10^{20} \log 10^{10}$

$= 10 \times 10^{20}$

$f_2(10^{10}) = 10^{10} (\log 10^{10})^{10}$

$= 10^{10} \times 10^{10}$

$f_1(n) \geq c\, g(n)$

$f_1(n) \geq c\, f_2(n) \Rightarrow f_1(n) = O(f_2(n))$

unacademy

$f_1(n) = n^2 \log n$ $\qquad f_2(n) = n (\log n)^{10}$

$n^2 \log n$ $\qquad n (\log n)^9$

n $\qquad (\log n)^9$

$\log n$ $\qquad \log (\log n)^9$

$\log n$ $\qquad 9 \log \log (n)$

$\log n > \log \log n$

GATE Q. $f_1(n) = 2^n$; $f_2(n) = n^{3/2}$; $f_3(n) = n\log_2 n$; $f_4(n) = n^{\log_2 n}$

What is the increasing order of asymptotic complexity.

Let $n = 2^{10}$

$f_1(n) = 2^{2^{10}} = 2^{1024}$

$f_2(n) = (2^{10})^{3/2} = 2^{15}$

$f_3(n) = 2^{10}\log_2 2^{10} = 10 \cdot 2^{10}$

$f_4(n) = 2^{10\log_2 2^{10}} = 2^{100}$

$\boxed{f_1 > f_4 > f_2 > f_3}$

GATE Q. Let $f(n) = n^2 \log n$ & $g(n) = n(\log n)^{10}$ be two positive functions of n. Which of the following statement is correct?

(a) $f(n) = O(g(n))$ and $g(n) \neq O(f(n))$

(b) $g(n) = O(f(n))$ and $f(n) \neq O(g(n))$

(c) $f(n) \neq O(g(n))$ and $g(n) \neq O(f(n))$

(d) $f(n) = O(g(n))$ and $g(n) = O(f(n))$

$n = 10^{10}$

$f(n) = 10^{20} \times \log 10^{10} = 10^{21}$

$g(n) = 10^{10}(\log 10^{10})^{10} = 10^{20}$

$f(n) > g(n)$

$\boxed{g(n) = O(f(n))}$

$n = 10^3$

$f(n) = 10^6 \log 10^3 = 3 \times 10^6$

$g(n) = 10^3 (\log 10^3)^{10} = 10^3 \times 3^{10} = 3^{10} \times 10^3$

3×10^6 $\quad$ $3^{10} \times 10^3$

3×10^6 $\quad$ $3 \times 3^9 \times 10^3$

10^3 $\quad$ 3^9

GATE Q. Which of them are true?

① $(n+k)^m = O(n^m)$

② $2^{n+1} = O(2^n)$

③ $2^{2n+1} = O(2^n)$

k & m are constants.

① $(n+k)^m = O(n^m)$

$(n+k)^m \leq c \cdot n^m$

$k = 1, \; m = 1$

$n + 1 \leq cn$ ✓

② $2^{n+1} = O(2^n)$

$2^{n+1} \leq c\,2^n$

$2^n \cdot 2 \leq c \cdot 2^n$

$2^n \cdot 2 \leq c \cdot 2^n$ ✓

$2^{n+1} = O(2^n)$

③ $2^{2n+1} = O(2^n)$

$2^{2n+1} \leq c\,2^n$

$2^n \cdot 2 \leq c \cdot 2^n$

$c = 2$

$2^n \cdot 2 \leq 2 \cdot 2^n$

$2n \leq n$ ✗

unacademy

Recurrence Relation :

→ A recurrence is an equation or inequality that describes a function in terms of its value on smaller inputs.

→ Arre, Mtlb agr simple language में बोलू तो बहुत सारे Algorithms are recursive in nature.

→ जब हम उन Algorithm को Analyse करते है तो अपने को एक Recurrence Relation Milta है।

→ Recurrence Relation Agr nikal liye toh smjho Time complexity भी Nikal gyi! कैसे? तो Vo aage btaunga dekhte jaao.

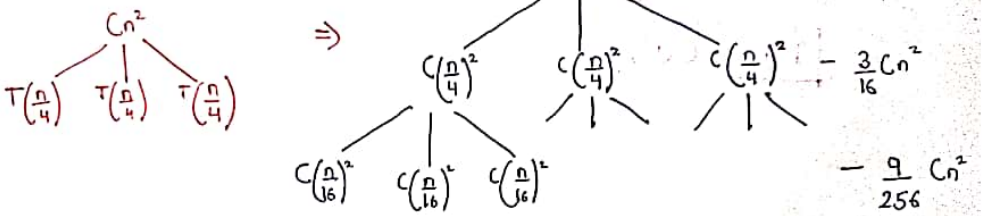
Solving Recurrence Relation

- Substitution Method
- Recurrence Tree Method
- Master's Theorem / Method

unacademy

Recurrence Tree Method :

① $T(n) = 3T\left(\frac{n}{4}\right) + n^2$ ← root



✱ Depth of the tree :

$\frac{n}{4^i} = 1 \Rightarrow n = 4^i$

$\log n = \log 4^i$

$\log n = i \log 4$

$\boxed{i = \log_4 n}$

✱ No. of leaves :

$3^i = 3^{\log_4 n}$

$\approx n^2$

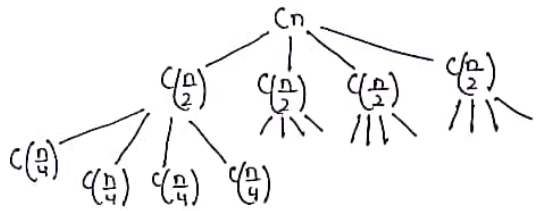
unacademy

∴ Total Cost

$$T(n) = Cn^2 + \frac{3}{16}Cn^2 + \left(\frac{3}{16}\right)^2 Cn^2 + \ldots\ldots + \theta(3^{\log_4 n})$$

$$\boxed{T(n) = \theta(n^2)}$$

(2.) $T(n) = 4T\left(\frac{n}{2}\right) + n$



Cn

$4\frac{Cn}{2} = 2Cn$

4Cn

⋮

✯ Depth of tree :

$\frac{n}{2^i} = 1$

$n = 2^i$

$\log n = \log 2^i$

$\log n = i \log 2$

$\boxed{i = \log_2 n}$

✯ No. of leaves :

$4^i = 4^{\log_2 n}$

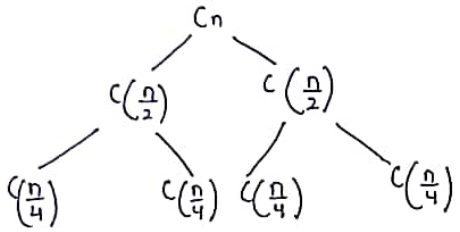
$= 2^{\log_2 n^2}$

$= n^2$

∴ Total cost = $Cn + 2Cn + 4Cn + \dots + \Theta(n^2)$

$\boxed{T(n) = \Theta(n^2)}$

(3.) $T(n) = 2T\left(\frac{n}{2}\right) + n$



— Cn

— Cn

— Cn

✯ Depth of the tree :

$\frac{n}{2^i} = 1 \Rightarrow \boxed{i = \log_2 n}$

✯ Level = Dept + 1

$= \log_2 n + 1$

✯ Total Cost = cost of each level x No. of level

$= Cn(\log_2 n + 1)$

$= Cn \log_2 n + Cn$

$\boxed{T(n) = Cn \log_2 n}$

Q.
```
fib(n)
  if(n=0)
    return 0
  if (n=1)
    return 1
  else
    return fib(n-1) + fib(n-2)
```

$$fib(n) = \begin{cases} 0, & \text{if } n=0 \\ 1, & \text{if } n=1 \\ fib(n-1) + fib(n-2), & n>1 \end{cases}$$

$T(n) = T(n-1) + T(n-2) + C$

Q. What is the time complexity?

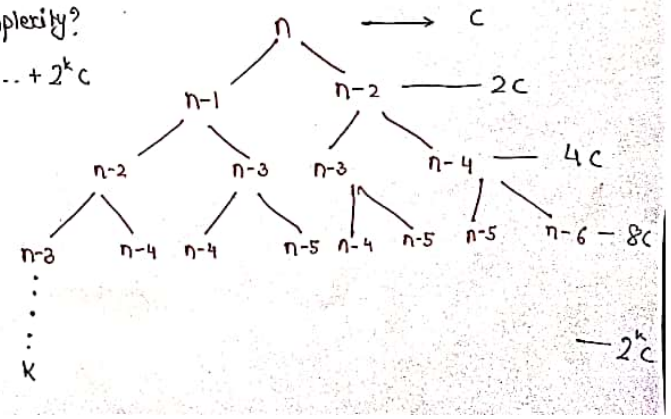
$C + 2C + 4C + 8C + \dots + 2^k C$

$C(1 + 2 + 4 + 8 + \dots 2^k)$

$C \cdot \frac{1(2^k - 1)}{2 - 1}$

$C \cdot (2^k - 1) \approx 2^k C$

$\boxed{O(2^n)}$



GATE-2007

Q. What is the time complexity of the following recursive function?

```
int DoSomething (int n)
{  if (n<=2)
     return 1;
   else
     return (DoSomething (floor (sqrt(n)) + n);
}
```

(a) $\Theta(n^2)$
(b) $\Theta(n \log_2 n)$
(c) $\Theta(\log_2 n)$
(d) $\Theta(\log_2 \log_2 n)$

$$T(n) = \begin{cases} T(\sqrt{n}) + C, & n > 2 \\ 1, & n <= 2 \end{cases}$$

$n \to n^{1/2} \to n^{1/4} \to \dots \to n^{1/2^k}$

$n^{\frac{1}{2^k}} = 2$

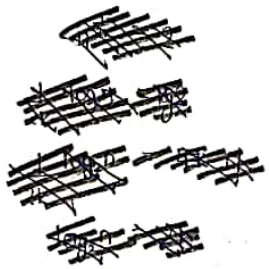
$\frac{1}{2^k} \log n = \log_2 2$

$\log_2 n = 2^k$

$\log \log_2 n = k \log 2$

$k = \log_2 \log_2 n$

$\Theta(k) = \Theta(\log_2 \log_2 n)$



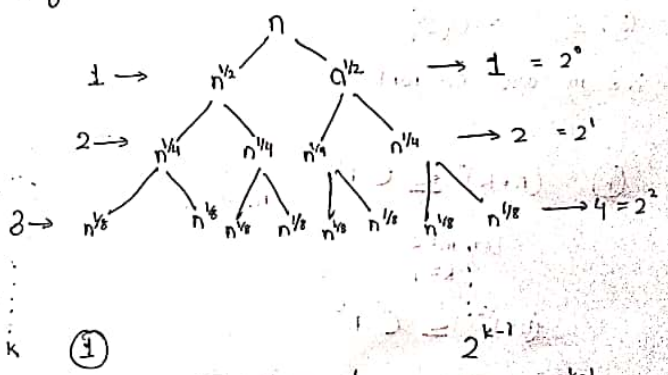
GATE

Q. Consider the following recurrence:

$$T(n) = 2T(\sqrt{n}) + 1, \quad T(1) = 1$$

Which one of the following is true?

(a) $T(n) = \Theta(\log \log n)$
(b) $T(n) = \Theta(\log n)$
(c) $T(n) = \Theta(\sqrt{n})$
(d) $T(n) = \Theta(n)$



$T(n) = 1 + 2^1 + 2^2 + \dots + 2^{k-1}$

$T(n) = 2^{k-1}$

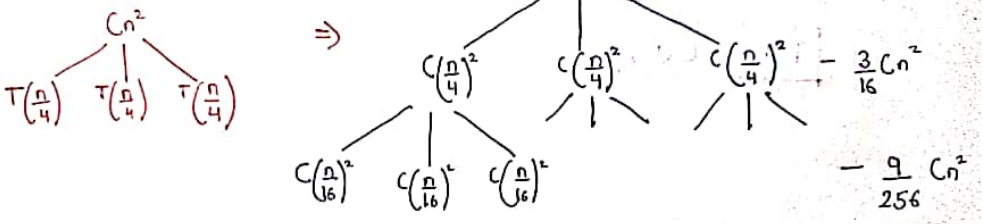
$= 2^{\log_2 \log_2 n - 1}$

$= \log_2 n - 1$

$T(n) = \Theta(\log_2 n)$

Recurrence Tree Method:

(1) $T(n) = 3T\left(\frac{n}{4}\right) + n^2$ ← root



$\Rightarrow$ $Cn^2 - Cn^2$

$C\left(\frac{n}{4}\right)^2 - \frac{3}{16}Cn^2$

$C\left(\frac{n}{16}\right)^2 - \frac{9}{256}Cn^2$

* Depth of the tree:

$\frac{n}{4^i} = 1 \Rightarrow n = 4^i$

$\log n = \log 4^i$

$\log n = i \log 4$

$i = \log_4 n$

* No. of leaves:

$3^i = 3^{\log_4 n}$

$\approx n^2$

unacademy

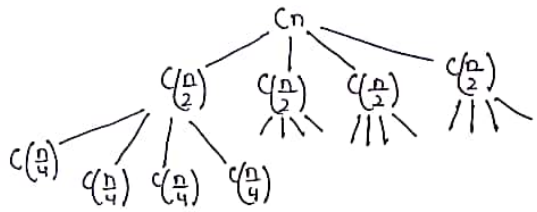
∴ Total Cost

$$T(n) = Cn^2 + \frac{3}{16}Cn^2 + \left(\frac{3}{16}\right)^2 Cn^2 + \ldots\ldots + \Theta(3^{\log_4 n})$$

$$\boxed{T(n) = \Theta(n^2)}$$

unacademy

(2.) $T(n) = 4T\left(\frac{n}{2}\right) + n$



Cn

$4\frac{Cn}{2} = 2Cn$

4Cn

⋮

✱ Depth of tree :

$\frac{n}{2^i} = 1$

$n = 2^i$

$\log n = \log 2^i$

$\log n = i \log 2$

$\boxed{i = \log_2 n}$

✱ No. of leaves :

$4^i = 4^{\log_2 n}$

$= 2^{\log_2 n^2}$

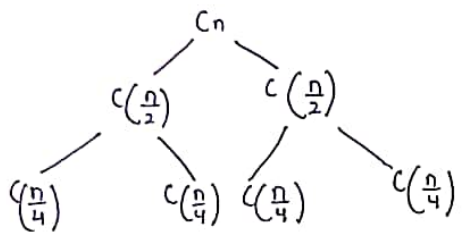
$= n^2$

∴ Total cost = $Cn + 2Cn + 4Cn + \ldots\ldots + \Theta(n^2)$

$\boxed{T(n) = \Theta(n^2)}$

unacademy

(3.) $T(n) = 2T\left(\frac{n}{2}\right) + n$



— Cn

— Cn

— Cn

✱ Depth of the tree :

$\frac{n}{2^i} = 1 \Rightarrow \boxed{i = \log_2 n}$

✱ Level = Dept + 1

$= \log_2 n + 1$

✱ Total Cost = cost of each level x No. of level

$= Cn(\log_2 n + 1)$

$= Cn\log_2 n + Cn$

$\boxed{T(n) = Cn\log_2 n}$

unacademy

Q. fib(n)

```
if(n=0)
  return 0
if (n=1)
  return 1
else
  return fib(n-1) + fib(n-2)
```

$$fib(n) = \begin{cases} 0, & \text{if } n=0 \\ 1, & \text{if } n=1 \\ fib(n-1) + fib(n-2), & n>1 \end{cases}$$

$$T(n) = T(n-1) + T(n-2) + C$$

Q. What is the time complexity?

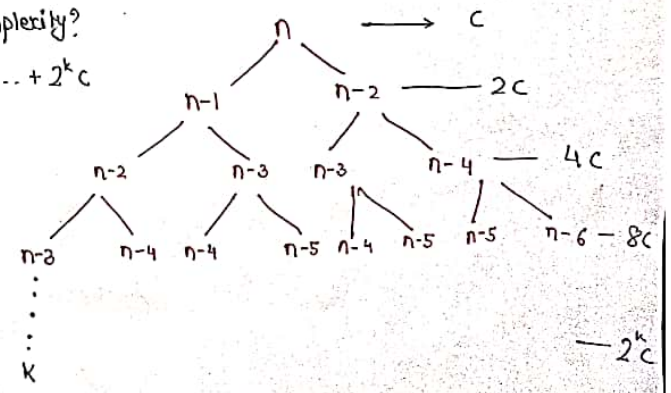
$C + 2C + 4C + 8C + \dots + 2^k C$

$C(1 + 2 + 4 + 8 + \dots + 2^k)$

$C \cdot \frac{1(2^k - 1)}{2-1}$

$C \cdot (2^k - 1) \approx 2^k C$

$O(2^n)$



GATE-2007

unacademy

Q. What is the time complexity of the following recursive function?

```
int DoSomething (int n)
{  if (n<=2)
     return 1;
   else
     return (DoSomething (floor (sqrt(n)) + n);
}
```

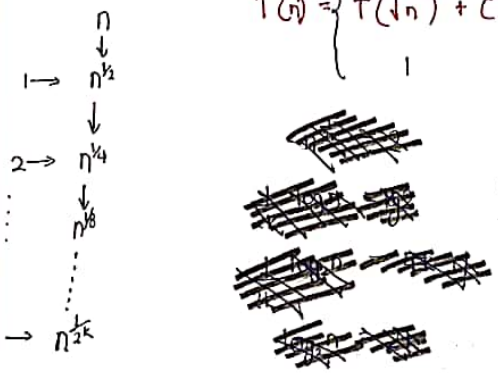
(a) $\Theta(n^2)$

(b) $\Theta(n \log_2 n)$

(c) $\Theta(\log_2 n)$

(d) $\Theta(\log_2 \log_2 n)$

$$T(n) = \begin{cases} T(\sqrt{n}) + C, & n>2 \\ 1, & n<=2 \end{cases}$$



$n^{\frac{1}{2^k}} = 2$

$\frac{1}{2^k} \log n = \log_2 2$

$\log_2 n = 2^k$

$\log \log_2 n = k \log 2$

$k = \log_2 \log_2 n$

$\Theta(k) = \Theta(\log_2 \log_2 n)$

unacademy

GATE

Q. Consider the following recurrence:

$T(n) = 2T(\sqrt{n}) + 1, \quad T(1) = 1$

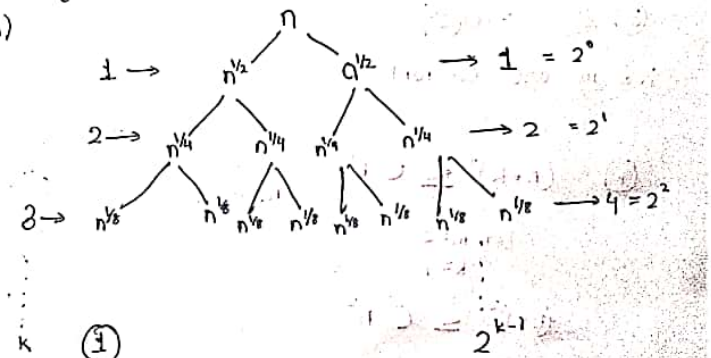
Which one of the following is true?

(a) $T(n) = \Theta(\log \log n)$

(b) $T(n) = \Theta(\log n)$

(c) $T(n) = \Theta(\sqrt{n})$

(d) $T(n) = \Theta(n)$



(1)

$T(n) = 1 + 2^1 + 2^2 + \dots + 2^{k-1}$

$T(n) = 2^{k-1}$

$= 2^{\log_2 \log_2 n - 1}$

$= \log_2 n - 1$

$T(n) = \Theta(\log_2 n)$

unacademy

Master's Theorem :

$$T(n) = a\,T\left(\frac{n}{b}\right) + F(n)$$

① Root level cost $>$ $f(n)$

$f(n) = O(n^{\log_b a - \varepsilon})$ for $\varepsilon > 0$

$T(n) = \Theta(n^{\log_b a})$

② Root level cost $=$ $f(n)$

$f(n) = \Theta(n^{\log_b a})$

$T(n) = \Theta(n^{\log_b a} \log n)$

③ Root level cost $<$ $f(n)$

$f(n) = \Omega(n^{\log_b a + \varepsilon})$

$T(n) = \Theta(F(n))$

unacademy

① $T(n) = T\left(\frac{n}{2}\right) + n$

$F(n) = n$, $f(n) = n^{\log_b a}$

$= n^{\log_2 1}$

$= n^0$

$T(n) = \Theta(n)$

② $T(n) = 4T\left(\frac{n}{4}\right) + n^2$

$F(n) = n^2$, $f(n) = n^{\log_b a}$

$= n^{\log_4 4}$

$= n$

$T(n) = \Theta(n^2)$

③ $T(n) = 27T\left(\frac{n}{3}\right) + n^3$

$F(n) = n^3$, $f(n) = n^{\log_3 27}$

$= n^{\log_3 3^3}$

$= n^3$

$T(n) = \Theta(n^3 \log n)$

④ $T(n) = 9T\left(\frac{n}{3}\right) + 4n^6$

$F(n) = 4n^6$, $f(n) = n^{\log_3 9}$

$= n^2$

$T(n) = \Theta(n^6)$

unacademy

Extended Master Theorem :

$$T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^k \log^p n)$$

① If $a > b^k$ then $T(n) = \Theta(n^{\log_b a})$

② If $a = b^k$ then

(i) $p > -1 \longrightarrow T(n) = \Theta(n^{\log_b a} \log^{p+1} n)$

(ii) $p = -1 \longrightarrow T(n) = \Theta(n^{\log_b a} \log\log n)$

(iii) $p < -1 \longrightarrow T(n) = \Theta(n^{\log_b a})$

③ If $a < b^k$

(i) $p \geq 0 \longrightarrow T(n) = \Theta(n^k \log^p n)$

(ii) $p < 0 \longrightarrow T(n) = \Theta(n^k)$

unacademy

Q. $T(n) = 2T\left(\frac{n}{2}\right) + n\log^2 n$

$a = 2 \;,\; b = 2$
$k = 1 \;,\; p = 2$

$a = b^k$
$2 = 2^1$ ✓✓ → case (2.)

$T(n) = \Theta(n^{\log_b a} \log^{p+1} n)$

$= \Theta(n^{\log_2 2} \log^{2+1} n)$

$= \Theta(n\log^3 n)$

$\boxed{T(n) = \Theta(n\log^3 n)}$ ✓

unacademy

Master's बाबा का एक और Version :

$\boxed{T(n) = aT(n-b) + O(n^k)}$ $\quad a>0;\; b\geq 1;\; k\geq 0$

Case1: if $a>1$ then $T(n) = O(n^k a^{n/b})$

Case2: if $a=1$ then $T(n) = O(n^{k+1})$

Case3: if $a<1$ then $T(n) = O(n^k)$

Q. $T(n) = 2T(n-1) + c$

$a = 2 \;,\; b = 1 \;,\; k = 0$

$T(n) = O(n^0 2^{n/1})$

$\boxed{T(n) = O(2^n)}$ ✓

unacademy

Greedy Technique :

→ Follow local optimal choice of each stage with intend of finding global optimum solution.

→ अपने को Feasible Solution नहीं Balki Optimal Solution chye!

→ Lagao अपना Dimaag Mil jaaega Optimal Solution.

Applications :
1. Knapsack Problem → Min. cost
2. Job Sequencing → Max Profit
3. Minimum Spanning Tree → Min Risk
4. Optimal Merge Pattern
5. Huffman Coding
6. Dijkstra's Algorithm

Huffman Coding :

unacademy

→ Message को Encryption और Decryption करने के लिए use krte!

→ No. of bits कम से कम लगे!

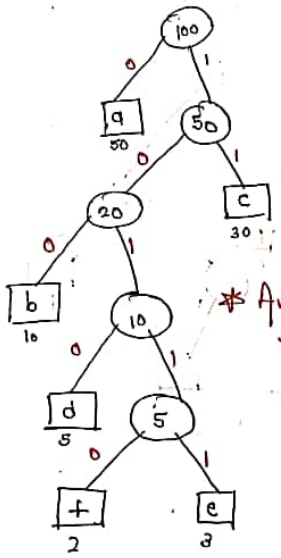
Q.

a = 50	a = 000
b = 10	b = 001
c = 30	c = 010
d = 5	d = 011
e = 3	e = 100
f = 2	f = 101

Total M = 100 ch^r

→ ASCII में 7 bit लगेगा एक के लिए!

→ 7 × 100 = 700 bits



a = 0	50 × 1 = 50
b = 100	10 × 3 = 30
c = 11	30 × 2 = 60
d = 1010	5 × 4 = 20
e = 10111	3 × 5 = 15
f = 10110	2 × 5 = 10
	185 bits.

* Avg. bits required to represent each character

$= \frac{185}{100}$

= 1.85 bits/character

unacademy

** Q. Consider the following message :

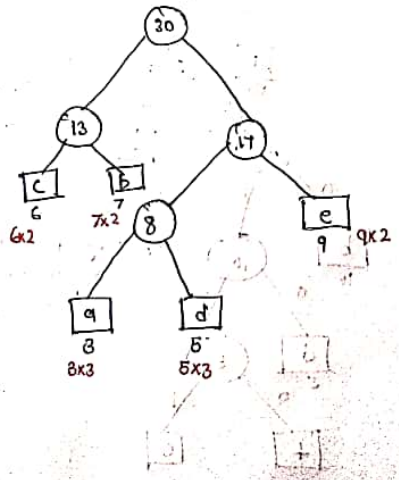
aa bbbb a bbb ccc ddd eee ccc eee dd eee

(i) Cal. the no. of bits required for Huffman encoding of above message?

(ii) Find the avg. bits required to represent

a = 3
b = 7
c = 6
d = 5
e = 9

M = 31



12
14
9
15
18
68

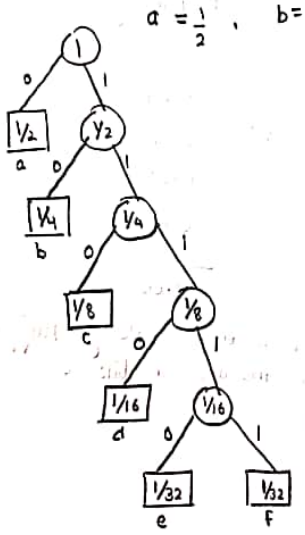
unacademy

GATE-2007

Q. Suppose the letters a, b, c, d, e, f have probabilities $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}, \frac{1}{32}$ respectively. What is the average length of the Huffman code for the letters a, b, c, d, e, f ?

