Design & Analysis of Algorithm

Topics:

1. Asymptotic Notation.
2. Time and Space Complexity.
3. Design Strategi:
   a). Divide & Conquer.
      (i) Binary Search
      (ii) All sorting Techniques.
      (iii) Heap tree.
   b). Greedy Method.
      (i) Job sequencing.
      (ii) Knap sack.
      (iii) Optimal merge Pattern.
      (iv) Huffman Coding.
      (v) Disjkstra Single source shortest path.
      (vi) MST
         Prim's
         Kruskal's Algorithm.
      (vii) DFS
      (viii) Connected component.
   c). Dymamic Programming.
      (i) All pairs shortest path
      (ii) Multistage graph.
      (iii) Optimal Binary search tree
      (iv) Travelling Sales Person.
      (v) 0/1 knap Sack problem.

* Algorithm :

- Finite set of steps to solve a problem is called algorithm.
- steps means instructions which contains fundamental operators i.e. (+, *, ÷, %, =, etc)

⇒ Characteristic of fundamental instructions :

1). Definiteness:

Every fundamental ins^{n}. must be define without any ambiguity.

e.g. `i = i + 1 ;` → Invalid ins^{n}.

↳ Invalid symbol.

`i = i + 1 ;` → valid ins^{n}.

2). Finiteness:

Every turn^{n}. ins^{n}. must be terminate with in finite amount of time.

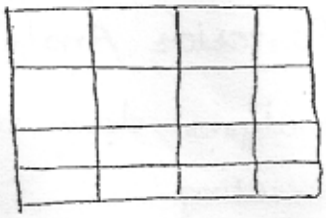
e.g.
```
i = 1
while (1)
{
    i = i+1
}
```
→ since it is executing infinitely. So, this kind of ins^{n}. not allowed in algo.

3). Every fundamental instruction must accept atleast '0' input & provides atmost '1' output.

* Steps for solving any problem :

1). Identifying Problem statement :

e.g. Arrange 4 Queens Q_1, Q_2, Q_3, Q_4 into 4×4 chess board.



2). Identifying Constraints:

e.g. No two queens on same row & on same column & on same diagonal.

3). Design logic:

Depends on the characteristics of the problem we can choose any one of the following design strategy for design logic.

(i) Divide & Conquer

(ii) Greedy Method.

(iii) Dynamic Programming.

(iv) Branch & Bound.

(v) Back tracking. etc...

4). Validation:

Most of algorithm validated by using mathematical indexian.

5). Analysis:

Process of comparing two algo. w.r.t time, space, no. of register, network bandwidth etc. is called analysis.

Priory Analysis

→ Analysis done before executing.

e.g. $x = x + 1$

Principle: frequency count of fundamental ins^{n}.

Since $x = x + 1$ being carried out only 1 time so it's complexity is $O(1)$ [order of 1]

→ It provides estimated values.

→ It provides uniform values.

→ It is independent of CPU, o/s & system architecture.

5). Implementation.

6). Testing & Debugg.

Posterior Analysis.

→ Analysis done after executing.

e.g. $x = x + 1$;

→ It provides exact values.

→ It provides non uniform values.

→ It is dependent.

* Asymptotic Notation:

To compare two Algorithms rate of growth with respect to time & space we need asymptotic notation.

Big - Oh (O) :-

Def^{n}: $f(n)$ is $O(g(n))$ iff $\exists$ some $c > 0$ and $k \geq 0$ such that $f(n) \leq c \cdot g(n)$; $\forall n \geq k$

Notes:

- $f(n)$
- that
- *** Lou

Ex: If

→ $f(r$

(I)

(II)

Note:

Eventhough $n^2 > n^3 > n^4$ are upperbounds to the $t(n) = n^2 + n + 1$; we have to take least upperbound only.

$\therefore t(n) = O(n^2)$

Shortcut: If $t(n) = a_0 + a_1 n + a_2 n^2 + \dots + a_m n^m \ (a_m \neq 0)$

then $t(n) = O(n^m)$

Ex: $t(n) = n^2 \log n$, $g(n) = n(\log n)^{10}$,

which of the following is true?

a) $t(n) = O(g(n))$, $g(n) \neq O(t(n))$

b) $t(n) \neq O(g(n))$, $g(n) = O(t(n))$

c) $t(n) = O(g(n))$, $g(n) = O(t(n))$

d) $t(n) \neq O(g(n))$, $g(n) \neq O(t(n))$

→ **Imp. Point:**

1. Low = O(high)
2. $t(n) = a_0 + \dots + a_m n^m \ (a_m \neq 0)$

   then $t(n) = O(n^m)$

→ $t(n) = n^2 \log n$ $\quad g(n) = n(\log n)^{10}$

$= n \log \times n$ $\quad = n \log n \ (\log n)^9$

Since $n > (\log n)^n$

$t(n) > g(n)$

$\Rightarrow g(n) < t(n)$

Low = O(high) $\Rightarrow g(n) = O(t(n))$

* Dor

Ex: t_1

t_2

in

(a)

(b)

→ t_2

t_3

Ex: If

→ $t(n$

Ex: If

→ $t(n$

	$f(n) = n^2$	$g(n) = 2^n$	
1.	1	2	$\Rightarrow f(n) < g(n)$
2.	4	4	$\Rightarrow f(n) = g(n)$
3.	9	8	$\Rightarrow f(n) > g(n)$
$n_0 =$ 4.	16	16	$\Rightarrow f(n) \leq g(n)$
5.	25	32	
6.	36	64	
7.	49	128	

$f(n) = O(g(n)) \Leftrightarrow f(n) \leq c \cdot g(n)$; $\forall n \geq k$.

Notes:

- $f(n) = O(g(n))$ means, $g(n)$ having higher growth rate than $f(n)$, for large values of 'n'.

*** Lower growth rate fun^n = O (Higher growth rate fun^n)
Only for Large 'n'.

Ex: If $f(n) = n^2 + n + 1$, then $f(n) = O(\quad)$.

$\rightarrow f(n) = O(g(n)) \Leftrightarrow f(n) \leq c \cdot g(n)$; $\forall n \geq k$

(I) $n^2 \leq n^2$

$n^2 + n \leq n^2 + n^2$

$n^2 + n + 1 \leq n^2 + n^2 + n^2$

$n^2 + n + 1 \leq 3 \cdot n^2$; $\forall n \geq 1$

$c = 3$, $k = 1$

$n^2 + n + 1 = O(n^2)$; $\forall n \geq 1$

(II) $n^2 \leq n^3$

$n^2 + n \leq n^3 + n^3$

$n^2 + n + 1 \leq n^3 + n^3 + n^3$

$n^2 + n + 1 \leq 3 \cdot n^3$; $\forall n \geq 1$

$n^2 + n + 1 = O(n^3)$

o the
sbound

$(a_m \neq 0)$

* Dominance Ranking:

$$2^{2^n} \geq n! \geq 4^n \geq 2^n \geq n^3 \geq n^2 \geq n \log n \geq n \geq \log k_n \text{ (where } k \geq 1) \geq \log \log n \geq \frac{\log n}{\log \log n} \geq 1$$

Ex: $t_1(n) = 2^n$, $t_2(n) = n^{3/2}$, $t_3(n) = n \log_2 n$, $t_4(n) = n^{\log_2 n}$. Arrange t_1, t_2, t_3, t_4 in the increasing order.

(a) $t_2\ t_3\ t_4\ t_1$ (c) $t_2\ t_1\ t_3\ t_4$

(b) $t_1\ t_2\ t_3\ t_4$ (d) $t_3\ t_2\ t_4\ t_1$

$\rightarrow t_2(256) = (256)^{3/2} = (2^8)^{3/2} = 2^{12}$ ...ii

$t_3(256) = 256 \times \log_2 256 = 2^8 \times 2^3 = 2^{11}$

$t_3 < t_2 < t_4 < t_1$ .....

Ex: If $t(n) = n!$ then $t(n) = O(\ )$

$\rightarrow t(n) = n!$

$= n(n-1)(n-2) \ldots 1$

$= n^n \{1\ (1-1/n)(1-2/n) \ldots 1/n\}$

$= O(n^n)$ $[\because t(n) = a_0 + \ldots + a_m n^m\ (a_m \neq 0)]$

Ex: If $t(n) = \log(n!)$ then $t(n) = O(\ )$

$\rightarrow t(n) = \log(n!)$

$= \log(O(n^n))$

$= O(\log(n^n))$

$= O(n \log n)$ [$\because$ log is constant fun^n which satisfies associative]

Ex: If $f(n) = n^5$ for $n \leq 100$ x - lower value
$= n^2$ for $n > 100$

and $g(n) = n$ for $n < 1000$ x - lower value
$= n^3$ for $n \geq 1000$

Which of the following is true!

a). $f(n) = O(g(n))$

b). $g(n) = O(f(n))$

Ex: If $f(n) = O(h(n))$, $g(n) = O(k(n))$

then $f(n) + g(n) =$

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

a). Max $(h(n), k(n))$. and $O(h(n) \cdot k(n))$ respectively.

b). Min $(h(n), k(n))$. and $O(h(n) + k(n))$ respectively.

→ $f(n) = n^2 + 1$, $h(n) = n^2$ $\quad g(n) = n + 1$, $k(n) = n$

$\therefore f(n) = O(h(n))$ $\quad g(n) = O(k(n))$

$f(n) + g(n) = n^2 + 1 + n + 1$

$= n^2 + n + 1$

$= O(n^2)$

$= O(\text{Max}(h(n), k(n))$

$f(n) \cdot g(n) = (n^2 + 1) \cdot (n + 1)$

$= n^3 + n^2 + n + 1$

$= O(n^3)$

$= O(h(n) \cdot k(n))$

Big - Omega (Ω):

$f(n)$ is $\Omega(g(n))$ iff $\exists$ some $c > 0$ and $k \geq 0$ such that $f(n) \geq c \cdot g(n)$; $\forall n \geq k$.

Ex: 1

→ n

n^2

n^3

n!

→

Note:

Ever

have

Shorte

It

then

The

f(n)

Ex: If

→ n^2

n^2

Ex: If $f(n) = n^2+n+1$, then $f(n) = \Omega(\ )$

$\rightarrow n^2 \geq n^2$

$n^2+n \geq n^2$

$n^2+n+1 \geq 1\ n^2 \ ; \forall n \geq 0$

$n^2+n+1 = \Omega(n^2)$ ✓

$\rightarrow n^2 \geq n$

$n^2+n \geq n$

$n^2+n+1 \geq 1 \cdot n \ ; \forall n \geq 0$

$n^2+n+1 = \Omega(n)$

Always take higher value from the lower frequency.

Note:

Even though n^2, n are lower bouns to $f(n)$ you have to take greatest lower bound only.

Shortcut:

If $f(n) = a_0 + a_1 n + a_2 n^2 + \ldots + a_m n^m \ (a_m \neq 0)$

then $f(n) = \Omega(n^m)$.

Theta (θ):

$f(n)$ is $\theta(g(n))$ iff $f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$.

$f(n) = \theta(g(n)) \Leftrightarrow \exists\ c_1, c_2 > 0$ and $k_1, k_2 \geq 0$ and $k_1 > 0$ such that

$c_1 g(n) \leq f(n) \leq c_2 \cdot g(n) \ ; \forall n \geq k_1$

Ex: If $f(n) = n^2+n+1$ then $f(n) = \theta(\ )$

$\rightarrow n^2+n+1 = O(n^2) \ ; \forall n \geq 1$ and for $c_2 = 3$

&

$n^2+n+1 = \Omega(n^2) \ ; \forall n \geq 0$ and for $c_1 = 1$

$1 \cdot n^2 \leq n^2 + n + 1 \leq 3 \cdot n^2 \; ; \; \forall n \geq 1$

$c_1 \qquad\qquad c_2 \qquad\qquad k_1$

$n^2 + n + 1 = \Theta(n^2)$

Shortcut:

$f(n) = a_0 + \ldots + a_m n^m \; (a_m \neq 0)$

then $f(n) = \Theta(n^m)$

Little-oh (o):

$f(n)$ is $o(g(n))$ iff $f(n) < c \cdot g(n)$, $\forall n \geq k$, $\forall c > 0$

O	o
→ $\leq$	$<$
→ ∃ some 'c'	→ ∀ c
→ e.g. $n^2 + n + 1 = O(n^2)$ only for $c = 3$	

Shortcut:

If $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$ then $f(n) = o(g(n))$

Little-omega (w):

$f(n)$ is $w(g(n))$ iff $f(n) > c \cdot g(n)$, $\forall n \geq k$, $\forall c > 0$

Ω	w
→ $\geq$	→ $>$
→ ∃ some 'c'	→ ∀ c

e.g. $n^2 + n + 1 = \Omega(n^2)$ only for $c = 1$

Shortcut

If

2007

Ex :

a).

b).

c).

d).

→ If

will

prin

orde

* Pro

Propert

Notation

O

Ω

Θ

o

w

Shortcut:

If $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$ then $f(n) = \omega(g(n))$

2007

Ex:

```
int Isprime (int n)
{
    int i, n;
    for (i = 2 ; i <= √n ; i++)
    {
        if (n % i == 0)
        {
            printf ("Not prime");
            return 0
        }
    }
    return 1;
}
```

a). $T(n) = \Theta(\sqrt{n})$, $T(n) = \Omega(\sqrt{n})$

b). $T(n) = O(\sqrt{n})$, $T(n) = \Omega(1)$

c). $T(n) = O(n)$, $T(n) = \Omega(\sqrt{n})$

d). none of the above.

→ If the i/p is divisible by 2 then for loop will be repeated only one time if i/p is prime no. the for loop will repeated in order of $\sqrt{n}$ time.

* Properties of Asymptotic Notation:

Property / Notation		Reflexivity	Symmetric	Transitive	Transpose
O	($\leq$)	✓	×	✓	$O \Rightarrow \Omega$
Ω	($\geq$)	✓	×	✓	$\Omega \Rightarrow O$
Θ	($=$)	✓	✓	✓	$\Theta \Rightarrow \Theta$
o	($<$)	×	×	✓	$o \Rightarrow \omega$
ω	($>$)	×	×	✓	$\omega \Rightarrow o$

Reflexive:

$a = a$

↳ (Ref)

$f(n) \leq 1 \cdot f(n)$, $\forall n \geq 0$

$f(n) = O(f(n))$

↳ (Ref)

$f(n) = O(g(n)) \Leftrightarrow f(n) \leq c \cdot g(n)$, $\forall n \geq k$.

Symmetric:

If $a = b \Rightarrow b = a$

↳ Symmetric

✗ If $f(n) = O(g(n)) \not\Rightarrow g(n) = O(f(n))$

Trasitive:

If $a = b$, $b = c \Rightarrow a = c$

↳ Trausitive

If $f(n) = O(g(n))$, $g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$

↳ Trausitive.

Transpose:

If $f(n) = O(g(n)) \Rightarrow g(n) = \Omega(f(n))$ ....

2009

Ex: Consider the following statements.

(i) $(n+k)^m = \theta(n^m)$, where k, m are constants.

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

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

Which of the following is true.

✓(a) (i) & (ii) (b) (ii) & (iii)

(c) (i) & (iii) (d) (i), (ii) & (iii)

→ (ii) $2^{n+1} = O(2^{n+1})$

↳ Reflexive

$= O(2^n \cdot 2)$

$= O(2^n)$

→ (iii)

2008

Ex:

→ Hei

Note:

Ref

* Tim

Meth

(i) S

Ex:

→ (iii) $2^{2n+1} = O(2^{2n+1})$

└→ Reflexive.

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

$= O(2^{2n}) = O(2^{3n}) = O(2^{4n}) = \dots$

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

2008

Ex: $f(n) = n!$, $g(n) = 2^{n}$, $h(n) = n^{\log_2 n}$

Which of the following is true!