(a) 3
(b) 2.1875
(c) 2.25
(d) 1.9375

$a = \frac{1}{2}$, $b = \frac{1}{4}$, $c = \frac{1}{8}$, $d = \frac{1}{16}$, $e = \frac{1}{32}$, $f = \frac{1}{32}$



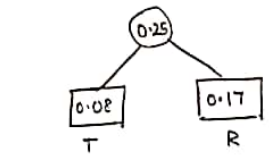
a = 0	$1 \times \frac{1}{2}$
b = 10	$2 \times \frac{1}{4}$
c = 110	$3 \times \frac{1}{8}$
d = 1110	$4 \times \frac{1}{16}$
e = 11110	$5 \times \frac{1}{32}$
f = 11111	$5 \times \frac{1}{32}$
	1.9375

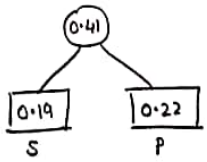
Q. A message is made up entirely of character set $X = \{P, Q, \ldots\}$. The table of probabilities for each is shown below:

Character	Probability
P	0.22
Q	0.34
R	0.17
S	0.19
T	0.08
Total	1.00

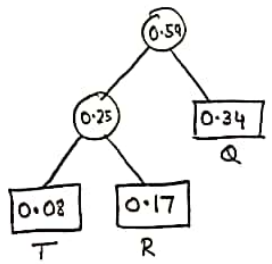
If a message of 100 characters over X is using Huffman coding, then the expected length of the encoded message in bits is

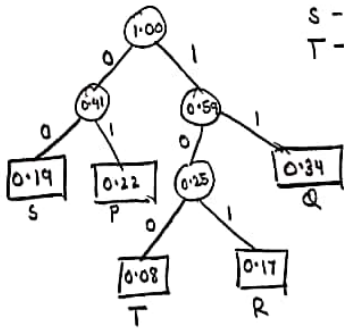
P = 0.22 Q = 0.34 R = 0.17 S = 0.19 T = 0.08





P - 01 — 2 × 0.22 = 0.44
Q - 11 — 2 × 0.34 = 0.68
R - 101 — 3 × 0.17 = 0.51
S - 00 — 2 × 0.19 = 0.38
T - 100 — 3 × 0.08 = 0.24
2.25



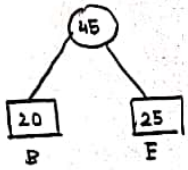


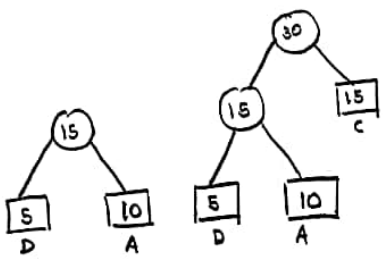
225

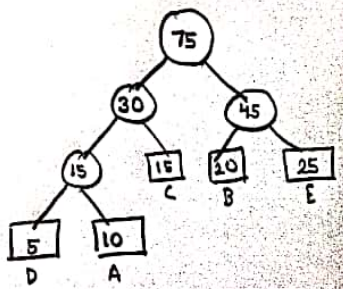
GATE-

Q. The minimum no. of record movements required to merge five files (containing A (10 records), B (20 records), C (15 records), D (5 records), E (25 records)) is

(a) 165
(b) 90
(c) 75
(d) 65







No. of movements = 15 + 30 + 45 + 75
= 165

Job Sequencing :

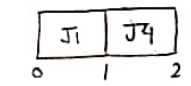
Jobs	J_1	J_2	J_3	J_4
Profit	50	15	10	25
Deadline	2	1	2	1

Algorithm :

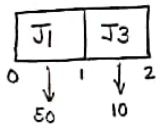
(1) Arrange All jobs in decreasing order of profit. $O(n \log n)$

(2) For each job (m_i), do linear search to find particular slot in array of size (n), where
n = maximum dead line $O(mn)$
m = total jobs

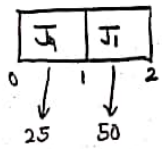
* Overall Time Complexity: $O(mn)$ or $O(n^2)$

* 

X

* 

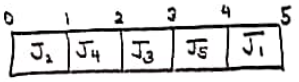
50 + 10 = 60

* 

25 + 50 = 75

Q.

	J_1	J_2	J_3	J_4	J_5	J_6
D:	5	3	3	2	4	2
P:	15	10	12	20	8	5

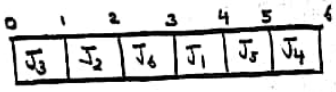


= 20 + 15 + 12 + 10 + 8

= 65

Q.

	J_1	J_2	J_3	J_4	J_5	J_6	J_7
D:	4	2	2	6	5	3	1
P:	60	40	20	70	50	30	10



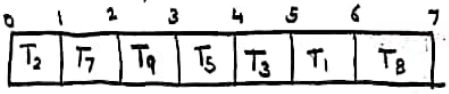
= 70 + 60 + 50 + 40 + 30 + 20

= 270

GATE Q. We are given 9 tasks $T_1, T_2, \ldots T_9$. The execution of each task requires one unit of time. We can execute one task at a time. Each task T_i has a profit P_i and a deadline d_i. Profit P_i is earned if the task is completed before the end of the d_i^{th} unit of time.

Task	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
Profit	15	20	30	18	18	10	23	16	25
Deadline	7	2	5	3	4	5	2	7	3

What is the maximum profit earned?



30 + 25 + 23 + 20 + 18 + 16 + 15

147

unacademy

GATE

Q. We are given 9 tasks $T_1, T_2, \ldots T_9$. The execution of each task requires one unit of time. We can execute one task at a time. Each task T_i has a profit P_i and a deadline d_i. Profit P_i is earned if the task is completed before the end of the d_i^{th} unit of time.

Task	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
Profit	15	20	30	18	18	10	23	16	25
Deadline	7	2	5	3	4	5	2	2	3

(a) All tasks are completed

(b) T_1 & T_6 are left out

(c) T_1 & T_8 are left out

(d) T_4 & T_6 are left out

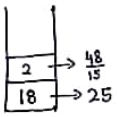
unacademy

Knapsack Problem:

Objects	obj1	obj2	obj3
Profit	25	24	15
Weight	18	15	10
P/w	1·3	1·6	1·5

Knapsack capacity
$M = 20$

★ Profit का Greedy करें



$= 25 + \frac{48}{15}$

$= 25 + 3.2$

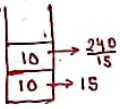
$= 28.2$

$15 - 24$

$1 - \frac{24}{15}$

$2 - \frac{24}{15} \times 2$

★ Weight का Greedy करें



$= 15 + \frac{240}{15}$

$= 15 + 16$

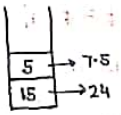
$= 31$

$15 - 24$

$1 - \frac{24}{15}$

$10 - \frac{240}{15}$

★ अब समझ आया Apun को Greedy कैसे Bane:



$= 24 + 7.5$

$= 31.5$

$10 - 15$

$1 - \frac{15}{10}$

$5 - \frac{15^3}{10_2} \times 5$

unacademy

Algorithm:

1. for $i=1$ to n
   Calculate P/w } $O(n)$

2. Sort objects in decreasing order of P/w } $O(n \log n)$

3. for $i=1$ to n
   if $M > 0$ & $w_i \le M$
   $M = M - w_i$
   $P = P + P_i$
   else break;
   if $(M > 0)$
   $P = P + P_i\left(\frac{M}{w_i}\right)$

★ Overall Time complexity:

$O(n \log n)$

unacademy

GATE-18
Q. Consider the weights and values of items listed below. Note that there is only one unit of each item.

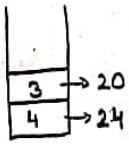
Item No	Weight (kg)	Value (in ₹)	V/W
1	10	60	6
2	7	28	4
3	4	20	5
4	2	24	12

The task is to pick a subset of these items such that their total weight is no more than 11 kgs and their total value is maximized. Moreover, no item may be split. The total value of items picked by an optimal algorithm is denoted by V_{opt}. A greedy algorithm sorts the items by their value-to-weight ratios in descending order and packs them greedily, starting from the first item in the ordered list. The total value of items picked by the greedy algorithm is denoted by V_{greedy}.

The value of $V_{opt} - V_{greedy}$ is _____.

unacademy

Item No.	Weight	Value	Value/Weight
4	2	24	12
1	10	60	6
3	4	20	5
2	7	28	4



~~4~~ 5

Knapsack capacity = 11

$V_{opt} = 60$ (Dekh के बता सकता कोई भी!)

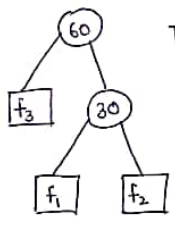
$V_{greedy} = 20 + 24 = 44$

$V_{opt} - V_{greedy} = 60 - 44 = 16$

unacademy

Optimal Merge Pattern:

f_1	f_2	f_3
10	20	30



Total record movement
= 60 + 30
= 90

∴ Time complexity = O(nlogn)

GATE
Q. The minimum no. of record movements required to merge five files A (with 10 records), B (with 20 records), C (with 15 records) and D (with 5 records) and E (with 25 records) is:

(a) 165 (b) 90
(c) 75 (d) 65

A	B	C	D	E
10	20	15	5	25

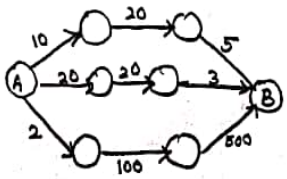
करलो रबुद!

Dynamic Programming :

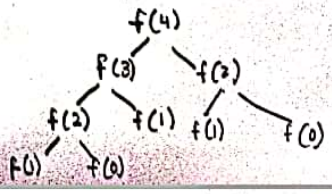
→ It divide the problem into series of overlapping sub-problems.

→ Its main feature :
- → Optimal substructure
- → Overlapping problems

→ Divide & Conquer की जैसा ही है पर यहा repeation of subproble हो सकती है जिसको अपन Dynamically solve करेंगे!



$$f(n) = \begin{cases} 0, & n=0 \\ 1, & n=1 \\ f(n-1)+f(n-2), & n>1 \end{cases}$$



→ LCS
→ Matrix Chain multiplication
→ 0/1 Knapsack Problem
→ TSP
→ Flyodd Warshall algorithm
→ Sum of Subset problem
→ optimal cost BST

Longest common Subsequence :

ABC

→ ϕ
→ A, B, C
→ AB, BC, AC
→ ABC

$2^3 = 8$

GATE

→ ϕ
→ G, A, T, E
→ GA, AT, TE, GE, AE, GT
→ GAT, GAE, ATE, GTE
→ GATE

$2^4 = 16$

for example :

$X = \langle A\ B\ C\ B\ D\ A\ B \rangle$

$Y = \langle B\ D\ C\ A\ B\ A \rangle$

LCS = < B D A B >

LCS = < B C B A >

LCS = < B C A B >

unacademy

* The recursive solution of LCS is :

if we assume $l[i,j]$ = length of LCS of x_i, y_j

Then

$$l[i,j] = \begin{cases} 0 , & \text{if } i=0 \text{ or } j=0 \\ l[i-1,j-1]+1 , & \text{if } i,j>0 \ \& \ x_i = y_j \\ \max(l[i-1,j], l[i,j-1]) , & \text{if } i,j>0 , \ x_i \neq y_j \end{cases}$$

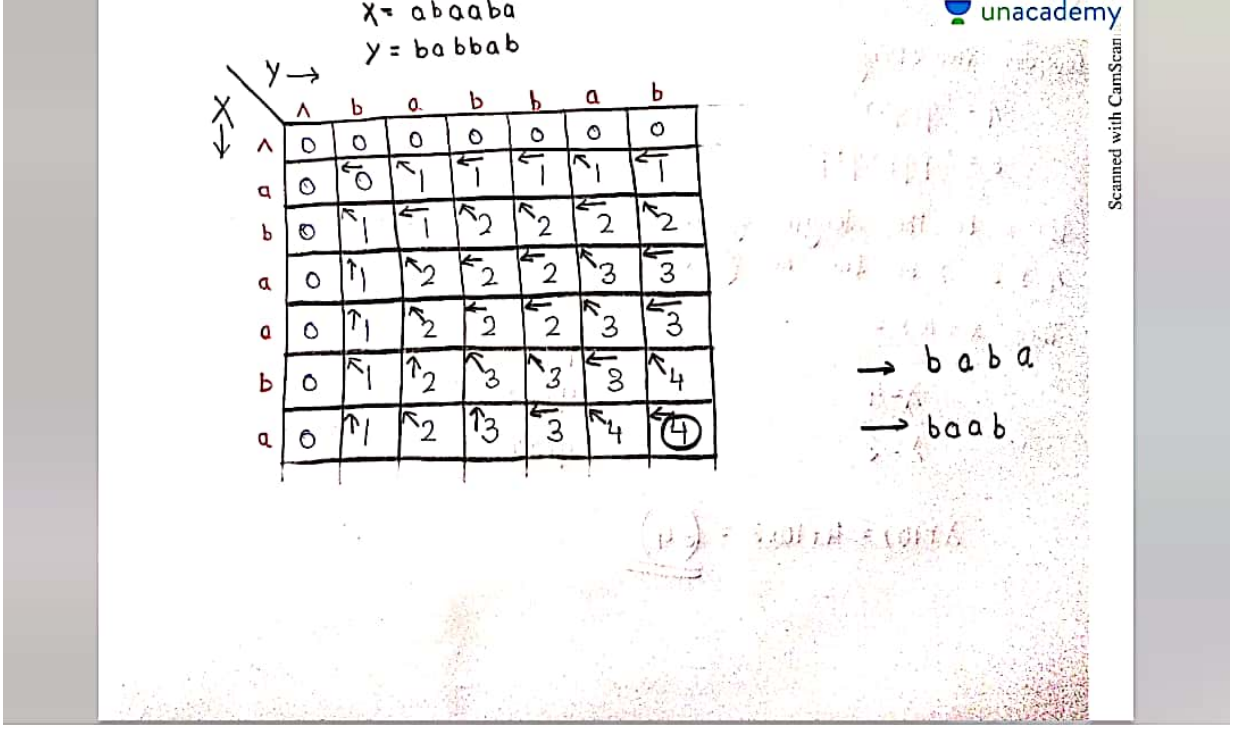
unacademy

LCS_length (X, Y) :

```
1.  n = |X|
2.  m = |Y|
3.  for i=0 to n do
4.      l[i,0] = 0
5.  for j=0 to m do
6.      l[0,j] = 0
7.  for i=1 to n do
8.      for j=1 to m do
9.          if x[i] = y[j] then
10.             begin l[i,j] = l[i-1,j-1] + 1
11.                 b[i,j] = "↖" end
12.         else if l[i-1,j] ≥ l[i,j-1] then
13.             begin l[i,j] = l[i-1,j] &
14.                 b[i,j] = "↑" end
15.         else
              begin l[i,j] = l[i,j-1]
                    b[i,j] = "←" end
16. Return l & b
```


unacademy

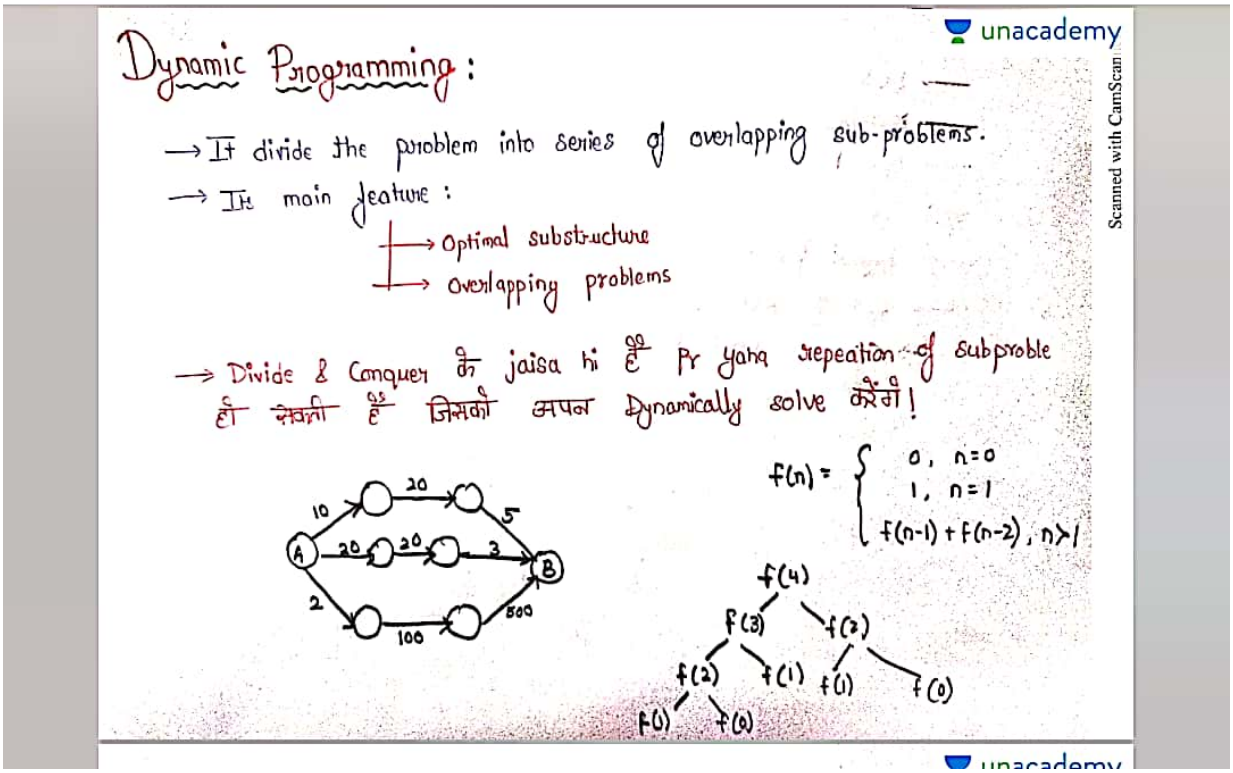
```
if ( x[i] == y[j])
      c[i][j] = 1 + c[i-1][j-1]

else
      c[i][j] = max(c[i][j-1] , c[i-1][j])
```



Test Preparation > GATE & ESE > Data Structures > Data Structure

Longest Common Subsequence with Gate-2014 question (in Hindi)

LESSON 50 OF 50



→ LCS
→ Matrix chain multiplication
→ 0/1 Knapsack Problem
→ TSP
→ Flyodd Warshall algorithm
→ Sum of Subset problem
→ Optimal cost BST

unacademy

Longest common Subsequence :

ABC

→ ϕ
→ A, B, C
→ AB, BC, AC
→ ABC

$2^3 = 8$

GATE

→ ϕ
→ G, A, T, E
→ GA, AT, TE, GE, AE, GT
→ GAT, GAE, ATE, GTE
→ GATE

$2^4 = 16$

for example :

$X = \langle A\ B\ C\ B\ D\ A\ B \rangle$

$Y = \langle B\ D\ C\ A\ B\ A \rangle$

LCS = < B D A B >
LCS = < B C B A >
LCS = < B C A B >

* The recursive solution of LCS is :

if we assume $L[i,j]$ = length of LCS of x_i, y_j

Then

$$L[i,j] = \begin{cases} 0, & \text{if } i=0 \text{ or } j=0 \\ L[i-1, j-1] + 1, & \text{if } i,j > 0 \ \& \ x_i = y_j \\ \max(L[i-1,j], L[i,j-1]), & \text{if } i,j>0,\ x_i \neq y_j \end{cases}$$

unacademy

LCS_length (X, Y) :

```
1.  n = |X|
2.  m = |Y|
3.  for i=0 to n do
4.      l[i,0] = 0
5.  for j=0 to m do
6.      l[0,j] = 0
7.  for i=1 to n do
8.      for j=1 to m do
9.          if x[i] = y[j] then
10.             begin l[i,j] = l[i-1, j-1] + 1
11.                 b[i,j] = "↖" end
12.         else if l[i-1, j] ≥ l[i, j-1] then
13.             begin l[i,j] = l[i-1, j] &
14.                 b[i,j] = "↑" end
15.         else
              begin l[i,j] = l[i, j-1]
                  b[i,j] = "←" end
16. Return l & b
```


unacademy

```
if ( x[i] == y[j])
      c[i][j] = 1 + c[i-1] [j-1]

else
      c[i][j] = max(c[i][j-1] , c[i-1] [j])
```

Scanned with CamScan

unacademy

X = a b a a b a

Y = b a b b a b

X ↓ \ Y →	^	b	a	b	b	a	b
^	0	0	0	0	0	0	0
a	0	←0	↖1	←1	←1	↖1	←1
b	0	↖1	←1	↖2	↖2	←2	↖2
a	0	↑1	↖2	←2	←2	↖3	←3
a	0	↑1	↖2	←2	←2	↖3	←3
b	0	↑1	↖2	↖3	↖3	←3	↖4
a	0	↑1	↖2	↑3	←3	↖4	←4

→ b a b a

→ b a a b

Scanned with CamScanner

GATE-14

Q. Consider two strings:

A = "qpqrr"

B = "pqprqrp"

Let x be the length of the longest common subsequence between A & B. y be the no. of such LCS between A & B

Then, $x + 10y =$

pqrr
qpqr
qprr

$X = 4$

$Y = 3$

$X + 10Y = 4 + 10 \times 3 = \boxed{34}$

GATE-09

Q. A subsequence of a given sequence is just the given sequence with some elements (possibly none or all) left out. We are given two sequences $X[m]$ & $Y[n]$ of lengths m and n, respectively with indexes of X & Y starting from 0.

We wish to find the length of the longest common sub-sequence (LCS) of $X[m]$ and $Y[n]$ as $l(m,n)$ where an incomplete recursive definition for the function $l(i,j)$ to compute the length of the LCS of $X[m]$ and $Y[n]$ is given below:

$l(i,j) = 0$, if either $i=0$ or $j=0$

$= expr1$, if $i,j>0$ & $X[i-1] = Y[j-1]$

$= expr2$, if $i,j>0$ & $X[i-1] \neq Y[j-1]$

Which one of the following options are correct?

(a) $expr1 = l(i-1,j)+1$

(b) $expr1 = l(i,j-1)$

(c) $expr2 = max(l(i-1,j), l(i,j-1))$

(d) $expr2 = max(l(i-1,j-1), l(i,j))$

GATE-09

Q. A sub-sequence of a given sequence ..... starting from 0.

मात्र कुछ Pichle ques जो Likha है!

The value of $l(i,j)$ could be obtained by dynamic programming based on the correct recursive definition of $l(i,j)$ of the form given below, using an array $L[M,N]$ where, $M = m+1$ & $N = n+1$ such that $L[i,j] = l(i,j)$

Which of the following statements would be TRUE regarding the dynamic programming solution for the recursive definition of $l(i,j)$?

(a) All elements of L should be initialized to 0 for the values of $l(i,j)$ to be properly computed.

(b) The values of $l(i,j)$ may be computed in a row major order or column major order of $L[M,N]$.

(c) The values of $l(i,j)$ cannot be computed in either row major order or column major order of $L[M,N]$.

(d) $L[p,q]$ needs to be computed before $L[r,s]$ if either $p<r$ or $q<s$.

unacademy

GATE-10

Q. The weight of a sequence $a_0, a_1, \ldots a_{n-1}$ of real no. is defined as $a_0 + a_1/2 + \ldots + a_{n-1}/2^{n-1}$. A subsequence of a sequence is obtained by deleting some elements from the sequence, keeping the order of the remaining elements the same. Let X denote the maximum possible weight of a subsequences of $a_0, a_1, \ldots a_{n-1}$ and Y the maximum possible weight of a subsequence of $a_1, a_2 \ldots a_{n-1}$. Then X is equal to

(a) $\max(Y, a_0 + Y)$
(b) $\max(Y, a_0 + Y/2)$
(c) $\max(Y, a_0 + 2Y)$
(d) $a_0 + Y/2$.

$S = \langle a_0, S_1 \rangle$

$S_1 = \langle a_1, a_2 \ldots a_{n-1} \rangle$

Case1: जब a_0 होगा :

$X = \text{weight}(\langle a_0, S_1 \rangle) = a_0 + \frac{1 \cdot Y}{2}$

Case 2: जब a_0 नही होगा :

$X = \text{weight}(\langle S_1 \rangle) = Y$

$\therefore$ Ans - $\boxed{\max\left(Y, a_0 + \frac{Y}{2}\right)}$

Scanned with CamScanner

unacademy

GATE-11

Q. An algorithm to find the length of the longest monotonically increasing sequence is in an array $A[0 : n-1]$ is given below:

Let L_i denote the length of the longest monotonically increasing sequence starting at index i in the array.

Initialize $L_{n-1} = 1$

For all i such that $0 \le i \le n-2$

$$L_i = \begin{cases} 1 + L_{i+1}, & \text{if } A[i] < A[i+1] \\ 1, & \text{otherwise} \end{cases}$$

$n = 5$
$L_4 = 1$

$0 \le i \le 3$

$L_0 = 1 + L_1$
$L_1 = 1 + L_2$
$L_3 = 1 + L_4$

Finally, the length of the longest monotonically increasing sequence is $\max(L_0, L_1, \ldots L_{n-1})$.

Which of the following statements is TRUE?

(a) The algo uses dynamic programming paradigm.
(b) The algo has a linear complexity and uses branch & bound paradigm.
(c) The algo has a non-linear polynomial complexity and uses B & B paradigm.
(d) The algo uses divide & conquer paradigm.

Scanned with CamScanner

unacademy

Matrix chain multiplication :

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}_{2\times3} \qquad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix}_{3\times2}$$

$2 \times 3 \quad | \quad 2\times3\times2 \quad | \quad 3 \times 2$

$$C = A \times B = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} \end{bmatrix}_{2\times2}$$

$A_1 \times A_2 \times A_3$

$2 \times 3 \quad 3 \times 4 \quad 4 \times 2$
$d_0 \; d_1 \quad d_1 \; d_2 \quad d_2 \; d_3$

$(A_1 \times A_2) \times A_3$ $\qquad$ $A_1 \times (A_2 \times A_3)$

Scanned with CamScanner

$A_1 \times A_2 \times A_3$

2 3 3 4 4 2

d_0 d_1 d_1 d_2 d_2 d_3

$(A_1 \times A_2) \times A_3$

$2 \times 3 \times 4 = (24)$

$A_1 \times (A_2 \times A_3)$

$3 \times 4 \times 2 = (24)$

*Time complexity :

→ n matrice

→ O(n) time to generate each of the $O(n^2)$ cost

→ ∴ Overall Time complexity = $O(n^3)$

$(A_1 \times A_2) \times A_3$

$2 \times 4 \quad 4 \times 2$

$2 \times 4 \times 2 = (16)$

$24 + 16 = (40)$

$A_1 \times (A_2 \times A_3)$

$2 \quad 3 \quad 3 \quad 2$

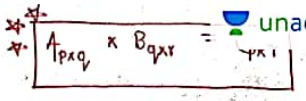
$2 \times 3 \times 2 = (12)$

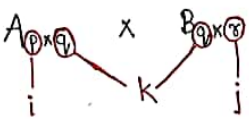
$24 + 12 = (36)$

इसका संतलब समझा !

multiplication = p*q*r

addition = p*(q-1)*r






```
for i=1 to p do
    for j=1 to r do
    {
        c[i,j] = 0;
        for (k=1 to q do)
        {
            c[i,j] = c[i,j] + a[i,k] * b[k,j];
        }
    }
```


$$m[i,j] = \min_{i \le k < j} \{ c[i,k] + c[k+1,j] + d_{i-1} \times d_k \times d_j \}$$

$$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min_{i \le k < j} \{ m[i,k] + m[k+1,j] + P_{i-1} P_k P_j \} & \text{if } i < j \end{cases}$$

where, $m[i,j]$ → min no. of multiplication need to compute $A_{i \ldots j}$

Q. Find the optimal parenthesization of a matrix chain p... whose sequence of dimension given below:

$\langle 4, 10, 3, 12, 20, 7 \rangle$

j →

	1	2	3	4	5
1	0	k=1 120	k=2 264	k=2 1080	k=2 1344
2		0	k=2 360	k=1 1320	k=1 1350
3			0	k=3 720	k=2 1140
4				0	k=4 1680
5					0

i ↓

4, 10, 3, 12, 20, 7
↓ ↓ ↓ ↓ ↓ ↓
P_0 P_1 P_2 P_3 P_4 P_5

unacademy

Scanned with CamScanner

★ $m[1,2] = m[1,1] + m[2,2] + P_0 P_1 P_2$
$= 0 + 0 + 4 \times 10 \times 3$
$= 120$

★ $m[2,3] = m[2,2] + m[3,3] + P_1 P_2 P_3$
$= 0 + 0 + 360 = 360$

★ $m[1,3] = m[1,2] + m[3,3] + P_0 P_1 P_3$
$= 120 + 0 + 4 \times 3 \times 12$
$= 264$

★ $m[1,5] = 1344$

$i=1, j=2$
$i \le k < j$
$1 \le k < 2$

$(A_1 A_2 A_3 A_4 A_5)$
↓ k=2
$(A_1 A_2)(A_3 A_4 A_5)$
↓ k=4
$(A_1 A_2)((A_3 A_4) A_5)$
↓
min. no. of multiplication.

unacademy

Scanned with CamScanner

Q. Find an optimal parenthesation of matrix chain multiplicat... whose sequence of dimension is given below:

$A_1 = 30 \times 35$
$A_2 = 35 \times 15$
$A_3 = 15 \times 5$
$A_4 = 5 \times 10$
$A_5 = 10 \times 20$
$A_6 = 20 \times 25$

$\langle P_0\ P_1\ P_2\ P_3\ P_4\ P_5\ P_6 \rangle$
$\langle 30\ 35\ 15\ 5\ 10\ 20\ 25 \rangle$

★ Space Complexity = Input + Extra
= <sequence of dim> + Extra
↳ n

Extra: Stack (n); matrix ⇒ $\frac{n^2}{2} \times 2$
1 for no.
2 for k

$= 2n + \frac{n^2}{2} \times 2$
$= O(n^2)$

unacademy

Scanned with CamScanner

unacademy

Matrix chain multiplication :

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}_{2\times3} \qquad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix}_{3\times2}$$

$2 \times 3 \quad | \quad 2\times3\times2 \quad | \quad 3\times2$

$$C = A \times B = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} \end{bmatrix}_{2\times2}$$

$$A_1 \times A_2 \times A_3$$

$2\times3 \quad 3\times4 \quad 4\times2$

$d_0 \; d_1 \quad d_1 \; d_2 \quad d_2 \; d_3$

$(A_1 \times A_2) \times A_3 \qquad A_1 \times (A_2 \times A_3)$

unacademy

$$A_1 \times A_2 \times A_3$$

$2 \quad 3 \quad 3 \quad 4 \quad 4 \quad 2$

$d_0 \; d_1 \quad d_1 \; d_2 \quad d_2 \; d_3$

$(A_1 \times A_2) \times A_3$

$2 \quad 3 \quad 3 \quad 4 \quad 4 \quad 2$

$2\times3\times4 = (24)$

$(A_1 \times A_2) \times A_3$

$2\times4 \quad 4\times2$

$2\times4\times2 = (16)$

$24 + 16 = (40)$

$A_1 \times (A_2 \times A_3)$

$2 \quad 3 \quad 3 \quad 4 \quad 4 \quad 2$

$3\times4\times2 = (24)$

$A_1 \times (A_2 \times A_3)$

$2 \quad 3 \quad 3 \quad 2$

$2\times3\times2 = (12)$

$24 + 12 = (36)$

इसका संतलब सम हुआ !

★ Time complexity :

→ n matrice

→ O(n) time to generate each of the $O(n^2)$ cost

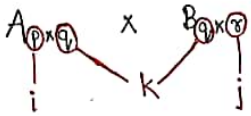
→ ∴ Overall Time complexity = $O(n^3)$

unacademy

$$A_{p\times q} \times B_{q\times r} = C_{p\times r}$$

multiplication = $p * q * r$

addition = $p * (q-1) * r$




```
for i=1 to p do
    for j=1 to r do
    {
        c[i,j] = 0;
        for (k=1 to q do)
        {
            c[i,j] = c[i,j] + a[i,k] * b[k,j];
        }
    }
```

$$m[i,j] = \min_{i \le k < j} \{ c[i,k] + c[k+1,j] + d_{i-1} \times d_k \times d_j \}$$

$$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min_{i \le k < j} \{ m[i,k] + m[k+1,j] + P_{i-1} P_k P_j \} & \text{if } i < j \end{cases}$$

where, $m[i,j] \rightarrow$ min no. of multiplication need to compute $A_{i \ldots j}$

Q. Find the optimal parenthesization of a matrix chain product whose sequence of dimension given below:

$\langle 4, 10, 3, 12, 20, 7 \rangle$

$j \rightarrow$

	1	2	3	4	5
1	0	k=1 120	k=2 264	k=2 1080	k=2 1344
2		0	k=2 360	k=1 1320	k=1 1350
3			0	k=3 720	k=2 1140
4				0	k=4 1680
5					0

$i \downarrow$

4, 10, 3, 12, 20, 7
P_0 P_1 P_2 P_3 P_4 P_5

★ $m[1,2] = m[1,1] + m[2,2] + P_0 P_1 P_2$
$= 0 + 0 + 4 \times 10 \times 3$
$= 120$

★ $m[2,3] = m[2,2] + m[3,3] + P_1 P_2 P_3$
$= 0 + 0 + 360 = 360$

★ $m[1,3] = m[1,2] + m[3,3] + P_0 P_1 P_3$
$= 120 + 0 + 4 \times 3 \times 12$
$= 264$

★ $m[1,5] = 1344$

$i=1, j=2$
$i \le k < j$
$1 \le k < 2$

$(A_1 A_2 A_3 A_4 A_5)$
$\downarrow k=2$
$(A_1 A_2)(A_3 A_4 A_5)$
$\downarrow k=4$
$(A_1 A_2)((A_3 A_4) A_5)$
$\downarrow$
min no. of multiplication.

Q. Find an optimal parenthesization of matrix chain multiplication whose sequence of dimension is given below:

$A_1 = 30 \times 35$

$A_2 = 35 \times 15$

$A_3 = 15 \times 5$

$A_4 = 5 \times 10$

$A_5 = 10 \times 20$

$A_6 = 20 \times 25$

$\langle P_0\ P_1\ P_2\ P_3\ P_4\ P_5\ P_6 \rangle$

$30\ \ 35\ \ 15\ \ 5\ \ 10\ \ 20\ \ 25$

✱ Space Complexity = Input + Extra

= <sequence of dim> + Extra

↳ n

Extra: stack (n); matrix ⇒ $\frac{n^2}{2} \times 2$ (1 for no., 2 for k)

$= 2n + \frac{n^2}{2} \times 2$

$= \boxed{O(n^2)}$

GATE-16

Q. Let A_1, A_2, A_3 & A_4 be four matrices of dimensions 10×5, 5×20, 20×10 & 10×5 respectively. The minimum no. of scalar multiplications required to find the product $A_1 A_2 A_3 A_4$ using the basic matrix multiplication method is

$A_1((A_2\ A_3)\ A_4)$

10 5 20 10 5

$P_0\ P_1\ P_2\ P_3\ P_4$

$m[i,j] = \min_{i \le k < j} \{ m[i,k] + m[k+1,j] + P_{i-1} P_j P_k \}$

	1	2	3	4
1	0	k=1 1000	k=1 1500	k=1 1500
2		0	k=2 1000	k=3 1250
3			0	k=3 1000
4				0

(i ↓, j →)

✱ $m[1,2] = \min\{ m[1,1] + m[2,2] + P_0 P_2 P_1 \}$, $1 \le k < 2$

$= \min\{ 0 + 0 + 10 \times 20 \times 5 \}$

$= \boxed{1000}$

✱ $m[2,3] = \min\{ m[2,2] + m[3,3] + P_1 P_3 P_2 \}$, $2 \le k < 3$

$= \min\{ 0 + 0 + 5 \times 10 \times 20 \}$

$= \boxed{1000}$

✱ $m[3,4] = \min\{ m[3,3] + m[4,4] + P_2 P_4 P_3 \}$, $3 \le k < 4$

$= \min\{ 0 + 0 + 20 \times 5 \times 10 \}$

$= \boxed{1000}$

✱ $m[1,3] = \min$, $1 \le k < 3$, $k = 1,2$

$m[1,1] + m[2,3] + P_0 P_3 P_1$

$0 + 1000 + 10 \times 10 \times 5$

$\boxed{1500}$

$m[1,2] + m[3,3] + P_0 P_2 P_2$

$1000 + 0 + 10 \times 10 \times 20$

$\boxed{3000}$

✱ $m[1,4]$, $1 \le k < 4$, $k = 1,2,3$

$= \begin{cases} 0 + 1250 + 10 \times 5 \times 5 = \boxed{1500} \\ 1000 + 1000 + 10 \times 5 \times 20 = 3000 \\ 1500 + 0 + 10 \times 5 \times 10 = 2000 \end{cases}$

✱ $m[2,4] = \min$, $2 \le k < 4$, $k = 2,3$

$m[2,2] + m[3,4] + P_1 P_4 P_2$

$0 + 1000 + 5 \times 5 \times 20 = \boxed{1500}$

$m[2,3] + m[4,4] + P_1 P_4 P_3$

$1000 + 0 + 5 \times 5 \times 10 = \boxed{1250}$

GATE-11

Q. Four matrices M_1, M_2, M_3, & M_4 of dimensions p×q, q×r, r×s and s×t respectively can be multiplied in several ways with different number of total scalar multiplications. For example when multiplied as $((M_1 \times M_2) \times (M_3 \times M_4))$, the total number of scalar multiplications is pqr + rst + prt, when multiplied as $(((M_1 \times M_2) \times M_3) \times M_4)$, the total no. of scalar multiplications is pqr + prs + pst.

If p=10, q=100, r=20, s=5 and t=80, then the minimum no. of scalar multiplications needed is

10 100 20 5 80

$P_0\ P_1\ P_2\ P_3\ P_4$

$m[i,j] = \min_{i \le k < j} \{ m[i,k] + m[k+1,j] + P_{i-1} P_j P_k \}$

	1	2	3	4
1	0	K=1 20000	K=1 15000	K=3 19000
2		0	K=2 10000	K=3 50000
3			0	K=3 8000
4				0

(i ↓, j →)

✱ $m[1,2] = P_0 P_2 P_1 = 10 \times 20 \times 100 = 20000$

✱ $m[2,3] = P_1 P_3 P_2 = 100 \times 5 \times 20 = 10000$

✱ $m[3,4] = P_2 P_4 P_3 = 20 \times 80 \times 5 = 8000$

✱ $m[2,4]$, $2 \le k < 4$, $k = 2,3$

$= \min\{ 0 + 8000 + 100 \times 80 \times 20 = 168000,\ 10000 + 0 + 100 \times 80 \times 5 = 50000 \}$

✱ $m[1,3]$, $1 \le k < 3$, $k = 1,2$

$= \min\{ 0 + 10000 + 10 \times 5 \times 100 = 15000,\ 20000 + 0 + 10 \times 5 \times 20 = 25000 \}$

✱ $m[1,4]$, $1 \le k < 4$

$= \min\{ 0 + 40000 + 10 \times 80 \times 100 = 120000,\ 20000 + 8000 + 10 \times 80 \times 20 = 44000 \}$

GATE-18

Q. Assume that multiplying a matrix G_1 of dimension $p \times q$ with another matrix G_2 of dimension $q \times r$ requires pqr scalar multiplications. Computing the product of n matrices $G_1, G_2, G_3 \ldots G_n$ can be done by parenthesizing in different ways. Define $G_i G_{i+1}$ as an explicitly computed pair for a given parenthesization if they are directly multiplied. For example, in the matrix multiplication chain $G_1 G_2 G_3 G_4 G_5 G_6$ using parenthesization $(G_1(G_2 G_3))(G_4(G_5 G_6))$, $G_2 G_3$ & $G_5 G_6$ are only explicitly computed pairs.

Consider a matrix multiplication chain F_1, F_2, F_3, F_4, F_5 where matrices F_1, F_2, F_3, F_4 & F_5 are of dimensions 2×25, 25×3, 3×16, 16×1, and 1×1000 respectively.

In the parenthesization of $F_1 F_2 F_3 F_4 F_5$ that minimizes the total no. of scalar multiplications, the explicitly computed pairs is/are

(a) $F_1 F_2$ & $F_3 F_4$ only

(b) $F_2 F_3$ only

(c) $F_3 F_4$ only

(d) $F_2 F_1$ & $F_4 F_5$ only

F_1 F_2 F_3 F_4 F_5

2 25 25 3 3 16 16 1 1 1000

1000 $(F_1 F_2 F_3 F_4 (F_5))$

48

F_1 F_2 F_3 F_4 F_5

2 25 25 3 3 16 16 1 1 1000

2 25 3 16 1 1000

P_0 P_1 P_2 P_3 P_4 P_5

$(F_1(F_2(F_3\ F_4)(F_5)))$

$m[i,j] = \min_{i \le k < j} \{ m[i,k] + m[k+1,j] + P_{i-1} P_i P_k \}$

* $m[1,2] = P_0 P_2 P_1 = 2 \times 3 \times 25 = 150$.
* $m[2,3] = P_1 P_3 P_2 = 25 \times 16 \times 3 = 1200$
* $m[3,4] = P_2 P_4 P_3 = 3 \times 1 \times 16 = 48$
* $m[4,5] = P_3 P_5 P_4 = 16 \times 1000 \times 1 = 16000$

	1	2	3	4	5	
1	0	K=1 150	K=2 246	K=1 175	K=4 2175	1
2		0	K=2 1200	K=2 123	K=2 9048	2
3			0	K=3 48	K=4 3048	3
4				0	K=4 16000	4
5					0	5

(i ↓, j →)

* $m[1,3] = \min_{1 \le k < 3} \begin{cases} k=1, & 2000 \\ k=2, & (246) \end{cases}$ (k=1,2)
* $m[2,4] = \min_{2 \le k < 4} \begin{cases} k=2, & (123) \\ k=3, & 1600 \end{cases}$ (k=2,3)
* $m[3,5] = \min_{3 \le k < 5} \begin{cases} k=3, & 64000 \\ k=4, & (3048) \end{cases}$ (k=3,4)
* $m[2,5] = \min_{2 \le k < 5} \begin{cases} k=2, & (9048) \\ k=3, & \text{बहुत ज्यादा !} \\ k=4, & 25123 \end{cases}$ (k=2,3,4)
* $m[1,4] = \min_{1 \le k < 4} \begin{cases} k=1, & (175) \\ k=2, & \times \\ k=3, & \times \end{cases}$ (k=1,2,3)
* $m[1,5] = \min_{1 \le k < 5} \begin{cases} k=1, \\ k=2, \\ k=3, \\ k=4, \end{cases}$ (k=1,2,3,4)

0/1 Knapsack Problem :

→ In 0/1 knapsack Problem item may not be broken into smaller pieces.

→ So, we may decide either to take an item or leave it (Binary choice 0/1) but may not take a fraction of item.

The recursive solution of knapsack is

$$k(i,c) = \begin{cases} \max\left[(p_i + k(i-1, c-w_i)),\ k(i-1,c)\right] \\ 0, \quad i=0 \text{ or } c=0 \\ k(i-1,c), \quad w_i > c \end{cases}$$

* Time Complexity of 0/1 Knapsack : $O(m \cdot n)$

P	10	12	28
W	1	2	4

C = 6

i \ C	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	10	10	10	10	10	10
2	0	10	12	22	22	22	22
3	0	10	12	22	28	38	40

* $k(1,1) = \max[(10 + k(0,0), k(0,1)]$
  $= \max[10+0, 0]$
  $= \max[10, 0]$
  $= 10$
* $k(1,2) = \max[(10 + k(0,1), k(0,2)]$
  $= \max[10, 0]$
  $= 10$
* $k(2,1) = k(1,1) = 10$
* $k(2,2) = [12 + k(1,0), k(1,2)] = 12$
* $k(3,1) = k(2,1) = 10$
* $k(3,6) = \max[28 + k(2,2), k(2,6)]$
  $= \max[28+12, 22]$
  $= \max(40, 22)$
  $= 40$

Q.

Weights = {3, 4, 6, 5}
Profits = {2, 3, 1, 4}

C = 8
n = 4

P_i	W_i	i \ C	0	1	2	3	4	5	6	7	8
		0	0	0	0	0	0	0	0	0	0
2	3 ✓	1	0	0	0	2	2	2	2	2	2
3	4 ✗	2	0	0	0	2	3	3	3	5	5
4	5 ✓	3	0	0	0	2	3	4	4	5	6
1	6 ✗	4	0	0	0	0	2	3	4	5	6

$x_i = \{1, 0, 0, 1\}$

6
6 − 4 = 2
2 − 2 = 0

max (3+0, 2) = max (3, 2) = 3	max (3+0, 2) = max (3, 2) = 3	max (3+0, 2) = max (3, 2) = 3	max (3+2, 2) = max (5, 2) = 5	max (3+2, 2) = max (5, 2) = 5
max (4+0, 3) = 4	max (4+0, 3) = 4	max (4+0, 5) = 5	max (4+2, 5) = 6	

Q.

Weights = {3, 4, 6, 5}
Profits = {2, 3, 1, 4}

C = 8
n = 4

P_i	W_i	i \ C	0	1	2	3	4	5	6	7	8
		0	0	0	0	0	0	0	0	0	0
2	3 ✓	1	0	0	0	2	2	2	2	2	2
3	4 ✗	2	0	0	0	2	3	3	3	5	5
4	5 ✓	3	0	0	0	2	3	4	4	5	6
1	6 ✗	4	0	0	0	2	3	4	4	5	6

$x_i = \{1, 0, 0, 1\}$

6
6 − 4 = 2
2 − 2 = 0

max (3+0, 2) = max (3, 2) = 3	max (3+0, 2) = max (3, 2) = 3	max (3+0, 2) = max (3, 2) = 3	max (3+2, 2) = max (5, 2) = 5	max (3+2, 2) = max (5, 2) = 5
max (4+0, 3) = 4	max (4+0, 3) = 4	max (4+0, 5) = 5	max (4+2, 5) = 6	

Subset Sum Problem:

Let A be an array or set which contains n non negative integers.
We have to find a subset 'x' of set 'A' such that sum of all elemen of $x = w$, here w is sum.

eg: $A = \{1, 2, 5, 9, 4\}$
$Sum(w) = 18$

→ Brute force : $O(2^n)$

→ Dynamic programming : $O(n \cdot sum)$

✱ $m[i][j] = 1$

→ $A[i] = j$

→ $A[i-1][j] = 1$

→ $A[i-1][j - A[i]] = 1$

Q. $A = \{2, 3, 5, 7, 10\}$
$Sum(w) = 14$

Sum(j) →

i ↓	2	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
2	3	1	0	1	1	0	(1)	0	0	0	0	0	0	0	0	0
3	5	1	0	1	1	0	1	0	1	1	0	1	0	0	0	0
4	7	1	0	1	1	0	1	0	1	1	1	1	0	0	0	1
5	10	1	0	1	1	0	1	0	1	1	1	1	0	0	1	(1) → Yes, sum exist!

$5-3 = (2) \rightarrow 1$ ✓
$6-3 = (3) \rightarrow 0$ ✓

GATE-05
Q. The subset-sum problem is defined as follows. Given a set of integers, $S = \{a_1, a_2, a_3 \ldots a_n\}$ and positive integer W, is there a subset of S whose elements sum to W? A dynamic program for solving this proble uses a 2-dimensional Boolean array, X with n rows and W+1 columns $X[i,j]$, $1 \le i \le n$, $0 \le j \le W$ is TRUE, if and only if there is a subset of $\{a_1, a_2, a_3 \ldots a_i\}$ whose elements sum to j.

(a) $X[i,j] = x[i-1, j] \vee X[i, j-a_i]$
(b) $X[i,j] = X[i-1, j] \vee X[i-1, j-a_i]$
(c) $X[i,j] = X[i-1, j] \forall \wedge X[i, j-a_i]$
(d) $X[i,j] = X[i-1, j] \wedge X[i-1, j-a_i]$

Q. The subset sum problem is defined as follows. Given a set of integers, $S = \{a_1, a_2, a_3, \ldots a_n\}$ and positive integer W, is there a subset S whose elements sum to W? A dynamic program for solving this program uses a 2-dimensional Boolean array X, with n rows and $W+1$ columns. $X[i,j]$, $1 \le i \le n$, $0 \le j \le W$ is TRUE, if and only if there is a subset of $\{a_1, a_2, a_3, \ldots a_i\}$ whose elements sum to j.

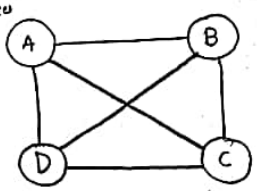
Which entry of the array X if TRUE, implies that there is a subset whose elements sum to W?

(a) $X[1, W]$
(b) $X[n, 0]$
(c) $X[n, W]$
(d) $X[n-1, n]$

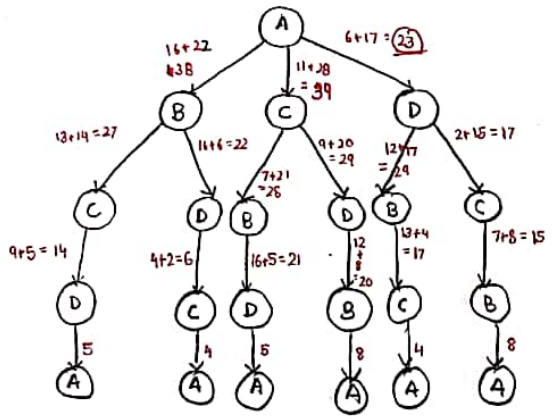
Travelling Salesman Problem :

$$g(i,S) = \min_{j \in S} [w(i,j) + g(j, \{S-j\})]$$

i → starting vertex
S → set of vertices salesman would visit



	A	B	C	D
A	0	16	11	6
B	8	0	13	16
C	4	7	0	9
D	5	12	2	0



* Time complexity = $(n-1)(n-2) \ldots 2 \cdot 1$
$= O(n!) * n$
$= O(n^n)$

$$g(i,S) = \min_{j \in S} [w(i,j) + g(j, \{S-j\})]$$

→ $g(A, \{B,C,D\}) = \min [w(A,B) + g(B, \{C,D\})] = 16+22 = 38$
$w(A,C) + g(C, \{B,D\}) = 11+28 = 39$
$w(A,D) + g(D, \{B,C\}) = 6+17 = 23$

→ $g(B, \{C,D\}) = w(B,C) + g(C, \{D\}) = 13+14 = 27$
$w(B,D) + g(D, \{C\}) = 16+6 = 22$

→ $g(C, \{D\}) = w(C,D) + g(D, \phi)$
$= 9 + 5 = 14$

→ $g(D, \{C\}) = w(D,C) + g(C, \phi)$
$= 2+4 = 6$

→ $g(C, \{B,D\}) = w(C,B) + g(B, \{D\}) = 7+21 = 28$
$w(C,D) + g(D, \{B\}) = 9+20 = 29$

→ $g(B, \{D\}) = w(B,D) + g(D, \phi) = 16+5 = 21$

	A	B	C	D
A	0	16	11	6
B	8	0	13	16
C	4	7	0	9
D	5	12	2	0

A → D → C → B

Floydd Warshall Algorithm:

$$D_{ij}^{\circ} = \begin{cases} 0 & , \text{ if } i=j \\ \infty & , \text{ if } i \neq j \ \& \ w_{ij} = 0 \\ w_{ij} & , \text{ if } i \neq j \ \& \ w_{ij} \neq 0 \end{cases}$$

```
for k=1 to V
    for i=1 to V
        for j=1 to V
```

$$D^{k}[i,j] = \min\left(D^{k-1}[i,j] \ , \ D^{k-1}[i,k] + D^{k-1}[k,j]\right)$$

✱ Time complexity of this algorithm = $\theta(V^3)$

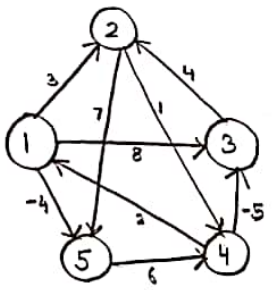
GATE-16

Q. The Floyd-Warshall algorithm for all pair shortest paths computation is based on:

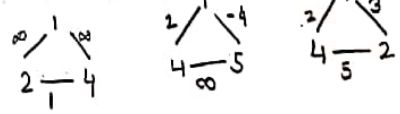
(a) Greedy paradigm
(b) Divide & Conquer paradigm
(c) Dynamic Programming paradigm
(d) Neither Greedy nor Divide and Conquer nor Dynamic Programming paradigm

$$W = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 3 & 8 & 0 & -4 \\ 0 & 0 & 0 & 1 & 7 \\ 0 & 4 & 0 & 0 & 0 \\ 2 & 0 & -5 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 \end{bmatrix} \end{array}$$

$$D^{\circ} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \end{array}$$



$$D^{1} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & \boxed{-2} \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \end{array}$$



D^2 =

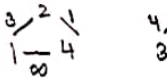
	1	2	3	4	5
1	0	3	8	(4)	-4
2	∞	0	∞	1	7
3	∞	4	0	(5)	(11)
4	2	5	-5	0	2
5	∞	∞	∞	6	0

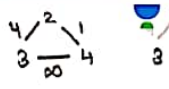
D^3 =

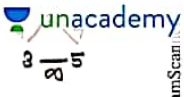
	1	2	3	4	5
1	0	3	8	4	-4
2	∞	0	∞	1	7
3	∞	4	0	5	11
4	2	(-1)	-5	0	-2
5	∞	∞	∞	6	0

D^4 =

	1	2	3	4	5
1	0	3	(-1)	4	-4
2	3	0	(-4)	1	(-1)
3	(7)	4	0	5	(3)
4	2	-1	-5	0	-2
5	(8)	(5)	(1)	6	0







D^5 =

	1	2	3	4	5
1	0	(1)	(-3)	(2)	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0

GATE-14

Q. Suppose you want to move from 0 to 100 on the number line. You either move right by a unit distance or you take shortcut (i,j), if you are at position i on the number line, you may directly move to j. Suppose T(k) denotes the smallest no. of steps needed to move from k to 100. Suppose, further that there is at most 1 shortcut involving any number and in particular, from 9 there is a shortcut to 15. Let y and z be such that

$$T(9) = 1 + \min(T(y), T(z))$$

Then, the value of the product yz is ——

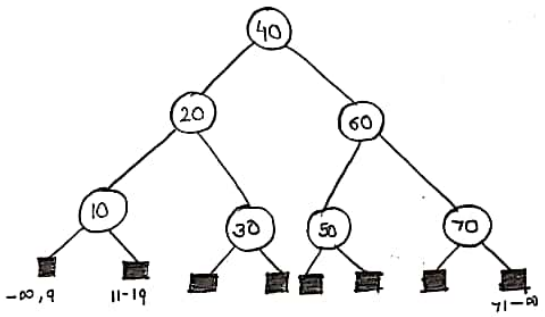
T(9) ⟶ Distance from 9 to 100

T(9) = 1 + min (Distance from y to 100, Distance from z to 100)

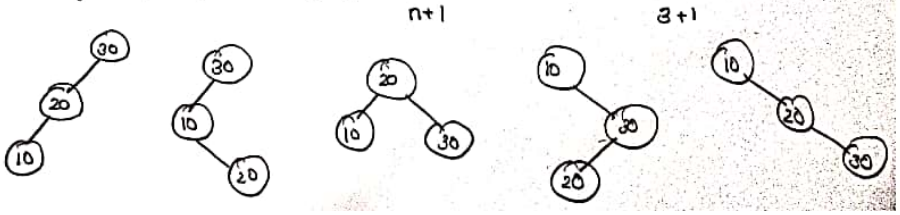
⟶ There are only 2 such values where we can reach from 9
- ↳ one is simple step to right on no. line (10)
- ↳ Another is shortcut (15)

∴ y = 10, z = 15 ⇒ yz = 10 × 15 = 150

OPTIMAL BINARY SEARCH TREE



Keys: 10, 20, 30 $\quad T(n) = \frac{^{2n}C_n}{n+1} \quad \Rightarrow T(3) = \frac{^{2\times3}C_3}{3+1} = 5$



→ For every key we have successful as well as unsuccessful probabilities which will be given.