(a) $f(n) = O(g(n))$, $g(n) = O(h(n))$

✓ (b) $g(n) = O(f(n))$, $h(n) = O(f(n))$

(c) $f(n) = \Omega(g(n))$, $g(n) = O(h(n))$

(d) $h(n) = O(f(n))$; $g(n) = \Omega(f(n))$

→ Here, $h(n) \leq g(n) \leq f(n)$

Note: If $(f(n)) = 2^{n}$, $g(n) = n!$, $h(n) = n^{\log_2 n}$

then option (d) is true.

Refer IT - 2008 & CSE - 2000

* Time Complexity:

Method :

- Simple for loop
- Nested for loop
- if else.
- Recursive algorithm.

(i) Simple for loop:

Ex: Find time complexity of the following.

```
Sum = 0;
for (i = 1;  i <= n ; i++)
```

(annotations above the loop header: $i = 1$ → 1, $i \leq n$ → $n+1$, $i{+}{+}$ → n)

```
{
    Sum = Sum + i;        → n
}
```

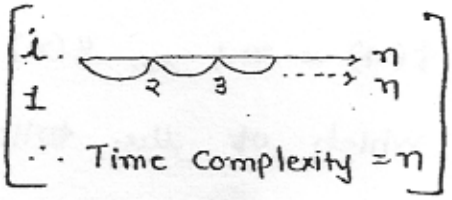
Ans:

Principle: Frequency count of fundamental instruction

$= 1 + 1 + (n+1) + n + n$

$= 3n^1 + 3$

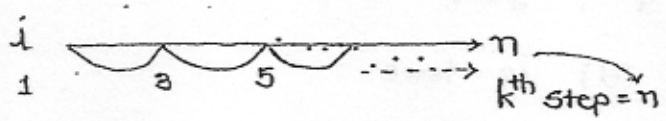
$\doteq O(n^1)$



Ex: Find Complexity:

```
Sum = 0;
for (i = 1; i <= n; i = i+2)
{
    Sum = Sum + i;
}
```

Ans:



$t_k = a + (k-1)d$ — k = # of step.

$n = 1 + (k-1)2$

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

∴ Total no. of comparision = $\frac{n+1}{2}$

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

$\therefore O(n+1)$

$= O(n)$ → Total time complexity.

Ex: Find total comparision & time complexity!

```
Sum = 0;
for (i=1; i <= n; i = i*2)
{
    Sum = Sum + i;
}
```

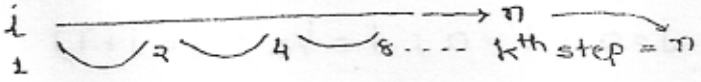
Ans

Ex:

Ans:

t_k

Ans:



$\therefore$ 1 2 4 8 ---- $k^{th} = n$

$a = 1$, $r = \frac{t_2}{t_1} = 2$

$$t_k = a \cdot r^{k-1}$$

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

$\Rightarrow k = \log_2(2n)$

$\Rightarrow \log_2 2 + \log_2 n$

$\Rightarrow 1 + \log_2 n$ ← total comparision.

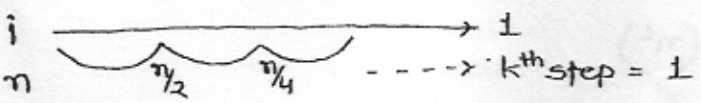
$\Rightarrow O(1 + \log_2 n)$

$= \Theta(\log_2 n)$ ← Time complexity.

Ex: Find total comparision & time Complexity!

```
Sum = 0 ....
for(i=n; i>0; i=i/2)
.....{
    Sum = Sum + i;
}
```

Ans:



$a = n$, $r = 1/2$

$t_k = a \cdot r^{k-1}$

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

$1 = n\left(\frac{1}{2^{k-1}}\right)$

$2^{k-1} = n$

$2^k = 2n$

$k = \log_2(2n)$

$\therefore k = 1 + \log_2 n$ → total comparision

$O(1 + \log_2 n) = \Theta(O(\log_2 n))$ → Time complexity.

Ex: for (i = n ; j = 0 ; i > 0 ; i = i/2 ; j = j + i)

Let val (j) denote the value stored in variable 'j' after termination of loop.

(a) Val (j) = $O(\log_2 n)$

(b) Val (j) = $O(n)$

(c) Val (j) = $O(\sqrt{n})$

(d) Val (j) = $O(\log n)$

Note: Time complexity for the above problem is depend on variable i so it is $O(\log_2 n)$.

Since

→ i ⟶ 1

n n/2 n/4 - - - - - - →

i → n n/2 n/4 - - - - - - 1

j → 0 0+n/2 0+n/2+n/4 - - - - - 0+n/2+n/4+...+1

Val (j) = $n/2 + n/4 + \ldots + 1$

= $O(n^1)$

(ii) Nested for loop:

Ex: Find time complexity:

```
Sum = 0;
for ( i = 1 ; i ≤ n ; i++ )
{
   for (j = 1 ; j ≤ n ; j++)
   {
      Sum = sum + j;
   }
}
```

Ans:

Ex: F

Ans: Q

Ex: Find

Ans: (value) $i \rightarrow 1 \quad 2 \quad 3 \quad \ldots \quad n$

(Times) $j \rightarrow n + n + n + \ldots + n$

$= n * n$

$= O(n^2)$ ← Time Comp.

Ex: Find time complexity.

```
Sum = 0;
for (i = 1; i <= n; i++)
{
    for (j = 1; j <= n; j = j + 2)
    {
        sum = sum + j;
    }
}
```

Ans: (value) $i \rightarrow 1 \quad 2 \quad \ldots \quad n$

(Time) $j \rightarrow \left(\frac{n+1}{2}\right) + \left(\frac{n+1}{2}\right) + \ldots + \left(\frac{n+1}{2}\right)$

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

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

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

$= O(n^2+n)$

$= O(n^2)$

Ex: Find time complexity:

```
Sum = 0;
for (i = 1; i <= n; i++)
{
    for (j = 1; j <= n; j = j * 3)
    {
        sum = sum + j;
    }
}
```

Ans: (value) $i \rightarrow 1 \quad 2 \quad \ldots\ldots \quad n$

(Time for execution) $j \rightarrow (1+\log_3 n) + (1+\log_3 n) + \ldots + (1+\log_3 n)$

$= n(1+\log_3 n)$

$= n + n\log_3 n$

$= O(n \log_3 n)$

Ex: Find time complexity:

```
Sum = 0;
for (i = 1; i <= n; i = i * 2)
{
   for (j = 1; j <= n; j = j + 2)
   {
      Sum = sum + j;
   }
}
```

Ans:

(Value) $i \rightarrow 1 \quad 2 \quad \ldots\ldots \quad 1+\log_2 n$

(Time) $j \rightarrow \left(\frac{n+1}{2}\right) + \left(\frac{n+1}{2}\right) + \ldots\ldots + \left(\frac{n+1}{2}\right)$

$= \left(1+\log_2 n\right)\left(\frac{n+1}{2}\right)$

$= \left(\frac{n+1}{2}\right) + \left(\frac{n+1}{2}\right)\log_2 n$

$= \frac{1}{2}\left(n+1+n\log_2 n + \log_2 n\right)$

$= O(n \log_2 n)$

Ex: F

Ans: (value) (Time)

Ex: Fin

Ex: Find time complexity.

```
Sum = 0;
for (i = 1 ; i ≤ n ; i = i+1)
{
   for (j = 1 ; j ≤ i ; j = j*2)
   {
      sum = sum + j;
   }
}
```

Ans:

(value) $i \rightarrow 1 \quad 2 \quad \dots\dots\dots \quad n$

(Time) $j \rightarrow (1+\log_2 1) + (1+\log_2 2) + \dots\dots + (1+\log_2 n)$

$= n + (\log_2 1 + \log_2 2 + \dots + \log_2 n)$

$= n + \log_2 (1 \cdot 2 \cdot 3 \cdot \dots\dots n)$

$= n + \log_2 (n!)$

Sol. ↓

$= O(n) + O(n \log n)$

$= O(n \log n)$

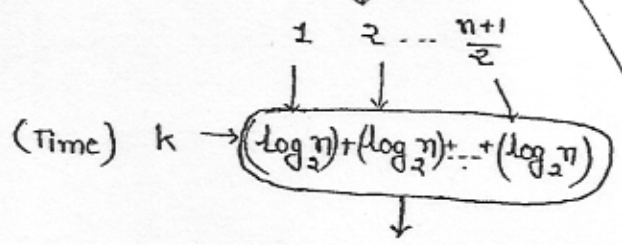
Ex: Find time complexity:

```
Sum = 0;
for(i = 1; i ≤ n; i++)
{
   for (j = 1 ; j ≤ n ; j = j+2)
   {
      for (k = 1 ; k ≤ n ; k = k*2)
      {
         sum = sum + k;
      }
   }
}
```

Ans:

(Value) $i \longrightarrow 1 \quad 2 \quad \ldots\ldots\ldots\ldots n$

(Time) $j \longrightarrow \left(\frac{n+1}{2}\right) + \left(\frac{n+1}{2}\right) + \ldots\ldots + \left(\frac{n+1}{2}\right)$



$$k \rightarrow \left[\frac{n+1}{2}\left(1+\log_2 n\right)\right] + \left[\frac{n+1}{2}\left(1+\log_2 n\right)\right] + \ldots\ldots + \left[\left(\frac{n+1}{2}\right)\left(1+\log_2 n\right)\right]$$

$$= n * \left(\frac{n+1}{2}\right)\left(1+\log_2 n\right)$$

$$= \left(\frac{n^2+n}{2}\right)\left(1+\log_2 n\right)$$

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

$$\therefore = O\left(n^2 \log_2 n\right)$$

Ex: Find time complexity:

```
Sum = 0
for (i = 0; i ≤ n; i++)
{
   for (j = 1; j < i; j++)
   {
      if (j % i == 0)
      {
         for (k = 1; k ≤ n; k++)
         {
            Sum = sum + k;
         }
      }
   }
```

Ans:
(Value)
(Time)
(Time
(Time

Note

Ex: In the

Ans:
(Value
(Time
(Time
(Time)

* Rec

int
{

Ans:

(value) $i \rightarrow 1 \quad 2 \quad 3 \quad 4 \quad \ldots \quad n$

(Time) $j \rightarrow 0 \quad 1 \quad 2 \quad 3 \quad \ldots \quad (n-1)$

(Time) if $\rightarrow 0 + 1 + 2 + 3 + \ldots + (n-1)$

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

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

$$= O\left(\frac{n^2-n}{2}\right)$$

$$= O(n^2)$$

$\left(\frac{n+1}{2}\right)\left(1+\log_2 n\right)$

Note: Inner most loop never been executed.

Ex: In above example change j loop for(j=1; j<=i; j++) then find time complexity.

Ans:

(Value) $i \rightarrow 1 \quad 2 \quad 3 \quad \ldots \quad n$

(Time) $j \rightarrow 1 \quad (1+2) \quad (1+2+3) \quad \ldots \quad n$

(Time) if $\rightarrow 1 \quad 2 \quad 3 \quad \ldots \quad n$

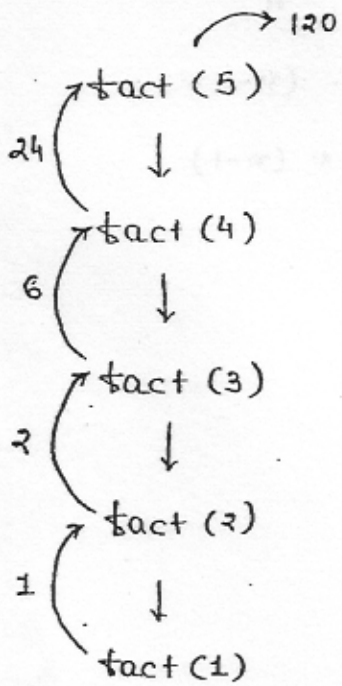
(Time) $k \rightarrow n + n + n + \ldots + n$

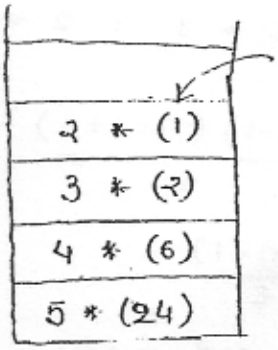
$$\Rightarrow O(n^2)$$

* Recursive Algorithm:

```
int fact (int n)
{
    if (n==0 || n==1)   // Base condition
    return 1;
    else
    return n * fact(n-1);
}
```

Here if we i/p $n = 5$ then,





Notes:

1). Time complexity of recursive algorithm =
No. of function call

∴ Time complexity of fact(n) = O(n)

2). Space complexity of recursive algorithm
= Depth of recursive tree
or
= No. of activation record.

3). Space complexity of fact(n) = O(n-1)
= O(n)

Ex: Find time complexity of recursive fun^n. of fibonacci sequence.

```
int fib (int n)
{
B.C. { if (n == 0)
       return 0;
       if (n == 1)
```

Ans:

Note: For

to

Si

of

Note:

Note:

Note:

1

fib(2)

+

fib(1)

```
    return 1
  else
I.C.{ return fib(n-1) + fib(n-2);
  }
```

(a) $O(n^2)$ (b) $O(2^n)$ (c) $O(n)$ (d) $O(n \log n)$

Ans: Here we take $n = 5$.

Note: For small values of n $fib(n) = O(n^2)$ & for large values of n $fib(n) = (2^n)$

Since our analysis is only for large values of n. So time complexity of $\boxed{fib(n) = O(2^n)}$

Note: No. of fun^n. calls on i/p size n in fibonacci sequence $= \underline{2 \cdot fib(n+1) - 1}$

e.g: $n = 5$, fun^n. call $= 15$

$= 16 - 1$

$= 2 \times 8 - 1$

$= 2 f(6) - 1$

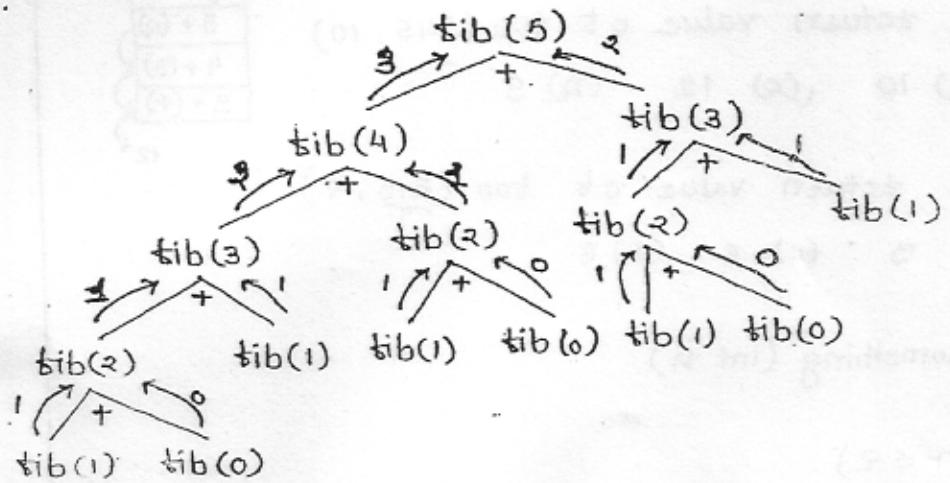
Note: No. of addition perform on input size n in

$fib(n) = fib(n+1) - 1$

e.g. $n = 5$, addition $= 7$

$= 8 - 1$

$= fib(6) - 1$



n	0	1	2	3	4	5	6	7
fib(n)	0	1	1	2	3	5	8	13

Function call = 15 (Total no of nodes)

Total Addition = 7

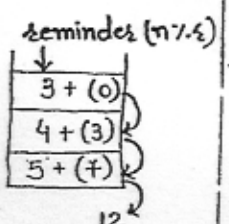
2011

Ex: Linked Question.

```
int foo (int n, int k)
{
   if (n>0)
   return ((n%k) + foo (n/k, k));
   else
   return 0;
}
```

1. What is the return value of foo (345, 10)

(a) 345 (b) 10 (c) 12 (d) 9



2. What is the return value of foo (513, 2)

(a) 2 (b) 5 (c) 6 (d) 8

2007

Ex:

```
int Dosomething (int n)
{
   if (n <= 2)
```

1. Fi
(a)
(c)
2. What

For Ans

```
    return 1;
    else
    return (DoSomething (floor(Sqrt(n))) + n);
}
```

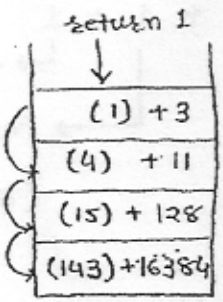
1. Find time complexity :

(a) $O(\log_2 n)$ (b) $O(\log_2 \log_2 n)$

(c) $O(n \log_2 n)$ (d) $O(n)$

2. What is the return value of DoSomething (16384) ?

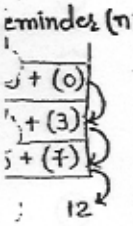
DS (16384)
↓ $\sqrt{16384}$
DS (128)
↓ $\sqrt{128}$
DS (11)
↓ $\sqrt{11}$
DS (3)
↓ $\sqrt{3}$
DS (1)



∴ Ans:2: 16527

For Ans : 1.

n	fun^n. call	Option
16384	5	X(D) $O(16384)$
		X(C) $O(16384 \times \log_2 16384)$
		X(B) $O(\log_2 \log_2 16384)$
		$= O(\log_2 \log_2 2^{14})$
		$= O(\log_2 14)$
		$= O(\log_2 2^4) = O(4)$
		✓(A) $O(\log_2 16384)$
		$= O(\log_2 2^{14}) = O(14)$



How to solve R. Algo.