→ We have to find a BST such that cost of searching any particular key should be minimum.

→ कैसे कर सकते : तो ek idea ye ho skta jitne bhi BST ho un sb का cost nikale aur jo minimum ho vo answer, pr ek din lgega agr key 5 ya 6 de diya jaaey!

→ Dynamic programming se yeh kaam krege jldi bina BST banaye!

$$c[i,j] = \min_{i<k\le j} \{ c[i,k-1] + c[k,j] + w[i,j] \}$$

$$w[i,j] = w[i,j-1] + p_j + q_j$$

Q. keys = {10, 20, 30, 40}

p_i = {3, 3, 1, 1} → successful search probabilities

q_i = {2, 3, 1, 1, 1} → unsuccessful search probabilities

j-i	0	1	2	3	4
0	$w_{00}=2$, $c_{00}=0$, $r_{00}=0$	$w_{11}=3$, $c_{11}=0$, $r_{11}=0$	$w_{22}=1$, $c_{22}=0$, $r_{22}=0$	$w_{33}=1$, $c_{33}=0$, $r_{33}=0$	$w_{44}=1$, $c_{44}=0$, $r_{44}=0$
1	$w_{01}=8$, $c_{01}=8$, $r_{01}=1$	$w_{12}=7$, $c_{12}=7$, $r_{12}=2$	$w_{23}=3$, $c_{23}=3$, $r_{23}=3$	$w_{34}=3$, $c_{34}=3$, $r_{34}=4$	
2	$w_{02}=12$, $c_{02}=19$, $r_{02}=1$	$w_{13}=9$, $c_{13}=12$, $r_{13}=2$	$w_{24}=5$, $c_{24}=8$, $r_{24}=3$		
3	$w_{03}=14$, $c_{03}=25$, $r_{03}=2$	$w_{14}=11$, $c_{14}=19$, $r_{14}=2$			
4	$w_{04}=16$, $c_{04}=32$, $r_{04}=2$				

* $w[i,j] = w[i,j-1] + p_j + q_j$

→ $w[0,1] = w[0,0] + p_1 + q_1$
$= 2 + 3 + 3$
$= 8$

→ $c[i,j] = \min_{i<k\le j} \{ c[i,k-1] + c[k,j] \} + w[i,j]$

→ $c[0,1] = \min_{0<k\le 1} \{ c[0,0] + c[1,1] \} + w[0,1]$
$= \min\{0+0\} + 8$
$= 8$

→ $c[1,2] = \min_{1<k\le 2} \{ c[1,1] + c[2,2] \} + w[1,2]$
$= \min\{0+0\} + 7 = 7$

* $c[0,2] = \min_{0<k\le 2,\ k=1,2} \{ c[0,0] + c[1,2],\ c[0,1] + c[2,2] \} + w[0,2]$
(k=1, k=2)

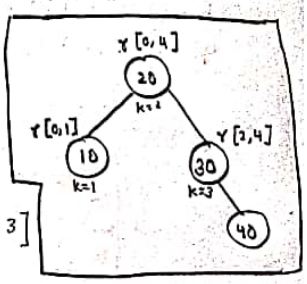
$= \min\{0+7, 8+0\} + 12$
$= \min\{7\} + 12 = 19$

* $c[0,3] = \min_{0<k\le 3} \{ k=1\ c[0,0] + c[1,3],\ k=2\ c[0,1] + c[2,3],\ k=3\ c[0,2] + c[3,3] \} + w[0,3]$
(0, 12; 8, 3; 19, 0)

$= \min\{12, 11, 19\} + 14$
$= 25$

* $c[0,4] = \min_{0<k\le 4} \{ k=1\ c[0,0] + c[1,4],\ k=2\ c[0,1] + c[2,4],\ k=3\ c[0,2] + c[3,4],\ k=4\ c[0,3] + c[4,4] \} + w[0,4]$
(0, 19; 8, 8; 19, 3; 25, 0)

$= 32$



* Min cost = $\frac{32}{16} = 2$

क्या है Hash Table और Hashing :

→ Data Structure
→ Faster access to elements → Insertion : O(1)
→ It uses array
→ It uses a hash function
→ Maps every data to key to get faster access to keys or elements.

eg: $F(x) = x \% 10$

elements : 67, 54, 13, 19, 20

→ $F(67) = 67 \% 10 = 7$
→ $F(54) = 54 \% 10 = 4$
→ $F(13) = 13 \% 10 = 3$
→ $F(19) = 19 \% 10 = 9$
→ $F(20) = 20 \% 10 = 0$

* Search : 15, 19, 51, 30
$F(15) = 15 \% 10 = 5$
↓
Hoga toh yahi hoga!
→ O(1)

HASH TABLE

0	20
1	
2	
3	13
4	54
5	
6	
7	67
8	
9	19

* Delete : 54, 13, 27, 19
$F(54) = 54 \% 10 = 4$
↓
Delete toh!
→ O(1)

Collision :

$F(x) = x \% 10$
Elements : 67, 54, 13, 19, 27

→ $F(x) = 54 \% 10 = 4$
→ $F(67) = 67 \% 10 = 7$ ←
→ $F(13) = 13 \% 10 = 3$ Collision
→ $F(19) = 19 \% 10 = 9$
→ $F(27) = 27 \% 10 = 7$ ←

HASH TABLE

0	
1	
2	
3	13
4	54
5	
6	
7	67
8	
9	19

Collision Resolution Techniques
- Separate Chaining / Open Hashing
- Open addressing / Closed Hashing
  - Linear Probing
  - Quadratic Probing
  - Double Hashing

Q किसको बोले एक अच्छा Hash Function :

→ Efficiency of hashing depends on the hash function.
→ Very common hash function :
$F(X) = X \%$ Hash tables size
→ Simple & Easy to compute
→ Should uniformly distribute keys.
→ Minimum number of collisions.

* Types of hash functions :

① Direct Hashing
→ No Algorithmic manipulation.
→ Min. no. of collision.
→ Limited to certain no. of keys.
→ Not suitable for large

② Modulo-Division
→ Also known as division remainder
→ works with any list size.
→ If the size list is prime no., km collision honge!

③ Mid-Square
→ middle of square
→ Square the key and middle of the required no. of address will be the add.
eg: k = 35 n = 2
$k^2 = 1225$ h(k) = 22

unacademy

④ Folding Hashing:

→ Fold shift hashing

* key value is divided into parts and then added to get required address.

eg: 123 456 789 , n=3

123
456
789
h(k)=368

discard → 1368

→ Fold Boundary hashing

* Left & right nos. are folded on a fixed boundary b/w them.

eg: 123 456 789

321
987
456
h(k)=764

discard ← 1764

⑤ Substraction:

→ Substract a fix no. from key.

$h(k) = k - C$

→ No collision

→ Suitable for small size list.

$h(k) = k - C$

Scanned with CamScanner

unacademy

① Linear Probing:

→ $F(x) = (x+i) \% 10$

where $i = 0, 1, 2, 3, 4 \ldots$

Elements : 67, 54, 14, 19, 25, 41

$F(67) = (67+0) \% 10 = 7$

$F(54) = (54+0) \% 10 = 4$

$F(14) = (14+0) \% 10 = 4$ → Collision आया!

$= (14+1) \% 10 = 5$ ✓

→ Searching : Mann Lo (22) search krna he!

$F(22) = (22+0) \% 10 = 2$ → नहीं है Table में!

$F(14) = (14+0) \% 10 = 4$ → nhi mila

$= (14+1) \% 10 = 5$ → Mil gya!

→ Deletion same

→ Problem : Clustering

HASH TABLE

0	
1	41
2	
3	
4	54
5	14
6	25
7	67
8	
9	19

Insert : 10, 20, 30, 40

0 0 0 0

1 1 1

2 2

3

Scanned with CamScanner

unacademy

② Quadratic Probing:

→ $F(x) = (x+i^2) \% 10$

where, $i = 0, 1, 2, 3, 4 \ldots$

→ Elements : 91, 55, 26, 31, 11

$F(91) = (91+0^2) \% 10 = 1$

$F(55) = (55+0^2) \% 10 = 5$

$F(26) = (26+0^2) \% 10 = 6$

$F(31) = (31+0^2) \% 10 = 1$ → collision

$= (31+1^2) \% 10 = 2$ ✓

$F(11) = (11+0^2) \% 10 = 1$ → collision

$= (11+1^2) \% 10 = 2$

$= (11+2^2) \% 10 = 5$

$= (11+3^2) \% 10 = 0$ ✓

HASH TABLE

0	11
1	91
2	31
3	
4	
5	55
6	26
7	
8	
9	

Scanned with CamScanner

क्या है Hash Table और Hashing :

→ Data structure
→ Faster access to elements → Insertion : O(1)
→ It uses array
→ It uses a hash function
→ Maps every data to key to get faster access to keys or elements.

eg: $F(x) = x \% 10$

elements : 67, 54, 13, 19, 20

→ F(67) = 67 % 10 = 7
→ F(54) = 54 % 10 = 4
→ F(13) = 13 % 10 = 3
→ F(19) = 19 % 10 = 9
→ F(20) = 20 % 10 = 0

* Search : 15, 19, 51, 30
F(15) = 15 % 10 = 5
↓
Hoga toh yahi hoga!
→ O(1)

HASH TABLE

0	20
1	
2	
3	13
4	54
5	
6	
7	67
8	
9	19

* Delete : 54, 13, 27, 19
F(54) = 54 % 10 = 4
↓
Delete toh!
→ O(1)

Collision :

$F(x) = x \% 10$

Elements : 67, 54, 13, 19, 27

→ F(54) = 54 % 10 = 4
→ F(67) = 67 % 10 = 7
→ F(13) = 13 % 10 = 3
→ F(19) = 19 % 10 = 9
→ F(27) = 27 % 10 = 7

Collision

HASH TABLE

0	
1	
2	
3	13
4	54
5	
6	
7	67
8	
9	19

Collision Resolution Techniques

- Separate chaining / Open Hashing
- Open addressing / Closed Hashing
  - Linear Probing
  - Quadratic Probing
  - Double Hashing

Q किसको बोले एक अच्छा Hash Function :

→ Efficiency of hashing depends on the hash function.
→ Very common hash function :
$F(X) = X \%$ Hash tables size
→ Simple & Easy to compute
→ Should uniformly distribute keys.
→ Minimum number of collisions.

* Types of hash functions :

① Direct Hashing
→ No Algorithmic manipulation.
→ Min. no. of collision.
→ Limited to certain no. of keys.
→ Not suitable for large key values.

② Modulo-Division
→ Also known as division remainder
→ works with any list size.
→ If the size list is prime no., km collision honge!

③ Mid-Square
→ middle of square
→ Square the key and middle of the required no. of address will be the add.
eg: k = 35 n = 2
$k^2 = 1225$ h(k) = 22

④ Folding Hashing:

→ Fold shift hashing

* key value is divided into parts and then added to get required address.

eg: 123 456 789 , n=3

123
456
789
1368

discard → 1368

h(k) = 368

→ Fold Boundary hashing

* Left & right nos. are folded on a fixed boundary b/w them.

eg: 123 456 789

321
987
456
1764

discard ← 1764

h(k) = 764

⑤ Substraction:

→ Substract a fix no. from key.

h(k) = k - C

→ No collision

→ Suitable for small size list.

h(k) = k - C

① Linear Probing:

→ F(x) = (x+i) % 10

where i = 0, 1, 2, 3, 4 ....

Elements : 67, 54, 14, 19, 25, 41

F(67) = (67+0) % 10 = 7

F(54) = (54+0) % 10 = 4

F(14) = (14+0) % 10 = 4 → Collision आया!

= (14+1) % 10 = 5 ✓

→ Searching : Mann Lo (22) search krna he!

F(22) = (22+0) % 10 = 2 → नहीं है Table में!

F(14) = (14+0) % 10 = 4 → nhi mila

= (14+1) % 10 = 5 → Mil gya!

→ Deletion same

→ Problem : Clustering

HASH TABLE

0	
1	41
2	
3	
4	54
5	14
6	25
7	67
8	
9	19

Insert : 10, 20, 30, 40

0 0 0 0
1 1 1
2 2
3

② Quadratic Probing:

→ F(x) = (x + i²) % 10

where, i = 0, 1, 2, 3, 4....

→ Elements : 91, 55, 26, 31, 11

F(91) = (91 + 0²) % 10 = 1

F(55) = (55 + 0²) % 10 = 5

F(26) = (26 + 0²) % 10 = 6

F(31) = (31 + 0²) % 10 = 1 → collision

= (31 + 1²) % 10 = 2 ✓

F(11) = (11 + 0²) % 10 = 1 → collision

= (11 + 1²) % 10 = 2

= (11 + 2²) % 10 = 5

= (11 + 3²) % 10 = 0 ✓

HASH TABLE

0	11
1	91
2	31
3	
4	
5	55
6	26
7	
8	
9	

Chaining:

3, 2, 9, 6, 11, 13, 7, 12

$h(k) = 2k+3$ $\boxed{m=10}$

$F(x) = h(k) \% 10$

→ $F(3) = (2\times3+3) \% 10 = 9$

→ $F(2) = (2\times2+3) \% 10 = 7$

→ $F(9) = (2\times9+3) \% 10 = 1$

→ $F(6) = (2\times6+3) \% 10 = 5$ ⟵ Collision

→ $F(11) = (2\times11+3) \% 10 = 5$ ⟵ Collision

→ $F(13) = (2\times13+3) \% 10 = 9$ → Collision

→ $F(7) = (2\times7+3) \% 10 = 7$ → Collision

→ $F(12) = (2\times12+3) \% 10 = 7$ → Collision

0		
1	9	
2		
3		
4		
5	6	→ 11
6		
7	2	→ 7 → 12
8		
9	3	→ 13

Double Hashing

A = 3, 2, 9, 6, 11, 13, 7, 12

$h_1(k) = 2k+3$ $\boxed{m=10}$

$h_2(k) = 3k+1$

→ Insert ki at first free place from $(u + v*i) \% m$, $\boxed{v = h_2(k) \% m}$

where, i = 0 to m−1

→ Linear probing: $(u+i) \% m$

→ Quadratic probing: $(u+i^2) \% m$

0	
1	9
2	
3	11
4	12
5	6
6	
7	2
8	
9	3

_, 9, _ 11, 12, 6, _ 2, _ 3

$v(7) = (3\times7+1) \% 10 = 2$

→ $(7+2*0) \% 10 = 7$

→ $(7+2*1) \% 10 = 9$

→ $(7+2*2) \% 10 = 1$

→ $(7+2*3) \% 10 = 3$

→ $(7+2*4) \% 10 = 5$

→ $(7+2*5) \% 10 = 7$

→ $(7+2*6) \% 10 = 9$

⋮

→ $(7+2*9) \% 10 = 5$

$v(9) = (3\times13+1) \% 10 = 0$

→ $(9+0*0) \% 10 = 9$

→ $(9+0*1) \% 10 = 9$

⋮

Key	location (u)	v	Probe
3	$(2\times3+3) \% 10 = 9$	–	1
2	$(2\times2+3) \% 10 = 7$	–	1
9	$(2\times9+3) \% 10 = 1$	–	1
6	$(2\times6+3) \% 10 = 5$	–	1
11	$(2\times11+3) \% 10 = 5$	4	3
13	$(2\times13+3) \% 10 = 9$	∝	∝ → nhi dalega hash Table
7	$(2\times7+3) \% 10 = 7$	2	∝ → nhi dalega
12	$(2\times12+3) \% 10 = 7$	7	2

$v(11) = (3\times11+1) \% 10 = 4$

→ $(5+4*0) \% 10 = 5$

→ $(5+4*1) \% 10 = 9$

→ $(5+4*2) \% 10 = 3$

Q. Consider a hash table with 100 slots. Collisions are resolved using chaining. Assuming simple uniform hashing, what is the probability that the first 3 slots are unfilled after the first 3 insertions?

(a) $(97 \times 97 \times 97)/100^3$
(b) $(99 \times 98 \times 97)/100^3$
(c) $(97 \times 96 \times 95)/100^3$
(d) $(97 \times 96 \times 95)/(3! \times 100^3)$

$$P = \frac{97}{100} \times \frac{97}{100} \times \frac{97}{100}$$

→ Same Question for linear probing

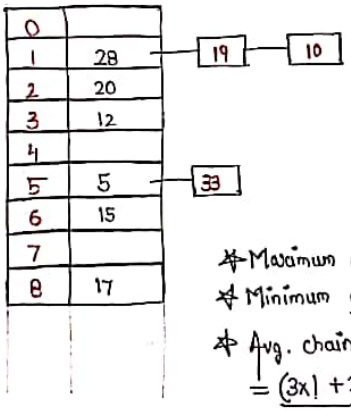
$$P = \frac{97}{100} \times \frac{96}{100} \times \frac{95}{100}$$

GATE-14

Q. Consider a hash table with 9 slots. The hash function is $h(k) = k \bmod 9$. The collisions are resolved by chaining. The following 9 keys are inserted in the order: 5, 28, 19, 15, 20, 33, 12, 17, 10. The maximum, minimum and average chain lengths in the hash table respectively are

(a) 3, 0 & 1
(b) 3, 3 & 3
(c) 4, 0 & 1
(d) 3, 0, 2

5 mod 9 = 5
28 mod 9 = 1
19 mod 9 = 1
15 mod 9 = 6
20 mod 9 = 2
33 mod 9 = 6
12 mod 9 = 3
17 mod 9 = 8
10 mod 9 = 1



* Maximum chain length = 3
* Minimum chain length = 0
* Avg. chain length $= \frac{(3\times1 + 2\times1 + 1\times4 + 0\times3)}{9}$

$= 1$

GATE-10

Q. A hash table of length 10 uses open addressing with hash function

$h(k) = k \bmod 10$

and linear probing. After inserting 6 values into an empty hash table, the table is shown below.

How many different insertion sequences of the key values using the same hash function & linear probing will result in the hash table shown above

(a) 10
(b) 20
(c) 30
(d) 40

→ इस में से 4 बिना collision के chle gye!

→ Ab bache sirf 52, 33

52 के option = 6
33 के option = 5 (including 52)

Total posibility = $6 \times 5 = 30$

0	
1	
2	42
3	23
4	34
5	52
6	46
7	33
8	
9	

GATE-09

Q. The keys 12, 18, 13, 2, 3, 23, 5 & 15 are inserted into an initially empty hash table of length 10 using open addressing with hash function $h(k) = k \bmod 10$ and linear probing. What is the resultant hash table?

(a)

0	
1	
2	2
3	23
4	
5	15
6	
7	
8	18
9	

(b)

0	
1	
2	12
3	13
4	
5	5
6	
7	
8	18
9	

(c)

0	
1	
2	12
3	13
4	2
5	3
6	23
7	5
8	18
9	15

(d)

0	
1	
2	12,2
3	13,3,23
4	
5	5,15
6	
7	
8	18
9	

GATE-10

Q. A hash table of length 10 uses open addressing with hash function $h(k) = k \bmod 10$, and linear probing. After inserting 6 values into an empty hash table, the table is shown as below:

Which one of the following choices gives a possible order in which the key values could have been inserted in the table?

(a) 46, 42, 34, 52, 23, 33
(b) 34, 42, 23, 52, 33, 46
(c) 46, 34, 42, 23, 52, 33
(d) 42, 46, 33, 23, 34, 52

0	
1	
2	42
3	23
4	34
5	52
6	46
7	33
8	
9	

(a) 46, 42, 34, 52, 23, 33

0	
1	
2	42
3	52
4	34
5	23
6	46
7	33
8	
9	

(b) 34, 42, 23, 52, 33, 46

0	
1	
2	42
3	23
4	34
5	52
6	33
7	46
8	
9	

(c) ✓ 46, 34, 42, 23, 52, 33

0	
1	
2	42
3	23
4	34
5	52
6	46
7	33
8	
9	

B.F.S & D.F.S :

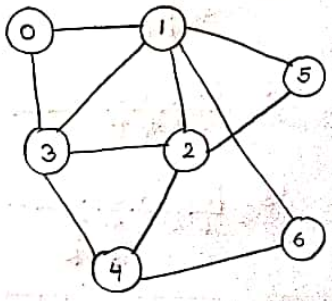
unacademy

Scanned with CamScan

① B.F.S (Breadth First Search) / Level order traversal :

Queue : | 0 | 1 | 3 | 2 | 5 | 6 | 4 |

O/P : 0 1 3 2 5 6 4



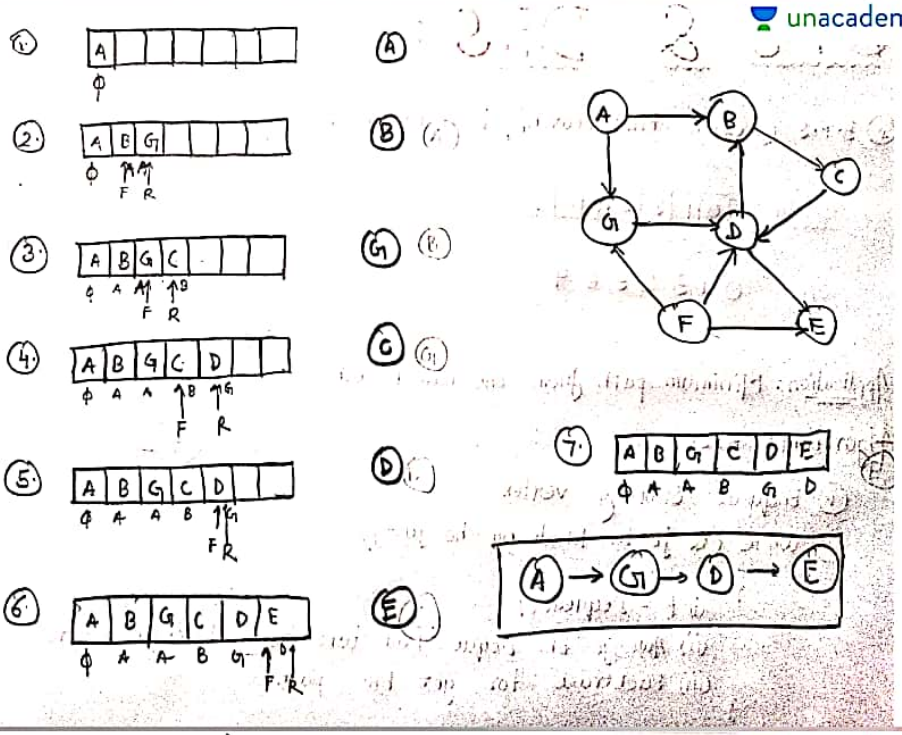
Application : Minimum path from one node to other.

Algorithm : BFS()

1. Enqueue Starting vertex.
2. While (Q jb tk khali na ho jaaey)
   {
   (i) P = Dequeue()
   (ii) Aur jo bhi Deque hua uska adjacent enqueue kro.
   (iii) Backtrack for get BFS path.
   }
3. Exit

unacademy

Scanned with CamScan

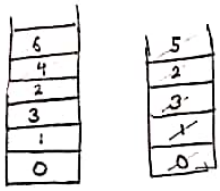


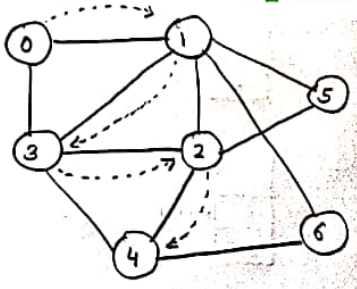
unacademy

Scanned with CamScan

② D.F.S (Depth First Search)

Stack :





O/P : 0, 1, 3, 2, 4, 6, 5

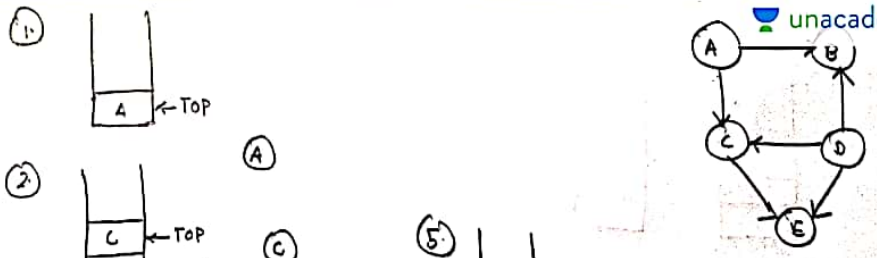
Application : Reachable node from any vertex.

Algorithm : DFS()

1. Push starting vertex
2. While (Stack jb tk khali na ho jaey)
   {
   (i) p = Top() ko pop kro
   (ii) Jo pop hua uska adjacent stack m dalo.
   (iii) Agr koi adj vertex nhi he toh pop kro.
   }

(iii) Agr koi adj vertex nhi he toh pop kro.

(3) Exit]



1.

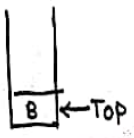
A

2.

C

3. 

E

4. 

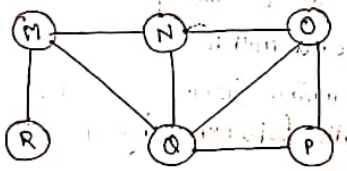
B

5. 

A - C - E - B

GATE-17

(36.) Breadth First Search (B.FS) has been implemented using queue data structure.



Which one of the following is a possible order of visiting the nodes in the graph above.

(a) M N O P Q R

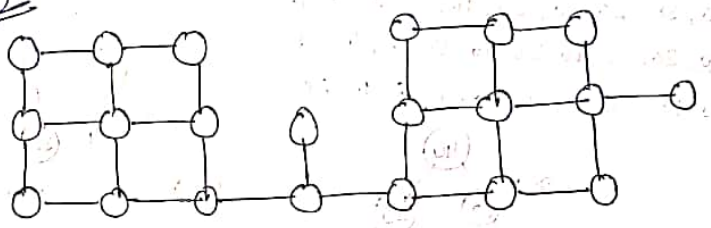
(b) N Q M P O R

(c) Q M N R O P

(d) P O Q N M R

GATE-14

(28.) Suppose depth first search is executed on the graph below starting at some unknown vertex. Assume that a recursive call to visit a vertex is made only after first checking that the vertex has not been visited earlier. Then the maximum possible recursion depth is (19)



unacademy

GATE-14

(43) Consider the tree arcs of a BFS traversal from a source node W in an unweighted, connected, undirected graph. The tree T formed by the tree arcs is a data structure for computing:

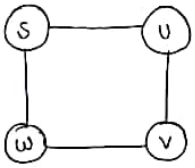
(a) the shortest path between every pair of vertices.
(b) the shortest path from W to every vertex in the graph.
(c) the shortest path from W to only those nodes that are leaves of T.
(d) the longest path in the graph.

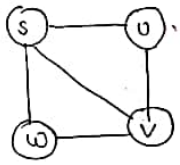
unacademy

GATE-15

(45) Let $G = (V, E)$ be a simple undirected graph and s be a particular vertex in it called the source. For $x \in V$, let $d(x)$ denote shortest distance in G from s to x. A BFS is performed starting at s. Let T be the resultant BFS tree. If (u, v) is an edge of G that is not in T, then which one of the following CANNOT be the value of $d(u) - d(v)$?

(a) -1
(b) 0
(c) 1
(d) 2



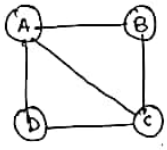


unacademy

GATE-14

(59) Let G be a graph with n vertices and m edges. What is the tightest upper bound on the running time on Depth First Search of G? Assume that the graph is represented using adjacency matrix.

(a) $\Theta(n)$
(b) $\Theta(n+m)$
(c) $\Theta(n^2)$
(d) $\Theta(m^2)$



Adjacency Matrix

	A	B	C	D
A	0	1	1	1
B	1	0	1	0
C	1	1	0	1
D	1	0	1	0

→ DFS में Apne ko ek ek Row traverse करना Pdega kiska Adjacency matrix ka!

$\Theta(n \times n)$

$\Theta(n^2)$

GATE-16

(60.) In an adjacent list representation of an undirected simple graph $G = (V, E)$, each edge (u, v) has two adjacency list entries : $[v]$ in the adjacency list of u, and $[u]$ in the adjacency list of v. These are called twins of each other. A twin pointer is a pointer from an adjacency list entry to its twin. If $|E| = m$ and $|V| = n$, and the memory size is not constraint, what is the time complexity of the most efficient algorithm to set the twin pointer in each entry in each adjacency list ?

(a) $\theta(n^2)$
(b) $\theta(m+n)$
(c) $\theta(m^2)$
(d) $\theta(n^4)$

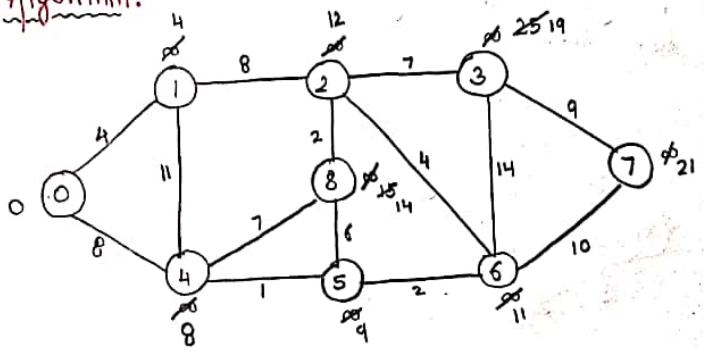
→ Agr apne ko twin pointer set krna hai [BFS] Algorithm lgega → Mtlb level by level traversal krna pdega!

→ BFS ka Time Complexit = $\theta(V+E) = \theta(m+n)$

Adjacency List

A	B, C, D
B	A, C
C	B, A, D
D	A, C

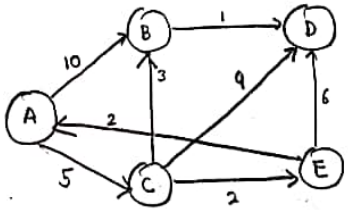
Dijkstra Algorithm:



if $(d(u) + c(u,v) < d(v))$

$d[v] = d(u) + c(u,v)$

* Not applicable for -ve wt graph.
* -ve edge wt ka cycle ho to surely don't work aur agar rege ho to bol nahi sakte !



	A	B	C	D	E
A	0	∞	∞	∞	∞
C		10 ←	5	∞	∞
E		8 ←		14	7
B		8	←	13	←
D				9	←

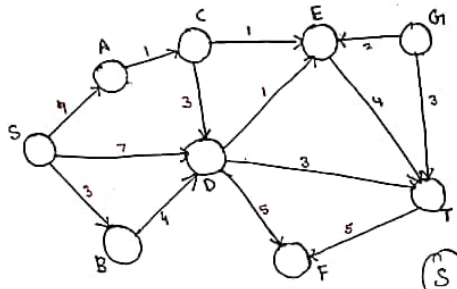
if $(d[u] + c(u,v) < d[v])$

$d[v] = d[u] + c(u,v)$

A — C — B — D

unacademy

GATE



Marked Vertex	S	A	B	C	D	E	F	G	T
S	0	∞	∞	∞	∞	∞	∞	∞	∞
B	4	3	∞	7	∞	∞	∞	∞	
A		4	∞	7	∞	∞	∞	∞	
C			5	7	∞	∞	∞	∞	
E				7	6	∞	∞	∞	
T					7	∞	∞	10 ←	

S —4— A —1— C —1— E —4— T

New value = min(old value, marked value + edge weight)

A $\min(\infty, 0+4) = 4$
B $\min(\infty, 0+3) = 3$
D $\min(\infty, 0+7) = 7$

D $\min(7, 3+4) = 7$
C $\min(\infty, 4+1) = 5$
D $\min(7, 5+3) = 7$
E $\min(\infty, 5+1) = 6$

T $\min(\infty, 6+4) = 10$

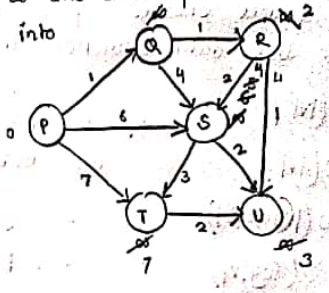
Scanned with CamScan

---

unacademy

GATE-04

Q. Suppose we run Dijkstra's single source shortest path algorithm on edge weighted directed graph with vertex P as the source. In what order do the nodes get included into the set of vertices for the shortest path distances are finalized?



(a) P, Q, R, S, T, U
(b) P, Q, R, U, S, T
(c) P, Q, R, U, T, S
(d) P, Q, T, R, U, S

GATE-05

Q. Let G(V, E) be an undirected graph with positive edge weights. Dijkstra's single source shortest path algorithm can be implemented using the binary heap data structure with time complexity:

(a) $O(|V|^2)$ → min dekho aur Array |V| times repeat
(b) $O((|E|+|V|)\log|V|)$ → Fibonacci Heap → decreases key operation
(c) $O(|V|\log|V|)$
(d) $O((|E|+|V|)\log|V|)$ → decreases key operation

Scanned with CamScan

---

unacademy

Q. Let G(V, E) an undirected graph with positive edge weights. Dij... Single source shortest path algorith can be implemented using sorted linked list data structure. What will be time complexity?

(a) $O(|V|^2)$
(b) $O(|V|^3)$
(c) $O(|V|\log|V|)$
(d) $O(\log|V|)$

* The basic operations of Dijkstra's algorithm are extract-min and decrease key (Relax)
→ Sorted list extract min would take O(1)
→ Relax operation would take O(V)
→ सब अपने को पता है Relax operation will be applied on every edge
∴ $O(VE)$
In worst case $E = |V|^2$
∴ $O(V^2 \cdot V) = O(|V|^3)$

In unsorted linked list: $O(|V|^2)$

Scanned with CamScan

Scanned with CamScanner

GATE-06
Q. To implement Dijkstra's shortest path algorithm on unweighted graphs so that it runs in linear time, the data structure to be used is:

(a) Queue (b) Stack (c) Heap (d) B-Tree

→ Using BFS algorithm we can find single source shortest path in unweighted graph by using Queue data structure.

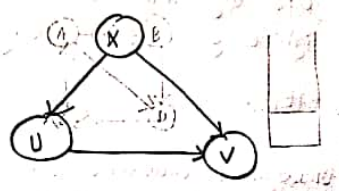
→ $O(|V|+|E|)$.

GATE-07
Q. In an unweighted, undirected connected graph, the shortest path from a node S to every node is computed most efficiently, in terms of time complexity by

(a) Dijkstra's algorithm starting from S.
(b) Warshall's algorithm.
(c) Performing a DFS starting from S.
(d) Performing a BFS starting from S.

GATE-07
Q. A depth first search is performed on a directed acyclic graph. d[v] denote the time at which vertex v is visited for the first time and f[v] the time at which at which the DFS call to the vertex v terminates. Which of the following statements is always TRUE for all edges (u, v) in the graph?

(a) $d[u] < d[v]$
(b) $d[u] < f[v]$
(c) $f[u] < f[v]$
(d) $f[u] \geq f[v]$



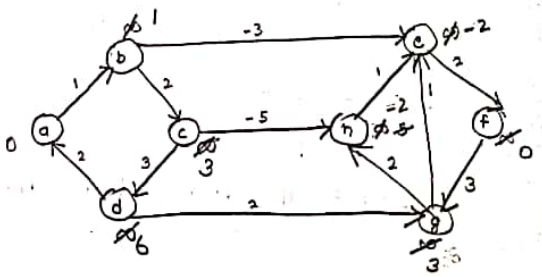
STATE-07
JGIC-12
Q. Consider a weighted, undirected graph with positive edge weights & let uv be an edge in the graph. It is know that the shortest path from the source vertex s to u has weight 58 and the shortest path from s to v has weight 65. Which one of the following is always TRUE?

(a) Weight $(u,v) \leq 12$ (b) Weight $(u,v) = 12$ (c) Weight $(u,v) \geq 12$ (d) Weight $(u,v) > 12$

If $w(u,v) < 12 \rightarrow \min w.s(s,v) = w(s,u) + w(u,v)$
$58 + < 12 = < 65$

GATE-08
Q. Dijkstra's single source shortest path algorithm when run from vertex a in the above graph computes the correct shortest path distance to

(a) only vertex a
(b) only vertices a, e, f, g, h
(c) only vertices a, b, c, d
(d) all the vertices



Bellman Ford Algorithm:

→ Single source shortest Path

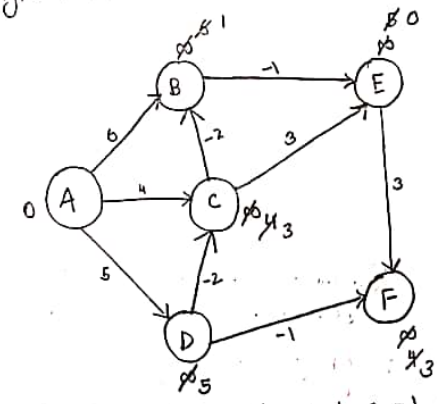
→ go on relaxing all the edges (n-1) times.

→ n = no. of vertices

→ Relaxing मतलब :

if (d[u] + c(u,v) < d[v])

d[v] = d[u] + c(u,v)



Edges: (A,B), (A,C), (A,D), (B,E), (C,E), (D,C), (D,F), (E,F), (C,B)

1st →
2nd →
3rd →
4th →
5th →

Scanned with CamScanner

unacademy

★ Time Complexity :

$O(E(|V|-1))$

if $E = V = n$

$\boxed{O(n^2)}$

GATE-13

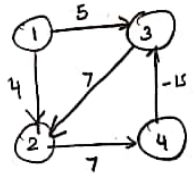
★ For complete Graph :

$E = \frac{n(n-1)}{2}$

$O\left[\frac{n(n-1)}{2} \cdot (n-1)\right]$

$\boxed{O(n^3)}$

★



→ -ve wt cycle ☹ Gir jaega!

GATE-09

Q. Which statement is correct regarding Bellman-Ford & shortest path algorithm?

P: Always find a negative wt cycle, if one exists.

Q: Finds whether any negative wt cycle is reachable from the source

(a) P only (b) Q only (c) Both (d) None

Scanned with CamScanner

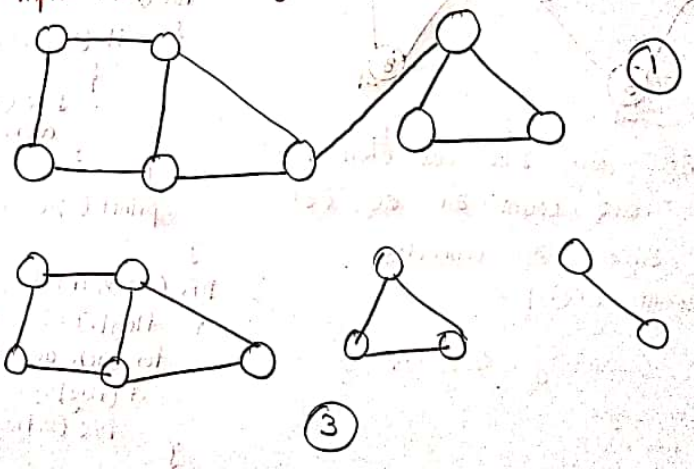
unacademy

Connected Components:

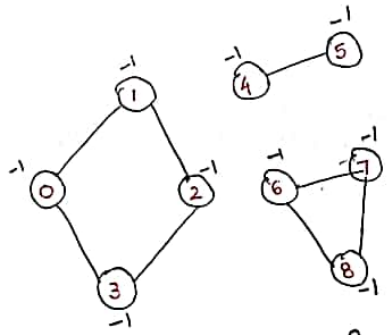
→ Set of vertices that are reachable.

→ It is a subgraph in which any two vertices are connected by a path and no other vertices is connected with any other vertices in supergraph.

→ It can be implemented either by DFS or BFS.



Scanned with CamScanner



→ जितनी बार DFS call होगा उतनी बार count को बढ़ा देंगे और उतने ही Connected Components होंगे!

GATE-08
→ Time Complexity : $\Theta(m+n)$

m → edges
n → vertices

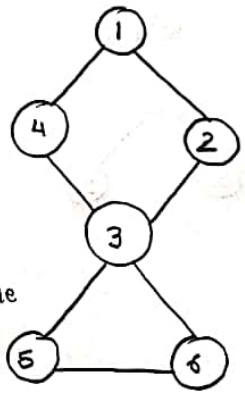
```
Connected_Components (G)
{ for each vertex v ∈ N
    flag[v] = -1
    count = 0
  for (int v = 0 ; v < N ; v++)
  { if (flag[v] == -1)
    {
      DFS (v, flag)
      count ++;
    }
  }
  printf ("%d", count);
}
DFS (int v, int flag)
{ flag[v] = 1 ;
  for each adjacent node u to v
    if (flag[u] == -1)
      DFS (u, flag)
}
```

unacademy

Scanned with CamScanner

Articulation Point & Biconnected Components

→ In a graph if there is any vertex whose removal will disconnect the graph into multiple components then that point is known as articulation point.

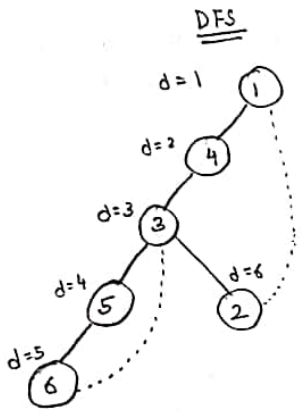


→ अगर कोई Graph represent कर रहा है कोई Network जैसे roadnetwork, railway network, computer network तो अपने को Articulation point पर Dhyan देना होगा becoz agr waha failure हुआ तो it will lead to break the graph into multiple components.

→ कैसे Remove करे Articulation point, join an edge between components.

unacademy

Scanned with CamScanner



	1	2	3	4	5	6
d =	1	6	3	2	4	5
L =	1	1	1	1	3	3

u, v → child
↓
Parent

$L[v] \geq d[u]$

then, u → articulation point

$L[5] \geq d[3]$

$3 \geq 3$

⇒ (3) is an articulation point

unacademy

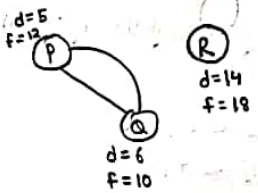
GATE-06

Q. Consider the DFS of an undirected graph with 8 vertices P, Q, and R. Let discovery time d(v) represent the time instant when the vertex v is first visited and finish time f(v) represent the time instant when the vertex v is last visited. Given that

d(P) = 5 unit f(P) = 12 unit

d(Q) = 6 unit f(Q) = 10 unit

d(R) = 14 unit f(R) = 18 unit



Which one of the following statements is TRUE about the graph?

(a) There is only one connected component

(b) There are two connected components, and P and R are connected

(c) There are two connected components, and Q and R are connected

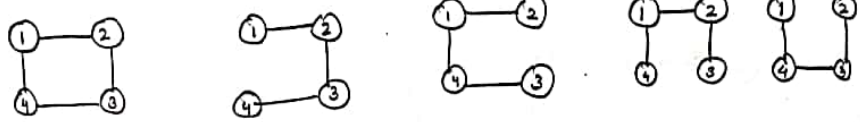
(d) There are two connected components, and P and Q are connected

unacademy

Minimum Cost Spanning Tree:

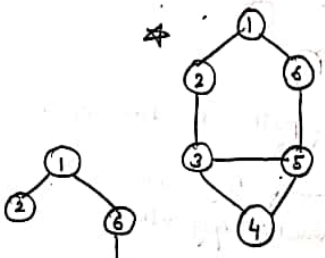
→ Spanning tree is a subgraph of a graph having all the vertices but (n-1) edges.

→ Result of that subgraph will always be a tree with no cycle.



|E| = 4

$^4C_3 = 4$



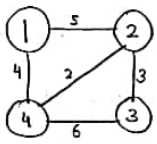
|E| = 7

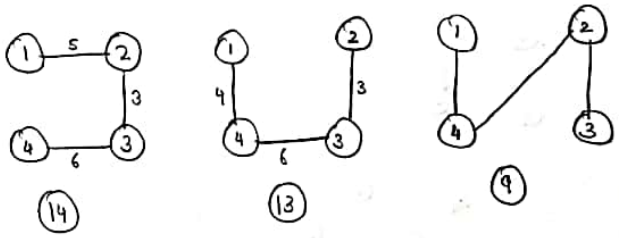
$^7C_5 - 2$

$^{|E|}C_{|V|-1}$ − no. of cycles

No. of spanning tree

unacademy

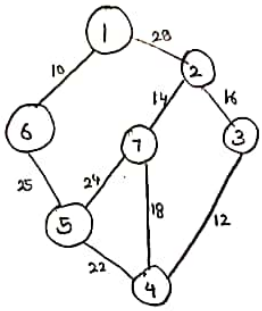




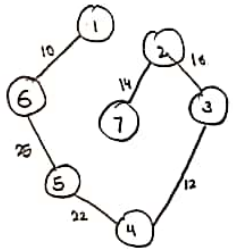
Greedy Algorithm to find Minimum Cost Spanning Tree:

→ Prim's Algorithm

→ Kruskal Algorithm

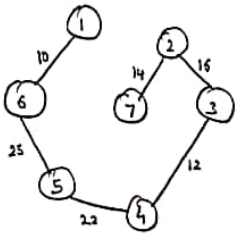


Prim's



99

kruskal's



99

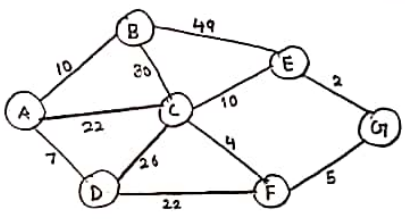
$\Theta(|V||E|) = \Theta(n^2)$

→ If we use min heap, no searching is required kyuki jb bhi delete krenge apne ~~ki~~ min. value milegi.

$\Theta(n \log n)$

GATE-04

Q. Consider the undirected graph below:



vertices visited = {A, D, B, C ....

Edge Choosen = {(AB) 10, (AC) 22, (AD) 7, DF, 22, BC, BE 20 49, CE, 10 (F) CD 26 ...

A	B, C, D
B	A, C, E
C	A, B, E, F, D
D	A, C, F
E	B, C, G
F	D, C, G
G	E, F

Using Prism's algorithm to construct a minimum spanning tree starting with node A which one of the following sequence of edges represents a possible order in which the edges would be added to construct the minimum spanning tree?

A. (E, G), (C, F), (F, G), (A, D), (A, B), (A, C)

B. (A, D), (A, B), (A, C), (C, F), (G, E), (F, G)

C. (A, B), (A, D), (D, F), (F, G), (G, E), (F, C)

D. (A, D), (A, B), (D, F), (F, C), (F, G), (G, E)

Topological Sort :

unacademy

→ It is a linear ordering of its vertices such that for every directed edge uv for vertex u to v, u must come before v in the ordering.

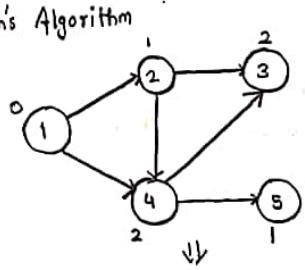
→ Graph should be DAG

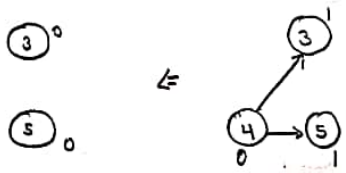
→ Every DAG will have atleast one topological ordering. U→V

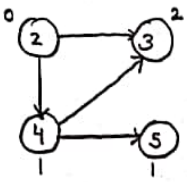
→ Time complexity = O(V+E) ← Kahn's Algorithm

1 2 4 3 5

1 2 4 5 3



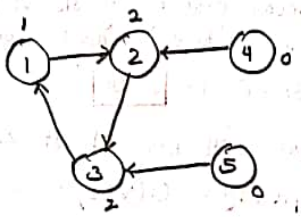




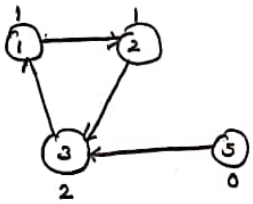
unacademy

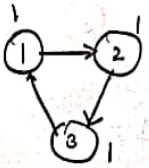
4 5 ?

5 4 ?



→ For graph having directed cycle topological ordering is not possible.

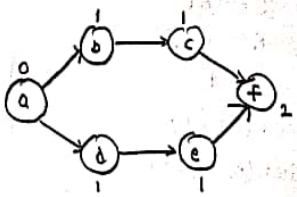




unacademy

GATE-16

Q. Consider the following directed graph:
The no. of different topological orderings of the vertices of the graph is ____.

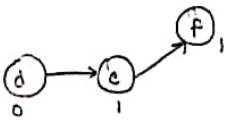


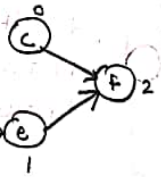
a → b → c → d → e → f

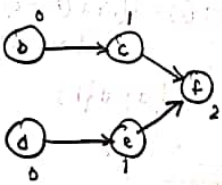
a → b → d → c → e → f

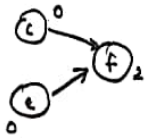
a → b → d → e → c → f

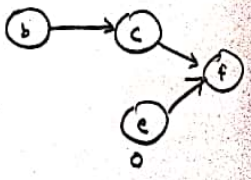
a → d





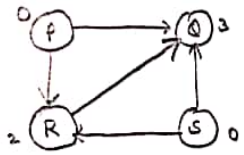






GATE-11

Q. Consider the directed graph below:



P S R Q

S P R Q

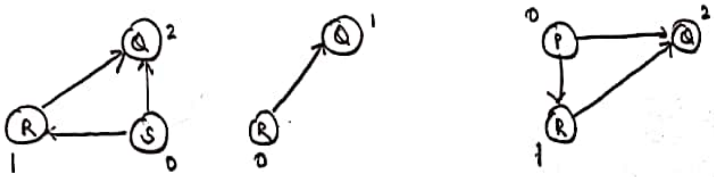
Which one of the following is TRUE?