1. Construct Recursion Tree.
   - T. Complexity = No. of fun^n. calls.
   - S. Complexity = Depth of R. Tree

1. Construct Recurrence Equation. Solve it by using.
   (a) Substitution Method
   (or)
   (b) Masters Theorem.

Construction of Recurrence eq^n.:

1. In the above problem,

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

   - 1 → For comparision ($n \leq 2$)
   - 1 → For store () + n into the stack

   (These two are contant value.

2. In the above problem. it,

   return (DoSomething ((n - 1) + n)) then,

$$\Rightarrow T(n) = T(n-1) + 1 + 1$$

$$\therefore T(n) = T(n-1) + c$$

3. In the above problem, it

   return (DoSomething ((n/2) + n * n)) then,

$$\Rightarrow T(n) = T(n/2) + 1 + 1$$

$$\therefore T(n) = T(n/2) + c$$

* Substitution Method:

$$T(n) = T(\sqrt{n}) + c \quad \text{----- } 1^{st}$$

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

$$T(n) = T(n^{1/2^2}) + 2c \quad \text{----- } 2^{nd}$$

$$\vdots \qquad \vdots$$

$$T(n) = T(n^{1/2^k}) + kc \quad \text{----- } k^{th} \qquad \text{---- (*)}$$

Put ⇒ Subs- ⇒ 7

Ex: Fin

(a)

Ans:

Here

Put $n^{1/2^k} = 2$

$\Rightarrow \frac{1}{2^k} = \log_n 2$

$\Rightarrow 2^k = \log_2 n$

$\Rightarrow k = \log_2 \log_2 n$

Substitute k in (*).

$\Rightarrow T(n) = T(2) + (\log_2 \log_2 n)\, c$

$= O(1 + c(\log_2 \log_2 n)\, c$

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

Ex: Find recurence eqn. using S.M.

```
int recursive (int n)
{
   if (n < 2)
   return 1;
   else
   return recursive (n-1) + recursive (n-1);
}
```

(a) $O(n^2)$ (b) $O(2^n)$ (c) $O(n \log n)$ (d) $O(n)$

Ans: $T(n) = T(n-1) + T(n-1) + 1 + 1$

$T(n) = 2T(n-1) + c$ ----- 1st

$= 2\{2T(n-2) + c\} + c$ ----- 2nd

$T(n) = 2^2 T(n-2) + 3c$ ----- 2nd

$= 2^3 T(n-3) + 7c$ ---- 3rd

$T(n) = 2^3 T(n-3) + (2^3 - 1)c$ ---- 3rd

$T(n) = 2^k T(n-k) + (2^k - 1)c$ ---- k^{th} ——— (*)

Here, $n - k = 1$ [b'coz $(n-1) < 2$]

$\Rightarrow k = n - 1$

Substitute k in (*)

$\therefore T(n) = 2^{n-1} T(1) + (2^{n-1} - 1) c$

$= 2^{n-1} + (2^{n-1} - 1) c$

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

$= O(2^n / 2)$

$= O(2^n)$

Shortcut: $T(n) = aT(n-b) + O(n^k)$

$a > 0$, $b \geq 1$, $k \geq 0$

(i) if $a = 1$, $T(n) = O(n^{k+1})$

(ii) if $a > 1$, $T(n) = O(n^k \cdot a^{n/b})$

(iii) if $a < 1$, $T(n) = O(n^k)$

For the above problem shortcut is.

$T(n) = 2T(n-1) + cn^0$ → For standard notation.

$= 2T(n-1) + O(n^0)$

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

⇒ Since $a = 2$ (> 1)

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

$= O(2^n)$

Ex: $T(1) = 1$ (if $n = 1$)

$T(n) = 2T(n-1) + n$ (if $n \geq 2$)

evaluates to,

(a) $2^{n+1} - n - 2$ (b) $2^{n+1} - 2n - 2$

(c) $2^n + n$ (d) $2^n - n$

→ Here we can't apply shortcut method b'coz options are not in that formate.

$n = 2 \Rightarrow$

which

Ex:

(a)

(c)

→

Put

Sub

T(

Ex: Let

Furth

$\forall n$.

$T(1) = 1$ (if $n=1$)

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

$n=2 \Rightarrow T(2) = 2T(1) + 2 = 2(1) + 2 = 4$

which option gives value 4 is the answer.

Ex: $T(n) = 2T(\sqrt{n}) + c$; if $n > 2$

$= 1$; if $n \leq 2$

(a) $O(\log_2 n)$ (b) $O(\log_2 \log_2 n)$

(c) $O(n \log n)$ (d) $O(n)$

$\rightarrow T(n) = 2T(\sqrt{n}) + c$ ---- 1st

$= 2\{2T(\sqrt{\sqrt{n}}) + c\} + c$ ---- 2nd

$= 2^2 T(n^{1/2^2}) + 3c$ ---- 2nd

$T(n) = 2^2 T(n^{1/2^2}) + (2^2-1)c$ --- 2nd

$T(n) = 2^k T(n^{1/2^k}) + (2^k-1)c$ ---- k^{th} ——— (*)

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

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

$k = \log_2 \log_2 n$

Substitute k in (*)

$T(n) = 2^{\log_2 \log_2 n} T(1) + (2^{\log_2 \log_2 n} - 1)c$

$= \log_2 n(1) + (\log_2 n - 1)c$ $\left[a^{\log_a b} = b \quad a=2,\ b=\log_2 n\right]$

$= O(\log_2 n)$

Ex: Let 'S' be a string containing either '0' or '1'. Further there are no two consecutive 0's in S. No. of solution on i/p size 'N' in S(N) is bounded by

(a) $O(n^2)$ (b) $O(n \log n)$ (c) $O(2^n)$ (D) $O(n)$

→

No. of bits (N)	Possible strings	No. of sol^n. in S(N)
0	NULL	1
1	0, 1 } 2	2
2.	0 0, 0 1, 1 0, 1 1 } 3	3
3	0 0 0, 0 0 1, 0 1 0, 0 0 0, 1 1 1, 1 1 0, 1 0 1, 0 1 1 } 5	5
....		

Let $T(n)$ be no. of possible sol^n. on i/p 'n' in S(N)

$$T(n) = \begin{cases} 1 & \text{if } n = 0 \\ 2 & \text{if } n = 1 \\ T(n-1) + T(n-2) & \text{if } n \geq 2 \end{cases}$$

$T(2) = T(1) + T(0)$

****

* Master Theorem:

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

where $a \geq 1$, $b > 1$, $k \geq 0$, 'p' is any real no.

case (i): If $a > b^k$, $T(n) = \Theta(n^{\log_b a})$

case (ii): If $a < b^k$ and

a) if $p \geq 0$; $T(n) = \Theta(n^k \log^p n)$

Case

Ex:

Ex:

Ex:

b). if $P < 0$; $T(n) = O(n^k)$

S(N) Case (iii): If $a = b^k$ and

a) If $P > -1$; $\Theta(n^{\log_b a} \log^{P+1} n)$

b) If $P = -1$: $\Theta(n^{\log_b a} \log \log n)$

c). If $P < -1$: $\Theta(n^{\log_b a})$

Case (i). Examples:

Ex: $T(n) = 16T\left(\frac{n}{4}\right) + n$

$= 16T\left(\frac{n}{4}\right) + \Theta(n \log^0 n)$

$a = 16, \; b = 4, \; k = 1, \; P = 0$

m S(N) From case (i) is $a > b^k$,

$16 > 4^1 \rightarrow$ yes

$\Rightarrow T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_4 16}) = \Theta(n^2)$

Ex: $T(n) = 4T\left(\frac{n}{2}\right) + \log n$

$= 4T\left(\frac{n}{2}\right) + \Theta(n^0 \log n)$

$a = 4, \; b = 2, \; k = 0, \; P = 1.$

Is $a > b^k$, $4 > 2^0$ (Yes)

no. $T(n) = \Theta(n^{\log_2 4}) = \Theta(n^2)$

Ex: $T(n) = \sqrt{2}\, T\left(\frac{n}{2}\right) + \log n$

$= \sqrt{2}\, T\left(\frac{n}{2}\right) + \Theta(n^0 \log n)$

$a = \sqrt{2}, \; b = 2, \; k = 0, \; P = 1$

Is, $a > b^k$, $\sqrt{2} > 2^0$ (Yes)

$\Rightarrow T(n) = \Theta\left(n^{\log_2 \sqrt{2}}\right) = \Theta(\sqrt{n})$

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

$T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n \log^0 n)$

$a = 3, \quad b = 2, \quad k = 1, \quad p = 0$

$\therefore$ Is $a > b^k$, $3 > 2$ (Yes)

$\Rightarrow T(n) = \Theta\left(n^{\log_2 3}\right)$

Case (ii) examples:

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

$a = 2, \quad b = 2, \quad k = 2, \quad p = 0$

Is, $a < b^k$, $2 < 2^2 \rightarrow$ yes

$p = 0 (\geq 0)$, case (ii) (a)

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

Ex: $T(n) = 6T\left(\frac{n}{3}\right) + n^2 \log n$

Is, $a < b^k$, $6 < 3^2 \rightarrow$ yes

$p = 1 (\geq 0)$, case (ii) (a)

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

Ex: $T(n) = 3T\left(\frac{n}{2}\right) + n^3 \log^2 n$

Is $a < b^k$, $3 < 2^3 \rightarrow$ yes

$\Rightarrow T(n) = \Theta(n^3 \log^2 n)$

Ex: $T(n) = 3T\left(\frac{n}{4}\right) + n \log n$

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

Ex: $T(n) = 3T\left(\frac{n}{2}\right) + \frac{n^2}{\log n}$

$= 3T\left(\frac{n}{2}\right) + n^2 \log^{-1} n$

$a = 3, \quad b = 2, \quad k = 2, \quad P = -1$

Is $a < b^k$, $3 < 2^2 \rightarrow$ yes

$P = -1 \ (< 0)$, case (ii) (b)

$\Rightarrow T(n) = O(n^k)$

$= O(n^2)$

case (iii): Examples:

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

$= 4T\left(\frac{n}{2}\right) + \theta(n^2 \log^0 n)$

$a = 4, \quad b = 2, \quad k = 2, \quad P = 0$

Is $a = b^k$, $4 = 2^2 \rightarrow$ yes.

$P = 0 \ (> -1)$, case (iii) a

$\Rightarrow T(n) = \theta\left(n^{\log_b a} \log^{P+1} n\right)$

$\Rightarrow T(n) = \theta\left(n^{\log_2 4} \log^{0+1} n\right)$

$= \theta(n^2 \log n)$

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

$a = 3, \quad b = 3, \quad k = 1, \quad P = 0$

Is $a = b^k$, $3 = 3^1 \rightarrow$ yes.

$P = 0 \ (> -1)$, case (iii) a

$\Rightarrow T(n) = \theta\left(n^{\log_3 3} \log^{0+1} n\right)$

$= \theta(n \log n)$

Ex: $T(n) = 3T\left(\frac{n}{3}\right) + \frac{n}{\log n}$

$a = 3, \quad b = 3, \quad k = 1, \quad p = -1$

Is $a = b^k$, $3 = 3^1 \longrightarrow$ yes.

$p = -1 \ (= -1)$, case (iii) (b)

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

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

Ex: $T(n) = 8T\left(\frac{n}{2}\right) + \frac{n^3}{\log^2 n}$

$a = 8, \quad b = 2, \quad k = 3, \quad p = -2$

$T(n) = 8T\left(\frac{n}{2}\right) + n^3 \log^{-2} n$

Is $a = b^k$, $8 = 2^3 \longrightarrow$ yes.

$p = -2 \ (< -1)$, case (iii) (c)

$\Rightarrow T(n) = \Theta\left(n^{\log_b a}\right) = \Theta\left(n^{\log_2 8}\right)$

$= \Theta(n^3)$

(II) 2005

Ex: $T(n) = 2T(n/2) + \sqrt{n}$

(A) $T(n) = \Theta(\log n)$ (B) $T(n) = \Theta(\sqrt{n})$

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

$a = 2, \quad b = 2, \quad k = 1/2, \quad p = 0$

case (i): $T(n) = \Theta\left(n^{\log_b a}\right)$

$= \Theta\left(n^{\log_2 2}\right)$

$= \Theta(n)$

Ex: $T(n) = T(n/3) + O(n)$

(A) $\Theta(n^2)$ (B) $\Theta(\log n)$ (C) $\Theta(n)$

Case (ii): $a < b^k$

2008

Ex: T(

(C)

(E)

Ex: T(

2005

Ex:

(A)

(C)

$\Rightarrow$ Sp

1). T(n

Sinc

So,

2). T

Here, 'a'

$P \geq 0$, case (ii) a

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

2008
Ex: $T(n) = \sqrt{2}\, T\left(\frac{n}{2}\right) + \sqrt{n}$

(A) $\sqrt{n}(\log n + 1)$ (B) $\sqrt{n} \log \sqrt{n}$

(C) $\sqrt{n} \log^2 n$ (D) $n \log \sqrt{n}$

(E) $\sqrt{n} \log n$

Ex: $T(n) = \sqrt{2}\, T\left(\frac{n}{2}\right) + \sqrt{n}$

$a = \sqrt{2}$, $b = 2$, $k = 1/2$, $P = 0$

Is $a = b^k$, $\sqrt{2} = 2^{1/2}$ → yes.

$P = 0$ $(P > -1)$, case (iii) (a)

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

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

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

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

2005
Ex: $T(n) = 2T(n/2) + O(n)$ which of the following is FALSE.

(A) $T(n) = O(n^2)$ (B) $T(n) = O(n \log n)$

(C) $T(n) = \Theta(n \log n)$ (D) $T(n) = \Omega(n^2)$

⇒ Special cases in Master Theorem:

1). $T(n) = 0.5\, T(n/2) + n^2$

Since $a = 0.5$ (< 1)

So, we can't apply master theorem.

2) $T(n) = 2^n\, T(n/2) + n^2$

Here, 'a' can't be a fun^n.

So, we can't apply M.T.

3). $T(n) = 2T(n/2) - n^2$

Negative fun^n. can't allow in M.T. So, we can't apply M.T.