A. The graph does not have any topological ordering.

B. Both PQRS and SRQP are topological orderings.

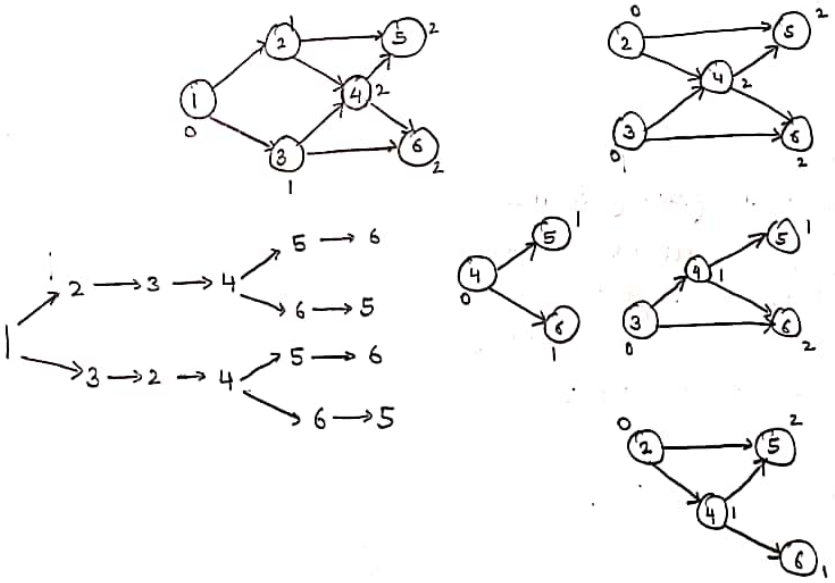
C. Both PSRQ and SPRQ are topological orderings.

D. PSRQ is the only topological ordering.



GATE-07

Q. Consider the DAG with V= {1, 2, 3, 4, 5, 6}.



Previou Years Questions on Graph Algorithm (in Hindi)

GATE-01

Q. Consider an unweighted graph G. Let Breadth-first traversal of G starting from a node r. Let $d(r,u)$ and $d(r,v)$ be the lengths of the shortest paths from r to u and v respectively in G. If u is visited before v during the breadth-first traversal, which of the following statements is correct?

A. $d(r,u) < d(r,v)$

B. $d(r,u) > d(r,v)$

C. $d(r,u) \leq d(r,v)$

D. None of the above

BFS shortest path Nikalne ke kaam ata hai.

→ Ques. me bola diya hai u pehle traverse hoga matlab u source se pass hoga

∴ $d(r,u) \leq d(r,v)$

GATE-05

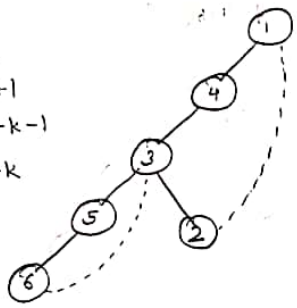
Q. In a depth first traversal of a Graph G with n vertices, are marked as tree edges. The no. of connected components in G is

a. k

b. k+1

c. n-k-1

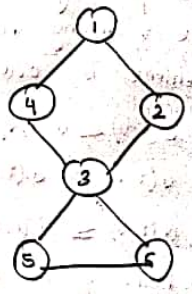
d. n-k



n = 6

k = 5

Here, you can clearly see the no. of connected component = 1

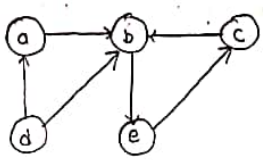


∴ n-k = 6-5 = 1

(d) n-k ✓

GATE TIFR-14

Q. Consider the following directed graph:



Suppose a depth first traversal of this graph is performed, assuming that whenever there is a choice, the vertex earlier in the alphabetical order is to be choosen. Suppose the number of tree edges is T, the no. of back edges B and the no. of cross edges is C. Then

a. B=1, C=1, and T=4

b. B=0, C=2, and T=4

c. B=2, C=1 and T=3

d. B=1, C=2 and T=3

e. B=2, C=2 and T=1

* जब भी बोला जाए choice toh alphabetical order lo!

A — B — E — C

No. of tree edge = 3 (T=3)

* Now, dekho koi cycle hai kya?

Haa bec

Milb Back edge hai C-B

B=1

* D को तो समझ से लगा दिए है।

D sirf cross edge bana skta. D-A & D-B

C=2

TIRF-13

Q. Given a weighted directed graph with n vertices where edge weights are integers, determining whether there are paths of arbitrarily large weight can be performed in time

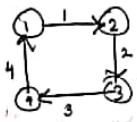
A. $O(n)$

B. $O(n \log n)$ but not $O(n)$

C. $O(n^{1.5})$ but not $O(n \log n)$

D. $O(n^3)$ but not $O(n^{1.5})$

E. $O(2^n)$ but not $O(n^3)$



→ ये काम इस कर सकते

→ यानी Bellman Ford

→ $O(VE)$

↓

-ve cycle wala edge we bana के !

→ $O(n^3)$

GATE-16

Q. BFS is started on a binary tree beginning from the root vertex. There is a vertex t at a distance four from the root. If t is the n^{th} vertex in this BFS traversal, then the maximum possible value of n is ___

Max. no. of nodes at height 'h'

$= \left\lceil \frac{n}{2^{h+1}} \right\rceil = \left\lceil \frac{n}{2^5} \right\rceil$

$n = 33$

Scanned with CamScanner

unacademy

Searching Algorithm

1. Linear Search or Sequential Search:

15	5	20	35	2	42	67	17
0	1	2	3	4	5	6	7

key = 42

```
for(i=0 ; i<n ; i++)
{   if(a[i] == data)
    {   printf("Element found at index : %d", i);
        break;
    }
}
if(i==n)
{   printf("Element not found");
}
```

★ Time complexity:

(i) Best case : $O(1)$

GATE (ii) Worst case : $O(n)$

GATE (iii) Avg. case : $\frac{1+2+\cdots n}{n}$

$\frac{n(n+1)}{2n}$

$O\left(\frac{n+1}{2}\right)$

unacademy

2. Binary Search:

★ Array sorted होना चाहिए !

```
int BinarySearch(a, n, data)
{   l=0 , r=n-1
    while (l<r)
    {   mid = (l+r)/2 ;
        if (data == a[mid])
            return mid;
        else if (data < a[mid])
            r = mid-1;
        else
            l = mid+1;
    }
    return -1;
}
```

5	9	17	23	25	45	59	63	71	89
0	1	2	3	4	5	6	7	8	9

l, mid, mid, mid, r

n = 10 , data = 17

l=0, r=9	l=0, r=3	l=2, r=3
while(0<9)	While (0<3)	while(2<3)
mid = 0+9/2 = 4	mid = 1	mid = 2
else if (17<25)	l = 2	if (17 = 17)
r = 3		17

★ Best case : $O(1)$

GATE-17 ★ Worst case : $O(\log n)$

Let A be an array of 31 no. jisme seq. of 0's & 1's है. Find the smallest index i such that A[i] = 1 by probing the min. no. of locations in A. The worst case no. of probes ___

IFR-17

Q. An array of n distinct element is said to be un-sorted if for i such that $2 \le i \le n-1$, either $A[i] > \max\{A[i-1], A[i+1]\}$ or $A[i] < \min\{A[i-1], A[i+1]\}$. What is the time complexity of the fastest algorithm that takes as input a sorted array A with n distinct elements and un-sorts A?

A. $O(n \log n)$ but not $O(n)$

B. $O(n)$ but not $O(\sqrt{n})$

C. $O(\sqrt{n})$ but not $O(\log n)$

D. $O(\log n)$ but not $O(1)$

E. $O(1)$

1	2	3	4	5	6	7	8

→ pairwise swap करो!

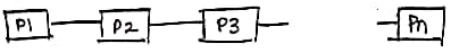
→ 2 1 4 3 6 5 8 7

→ क्योंकि हर एक element exactly एक बार swap हो रहा!

→ Time Complexity = $O(n)$

IFR-13

Q. Suppose n processors are connected in a linear array. Each p[illegible] has a no. The processors need to exchange numbers so that the numbers eventually appear in ascending order (processor P1 should have the minimum value and the processor P_N should have the maximum value).



The algorithm to be employed is the following: Odd numbered processors and even numbered processors are activated alternate steps; assume that in the first step all the even numbered processors are activated. When a processor is activated, the no. it holds is compared with the no. held by its right hand neighbour and the smaller of the two no. is retained by the activated processor and the bigger stored in its right hand neighbour. How long does it take for processors to sort the value?

(a) $n \log n$ steps

(b) n^2 steps

(c) n steps

(d) $n^{1.5}$ steps

4 3 2 1

3 4 1 2 — ①

3 4 1 2 — ②

3 4 1 2 ⇔ 3 1 2 4 — ③

3 1 4 2 ⇔ 1 2 3 4 — ④

IFR-12

Q. An array A contains n-integers. We wish to sort A in ascen[illegible] We are told that initially no element of A is more than a distance k away from its final position in the sorted list. Assume that n and k are large and k is much smaller than n. Which of the following is true for the worst case complexity of sorting A?

(a) A can be sorted with constant kn comparison but not with fewer comparisons.

(b) A cannot be sorted with less than constant $n \log n$ comparisons.

(c) A can be sorted with constant n comparisions

(d) A can be sorted with constant $n \log k$ comparisons but not with fewer comparison

(e) A can be sorted with constant $k^2 n$ comparisons but not fewer.

अगर बहुत ध्यान से सोचो तो ये कुछ नहीं :

Sort करना है एक k-sorted array को!

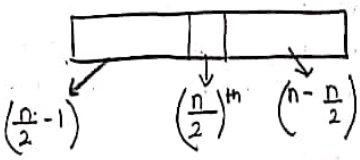
↳ We can sort such array most efficiently with heap sort.

↳ $O(n \log k)$

IIFR-14,12

Q. Consider, the quick sort algorithm on a set of n numbers, where recursive subroutine of the algorithm, the algorithm chooses the median of that set as the pivot. Then, which of the following statement is TRUE?

(a) The running time of the algorithm is $\Theta(n)$
(b) The running time of the algorithm is $\Theta(n \log n)$
(c) The running time of the algorithm is $\Theta(n^{1.5})$
(d) The running time of the algorithm is $\Theta(n^2)$



$$T(n) = T\left(\frac{n}{2}-1\right) + T\left(n-\frac{n}{2}\right) + \Theta(n)$$

$$= \boxed{\Theta(n \log n)}$$

TIFR-11

Q. The first n cells of an array L contain positive integers in decreasing order, and the remaining m-n cells contain 0. Then, given an integer x, how many comparisons can one find the position of x in L?

A. At least n comparisons are necessary in the worst case.
B. At least $\log m$ comparisons are necessary in the worst case.
C. $O(\log(m-n))$ comparisons suffice.
D. $O(\log n)$ comparisons suffice.
E. $O(\log(m/n))$ comparisons suffice.

2	1	0

$n = 3$
$m = 4$
$x = 1$

→ Search के लिए क्या lagaoge?

→ Binary Search : $O(\log n)$

TIFR-11

Q. Given, a set of $n = 2^k$ distinct numbers, we would like to determine the smallest & the second smallest using comparisons. Which of the following statements is TRUE?

A. Both these elements can be determined using $2k$ comparisons.
B. Both these elements can be determined using $n-2$ comparisons.
C. Both these elements can be determined using $n+k-2$ comparisons.
D. $2n-3$ comparisons are necessary to determine these two elements.
E. nk comparisons are necessary to determine these two elements.

$n = 2, k = 1$

↓

mtlb agr do hi element दे तो btao kitne comparison lagenge?

$n = 4, k = 2$

1	2	3	4

Linked Lists

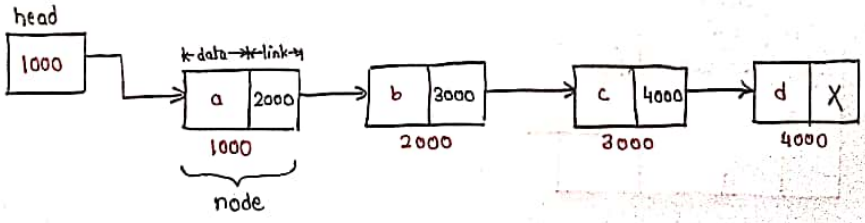
unacademy

→ Linear Data Structure in which elements are connected in linear order.

- → Singly linked list (SLL)
- → Doubly linked list (DLL)
- → Circular linked list
  - → Circular singly linked list (CSL)
  - → Circular doubly linked list (CDL)

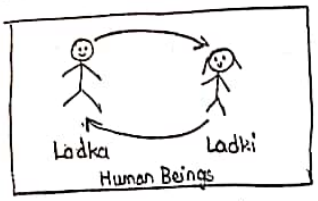
→ Linked list/one way linked list is a linear collection of data items.

→ Elements are not stored in contiguous location.

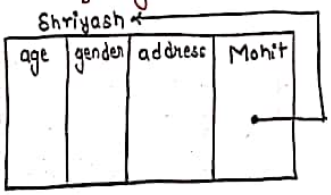


unacademy

→ Self referential structure:




```
Struct Person
{   int age;
    char Naam, pata[20];
    Struct Person * Mohit;
} Shriyash;
```



Operations on Linked lists:

- → Traverse a node
- → Insert a new node
- → Delete existing node
- → Search for a node
- → Sort Nodes in ascending or descending order

unacademy

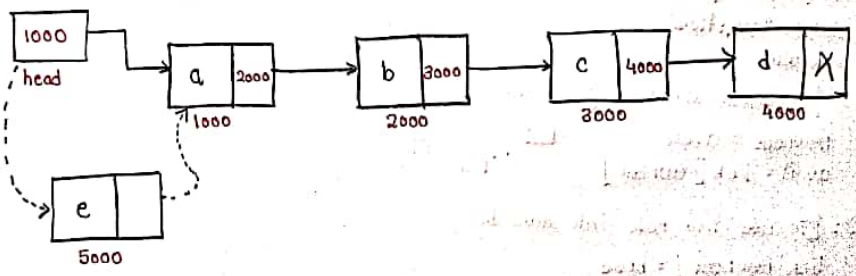
(1) Singly Linked List:

→ A singly linked list is one in which all the nodes are linked together in some sequential manner.

→ First node is accessed by external pointer (head).

→ Last node contains NULL value.

✡ Insertion at beginning:



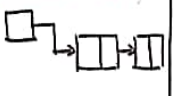
unacademy

Insert_Begin (head, data, link, avail, item)

[head: pointer pointing to first node
data: data part of the node
link: address part of the node
avail: avail list for getting new node
item: item to be inserted

Step1: [To check overflow]
if (avail=-1) then
write "overflow"
return

Step2: [To break the node from avail list]
newnode = avail
avail = link[newnode]



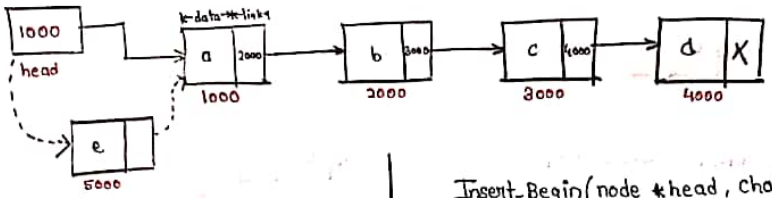
Step3: [To add new node into data list]
link[newnode] = head
head = newnode

Step4: [To add item]
data[newnode] = item

Step5: Exit

unacademy

C- program implementation:

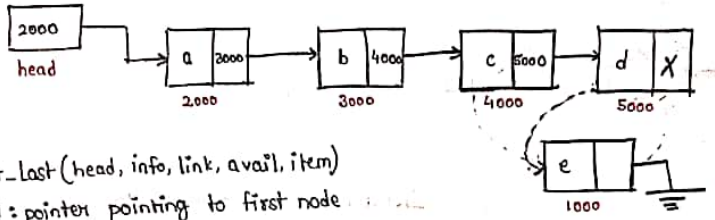



```
Struct node
{   char data;
    struct node *link;
}n1, *n2, *n3 ;

typedef structnode node ;

node*  nodebanao (char info)
{       node *p;
        p = (node*) malloc (size_of(node));
        p→ data = 'a';
        p → link = NULL;
}       return p;
```

```
Insert_Begin(node *head, char info)
{  node  * newnode;
   newnode = (node*) malloc (sizeof (node));
   newnode → data = info;
   newnode → link = head ;
   head = newnode;
   return head ;
}
```


unacademy

☆ Insertion at last:



Insert_Last (head, info, link, avail, item)

[head: pointer pointing to first node
data: data part of the node
Link: address part of the node
avail: avail list for getting new node
item: item to be inserted

Step1: [To check overflow]
if(avail=-1) then
write "overflow"
return

Step2: [To break the node from avail list]
newnode = avail
avail = link[newnode]

Step3: [To reach to the end of data list]
if (~~start~~ head == -1)
~~start~~ head = newnode
link[newnode] = -1

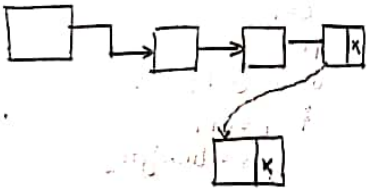
else
ptr = start
repeat step while (link[ptr] ≠ -1)
ptr = link[ptr]

Step4: link[ptr] = newnode
link[newnode] = -1

Step5: info[newnode] = item

Step6: Exit

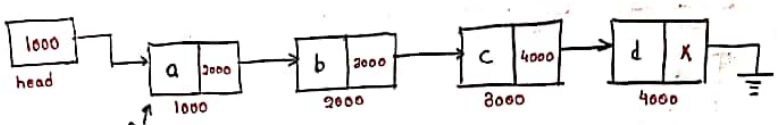
* C- programming implementation :



```
Insert_End (node *head, char info)
{
    node *p;
    p = (node *) malloc (sizeof(node));
    p -> data = info;
    p -> link = NULL;
    node *Q = head;
    while (Q -> link != NULL)
    {
        Q = Q -> link;
    }
    Q -> link = p;
    return head;
}
```

* Insertion at the middle of the list :

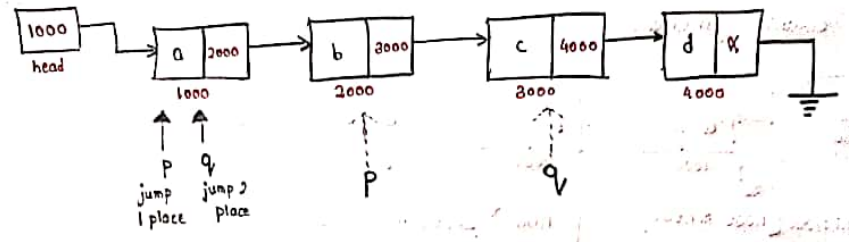
1. Using count function:

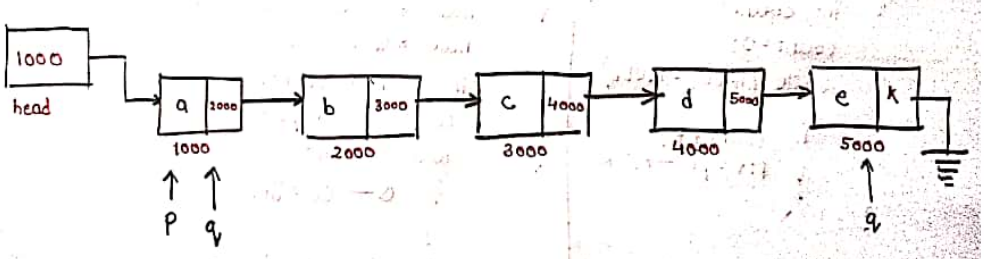



```
Countnode(node *head)
{
    int count;
    count = 0;
    While (head != NULL)
    {
        count++;
        ptr = ptr -> link;
    }
}
```

```
node* Insertmiddle (node *head, char, info)
{
    node *p = Createnode (info);
    node *Q = head;
    int count = countnode (head);
    count = count/2;
    while (--count)
    {
        Q = Q -> link;
    }
    p -> link = Q -> link;
    Q -> link = p;
    return head;
}
```

2. Using pointer :



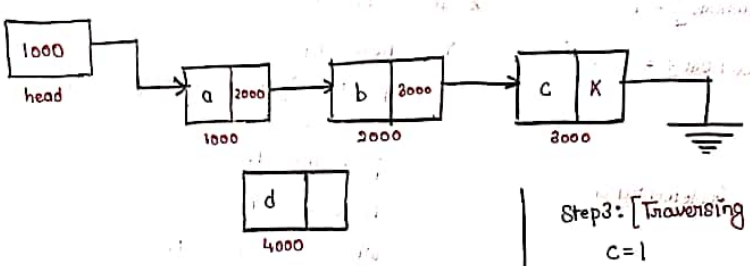


```
node * middle_chalo(node *head)
{   node *p, *q;
    p = q = head;
    while ((q->link != NULL) && (q->link->link != NULL))
    {       q = q->link->link;
            p = p->link;
    }
    return p;
}
Insertmiddle(node *head, char info)
{   node *p = createnode(info);
    node *Q = middle_chalo(head);
    p->link = Q->link;
    Q->link = P;
    return head;
}
```

unacademy

Scanned with CamScanner

* Insertion at any position :



Insert_At_any-pos (head, info, link, avail, pos, item)

Step1: [To check overflow]
if (avail=-1) then
write "Overflow"

Step2: [To break the node from avail list]
newnode = avail
avail = link[newnode]

Step3: [Traversing to given position]
c=1
ptr = start
while (c ≠ pos-1)
{ c=c+1
ptr = link[ptr]
}

Step4: link[newnode] = link[ptr]
link[ptr] = newnode

Step5: info[newnode] = item

Step6: Exit

unacademy

Scanned with CamScanner

C - programming implementation :

```
Insert_at_any_pos(node *head, char info)
{   node *p = Createnode(info);
    Node *q = head;
    for (i=1; i< pos-1 && q!= NULL; i++)
    {   q->link = p;
        return head;
    }
}
```

unacademy

Scanned with CamScanner

Linked Lists

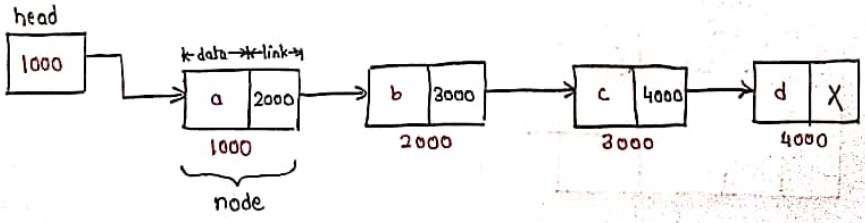
unacademy

→ Linear Data Structure in which elements are connected in linear order.

- → Singly linked list (SLL)
- → Doubly linked list (DLL)
- → Circular linked list
  - → Circular singly linked list (CSL)
  - → Circular doubly linked list (CDL)

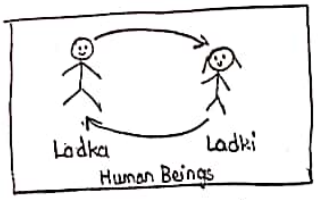
→ Linked list/one way linked list is a linear collection of data items.

→ Elements are not stored in contiguous location.

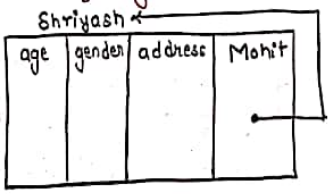


unacademy

→ Self referential structure:




```
Struct Person
{   int age;
    char Naam, pata[20];
    Struct Person * Mohit;
} Shriyash;
```



Operations on Linked lists:

- → Traverse a node
- → Insert a new node
- → Delete existing node
- → Search for a node
- → Sort Nodes in ascending or descending order

unacademy

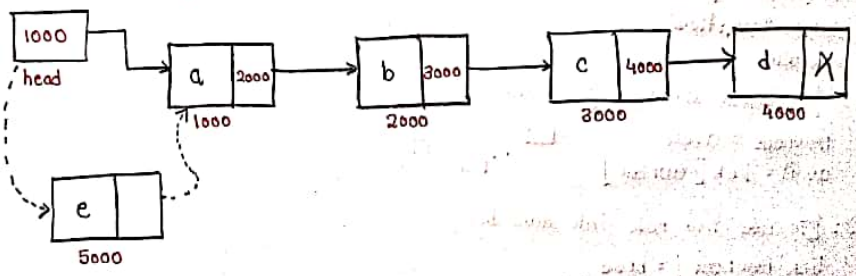
(1) Singly Linked List:

→ A singly linked list is one in which all the nodes are linked together in some sequential manner.

→ First node is accessed by external pointer (head).

→ Last node contains NULL value.

✡ Insertion at beginning:



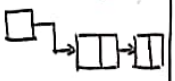
unacademy

Insert_Begin (head, data, link, avail, item)

[head: pointer pointing to first node
data: data part of the node
link: address part of the node
avail: avail list for getting new node.
item: item to be inserted

Step1: [To check overflow]
if (avail = -1) then
write "overflow"
return

Step2: [To break the node from avail list]
newnode = avail
avail = link[newnode]



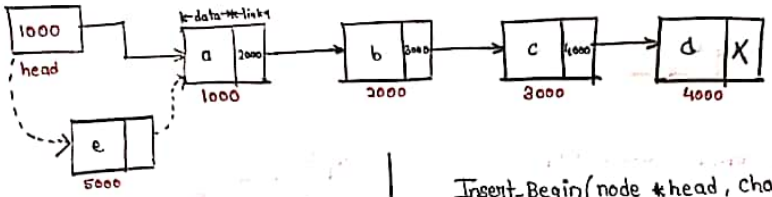
Step3: [To add new node into data list]
link[newnode] = head
head = newnode

Step4: [To add item]
data[newnode] = item

Step5: Exit

unacademy

C- program implementation:

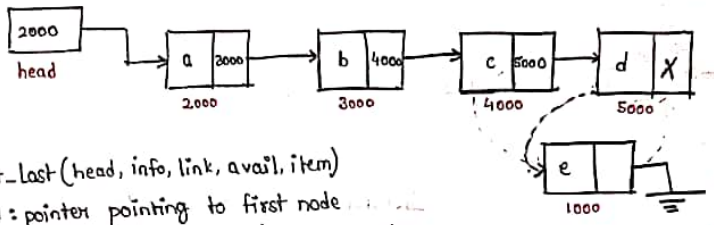



```
Struct node
{   char data;
    struct node *link;
};n1, *n2, *n3;

typedef structnode node;

node* nodebanao (char info)
{       node *p;
        p = (node*) malloc (size_of(node));
        p→ data = 'a';
        P → link = NULL;
}       return p;
```

```
Insert_Begin(node *head, char info)
{  node  *newnode;
   newnode = (node*) malloc (sizeof (node));
   newnode → data = info;
   newnode → link = head;
   head = newnode;
   return head;
}
```


unacademy

✱ Insertion at last:



Insert_Last (head, info, link, avail, item)

[head: pointer pointing to first node
data: data part of the node
Link: address part of the node
avail: avail list for getting new node
item: item to be inserted

Step1: [To check overflow]
if (avail = -1) then
write "overflow"
return

Step2: [To break the node from avail list]
newnode = avail
avail = link[newnode]

Step3: [To reach to the end of data list]
if (~~start~~ head == -1)
~~start~~ head = newnode
link[newnode] = -1

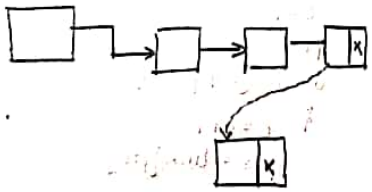
else
ptr = start
repeat step while (link[ptr] ≠ -1)
ptr = link[ptr]

Step4: link[ptr] = newnode
link[newnode] = -1

Step5: info[newnode] = item

Step6: Exit

* C- programming implementation :



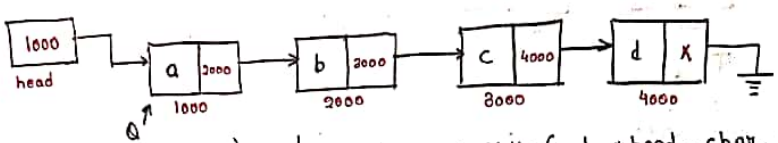
```
# Insert_End (node *head, char info)
{
    node *p;
    p = (node *) malloc (sizeof(node));
    p -> data = info;
    p -> link = NULL;
    node *Q = head;
    while (Q -> link != NULL)
    {
        Q = Q -> link;
    }
    Q -> link = p;
    return head;
}
```

unacademy

Scanned with CamScanner

* Insertion at the middle of the list :

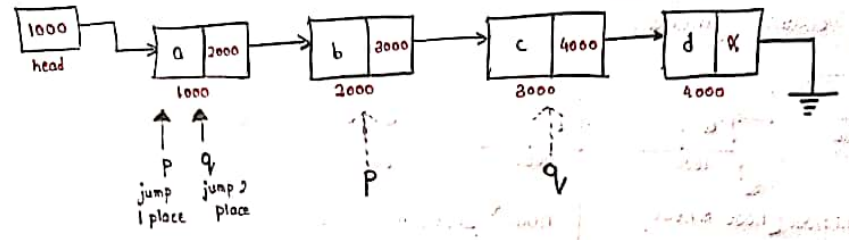
① Using count function:

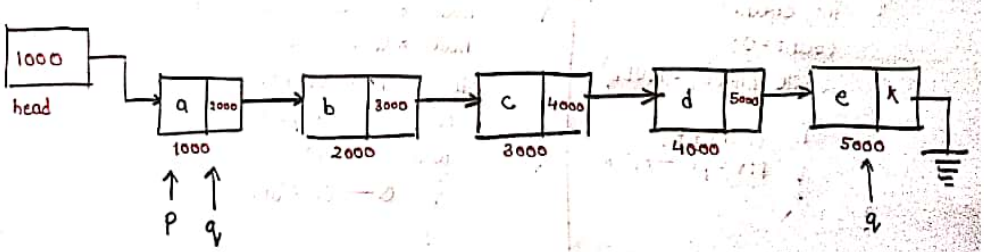



```
Countnode(node *head)
{
    int count;
    count = 0;
    While (head != NULL)
    {
        count++;
        ptr = ptr -> link;
    }
}
```

```
node* Insertmiddle (node *head, char, info)
{
    node *p = Createnode (info);
    node *Q = head;
    int count = countnode (head);
    count = count/2;
    while (--count)
    {
        Q = Q -> link;
    }
    P -> link = Q -> link;
    Q -> link = P;
    return head;
}
```

② Using pointer :



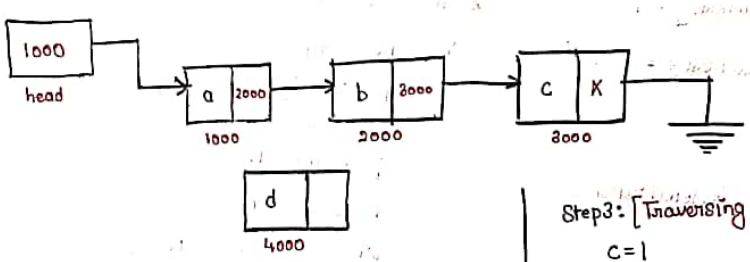


```
node * middle_chalo(node *head)
{   node *p, *q;
    p = q = head;
    while ((q->link != NULL) && (q->link->link != NULL))
    {       q = q->link->link;
            p = p->link;
    }
    return p;
}
Insertmiddle(node *head, char info)
{   node *p = Createnode(info);
    node *Q = middle_chalo(head);
    p->link = Q->link;
    Q->link = P;
    return head;
}
```

unacademy

Scanned with CamScanner

* Insertion at any position :



Insert_At_any-pos (head, info, link, avail, pos, item)

Step1: [To check overflow]
if (avail=-1) then
write "Overflow"

Step2: [To break the node from avail list]
newnode = avail
avail = link[newnode]

Step3: [Traversing to given position]
c=1
ptr = start
while (c≠ pos-1)
{ c=c+1
ptr = link[ptr]
}

Step4: link[newnode] = link[ptr]
link[ptr] = newnode

Step5: info[newnode] = item

Step6: Exit

unacademy

Scanned with CamScanner

C - programming implementation :

```
Insert_at_any_pos(node *head, char info)
{   node *p = Createnode(info);
    node *q = head;
    for (i=1; i< pos-1 && q!=NULL; i++)
    {   q->link = p;
        return head;
    }
}
```

Linked Lists

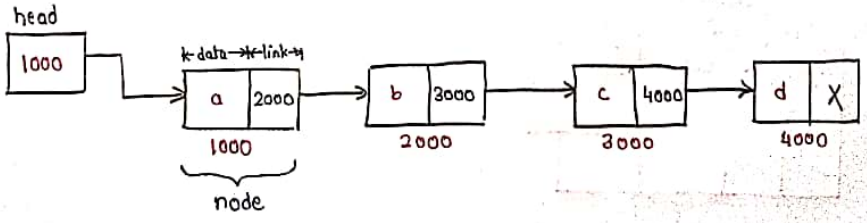
unacademy

→ Linear Data Structure in which elements are connected in linear order.

- → Singly linked list (SLL)
- → Doubly linked list (DLL)
- → Circular linked list
  - → Circular singly linked list (CSL)
  - → Circular doubly linked list (CDL)

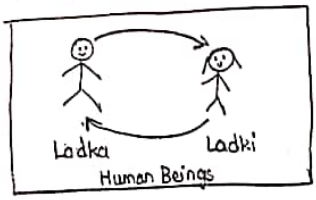
→ Linked list/one way linked list is a linear collection of data items.

→ Elements are not stored in contiguous location.

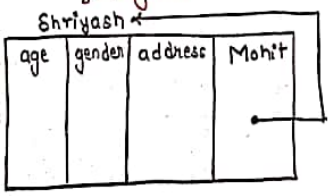


unacademy

→ Self referential structure:




```
Struct Person
{   int age;
    char Naam, pata[20];
    Struct Person * Mohit;
} Shriyash;
```



Operations on Linked lists:

- → Traverse a node
- → Insert a new node
- → Delete existing node
- → Search for a node
- → Sort Nodes in ascending or descending order

unacademy

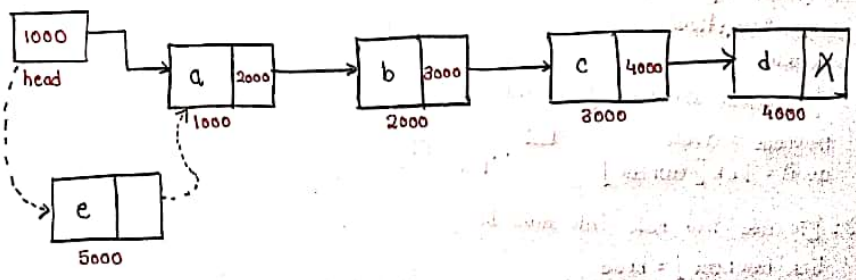
(1) Singly Linked list:

→ A singly linked list is one in which all the nodes are linked together in some sequential manner.

→ First node is accessed by external pointer (head).

→ Last node contains NULL value.

✡ Insertion at beginning:

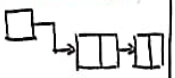


Insert_Begin (head, data, link, avail, item)

[head: pointer pointing to first node
data: data part of the node
link: address part of the node
avail: avail list for getting new node
item: item to be inserted

Step1: [To check overflow]
if (avail=-1) then
write "overflow"
return

Step2: [To break the node from avail list]
newnode = avail
avail = link[newnode]

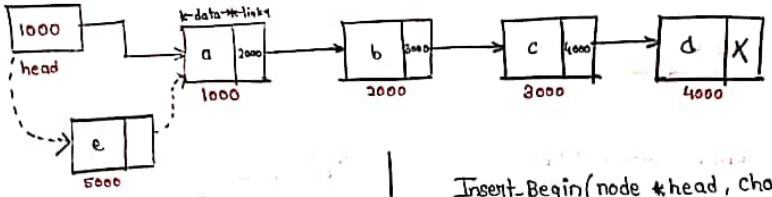


Step3: [To add new node into data list]
link[newnode] = head
head = newnode

Step4: [To add item]
data[newnode] = item

Step5: Exit

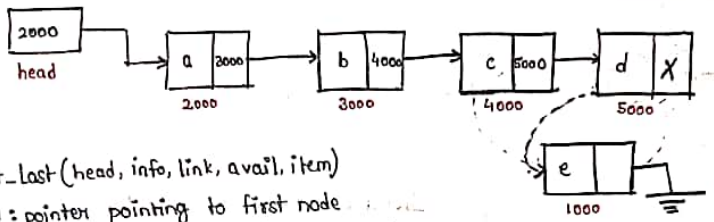
C- program implementation:




```
Struct node
{   char data;
    Struct node *link;
}n1, *n2, *n3;

typedef structnode node;

node* nodebanao (char info)
{       node *p;
        p = (node*) malloc (size_of(node));
        p→ data = 'a';
        p → link = NULL;
}       return p;
```

```
Insert_Begin(node *head, char info)
{ node *newnode;
  newnode = (node*) malloc (sizeof (node));
  newnode → data = info;
  newnode → link = head;
  head = newnode;
  return head;
}
```


✱ Insertion at last:



Insert_Last (head, info, link, avail, item)

[head: pointer pointing to first node
data: data part of the node
link: address part of the node
avail: avail list for getting new node
item: item to be inserted

Step1: [To check overflow]
if(avail=-1) then
write "overflow"
return

Step2: [To break the node from avail list]
newnode = avail
avail = link[newnode]

Step3: [To reach to the end of data list]
if (~~start~~ head == -1)
~~start~~ head = newnode
link[newnode] = -1

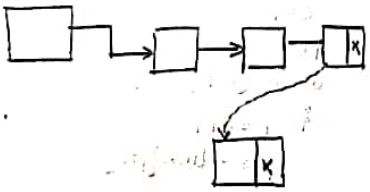
else
ptr = start
repeat step while (link[ptr] ≠ -1)
ptr = link[ptr]

Step4: link[ptr] = newnode
link[newnode] = -1

Step5: info[newnode] = item

Step6: Exit

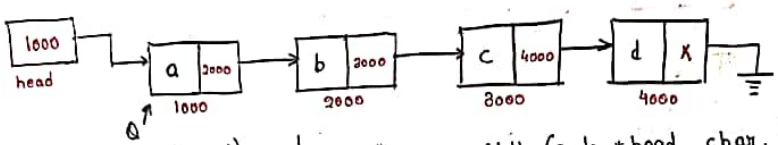
* C- programming implementation :



```
Insert_End (node *head, char info)
{
    node *p;
    p = (node *) malloc (sizeof(node));
    p -> data = info;
    p -> link = NULL;
    node *Q = head;
    while (Q -> link != NULL)
    {
        Q = Q -> link;
    }
    Q -> link = p;
    return head;
}
```

* Insertion at the middle of the list :

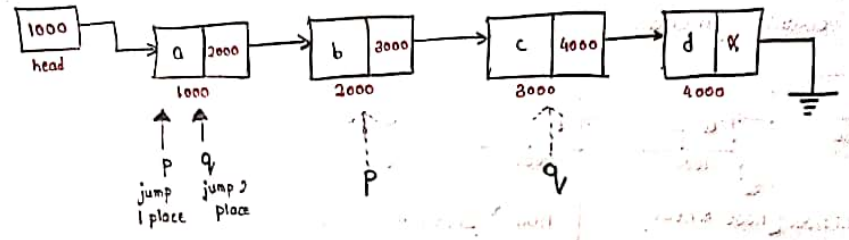
① Using count function:

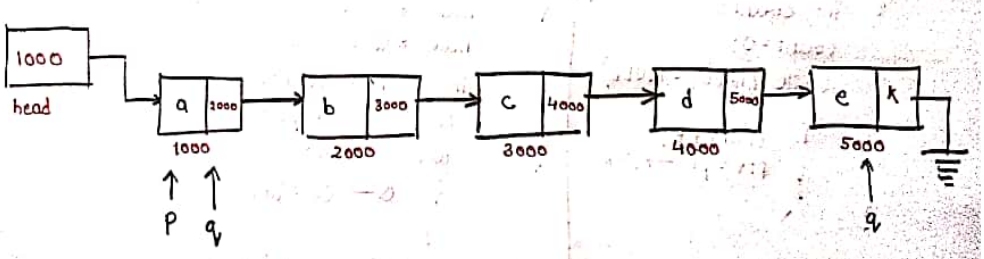



```
Countnode(node *head)
{
    int count;
    count = 0;
    while (head != NULL)
    {
        count++;
        ptr = ptr -> link;
    }
}
```

```
node* Insertmiddle (node *head, char, info)
{
    node *p = Createnode (info);
    node *Q = head;
    int count = countnode (head);
    count = count/2;
    while (--count)
    {
        Q = Q -> link;
    }
    p -> link = Q -> link;
    Q -> link = p;
    return head;
}
```

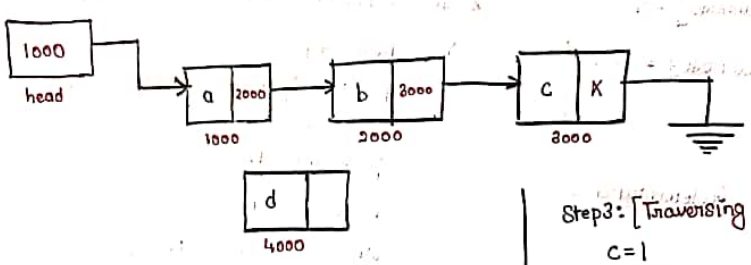
② Using pointer :





```
node * middle_chalo(node *head)
{   node *p, *q;
    p = q = head;
    while ((q->link != NULL) && (q->link->link != NULL))
    {       q = q->link->link;
            p = p->link;
    }
    return p;
}
Insertmiddle(node *head, char info)
{   node *p = createnode(info);
    node *Q = middle_chalo(head);
    p->link = Q->link;
    Q->link = P;
    return head;
}
```


* Insertion at any position :



Insert_At_any-pos (head, info, link, avail, pos, item)

Step1: [To check overflow]
if (avail=-1) then
write "Overflow"

Step2: [To break the node from avail list]
newnode = avail
avail = link[newnode]

Step3: [Traversing to given position]
c=1
ptr = start
while (c≠ pos-1)
{ c=c+1
ptr = link[ptr]
}

Step4: link[newnode] = link[ptr]
link[ptr] = newnode

Step5: info[newnode] = item

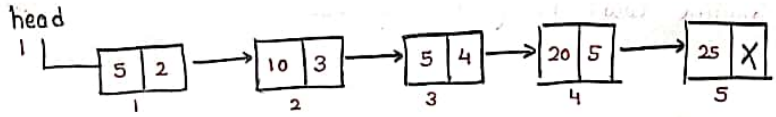
Step6: Exit

C - programming implementation :

```
Insert-at-any-pos(node *head, char info)
{   node *p = Createnode(info);
    Node *q = head;
    for (i=1; i< pos-1 && q!= NULL; i++)
    {   q->link = p;
        return head;
    }
}
```

unacademy

Q. Consider the following Linked list:



(i) p = head → link → link → link

p = 4

(ii) head → link → link = p → link

4 → 5

(iii) p → link → link = p

Null = 4

unacademy

Q Fill in the blanks:

The following routine count no. of node in SLL.

```
int count()
{
  struct node *p = head;
  int c=0;
  while ( p )
  {
     ++c;
     p = p→link;
  }
```

(A) p

(B) !p

(C) p→link

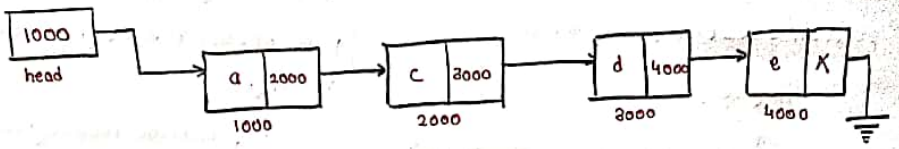
(D) !p→link

Q Fill in the blanks:

```
int Do(struct node *p)
{
  if( p )
    return 1+ Do(p→link);
  else
    return 0;
}
void main()
{
  printf("%d", Do(head));
}
```


unacademy

Printing Elements of linked list:




```
print_kr_de_bhai(node *head)
{
   while (head→link != NULL)
   {
     printf("%c", head→data);
     head = head→link;
   }
   printf("%c", head→data);
}
```

|| while (head != NULL)

O/P: a c d e

→ Using recursion :

```
print_kr_de_bhai(node *head)
{
    if (head)
    {
        printf("%c", head -> data);
        print_kr_de_bhai(head -> link);
    }
}
```

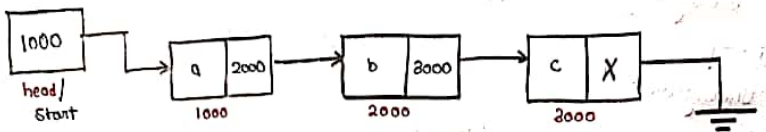
* Reverse भी print कर सकते है।

```
print_kr_de_reverse(node *head)
{
    if(head)
    {
        print_kr_de_reverse(head -> link);
        printf("%c", head -> data);
    }
}
```

ⓔ ⓓ ⓒ ⓐ

P(1000)	P(2000)	P(3000)	P(4000)	P(NULL)
if( ) ✓	if() ✓	if() ✓	if() ✓	if() X
ⓐ	ⓒ	ⓓ	ⓔ	
P(2000)	P(3000)	P(4000)	P(NULL)	

Deletion at first :



Del at first (info, link, avail, start)

Step1: [To check underflow]
if (start = -1) then
write " Underflow"
return

Step2: node = start
start = link[node]

Step3: [To add free node into avail list]
link[node] = avail
avail = node

Step 4: Exit

Del at_last (info, link, avail, start)

Step1: [To check underflow]
if (start = -1) then
write "Underflow"
return

Step2: [Traversing to last element]
ptr = start
repeat step while (link[ptr] ≠ -1)
ptr1 = ptr
ptr = link[ptr]

Step3: link[ptr1] = -1
link[ptr] = avail
avail = ptr

Del at any Pos (info, link, avail, start)

Step1: [To check underflow]
if (start = -1)
write " Underflow"
return

Step2: c = 1, ptr1 = -1
ptr = start
while (c ≠ pos)
{
ptr1 = ptr
c = c+1
ptr = link[ptr]
}

Step3: link[ptr1] = link[ptr]

Step4: [To add free node to avail list]
link[ptr] = avail
avail = ptr

Step 5: Exit

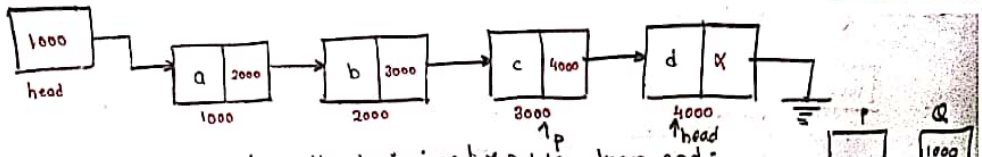
unacademy

Searching in singly linked list:


```
Struct node {
  int data;
  struct node *next;
};
typedef struct node node;

int search(node *head, int info)
{  if (head == NULL) return 0;
   while (head != NULL)
   {  if (head -> data == info) return 1;
      head = head -> next;
      return 0;
   }
}
```

```
node * search (node *head, int info)
{  if (head == NULL) return NULL;
   while (head != NULL)
   {  if (head -> data == info)
      {  return head;
         head = head -> next;
      }
      return NULL;
   }
```


unacademy



☆ Delete a node from the beginning:

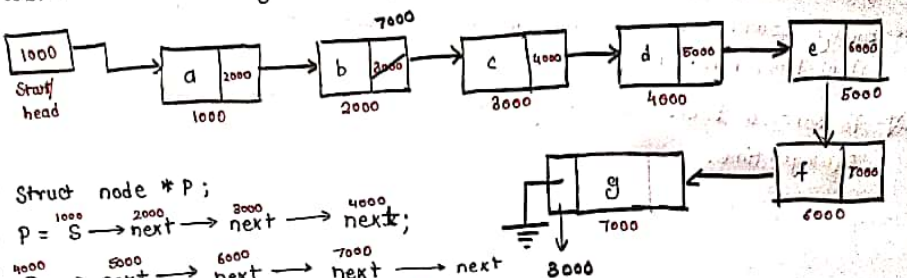
```
node * Delete_begin (node * head)
{
    node *p;
    p = head;
    head = head -> link;
    free(p);
    return head;
}
```

☆ Delete from end:

```
node * Delete_End (node *head)
{   node *p; node *Q = head;
    while (head -> link != NULL)
    {  p = head;
       head = head -> link;
    }
    free (head);
    p -> next = NULL;
    return Q;
```


unacademy

*** Q. Consider the linked list given below:



(1) Struct node *P;

(2) P = S -> next -> next -> next;
(1000, 2000, 3000, 4000)

(3) P -> next -> next -> next -> next
(4000, 5000, 6000, 7000)
= S -> next -> next;
(1000, 2000, 3000)

(4) S -> next -> next = P -> next -> next -> next;
(1000, 2000, 4000, 5000, 6000, 7000)

(5) Print(P -> next -> next -> next -> next -> next -> next -> next -> data)
(4000, 5000, 6000, 7000, 3000, 4000, 5000, 6000)

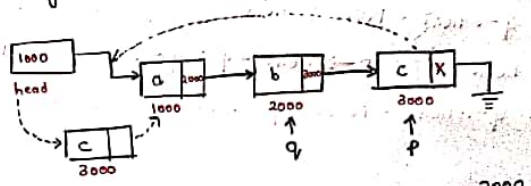
(f)

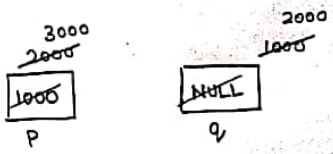
- Linear search works on linked lists. O(n)
- Binary search can be implemented on linked list but will take O(n) time, hence not used.
- To sort a linked list, merge sort can be used which will take O(nlogn) time.

GATE-10

Q. The following C function takes a singly linked list as input and modifies the list by moving the last element to the front of the list and returns the modified list. Some part of the code is left blank. Fill that blank:

```
typedef struct node {
int value;
struct node *next;
} node;
node *move_to_front(node *head)
{
    node *p, *q;
    if((head == NULL) || (head -> next == NULL))
        return head;
    q = NULL;
    p = head;
    while (p -> next != NULL)
    {
        q = p;
        p = p -> next;
    }
    return head;
```






```
p -> next = head;
q -> next = NULL;
head = p;
```

order chlega!

(a) q = NULL; p -> next = head; head = p;

(b) q -> next = NULL; head = p; p -> next = head;

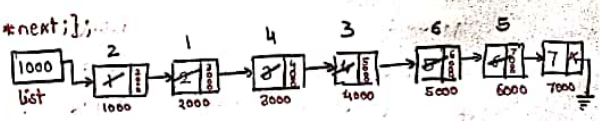
(c) head = p; p -> next = q; q -> next = NULL;

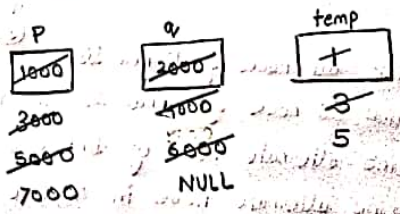
(d) q -> next = NULL; p -> next = head; head = p;

GATE-05

Q. The following C-function takes a singly linked list of integers as a parameter & rearrange the elements of the list. The list is represented as pointer to a structure. The function is called with the list after containing the integers 1,2,3,4,5,6,7 in the given order. What will be the contents of the list after the function completes execution?

```
struct node { int value; struct node *next;};
void rearrange (struct node *list)
{ struct node *p, *q;
  int temp;
  if (!list || !list -> next) return;
  p = list;
  q = list -> next;
  while (q) {
  temp = p -> value;
  p -> value = q -> value;
  q -> value = temp;
  p = q -> next;
  q = p ? p -> next : 0;
}}
```



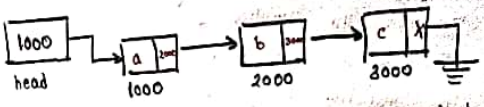


A) 1,2,3,4,5,6,7
B) 2,1,4,3,6,5,7
C) 1,3,2,5,4,7,6
D) 2,3,4,5,6,7,1

GATE

Q. What does the following function do for a given linked list with first node as head?

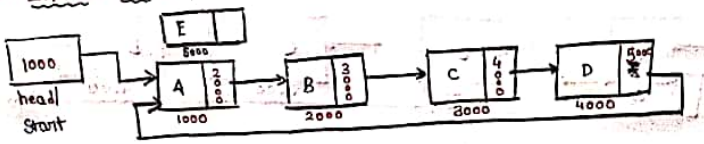
```
void fun1 (struct node *head)
{ if (head == NULL)
    return;
  fun1 (head -> next);
  printf ("%d", head -> data);
}
```



fun1(1000) | fun1(2000) | fun1(3000) | fun1(NU
if() X | if() X | if() X | if()✓
fun1(2000) | fun1(3000) | fun1(NULL)
a ③ | b ② | c ①

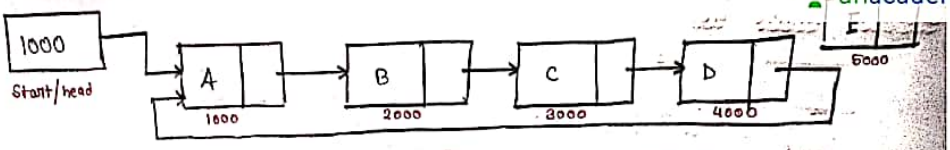
A. Prints all nodes of linked list
B. Prints all nodes of linked list in reverse order.
C. Prints alternate nodes of linked list.
D. Prints alternate nodes in reverse order.