4). $T(n) = 2T(n/2) + 2^n$ ← exponential fun^n, then put in directly in Answer.

$T(n) = 2T(n/2) + 2^n$

Ans: $O(2^n)$

5). $T(n) = 2T(n/2) + n!$

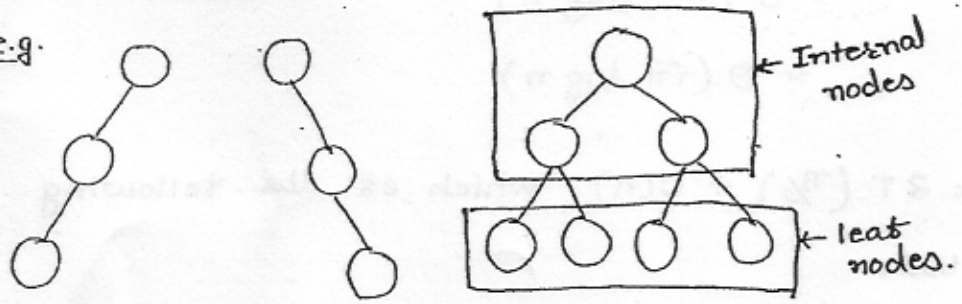
Ans: $O(n!)$

Divide & Conquer:

* **Binary Tree:**

In a binary tree every node must have atmost two children.

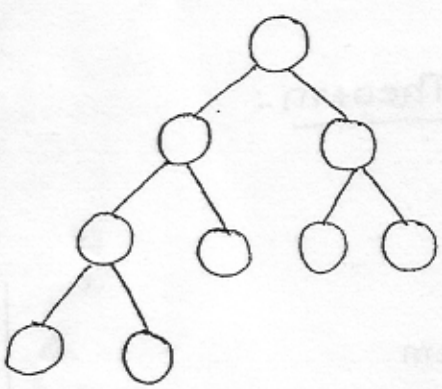
e.g.



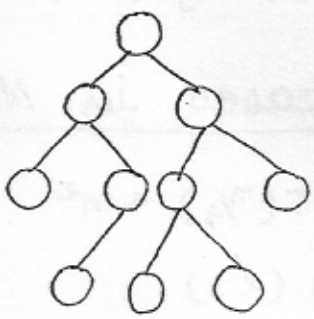
Left skewed B.T.

Right skewed B.T.

2-array tree Full B.T.



Complete B.T.

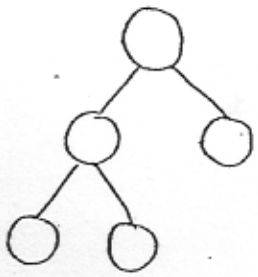


Not C.B.T.

→ 2-array tree:

- In a 2-array tree every internal node have exactly two childern.
- Every Full B.T is 2-array tree but converse is not true.

e.g.

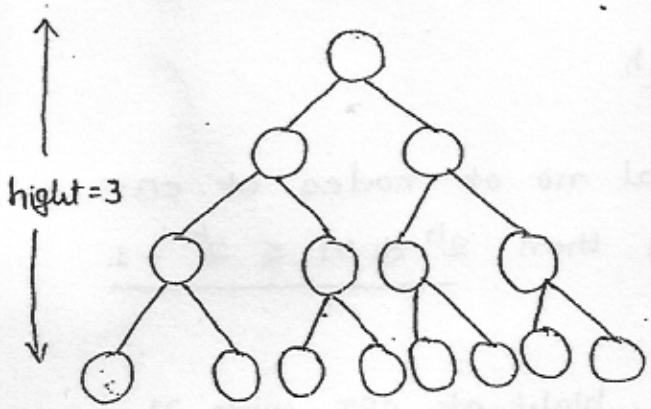


← It's 2-array tree but not F.B.T.

→ Complete Binary Tree. (C.B.T)

- In a CBT all levels are full except last level & it is filling from left to right.
- Every FBT is a CBT but converse need not be true.

⇒ Properties of FBT:



Level	No. of nodes
0	$1 = 2^0$
1	$2 = 2^1$
2	$4 = 2^2$
3	$8 = 2^3$

i) Hight:

- hight of leaf node = 0
- hight of internal node = logest path from that node to leaf.
- Height of tree = hight of root node.
  $= \log_2$(no. of leaf nodes)

$= \log_2(\text{no. of internal nodes} + 1)$

ii) Maximum no. of nodes of FBT whose hight is (h)

$= 2^0 + 2^1 + 2^2 + \ldots + 2^h$

$S_n = \frac{a(r^n - 1)}{r - 1}$; $|r| > 1$

$S_{h+1} = \frac{1(2^{h+1} - 1)}{2 - 1} = 2^{h+1} - 1$

⇒ Properties of CBT:

i) Maximum No. of nodes of CBT whose hight is (h) $= 2^{h+1} - 1$

ii) Minimum No. of nodes of CBT whose hight is (h) $= 2^h$

$= (2^0 + 2^1 + 2^2 + \ldots + 2^{h-1}) + 1$

$\left(S_h = \frac{1(2^h - 1)}{2 - 1}\right) + 1$

$= 2^h - 1 + 1 = 2^h$

iii) If n represent total no. of nodes of CBT whose hight is (h) then $2^h \leq n \leq 2^{h+1} - 1$

Theorm:
Prove that maximum hight of CBT with n nodes is bounded by $O(\log_2 n)$.

or

Prove that in a CBT with 'n' nodes and height 'h' satisfies the following.

$\log_2(n+1) - 1 \leq h \leq \log_2 n$

Proof: $2^h \leq n \leq 2^{h+1} - 1$

$2^h \leq n$

$\Rightarrow h \leq \log_2 n$ ....(i)

$n \leq 2^{h+1} - 1$

$n + 1 \leq 2^{h+1}$

$\log_2 (n+1) \leq h + 1$

$\log_2 (n+1) - 1 \leq h$ ....(ii)

Combining (i) & (ii)

$\log_2 (n+1) - 1 \leq h \leq \log_2 n$

$h = O(\log_2 n)$

*** hight of CBT $= O(\log_2 n)$

* Binary Search:

Search $x = 151$ from the following array.

-10	0	5	9	14	28	52	76	84	98	111	122	136	151
1	2	3	4	5	6	7	8	9	10	11	12	13	14

$x = 52$ (Average case)

$x = -10$ (Best case)

$x = 151$ (worst case)

Seq^n. search	Best case	Worst case	Average case
Successful	$O(1)$	$O(n)$	$O(n/2) = O(n)$
Unsuccessful		$O(n)$	

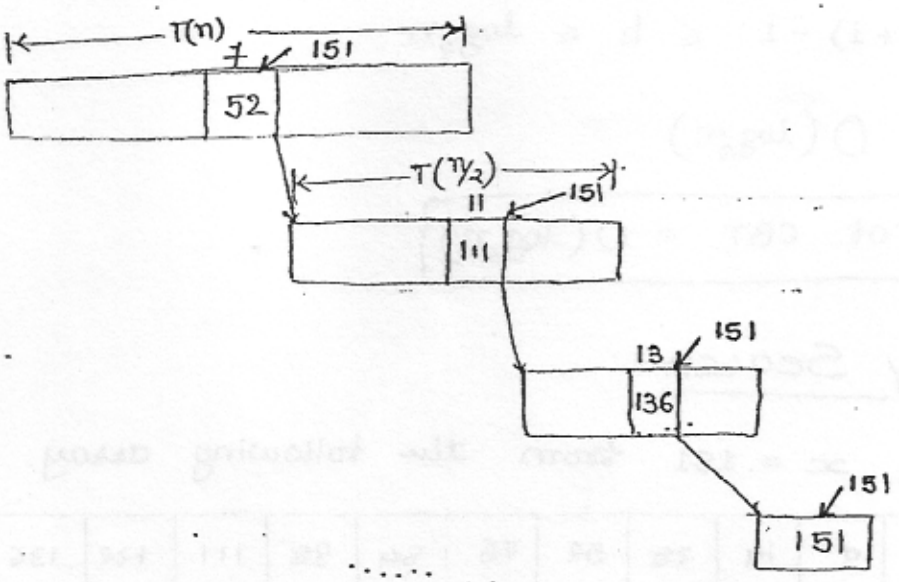
Binary Search	Best case	Worst case	Average Case
Successful	$O(1)$	$O(\log_2 n)$	$O(\log_2 n)$
Unsuccessful		$O(\log_2 n)$	

→ Binary search:

Low	High	Mid
1	14	7, a[7] = 52
8	14	11, a[11] = 111
12	14	13, a[13] = 136
14	14	14, a[14] = 151

x = 151

$$Mid = \left\lfloor \frac{(Low + High)}{2} \right\rfloor$$



.....

Comparision = Level + 1

= Hight of tree + 1

= Hight of CBT + 1

$= (\log_2 n + 1)$

$= O(\log_2 n)$

$T(n) = T(n/2) + 1$

$= T(n/2) + \Theta(n^0 \log^0 n)$

$a = 1, b = 2, k = 0, p = 0$

Is $a = b^k$, $p = 0 (> -1)$, case (iii)(a)

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

$= \Theta(n^{\log 1/2} \log^{0+1} n) = \Theta(\log n)$.

Ex:

Ex:

Application

Ex:

Ex: Let A[n] be array of n elements in increasing order where $n \in [2^{k-1}, 2^k]$ in order to search a key from that array is worst case. How many comparision are reqd. by using most efficient algorithm.

(a) n (b) k-1 (c) 2^k (d) k

→ In the previous example array contains 14 elements which are in the increasing order.

So, clearly $14 \in (2^3, 2^4]$.

To search a key x = 151, we need 4 comparisions. Therefore in worst case we need k comparision.

Ex: Suppose that we have no. betn. 1 & 100 in a binary search tree & we want to search for the no. 55. Which of the following seqn. of nodes can't be examined.

(A) {10, 75, 64, 43, 60, 57, 55}

(B) {9, 85, 47, 68, 43, 57, 55}

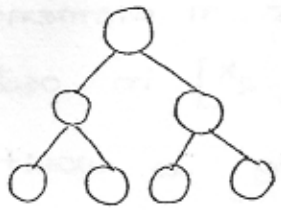
(C) {90, 12, 68, 34, 62, 45, 55}

(D) {79, 14, 72, 56, 16, 53, 55}

Application of B.T.

Ex: In a complete k-array tree every internal node has exactly k childrens. No. of leaf nodes in such a tree with n internal node.

(A) nk (B) (n-1) k+1 (C) n(k-1)+1 (D) n(k-1)



$k = 2$ (2-array)

$n = 3$

leaf node $= 4$

So, $n(k-1)+1$ gives answer 4.

Ex: If $L = 41$, $I = 10$ where L = no. of leaf node, I = no. of internal nodes of n-array tree. What is the value of n.

(a) 3 (b) 4 (c) 5 (D) 6.

No. of leaf nodes $= n(k-1)+1$

$L = I(n-1)+1$

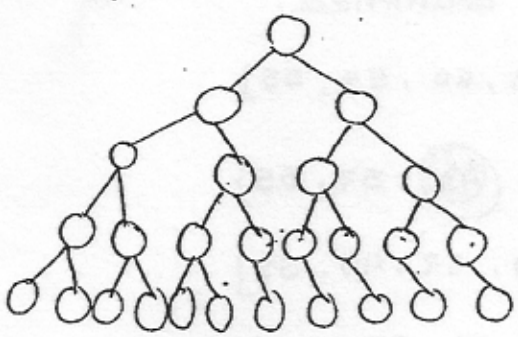
$41 = 10(n-1)+1$

$\Rightarrow n = 5$

IT 2006

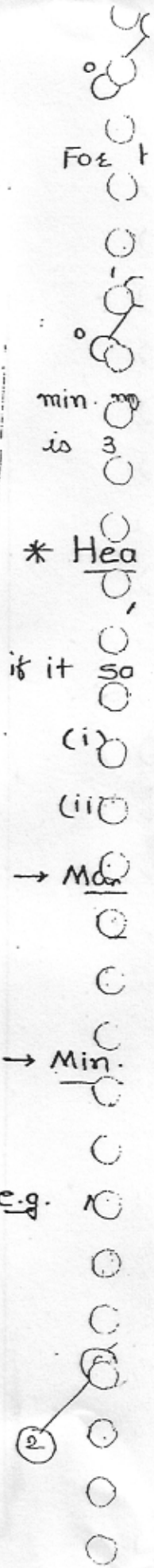
Ex: In a B.T. No: of internal nodes of degree 2 is 10 & no. of internal nodes of degree 1 is 5. what is the leaf node in search tree.

(A) 10 (B) 11 (C) 12 (D) 15



Ex: In a binary tree every node the difference bet^n the no. of nodes in a left & right sub tree is atmost 2. If the hight of the tree $h > 0$. Then the minimum no. of nodes in a tree —

(A) 2^h (B) $2^h - 1$ (C) 2^{h-1} (D) $2^{h-1} + 1$



For h = 1

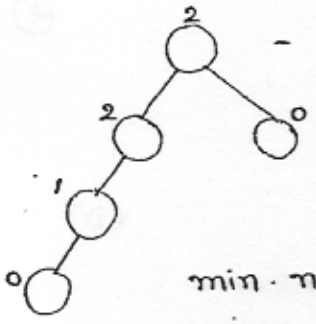
| LST nodes − RST nodes | ≤ 2

minimum node in h = 1 is 2 which option gives ans 2 is right.

For h = 2

min. nodes in h=2 is 3

For h = 3



min. nodes in h = 3 is 5.

* Heap Tree :

A binary tree is said to be heap tree if it satisfy following property.

(i) Structuring Property.

(ii) Ordering property. (Max-heap / Min-heap)

→ **Max. heap:**

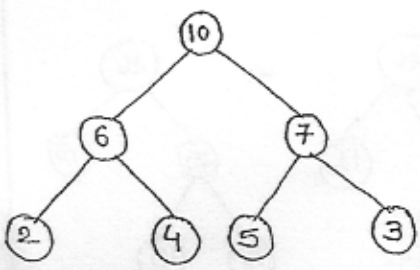
In max. heap tree

Parent > children.

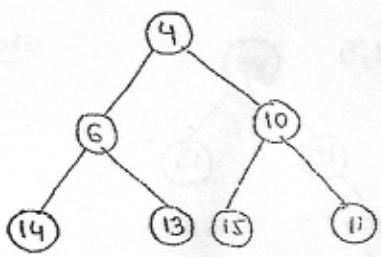
→ **Min. heap:**

Parent < children.

e.g. Max-heap:

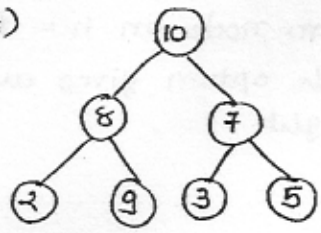


e.g. Min-heap.

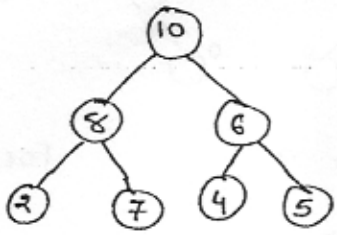


Ex: Which of the following is Max-heap.

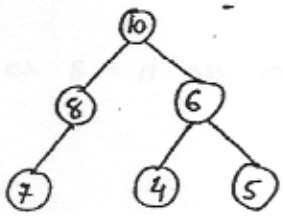
(A)



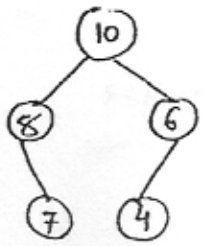
(B)



(C)



(D)



Not maintaining structuring property.
So, Not heap tree.

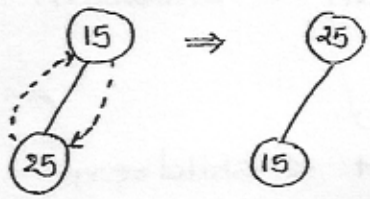
Note: By detaut every heap is Max-heap.