Circular linked list :



Insert_Begin (info, link, start, avail, item)

1. [To check overflow]
   if (avail == -1) then
   write "overflow" return
2. [To break the node from avail list]
   newnode = avail
   avail = link[newnode]
3. if start == -1 then
   start = newnode
   link[newnode] = start
   else
   ptr = start
   repeat step while (link[ptr] ≠ start)
   ptr = link[ptr]
4. link[newnode] = start
   start = newnode
   link[ptr] = start
5. Exit



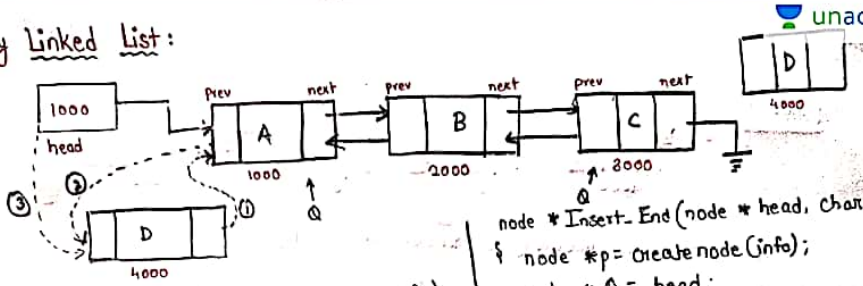
Insert_Last (info, link, start, avail, item)

1. [To check overflow]
   if (avail == -1) then
   write "Overflow"
   return
2. [To break the node from avail list]
   newnode = avail
   avail = link[newnode]
3. if start == -1
   start = newnode
   link[newnode] = start
   else
   ptr = start
   repeat step while link[ptr] ≠ start
   ptr = link[ptr]
4. link[ptr] = newnode
   link[newnode] = start
5. Exit

```
node *Insert_Begin (node *head, char info)
{
    node *p = Createnode (info);
    if (head == null)
    { head = newnode;
      newnode → next = head;
    }
    else {
        ptr = head;
        while (ptr → next != head)
        ptr = ptr → next;
    }
    newnode → next = head
    head = newnode
    ptr → next = head
}
```

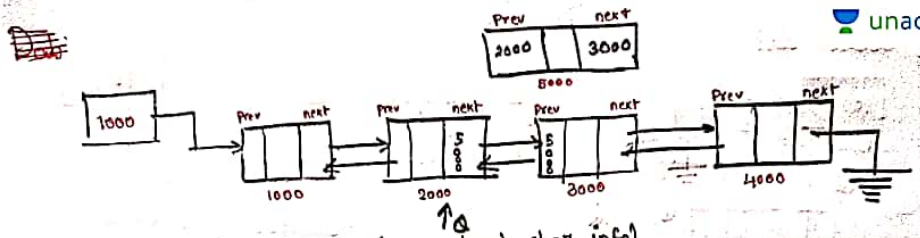
```
node *Last (node *head, char info)
{
    node *p = Createnode (info);
    node *Q = head;
    if (head == null)
    {
        head = newnode;
        newnode → next = head;
    }
    else {
        ptr = head;
        while (ptr → link != head)
            ptr = ptr → next;
    }
    ptr
    newnode → next = newnode;
    newnode → next = head;
}
```

Doubly Linked List:




```
node * Insert_Begin (node *head, char info)
{   node *newnode;
    newnode = (node*) malloc (sizeof (node));
    newnode → data = info;
    newnode → prev = null;
    newnode → next = null;
    newnode → next = head;
    head → prev = newnode
    head = newnode;
    return head;
}
```

```
node * Insert_End (node * head, char info)
{ node *p = Create node (info);
  node * Q = head;
if (head == null) { head = p; return head; }
else {
    while (Q → next != NULL)
    { Q = Q → next;
    }
    Q → next = p;
    p → prev = Q;
    return head;
  }
}
```



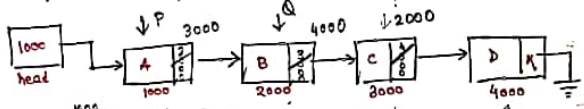

```
node * Insert_middle (node *head, char info)
{   node *p = Createnode(info);
    node *Q = reachmiddle(head);
    p -> prev = Q;
    p -> next = Q -> next;
    Q -> next -> prev = p;
    Q -> next = p
    return head;
}
```

GATE-05
Q. Which of the following statements is true?

I. As the no. of entries in a hash table increases, the no. of collission increases.
II. Recursive programs are efficient.
III. The worst case complexity of Quicksort is $O(n^2)$
IV. Binary search using a linked list is efficient

A. I and II B. II & III C. I & IV D. I & III

GATE
Q. Let p be the pointer as shown



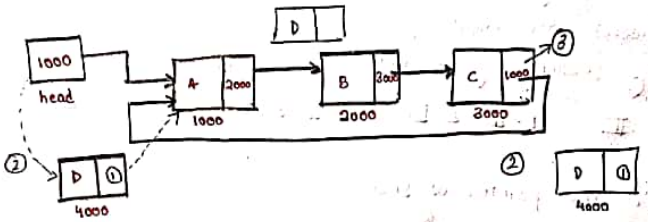
q = p -> next : 2000
p -> next = q -> next
q -> next = (q -> next) -> next
(p -> next) -> next = q

Swaps the two nodes next to p in the linked list

इस Code क्या कर रहा है?

GATE
Q. In a circular linked list organisation, insertion of a record involves modification of :

A. One pointer B. Two pointers C. Multiple pointers D. No pointers.



→ At beginning = 3
→ At last = 2
→ At middle = 2

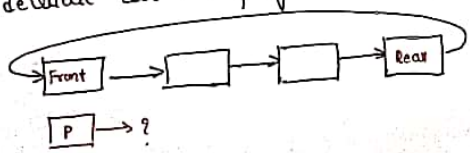
GATE UGC-NET
Q. Linked lists are not suitable data structure for which one of the following problems?

A. Insertion sort B. Binary Search C. Radix sort D. Polynomial manipulation

GATE-04
Q. Suppose each set is represented as a linked list with elements in order. What Which of the operations among union, intersection, membership, cardinality will be the slowest?

A. union only
B. intersection, membership
C. membership, cardinality $O(n_1 + n_2)$
D. union, intersection $O(n_1 \times n_2)$

GATE-04
Q. A circularly linked list is used to represent a Queue. A single variable p is used to access the Queue. To which node should p point such that both the operations enQueue & deQueue can be performed in constant time?



A. rear node
B. front node
C. not possible with a single ptr.
D. node next to front

* EnQueue : Insert newNode after Rear and make Rear point to inserted node :

```
newNode → next = rear → next;
rear → next = newNode;
rear = newNode;
```

* DeQueue : Delete the front node & make the second node the front node.

```
front = rear → next;
rear → next = front → next;
free(front);
```

Which one of the following is the time complexity of the most implementation of 'enqueue' and 'dequeue' respectively for this data structure.

(A) $\Theta(1)$, $\Theta(1)$
(B) $\Theta(1)$, $\Theta(n)$
(C) $\Theta(n)$, $\Theta(1)$
(D) $\Theta(n)$, $\Theta(n)$

→ For enqueue operation, → constant amount of time $\Theta(1)$ because it modifies only 2 pointers.

→ For Dequeue operation → we need address of second last node of single link list to make NULL, mtlb traversing krni hogi $\Theta(n)$

GATE-04

Q. Let P be a singly linked list. Let Q be the pointer to an inter... x in the list. What is the worst case time-complexity of the best known algorithm to delete the node x from the list?

A. O(n)
B. O(log² n)
C. O(log n)
D. O(1)

→ Traverse the LL until you find the node jisko delete karna hai।

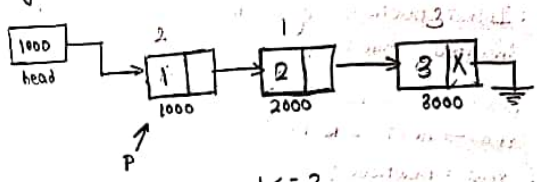
→ Lekin iske liye pointer should point to head node mtlb kuch gadbad hai।

→ Yaha pointer intermidiate node में है iss pointer को Brahmastra की तरह उपयोग करके O(1) में Delete कर सकते है।

GATE-03

Q. Consider the function f defined below:

```
struct item {
    int data;
    struct item *next;
};
int f(struct item *p) {
    return ((p == NULL) || (p->next == NULL) || ((p->data <= p->next->data) &&
                                                  f(p->next)));
}
```



For a given linked list p, the function f returns 1 if & only if

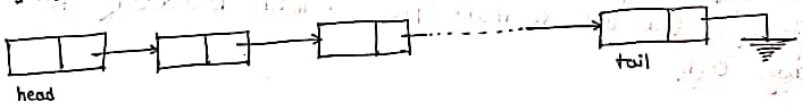
A. the list is empty or has exactly one element.
B. the elements in the list are sorted in non-decreasing order of data value.
C. the elements in the list are sorted in non-increasing order of data value.
D. not all elements in the list have the same data value.

The function f() works :

1) If linked list is empty return 1
2) Else if linked list has only one element return 1
3) Else if node→ data is smaller than equal to node→next→ data return 1
4) Else return 0

GATE-18

Q. A queue is implemented using a non-circular singly linked list. The queue has a head pointer and a tail pointer. Let n denote the number of nodes in the queue. Let 'enqueue' be implemented by inserting a new node at the head and 'dequeue' be implemented by deletion of a node from the tail.



GATE-17

Q. Consider the C code fragment given below:

```
typedef struct node {
  int data;
  node * next;
} node;

void join (node *m, node *n) {
  node *p = n;
  while (p -> next != NULL) {
    p = p -> next;
  }
  p -> next = m;
}
```

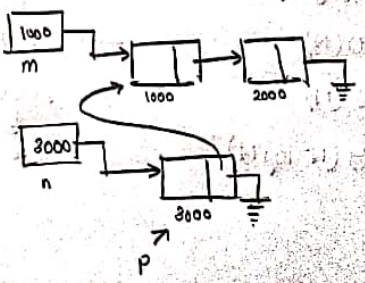
Assuming that m & n point to ~~m~~ & ~~n~~ valid NULL terminated linked lists, invocation of join will

A. append list m to the end of list n for all inputs.

B. either cause a null pointer dereference or append list m to the end of list n.

C. cause a null pointer dereference for all inputs.

D. append list n to the end of list m for all inputs



GATE-16

Q. N elements are stored in a sorted doubly linked list. For ~~a~~ operation, a pointer is provided to the records to be deleted. For a decrease-key operation, a pointer is provided to the record on which the operation is performed.

An algorithm performs the following operations on the list in this order: $\Theta(N)$ delete, $O(\log N)$ insert, ~~$O(\log N)$~~ find and $\Theta(N)$ decrease key. What is the time complexity of all these operations put together?

A. $O(\log^2 N)$

B. $O(N)$

C. $O(N^2)$

D. $\Theta(N^2 \log N)$

→ Pointer mila ~~है~~ Delete ~~करने~~ के ~~लिए~~

• Delete: $O(1)$

→ Insert: $O(N)$ → $O(N \log N)$, Find $O(N)$ → $O(N \log N)$ $O(1 \times N) = O(N)$

→ Decrease key: $O(N) \rightarrow O(N \times N) \rightarrow O(N^2)$

(Delete + insert)

Stack: LIFO/FILO

→ A stack is an Abstract data type commonly used in most programming languages.

→ It is name stack as it behaves like a real world stack.
For eg: a deck of cards or a pile of plates etc.

→ It allows operations at only one end.

→ A stack can be implemented by means of Array, structure, pointer & linked list और बाकी Questions में पता चलेगा!

→ Insertion operation is called Push & removal operation is known as POP.

→ It can be either fixed sized or dynamic ~~size~~ resizing.

★ Top pointer हमेशा top of the element को point करेगा!

Basic operations :

- Push() – pushing (storing) an element on the stack.
- Pop() – removing (accessing) an element from the stack.
- peek() – get the top data element of the stack without removing it.
- isFull() – Check if stack is full
- isEmpty() – Check if stack is empty.

(4)
```
int peek()
{ return stack[top];
}
```

(2)
```
bool isFull()
{ if (top==MAXSIZE)
    return true;
  else
    return false;
}
```

(3)
```
bool isEmpty()
{ if(top==-1)
    return true;
  else
    return false;
}
```

Push operation :

Push(Stack, Top, Maxstk, item)

Stack: Array
Top: Pointer ~~pointing~~ to top of stack index value
item: element to be inserted
Maxstk: max size of stack

(1) [To check overflow]
if(Top = Maxstk - 1) then
write "OverHow"

(2) Top = Top+1
Stack[Top] = item

(3) Exit

C-implementation :

```
void push (int data)
{ if (!isFull())
  {
    top = top+1;
    stack[top] = data;
  }
  else
  {
    printf(" नहीं Daal skte
            Stack full है");
  }
}
```

Pop Operation :

→ Accessing the content while removing it from stack is known as pop operation.

→ In an array implementation of pop() operation, the data element is not actually removed, instead top is decremented to a lower position in the stack to point to the next value.

→ But, in linked list implementation pop() actually removes data element and delocates memory space.

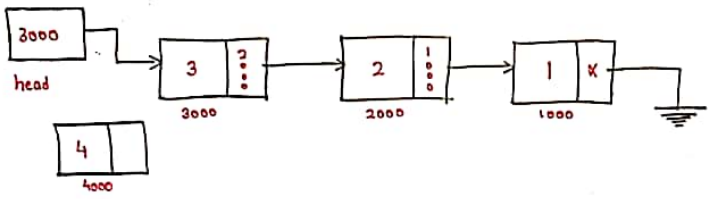
(1) [To check underflow]
if(Top==-1) then
write "Underflow"

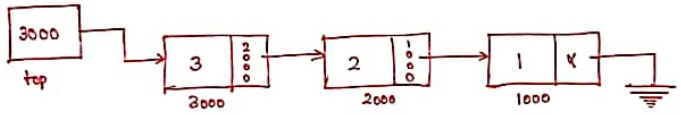
(2) Top = Top-1

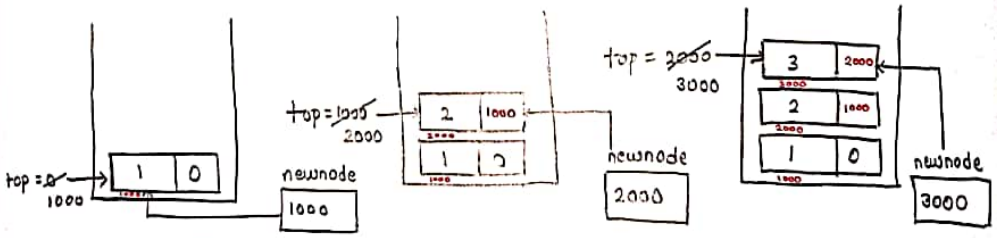
(3) Exit

```
int pop(int data)
{  if(!isEmpty())
   {
     data = stack[top];
     top = top-1;
     return data;
   }
   else { pf(" Nhi nikal skte khali
              ~~है~~ stack");
   }
}
```

Implementation of stack using linked list:

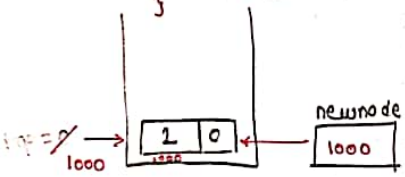







```
Struct node
{  int data;
   Struct node *link;
}
Struct node *top=0;
void push(int x)
{  Struct node *newnode;
   newnode = (Struct node*) malloc (sizeof(Struct node));
   newnode → data = X;
   newnode → next = top;
   top = newnode;
}
```

```
void main()
{  push(2);
   push(3);
   push(10);
   display();  10 3 2
   peek();  10
   pop();  10
   peek();  3
   display();  3 2
}
```

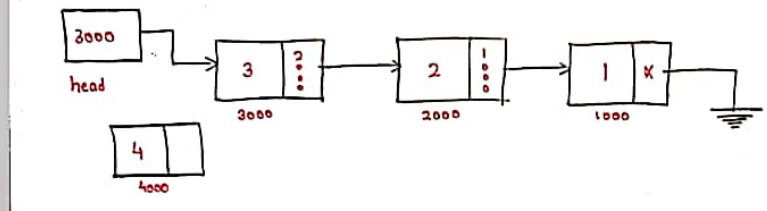


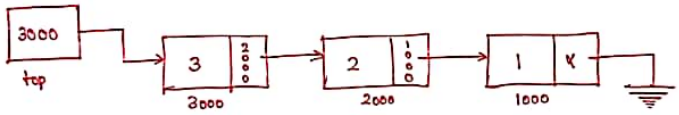

```
Void pop()
{  Struct node *temp;
   temp = top;
   if(top==0)
   {  pf("kuch nahi h");
   }
   else
   {  printf("%d", top→data);
      top = top→link;
      free(temp);
   }
}
```

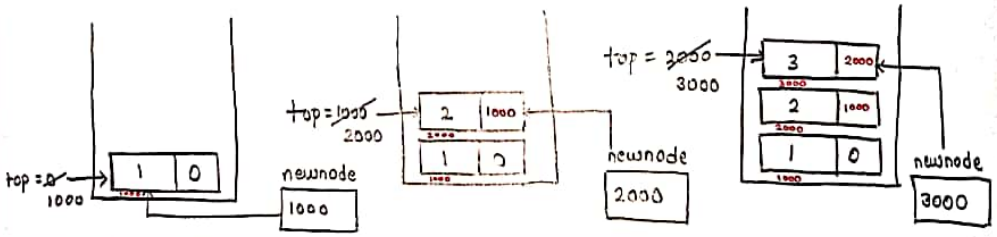
```
void display()
{ struct node *temp;
  temp = top;
  if (top==0)
  { pf("kuch nhi h");
  }
  else
  { while(temp!=0)
    { printf("%d", temp->data);
      temp = temp->link;
    }
  }
}
```

```
void peek()
{ if (top==0)
  { pf("kuch nhi h");
  }
  else
  { pf("top element is %d", top->data);
  }
}
```


Implementation of stack using linked list:

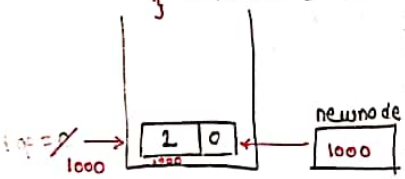







```
Struct node
{ int data;
  Struct node *link;
}
Struct node *top=0;
void push(int x)
{ Struct node *newnode;
  newnode = (Struct node*) malloc (sizeof(Struct node));
  newnode -> data = X;
  newnode -> next = top;
  top = newnode;
}
```

```
void main()
{ push(2);
  push(3);
  push(10);
  display();  10 3 2
  peek();  10
  pop();  10
  peek();  3
  display();  3 2
}
```



```
Void pop()
{ struct node *temp;
  temp = top;
  if (top==0)
  { pf("kuch नहीं है");
  }
  else
  { printf("%d", top->data);
    top = top->link;
    free(temp);
  }
}
```

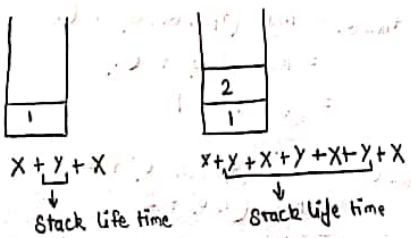
```
void display()
{ struct node *temp;
  temp = top;
  if (top==0)
  { pf("kuch nhi है");
  }
  else
  { while(temp!=0)
    { printf("%d", temp->data);
      temp = temp->link;
    }
}
```

```
void peek()
{ if (top==0)
  { pf("kuch नहीं है");
  }
  else
  { pf("top element is %d", top->data);
  }
}
```

IATE-03

Q. Let S be a stack of size $n \geq 1$. Starting with the empty stack, suppose we push the first n natural numbers in sequence and then perform n pop operations. Assume that Push & Pop operations take X seconds each and Y seconds elapsed between the end of one such stack operation and the start of the next operation. For $m \geq 1$, define the stack life of m as the time elapsed from the end of Push(m) to the start of the pop operation that removes m from S. The average stack-life of an element of this stack is

A. $n(X+Y)$ B. $3Y+2X$ C. $n(X+Y)-X$ D. $Y+2X$



GATE-05
Q. A function f defined on stacks of integers satisfies the following properties. $f(\phi) = 0$ & $f(push(s, i)) = \max(f(s), 0) + i$ for all stacks s & integers i.

If a stack S contains the integers 2, -3, 2, -1, 2 in order from bottom to top, what is f(S)?

$f(\phi) = 0 \longrightarrow$ stack is empty

i → element to be pushed

(1) $f_{new}(S) = \max(f_{prev}(S), 0) + 2$
$= \max(0, 0) + 2$
$= 2$

(2) $f_{new}(S) = \max(f_{prev}(S), 0) + (-3)$
$= \max(2, 0) + -3$
$= -1$

(3) $f_{new}(S) = \max(f_{prev}(S), 0) + (2)$
$= \max(-1, 0) + 2$
$= 2$

(4) $f_{new}(S) = \max(f_{prev}(S), 0) + (-1)$
$= \max(2, 0) + (-1)$
$= (2) - 1 = 1$

(5) $f_{new}(S) = \max(f_{prev}(S), 0) + (2)$
$= \max(1, 0) + 2$
$= (3)$

Q. Consider n elements that are equally distributed in k stacks. Each stack, elements of it are arranged in ascending order (min is at the top in each of the stack & then increasing downwards). Given a queue of size n in which we have to put all n elements in increasing order. What will be the time complexity of the best known algorithm?

A. $O(n \log k)$
B. $O(nk)$
C. $O(n^2)$
D. $O(k^2)$

→ $O(n \log k)$ by creating a min heap of size k & adding all the top-elements of all the stacks.

→ $O(k)$ → For creating heap of size k
→ $(n-k) \log k$ → Insertions into the heap

$O(n \log k)$

SRO-17
Q. The minimum no. of stacks needed to implement a queue is 2

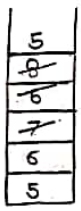
GATE-02
Q. To evaluate an expression without any embedded function calls:

A. one stack is enough
B. Two stack are needed
C. As many stacks as the height of the expression tree are needed.
D. A turning machine is needed in the general case.

Q. Suppose a stack is to be implemented with a linked list instead of an array. What would be the effect on the time complexity of the push & pop operations of the stack implemented using linked list?

A. $O(1)$ for insertion & $O(n)$ for deletion
B. $O(1)$ for insertion & $O(1)$ for deletion
C. $O(n)$ for insertion & $O(1)$ for deletion
D. $O(n)$ for insertion & $O(n)$ for deletion

Q. Consider an efficient implementation of a data structure STACK that support an operation "max()" that reports the current maximum among all elements in the stack. Normal stack operations ie, push, pop are also to be supported. The size of above data structure after performing operation push(5), push(6), push(7), pop, max, push(6), push(8), pop, pop, max, push(5) is ____ bytes. Assume that an integer can be stored in 4 bytes.

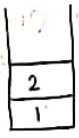


6 → 6 elements are left

6 × 4 = 24 bytes

GATE-03
Q. Let S be a stack of size $n \geq 1$. Starting with the empty stack, suppose we push the first n natural numbers in sequence and then perform n pop operations. Assume that Push & Pop operations take X seconds each and Y seconds elapsed between the end of one such stack operation and the start of the next operation. For $m \geq 1$, define the stack life of m as the time elapsed from the end of Push(m) to the start of the pop operation that removes m from S. The average stack-life of an element of this stack is

A. $n(X+Y)$ B. $3Y+2X$ C. $n(X+Y)-X$ D. $Y+2X$



$X + Y + X$ (Stack life time)

$X + Y + X + Y + X + Y + X$ (Stack life time)

GATE-05
Q. A function f defined on stacks of integers satisfies the following properties. $f(\phi) = 0$ & $f(\text{push}(S, i)) = \max(f(S), 0) + i$ for all stacks S & integers i.

If a stack S contains the integers 2, -3, 2, -1, 2 in order from bottom to top, what is f(S)?

$f(\phi) = 0 \rightarrow$ stack is empty

i → element to be pushed

(1) $f_{new}(S) = \max(f_{prev}(S), 0) + 2$
$= \max(0, 0) + 2$
$= 2$

(2) $f_{new}(S) = \max(f_{prev}(S), 0) + (-3)$
$= \max(2, 0) + -3$
$= -1$

(3) $f_{new}(S) = \max(f_{prev}(S), 0) + (2)$
$= \max(-1, 0) + 2$
$= 2$

(4) $f_{new}(S) = \max(f_{prev}(S), 0) + (-1)$
$= \max(2, 0) + (-1)$
$= 2 - 1 = 1$

(5) $f_{new}(S) = \max(f_{prev}(S), 0) + (2)$
$= \max(1, 0) + 2$
$= 3$

Q. Consider n elements that are equally distributed in k stack. unacademy
stack, elements of it are arranged in ascending order (min is at the top in each of the stack & then increasing downwards). Given a queue of size n in which we have to put all n elements in increasing order. What will be the time complexity of the best known algorithm?

A. O(nlogk)
B. O(nk)
C. O(n²)
D. O(k²)

→ O(nlogk) by creating a min heap of size k & adding all the top-elements of all the stacks.

→ O(k) → For creating heap of size k

→ (n-k) log k → Insertions into the heap

O(nlogk)

SRO-17
Q. The minimum no. of stacks needed to implement a queue is ___2___

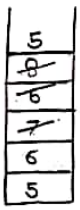
GATE-02
Q. To evaluate an expression without any embedded function calls: unacademy

A. One stack is enough
B. Two stack are needed
C. As many stacks as the height of the expression tree are needed.
D. A turing machine is needed in the general case.

Q. Suppose a stack is to be implemented with a linked list instead of an array what would be the effect on the time complexity of the push & pop operations of the stack implemented using linked list?

A. O(1) for insertion & O(n) for deletion
B. O(1) for insertion & O(1) for deletion
C. O(n) for insertion & O(1) for deletion
D. O(n) for insertion & O(n) for deletion

Q. Consider an efficient implementation of a data structure STACK unacademy
that support an operation "max()" that reports the current maximum among all elements in the stack. Normal stack operations ie, push, pop are also to be supported. The size of above data structure after performing operation push(5), push(6), push(7), pop, max, push(6), push(8), pop, pop, max, push(5) is ___ bytes. Assume that an integer can be stored in 4 bytes.



→ 6 elements are left

6 → 6 × 4 = 24 bytes