Ex: Construct heap tree tor the following key by inserting one after another in the given oreder.

15, 25, 13, 12, 26, 9, 16, 30

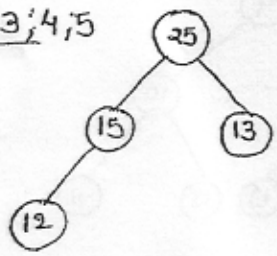
Step-1

Step-2

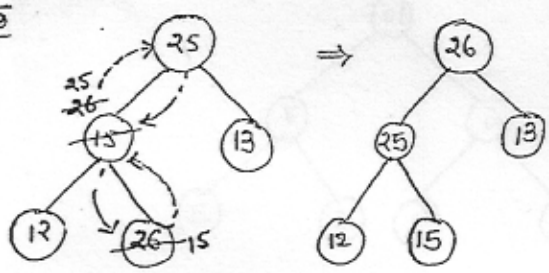


· b'coz parent is always big.
that's why we do swapping.

Step-3,4,5



Step-6



Step-9

→ A

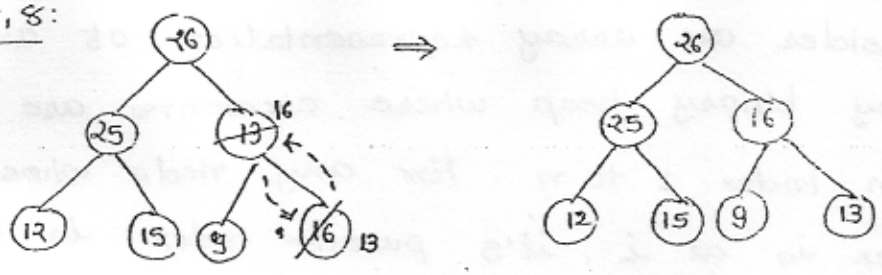
1).

2).

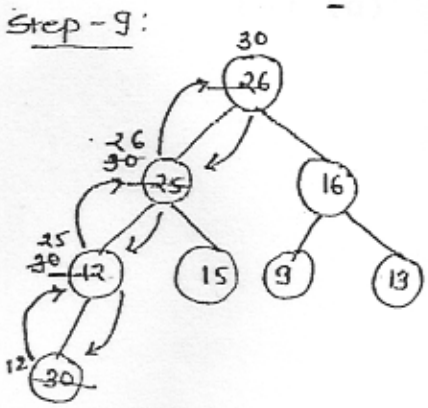
3).

* Arra

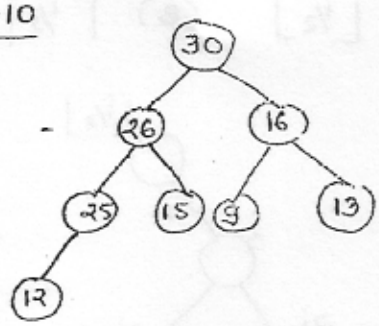
Step - 7, 8:



Step - 9:



Step - 10



→ Analysis of Heap tree:

1). To insert a key into empty heap takes the complexity $O(1)$.

e.g. Inserting node 15 takes $O(1)$.

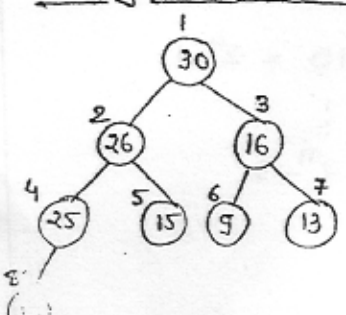
2). To insert a key into already constructed heap in the worst case it requires $\log n$ comparisions & $\log n$ swapings.

So, total time is $O(\log_2 n + \log_2 n) = O(\log_2 n)$

3). Since we are constructing heap tree with 'n' elements by inserting one after another.

So the total time complexity is $O(n \log n)$.

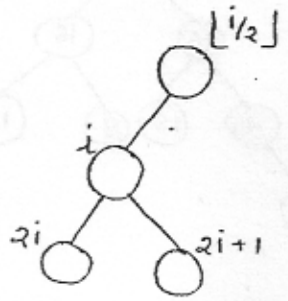
* Array Representation of Heap tree:



1	2	3	4	5	6	7	8
30	26	16	25	15	9	13	12

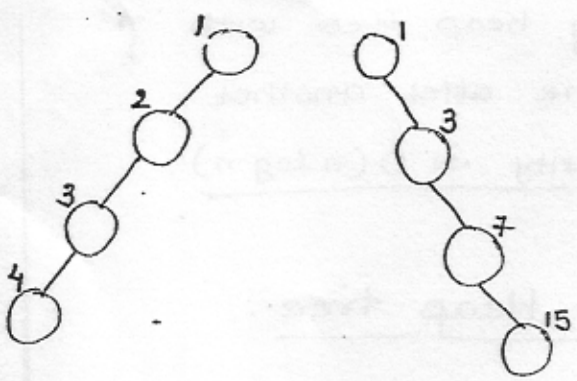
2001
Ex: Consider an array representation of an n array binary heap where elements are stored from index 1 to n. For any node whose index is at i, it's parent index is h.

(A) $\lfloor i/2 \rfloor$ (B) $\lceil i/2 \rceil$ (C) $\frac{i-1}{2}$ (D) $i-1$



Ex: A scheme of storing B.T. in an array X is as follows.
Indexing of x is starting from '1'.
For any node whose index is 'i' its left child is at X[2i], and right child is at X[2i+1]. To be able to store any B.T. with 'n' nodes minimum size of X ___

(A) n (B) $\log_2 n$ (C) $2n-1$ (D) $2n$



for left child X[2i] &
for right child X[2i+1]

n	Size
3	$7 = 2^3-1$
4	$15 = 2^4-1$
⋮	⋮
n	2^n-1

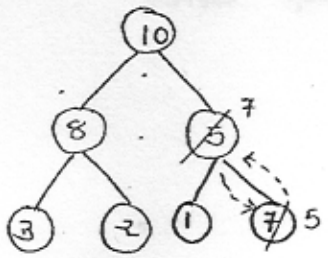
So, maximum $= 2^n-1$ &
Minimum $= n$

Ex:
Ex:
Ex:
CSE 2006
Ex:

Ex: A priority queue is implemented as a Max-heap. The level order traversal is 10, 8, 5, 3, 2. Two new elements '1' & '7' are added to traversal of heap. What is the level order after insertion.

(A) 10, 8, 7, 5, 3, 2, 1

(B) 10, 8, 7, 3, 2, 1, 5



10, 8, 7, 3, 2, 1, 5

Ex: which of the following sequence of array element represent Max-heap.

(A) {23, 17, 14, 6, 13, 10, 1, 12, 7, 5}

(B) {23, 17, 14, 7, 13, 10, 1, 12, 5, 7}

(C) {23, 17, 14, 7, 13, 10, 1, 5, 6, 12}

CSE 2006 Linked quen:

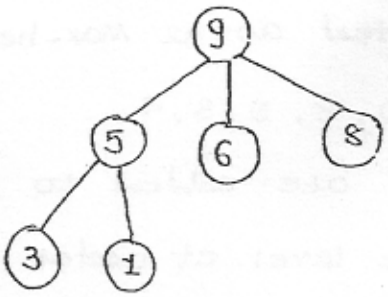
Ex: A three array Max-heap is implemented as follows. The root is stored from the location a[0]. and nodes in the next location is started from a[1] to a[n]. Which of the following represent 3-array Max-heap.

(A) 1, 3, 5, 6, 8, 9

(B) 9, 3, 6, 8, 5, 1

(C) 9, 6, 3, 1, 8, 5

(D) 9, 5, 6, 8, 3, 1

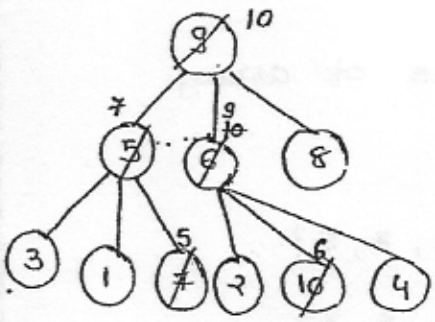


3-array means every node has 3-child.

Q-2 Suppose the elements 7, 2, 10 & 4 are inserted into the valid Max-heap found in the above question. What is the resultant heap after insertion.

(A) 10, 8, 6, 9, 7, 2, 3, 4, 1, 5.

(B) 10, 7, 9, 8, 3, 1, 5, 2, 6, 4



* Heap Sort:

Short the following array using heap sort.

1	2	3	4	5	6	7	8
15	25	13	12	26	9	16	30

Step-1: Construct heap tree by inserting keys one after another in the given order.

In each iteration, Delete root node by replacing it with last leafnode, & the Delelet node will be placed in highest empty index of the array.

Step-3/Ayi

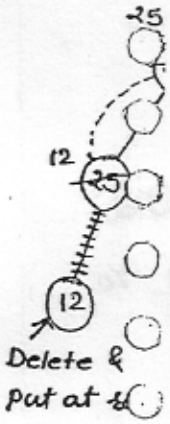
Step-4:Co

ar

Time

St

St



Step-3: Adjust the CBT such that it maintains Heap tree property.

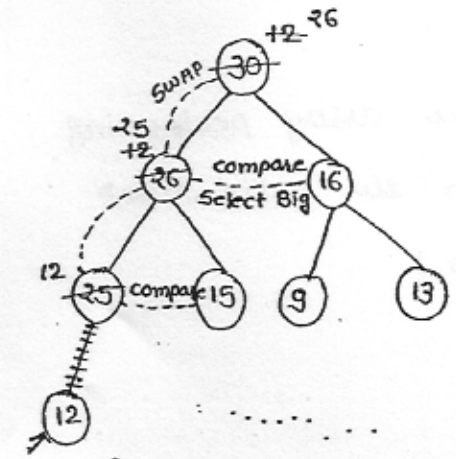
Step-4: Continue step-2 & step-3 untill all elements are sorted.

Time complexity for every step:

Step-1 = $O(n \log n)$

step-2 = $O(\log_2 n)$

Step-3 = $O(n \log_2 n)$



Delete & put at root node. ⇓

Comparision bet^n. Sibling to Sibling

= 2 = $\log_2 n$

Comparision bet^n parent to child

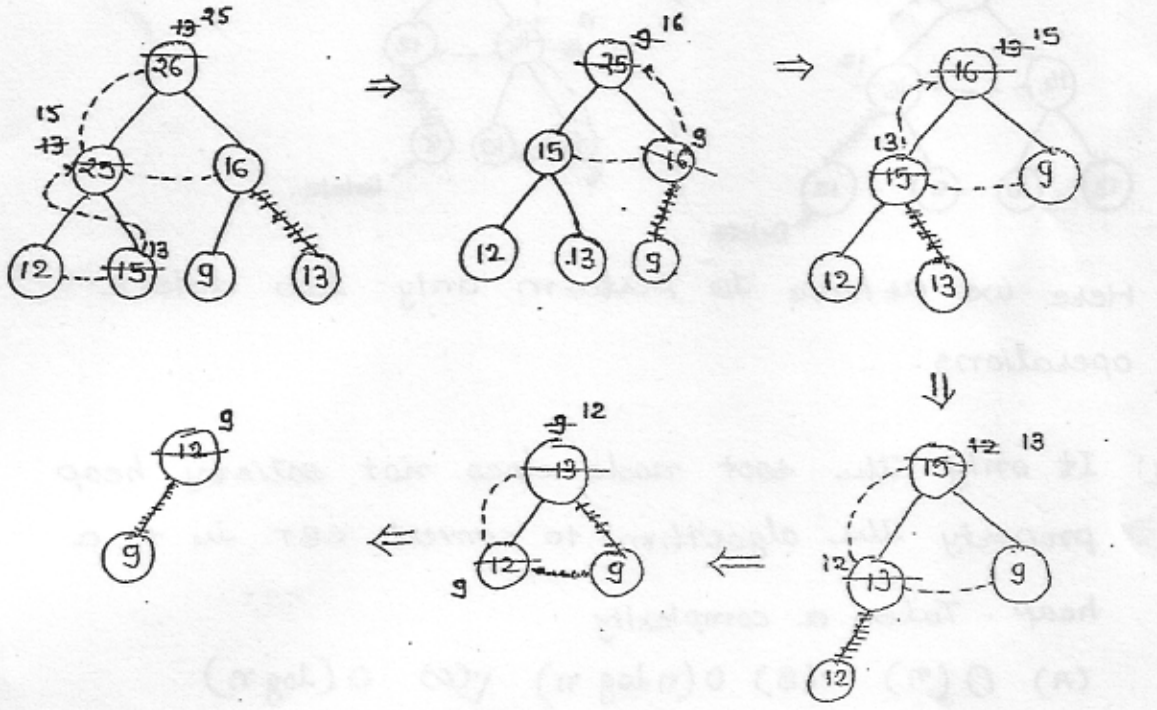
= 2 = $\log_2 n$

Swaping bet^n. parent to child

= 2 = $\log_2 n$

Total = $3 \log_2 n$

= $O(3 \log_2 n) = \log_2 n$



So,

1	2	3	4	5	6	7	8
9	12	13	15	16	25	26	30

Total time = step-1 + step+4

$= n \log n + n \log n$

$= O(2n \log n)$

$= O(n \log n)$

Ex:

Q-1: Which of the following seq^n. of arrays represent Max-heap.

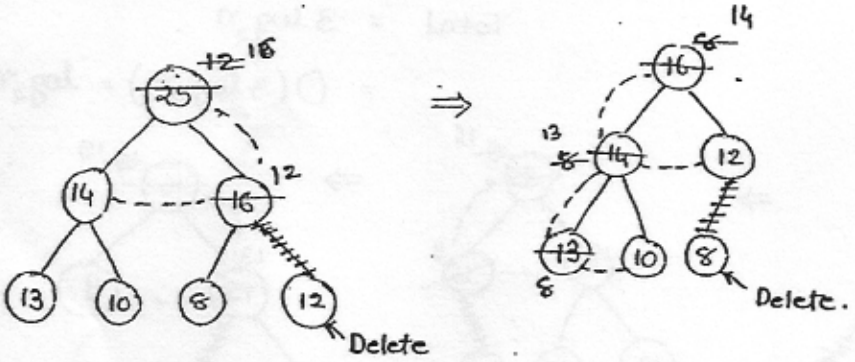
(A) {25, 12, 16, 13, 10, 8, 14}

(B) {25, 14, 16, 13, 10, 8, 12}

Q-2: Which is the content of the array performing after two delete operation on the valid heap found in the above question?

(A) {14, 13, 12, 10, 8}

(B) {14, 13, 12, 8, 10}



Here we have to perform only two delete operations.

Ex: If only the root node does not satisfy heap property the algorithm to convert CBT is to a heap. Takes a complexity

(A) $O(n)$ (B) $O(n \log n)$ (C) $O(\log n)$

(D) $O(n \log \log n)$

2003 Ex:

2007 Ex:

2003
Ex: In a heap with n elements smallest element is at the root. The 7^{th} smallest element can be found in time.

(A) $O(\log n)$ (B) $O(1)$ (C) $O(n)$ (D) $O(n \log n)$

→ To find the smallest element from the heap we have to perform 7 delete operation. Since each delete operation takes $\log n$ time,

So, total delete time $= O(7 \log n)$

$= O(\log n)$

Find	Max-heap (from array)	Min-heap (From array)
Max element	$O(\log n)$	$O(n \log n)$
Min element	$O(n \log n)$	$O(\log n)$

Ex: Time complexity of finding smallest element from the heap min-heap.

(A) $O(1)$ (B) $O(1)$ (C) $O(n \log n)$ (D) $O(\sqrt{n})$

→ Only one delete operation we have to perform.

2007
Ex: Consider the process of inserting an element into a max heap where the max-heap is represent by an array. Suppose we perform binary search on the path from the new leaf linked to the root node. To find position from newly inserted element no. of comparision perform.

(A) $O(\log_2 n)$ (B) $O(n)$

(C) $O(\log_2 \log_2 n)$ (D) $O(n \log_2 n)$

→ $O(\log_2 n)$ Seq^n. search	$O(n)$	$O(k)$	O (srinivas)
$O(\log_2 \log_2 n$ Binary search	$O(\log \frac{n}{2})$	$O(\log k)$	$O(\log_2$ srinivas)

2005 Ex: (∈

2008

Ex: We have a binary heap of n elements & wish to insert n more element (need not necessarily insert one after another) into this heap). Total time required is -

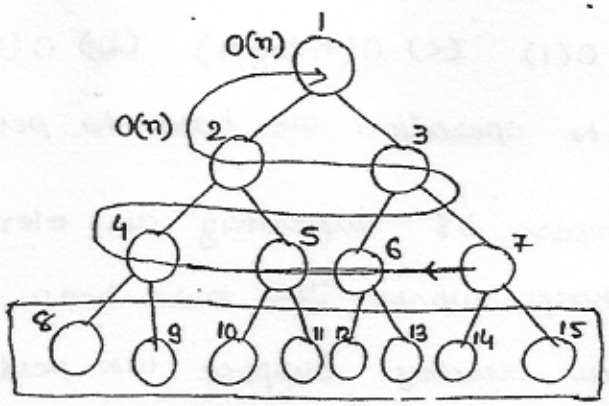
(A) $O(\log_2 n)$ (B) $O(n \log_2 n)$

(C) $O(n)$ (D) $O(n^2)$

Heap Tree Construction

insert key one after another in given order → $O(n \log n)$

keys need not necessarily insert one after another → $O(n)$



Here we have not consider leaf node.

Max-heap : parent > child.

∴ for ($i = \lfloor n/2 \rfloor$; $i > 0$; $i--$)

{

}

it takes $O(n/2)$ times

∴ $O(n)$

2005

Ex: Suppose there are $\log n$ sorted list & each list contains $\frac{n}{\log n}$ elements. The time complexity of producing a sorted list of all these elements is (Hint : Use heap data structure)

(A) $O(n \log \log n)$ (B) $\Omega(n \log n)$

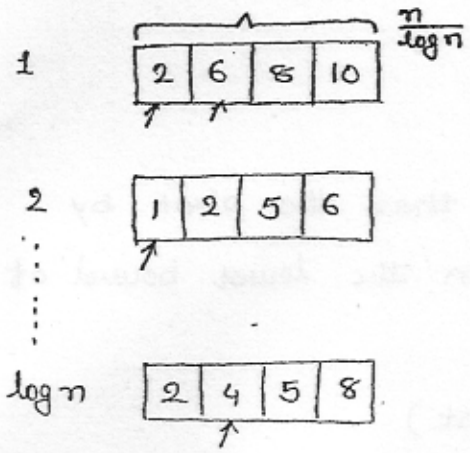
(C) $O(n \log n)$ (D) $\Omega(n^{3/2})$

→ Build heap tree with $\log n$ heaps takes the time complexity $O(\log n)$ [keys need not insert one after another].

→ Perform delete operation on heap tree with $\log n$ node it takes the complexity O

If we perform delete operation it returns the list whose 1st element is smallest among all then delete the current list first element & insert the remaining sorted list based on next element in the list. Perform this operation repeatedly until heap is empty.

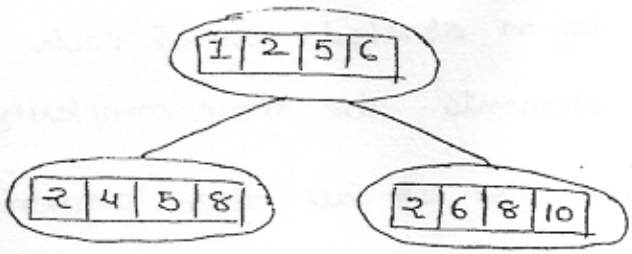
Since there are n each operation total time complexity = $O(n \log \log n)$



Total no. of elements

$= \log n * \frac{n}{\log n}$

$= n$



Remove first node from first non-leaf tree then remove first element from that node & leave other as it is.

1). $O(\log n)$

n { 2). $\log \log n$; 3). $\log \log n$

$= \log n + n \log \log n + n \log \log n$

$= \log n + 2n \log \log n$

$= O(2n \log \log n)$

$= O(n \log \log n)$

* Quick Sort:

Sort the following.

65 70 75 80 85 60 55 50 45

Step-1: Select any 1 for pivot, say a[k] where $L \leq k \leq U$, L = Lower bound of array, U = Upper bound of array.

Now hide pivot at a[0], to do this, swap (a[k], a[0]).

Step-2: Let i = L, j = U - 1

Step-3: Select bigger element than the pivot by searching forward from the lower bound of array.

i.e. while (a[i] ≤ pivot)

i++;

Step-4: Select smaller element than pivot by searching backward from the upper bound of the array.

i.e.

Step-5

Step-6

Step-7

Step-8

Step-9:

L
65

L
45 7
i

45 5

45

45 57

45

45 50

Left ≤

- Now

subt

i.e. while (a[i] >= pivot)
j--;

Step-5: Exchange a[i] & a[j]

Step-6: Repeatdly select elements in opposite ways until 'i' & 'j' are cross each other

Step-7: Now replace pivot at a[i].

Step-8: Divide the array into two sub arrays at pivot position say they are left & right sub arrays.

Step-9: Continue above process recursively on left & right sub arrays until all elements are sorted.

L U
65 70 75 80 85 60 55 50 45 ....

Let a[k] = 65, SWAP L & U

L U-1
45 70 75 80 85 65 55 50 | 65 ← Hide.
i i j

45 50 75 80 85 60 55 70 65
i j

45 50 55 80 85 60 75 70 65
i j

45 50 55 60 65 80 75 70 85
i j

45 50 55 60 (65) 80 75 70 85

45 50 55 60 — Left sub tree

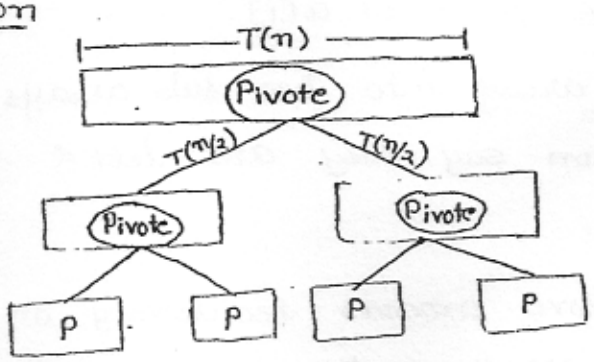
80 75 70 85 — Right sub tree.

- Now recursively apply above process on left & right subtree array until all elements are sorted.

⇒ Analysis of Quick sort:

Best case:

In each iteration if selected pivot placed in exactly in middle most position then it is called best case in quick sort.



Total time complexity for n element is T(n).

$T(n) = T(n/2) + T(n/2) + O(n)$

↳ Time required for placing pivote.

$= 2T(n/2) + O(n)$

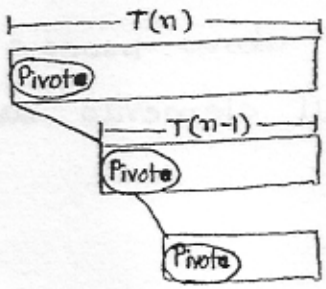
$\therefore O(n \log n)$

No. of iteration required for sorting n elements in best case of quick sort = hight of CBT + 1

$= \log_2 n + 1$

Worst case:

In each iteration if the selected pivot placed in either 1st or Last position then it is called worst case of quick sort.



→ e.g.

$T(n) = T(n-1) + O(n)$

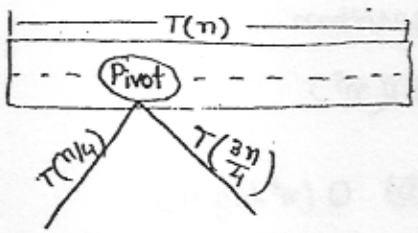
$= O(n^2)$ $\left[\begin{array}{l} T(n) = aT(n-b) + O(n^k) \\ \text{if } a=1,\ T(n) = O(n^{k+1}) \end{array}\right]$

In worst case of quick sort to sort n elements we need n iterations.

Average Case:

In each iteration if the selected pivot placed in $n/4^{th}$ position or $3n/4^{th}$ position then it is called as average case.

Iteration	No. of elements sorted
1	



$T(n) = T(n/4) + T\left(\frac{3n}{4}\right) + O(n)$

↳ Constructive method.

$= O(n \log n)$ ....

Note:

Worst case behavior of best case behaviour $= O(n \log n)$

Worst case behavior of Avg. case behaviour $= O(n \log n)$

Worst case behavior of Worst case behaviour $= O(n^2)$

2006

Ex: Median of 'n' elements can be found in $O(n)$ time. Which of the following is correct about time complexity of quick sort.

(A) $O(n \log n)$ (B) $O(n)$

(C) $O(n^2)$ (D) $O(n^3)$

→ e.g. 2 1 6 3 5

Median ⇓ To find median 1st arrange into accending order.

1 2 (3) 5 6

To find median we can apply any sorting method. So minimum time = $O(n \log n)$, Maximum time = $O(n^2)$

In quick sort example we found median 65 in $O(n)$ without sorting all elements. It is possible only in the case of best case of quick sort so time complexity is order of $O(n \log n)$.

2009
Ex: In quick sort for sorting n elements the $n/4^{th}$ smallest element is selected as pivot using $O(n)$ time then what is the worst case time complexity of quick sort algorithm.

(A) $O(n)$ (B) (n^2)

(C) $O(n \log n)$ (D) $O(n^2 \log n)$

... Worst case of average case = $O(n \log n)$

Depends on pivot position.

2008
Ex: Consider the quick sort algorithm. Suppose there is a procedure to find a pivot element which splits list into two sublist. Each of which contains atleast $1/5^{th}$ elements. Let $T(n)$ be the no. of comparision required.

(A) $T(n) \leq 2T(n/5) + n$

(B) $T(n) \leq 2T(4n/5) + n$

(C) $T(n) \leq 2T(n/2) + n$

(D) $T(n) \leq T(n/5) + T(4n/5) + O(n)$

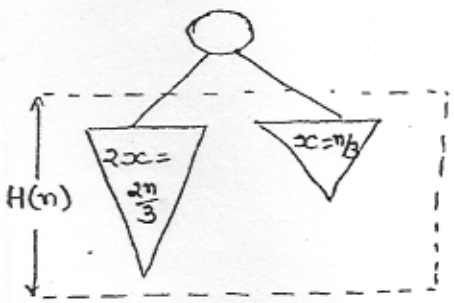
2002
Ex: A weight balanced tree is a B.T. in which each node the no. of nodes in the left sub tree is atleast half & atmost twice the no. of nodes in the right sub tree. The maximum possible

High
(A)
Let
the
left
node
hight
H(n)
H
$(2/3)^k$
$n = 9$
$k = 4$

hight of tree is described by.

(A) $\log_2 n$ (B) $\log_3 n$ (C) $\log_{4/3} n$ (D) $\log_{3/2} n$

Let H(n) represent maximum possible hight of the tree with n nodes. If we assume that left subtree contains atmost twice the no of nodes in the right sub tree then the maximum hight is possible only in direction of left sub tree.



$2x + x = n$

$\Rightarrow 3x = n$

$\Rightarrow x = n/3$

$H(n) = H(2/3\, n) + 1$ ---- 1st

$= H\left((2/3)^2 n\right) + 1 + 1$

$= H\left((2/3)^2 n\right) + 2$ ---- 2nd

$\vdots$

$= H\left((2/3)^k n\right) + k$ ---- kth ____(*)

$(2/3)^k n = 1$

$n = (3/2)^k$

$k = \log_{3/2} n$

Put k in (*)

$= H(1) + \log_{3/2} n$

$= 0 + \log_{3/2} n$

$= O(\log_{3/2} n)$

GREEDY METHOD

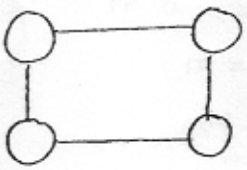
* Spanning Tree:

Let G(V, E) be an undirect connected graph A subset T(V, E') of G(V, E) is said to be a spanning tree iff T is a tree.

→ Connected graph:

In a connected graph bet^n. every pair of vertex there exist a path.

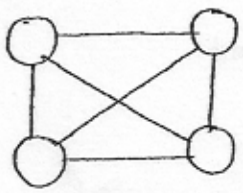
e.g.



→ Complete graph:

In a complete graph bet^n. every pair of vertices there exists an edge.

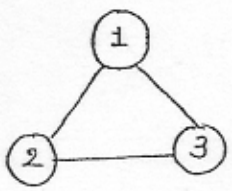
e.g.



→ Total no of edges in a complete undirected graph with n vertices = $(n-1) + (n-2) + \ldots + 0$

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

e.g.



vertex : 1 2 3

outdeg : 2 + 1 + 0 = 3

→ Total no of edges in a complete directed graph with n vertices = $2 \times \left(\frac{n(n-1)}{2}\right)$

$$= n(n-1)$$

e.g

* The

Prove

with

e.g. T

⇒

cas

Cas

Case

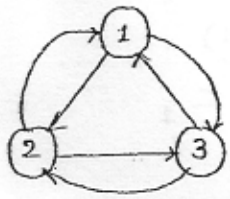
Cas

∴ T

Proof:

(i) G

e.g.



$\Rightarrow 2 \times 3 = 6$ edges

graph
be

* Theorm:

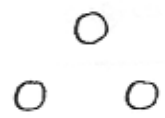
Prove that maximum no. of undirected graphs with 'n' vertices = $2^{\text{no. of edges}}$

e.g. Take $n = 3$ vertices.

$\Rightarrow$ Maximum no. of edges $= \frac{n(n-1)}{2} = \frac{3(3-1)}{2} = 3$ edges

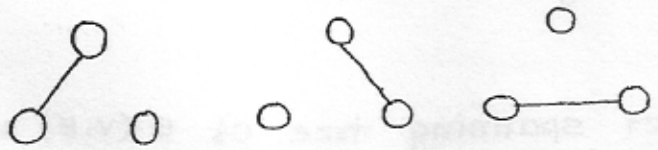
↳ undirected graph.

case (i) : Graph with '0' no of edges.



$= {}^3C_0 = 1$

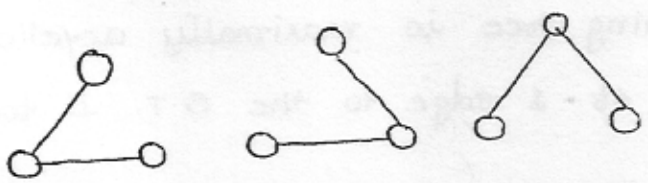
case (ii) : Graph with '1' no. of edges.



$= {}^3C_1 = 3$

vertices

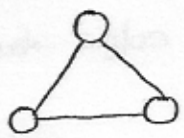
case (iii) : Graph with '2' no. of edges.



$= {}^3C_2 = 3$

graph

case (iv) : Graph with '3' no. of edges.



$= {}^3C_3 = 1$

$\therefore$ Total no. of graph $= 1 + 3 + 3 + 1 = 8$

Proof: n = vertices

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

(i) Graph with '0' edge $= {}^{\frac{n(n-1)}{2}}C_0$

(ii) Graph with '1' edge = $^{\frac{n(n-1)}{2}}C_1$

⋮

Graph with $\frac{n(n-1)}{2}$ edges = $^{\frac{n(n-1)}{2}}C_{\frac{n(n-1)}{2}}$

∴ Total no. of graphs = $^{\frac{n(n-1)}{2}}C_0 + {}^{\frac{n(n-1)}{2}}C_1 + \dots + {}^{\frac{n(n-1)}{2}}C_{\frac{n(n-1)}{2}}$

$= 2^{\frac{n(n-1)}{2}}$ $\left[\because {}^nC_0 + {}^nC_1 + \dots + {}^nC_n = 2^n\right]$

∴ Total no. of graphs = $2^{\text{No. of edges}}$.

* Properties of Spanning tree:

1). Every spanning tree of G(V, E)

2). To construct spanning tree of G(V, E) we have to remove (E − V + 1) no. of edges from G(V, E)

3). Every spanning tree is maximally acyclic that is by addition of 1 edge to the S.T. it forms a cycle.

4). Every spanning tree (S.T) is minimally connected. i.e. by the removal of 1 edge from ST it becomes disconnected.

5). Maximum no. of possible ST of the complete graph G(V, E) is V^{V-2}.

6). To identify minimum cost ST out of V^{V-2} ST we can take help of either prim's or kruskal's algorithm.

e.g.

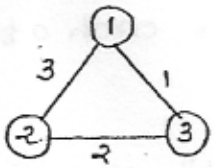
Ex:

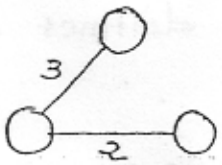
→ e.g.

cost

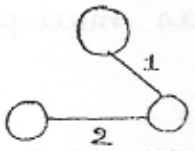
Ex:

e.g.

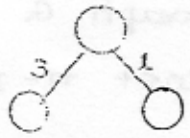




T (3, 2)
(V, E')
cost = 5



T' (3,2)
cost = 3



T'' (3,2)
cost = 4

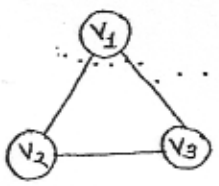
E	V	remove = E-V+1
3	3	1
6	4	3

V	S.T
3	$3 = 3^{3-2} = V^{V-2}$

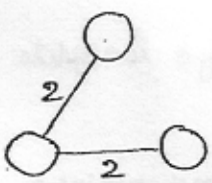
Ex: Consider a weighted complete graph G on the vertex set $\{V_1, V_2, \ldots V_n\}$ such that weight of an edge $\langle V_i, V_j \rangle = 2|i-j|$. Then weight of minimum cost spanning tree (MST) =

(A) n^2 (B) $2n-1$ (C) $2n-2$ (D) $n/2$

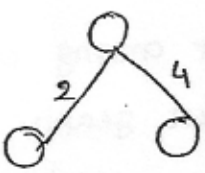
→ e.g.



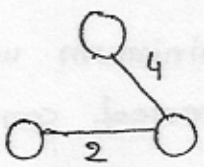
$\Rightarrow \langle V_i, V_j \rangle = 2|i-j|$
$= 2|1-2|$
$= 2$



cost = 4



cost = 6



cost = 6

⇓
n = 3

Put n = 3 in options & check ans gives 4 or not.

Ex: Consider an undirected graph G with n nodes. It's adjacency matrix is given by n x n square matrix whose principle diagonal elements are 0 & remaining elements are 1 which of the following is true.

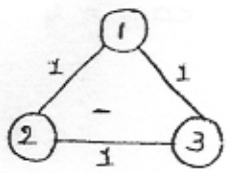
(A) Graph G has no MST.

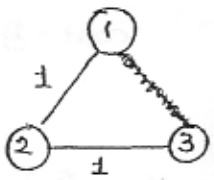
(B) Graph G has unique MST of cost n-1

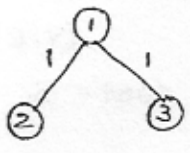
(C) Graph G has multiple distinct MSTs each of cost $n-1$

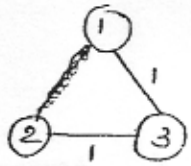
(D) Graph G has multiple STs of diff^{n}. cost.

→









cost = 2 = 2 = 2

= 3-1 = n-1 = n-1

= n-1

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \end{array}$$

Graph

- Adjacency Matrix
  - Unweighted
  - Weighted
- Adjacency list

Unweighted

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \end{array}_{3\times 3}$$

Weighted

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 2 & 3 \\ 2 & 0 & 4 \\ 3 & 4 & 0 \end{bmatrix} \end{array}$$

Ex: Let 'w' be minimum weight among all edge weights in an undirected connected graph.

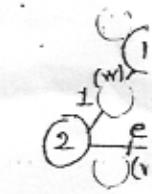
Let 'e' be a specific edge which contains weight 'w'. Which of the following is false.

(A) There is a MST which contains an edge 'e'

(B) Every MST contain an edge of weight 'w'.

(C) If 'e' is not in MST, 'T' then by adding e to 'T' it forms a cycle.

(D) Every MST contain an edge of 'e'.



Hel

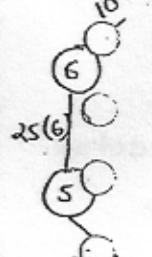
we

H.W:

*

F

→



→ Ana

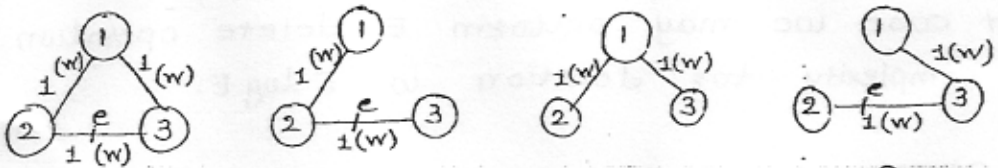
1). Adj

order

2). In

queu

pari



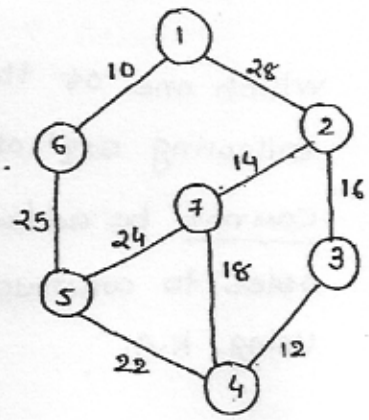
= 2 = 2 = 2

Here we have to take same weight edges b'coz we want more than 1 MST.

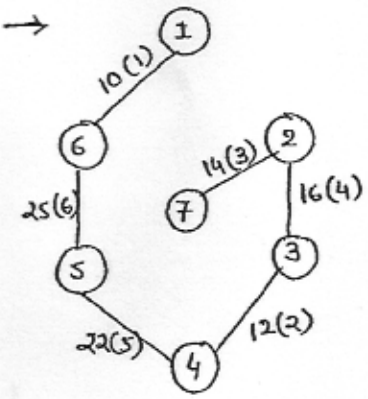
H.W: Refer CSE 2011 problem:

* Kruskal's Algorithm:

Find MST for the following graph.



→



It's a MST =

COST = 99

It satisfies all the properties of spanning tree.

→ Analysis of Kruskal's Algorithm:

1). ~~Adjust~~ Edges must be arrange in the increasing order of their weight with order of $E \log E$ time.

2). In each iteration delete root node from priority queue with $\log E$ time and include it into the partially constructed forest without forming a cycle.

In worst case we may perform E delete operation.
So time complexity for deletion is E log E.

3. Time complexity of K.A. = step 1 + step 2

= E log E + E log E

= O(E log E)

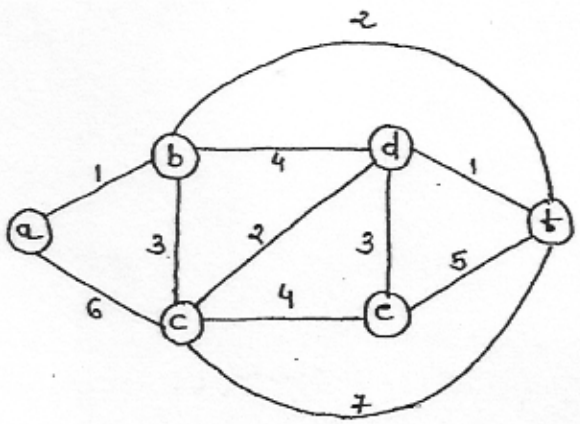
Priority Queue (Heap tree).

10 12 14 16 18 22 24 25 28

<1 6> <3 4> <2 7> <2 3> <7 4> <5 4> <5 7> <5 6> <1 2>

Delete but not include in heap b'coz it forming a cycle.

2006
Ex:



Which one of the following seq^n. of edges cannot be added in order to construct MST using K.A.

(A) (a-b)^1 (d-f)^1 (b-f)^2 (d-c)^2 (d-e)^3

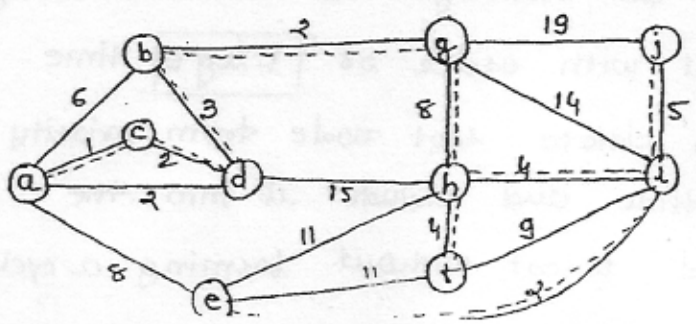
(B) (a-b)^1 (d-f)^1 (d-c)^2 (b-f)^2 (d-e)^3

(C) (d-f)^1 (a-b)^1 (d-c)^2 (b-f)^2 (d-e)^3

× (D) (d-f)^1 (a-b)^1 (b-f)^2 (d-e)^3 (d-c)^2

2009
Ex: Refer.

2003
Ex: What is the weight of MST for the following graph.



→ Com
1. Stru
alwa
weig
2. K.A
algo

* Ana
1). F
so
2). F

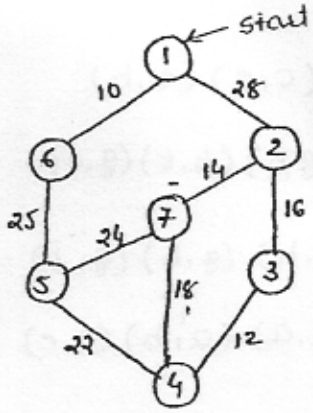
2008
Ex: Fo
whi
cor
MS

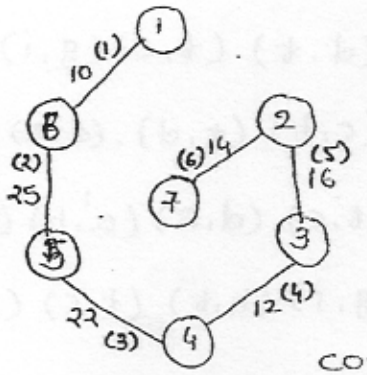
operation.

(A) 30 (B) 31 (C) 32 (D) 41

* Prim's Algorithm:

Apply Prim's algo. on the following graph. On starting vertex 1.





cost = 99

→ Comparision bet^{n}. Prim's & K. Algo.

1. Structure of MST W.r.T. Prim's & K.A. is always same if graph contains distinct edge weights.

2. K.A. does not maintain continuty where prim's algo. maintain continuty.

* Analysis of Prim's Algo:

1). By using adjacency matrix:

Since adjacency matrix contains n^2 elements so time complexity bounded by $O(n^2)$.

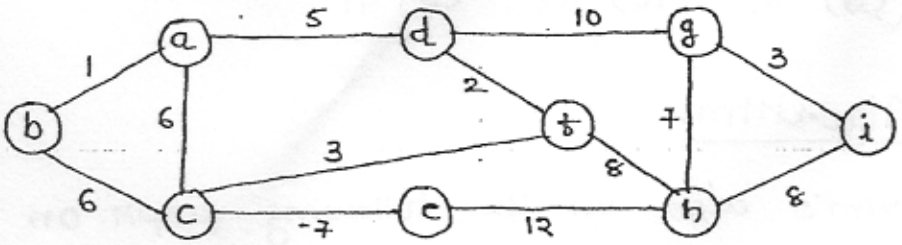
2). By using binary heap (Priority queue):

By using binary heap time complexity.

$$= O((V+E) \log V)$$

2008

Ex: For the undirected weighted graph given below which of the following seq^{n}. of edges represent correct execution of prim's algo. to construct MST.



× (A) (a', b) (d, f) (f, c) (g, i) (d, a) (c, e) (f, h)

(B) (c, e) (c, f) (f, d), (d, a) (a' b) (g, h) (h, f) (g, i)

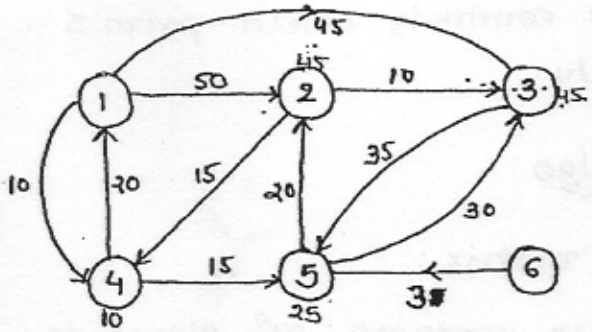
✓ (C) (d, f) (f, c) (d, a) (a', b) (c, e) (f, h) (g, h) (g, i)

(D) (h, g) (g, i) (h, f) (f, c) (f, d) (d, a) (a', b) (c, e)

→ No. of edges = V − 1 = 8

* Disjkstra Algo:

Find the shortest path distance from source vertex ① to remaining all vertex of graph.



→ DSSSP algo. can be implimented only on +Ve weight edged graph.

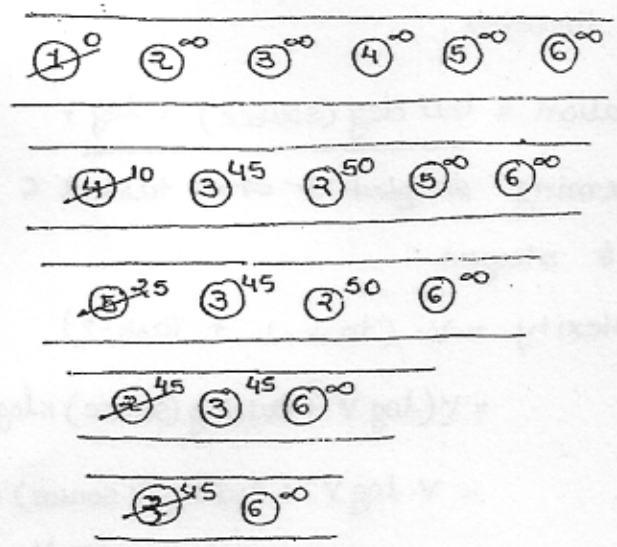
Cost Matrix Representation

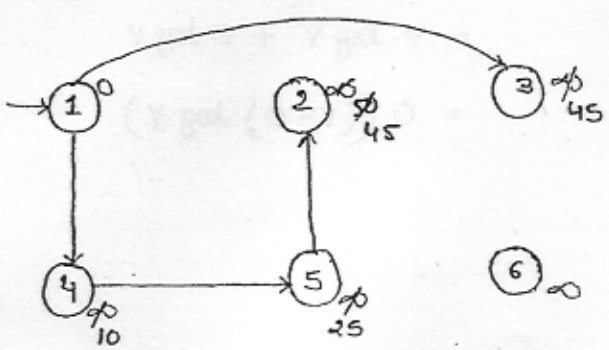
	2	3	4	5	6
S = {1}	50	45	10*	∞	∞
S = {1,4}	50	45	10*	25*	∞
S = {1,4,5}	45*	45	10*	25*	∞
S = {1,4,5,2}	44*	45*	10*	25*	∞
S = {1,4,5,2,3}	44*	45*	10*	25*	∞

- Time
- Draw
- cost
- bind
- given
- path
- We
- over

Source	Destination	Path
1	2	1 - 4 - 5 - 2
1	3	1 - 3
1	4	1 - 4 -
1	5	1 - 4 - 5
1	6	No path.

- Time complexity of DSSSP using cost matrix is $= O(n^2)$
- Drow back of cost matrix represent is we can't find shortest path implicitly. i.e. we can find a vertices which are reachable from the given source but it can not generate shortest path.
- We have to construct shortest path explicitly. To overcome this drawback we can use graph method.





→ Analysis:

1. To delete a key in Priority queue (Heap) = $\log V$
2. To adjust a key in Priority queue (Heap) = $\log V$
3. Out degree of all vertices (V) = E (i.e. total no. of edges)

At each stage (i.e each vertex) we are performing two task:

task-1: Deleting a vertex from P.Queue whose lable is minimum with order of $\log V$ time.

tas-2: Identify all vertices which are having direct age from the source & update their cost label.

→ No. of updated vertices = Out deg (Source vertex)

Since to update each vertex lable in P.Queue takes order of $\log V$ time therefor,

→ Total time to updation = Out deg (source) x $\log V$

task-3: Since we are performing ~~stage-1 & step~~ task-1 & task-2 in V no. of stages

∴ Total time complexity = V · (task-1 + Task-2)

$= V(\log V + \text{Outdeg (source)} \times \log V)$

$= V \cdot \log V + \text{Outdeg (V.source)} \times \log V$

$= V \cdot \log V + E \log V$

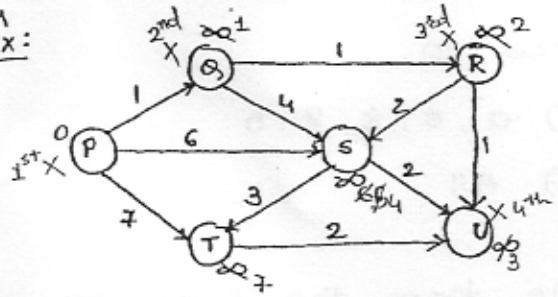
$= O((V+E) \log V)$

2004 Ex:

2006 Ex:

2008 Ex:

2004 Ex:



It we run DSSSP algo. of vertex 'P' in what order do the nodes get included, in order to compute shortest path.

(A) P Q R S T U

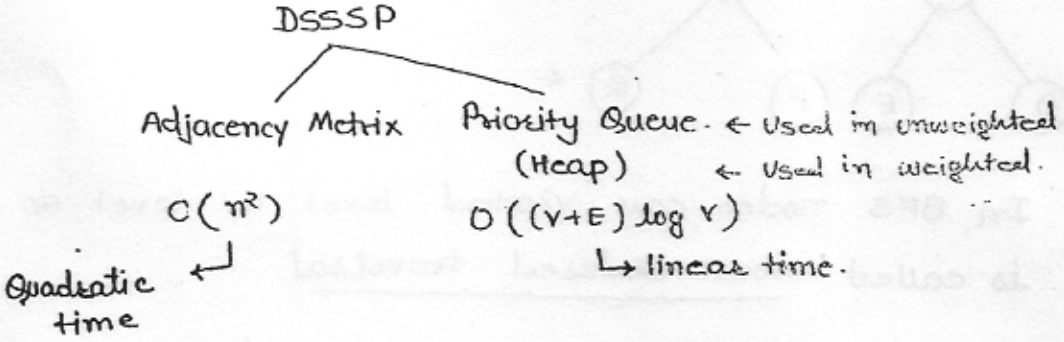
(B) P Q R U T S

(C) P Q R U S T

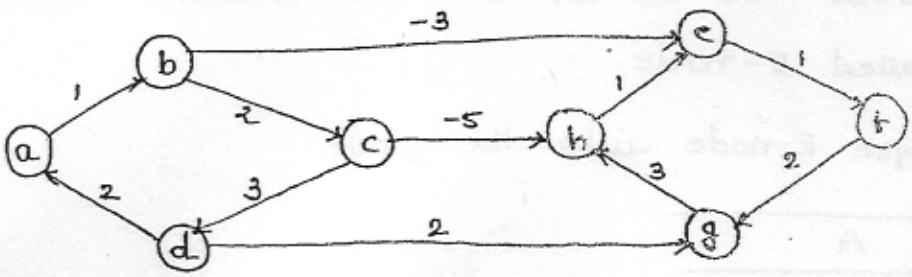
(D) P Q T R U S

2006 Ex: To implement DSSSP algo. on weighted graph so that it runs on linear time, what is the data structure to be used.

(A) Queue (B) stack (C) B-Tree (D) Heap



2008 Ex:



It we run DSSSP on vertex 'a', shortest path

distance can be computed to.

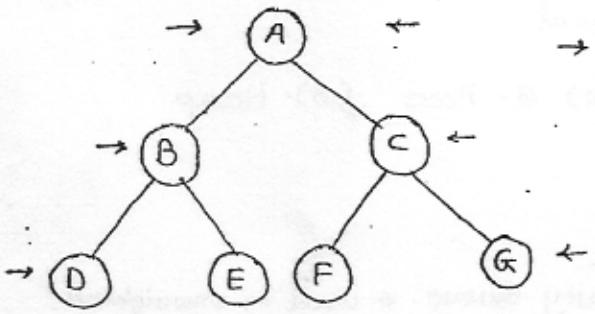
(A) a, b, c, d (B) a, e, f, g, h

(C) a (D) All

→ If any vertex is reachable from the given source with the +Ve & -Ve weight edges (i.e. vertex 'g', +Ve path is a-b-c-d-g & -Ve path is a-b-e-f-g) for such type of vertices we can not find shortest path distance. Ignor that vertex.

* BFS - Breath Fist Search :

1. BFS is useful for finding shortest path distance in the unweighted graph.



→ A B C D E F G

or

A C B G F E D ←

2. In BFS nodes can visited level by level so it is called level ordered traversal.

3. To impliment BFS we use queue data structure.

E-node : Nodes not at explore it's childsen are called E-node.

→ Enque E-node into the queue.

A

↳ E-node

Live

Dead

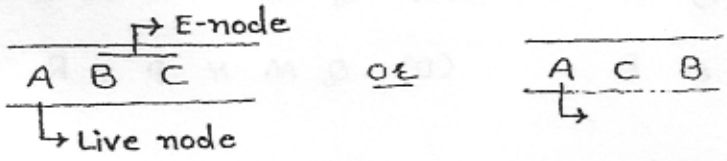
- Contin

⇒ Time

Ex:

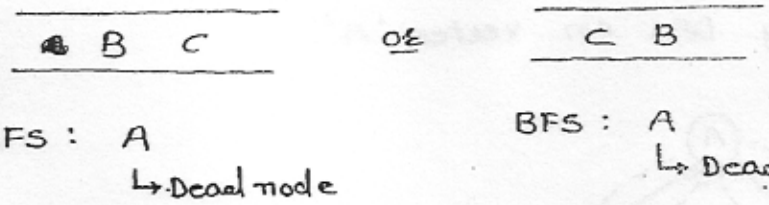
Ex:

Live node : Nodes currently being exploring its children are called live node.



Dead node : Node which can not explore further is called dead node. Dequeue dead node from the queue & append them into BFS sequence.

e.g.



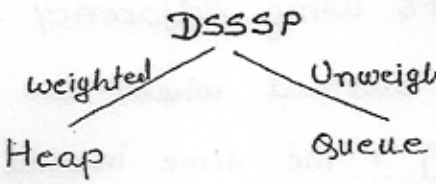
- Continue above process until all nodes are visited.

⇒ Time complexity of BFS using adjacency list :

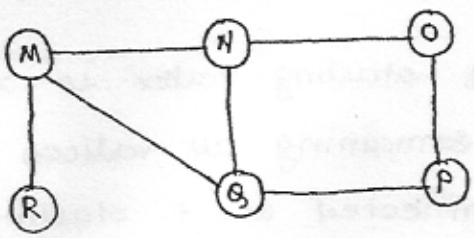
Time complexity of BFS using adjacency list = $O(V+E)$

Ex: To implement DSSSP on unweighted graph so that it runs in a linear time data structure to be used ---

(A) Queue (B) stack (C) B-Tree (D) Heap.



Ex:



Which of the following is valid BFS.

(A) M N O P Q R ✗ (B) Q M N P R O ✓

(C) N Q M P O R ✗ (D) Q M N P O R

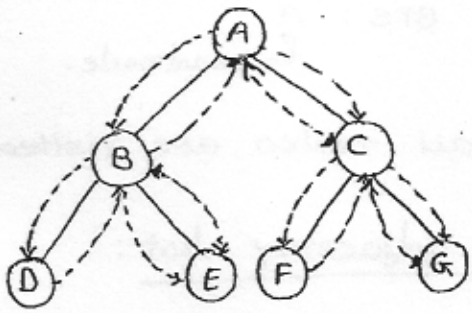
M $\xrightarrow{\text{child}}$ R Q N

N $\xrightarrow{\text{child}}$ M Q O

Q $\xrightarrow{\text{child}}$ M N P

* DFS - Depth First Search:

Apply DFS on vertex 'A'.



DFS seq^{n}: A B D E C F G

1). In DFS nodes can visited in deepest direction until it reaches leaf node & then it applies backtracking for processing remaining nodes.

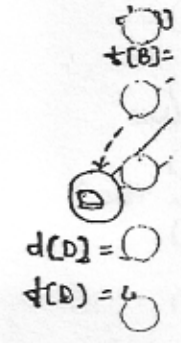
2). To impliment DFS we use stack data structure.

3). Time complexity of DFS using Adjacency list = O(V+E)

4). Let d[v] = The time instant when the node u first visited. & f[v] = The time instant when the node last visited.

Note: If the finishing time of starting vertex is > the finishing time of remaining all vertices then the graph is said to connected w.r.t. starting vertex.

5). N(-)



Ex:

(A)

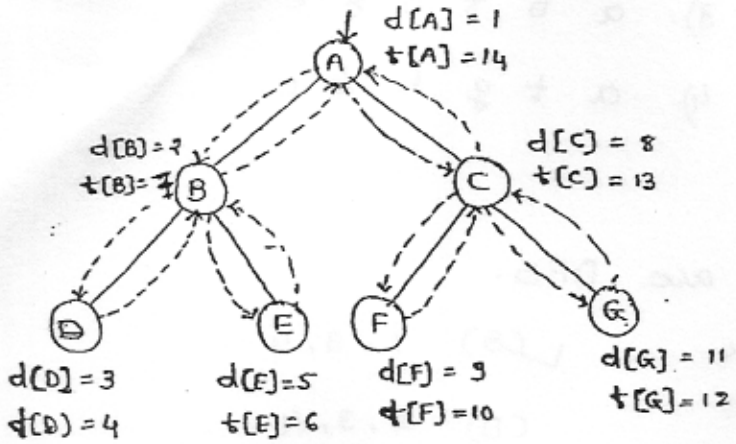
(B)

(C) ✓

(D)

2003
Ex: Con

5). No. of connected components in graph G(V, E)
= No. of DFS calls required on diff^n. vertices for processing all vertices of the graph.



d → Discover time
t → finish time

Ex: Consider DFS of undirected graph with 3 vertices P, Q & R. d[P] = 5 units, t[P] = 12 units
d[Q] = 6 units t[Q] = 10 units
d[R] = 14 units. t[R] = 18 units

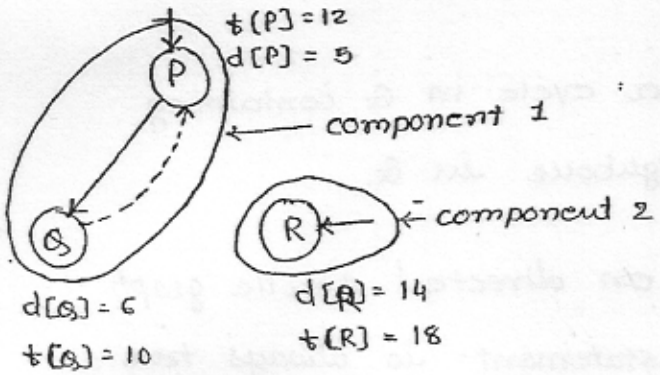
Which of the following is true.

(A) There is only 1 - connected component.

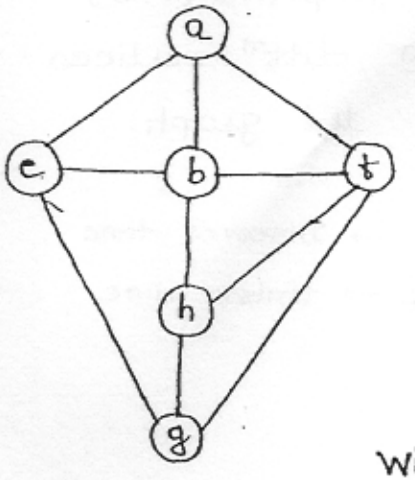
(B) There are 2 - connected components and 'P' & 'R' are connected.

(C) ----- 'P' & 'Q' are connected.

(D) ---- 'Q' & 'R' are connected.



2003
Ex: Consider the following graph.



1). a b e g h f

2). a b f e h g

3). a b f h g e

4). a f g h b e

Which are DFS.

(A) 1, 2, 4 ✓(B) 1, 3, 4

(C) 1, 4 (D) 2, 3, 4

2006

Ex: Let 'T' be a DFS in an undirected graph G. Vertices 'U' & 'V' leves of this tree. The degree of 'U' & 'V' are atleast 2. Which

✓(A) There must exists vertex 'W' adjacent to both 'U' & 'V' in G.

(B) There must exists a vertex 'W' whose removal disconnect 'U' & 'V'.

(C) There must exists a cycle in G containing 'U' & 'V'.

(D) There must exist a cycle in G containing 'U' & all its neighbour in G.

IT 2007

Ex: A DFS is performed on directed acyclic graph. Which of the following statement is always true for all edges (U, V) in the graph.

(A) d[U] < d[V] (B) d[U] > d[V]

(C) f[U] < f[V] (D) f[U] > f[V]

Ex: T

c

e

(P

Ex: Wh

tre

edges

Vertices

The a

have

do fo

∴ For

* A

A

remo

the g

Ex: The most efficient algo. for finding no. of connected component in an undirected graph on 'n' vertices & 'm' edges. takes the complexity.

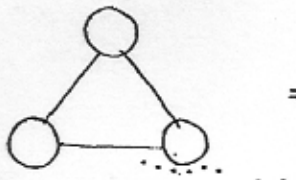
(A) $O(n)$ (B) $O(m)$ ✓(C) $O(n+m)$ (D) $O(nm)$

↓ DFS $\longrightarrow O(\underset{n+m}{V+E})$

Connected components

Ex: What is the largest integer 'm' such that every simple connected graph with 'n' vertices & 'n' edges contains atleast 'm' diff^{n}. spanning trees.

(A) 1 (B) 2 (C) 3 ✓(D) n



$\Rightarrow$ We get m diff^{n} spanning trees.

Here, $m = 3 = n$

edges $n = 3$

vertices $n = 3$

The above graph contains 3 edges & 3 vertices, we have to remove one edge of the graph, we can do this in three ways.

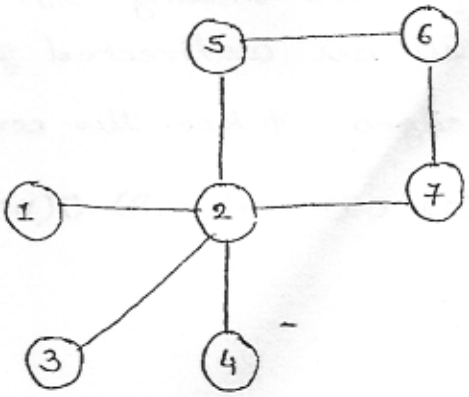
∴ For 'n' edges, we can do it in 'n' ways

∴ $\underline{m = n}$

* Articulation Point:

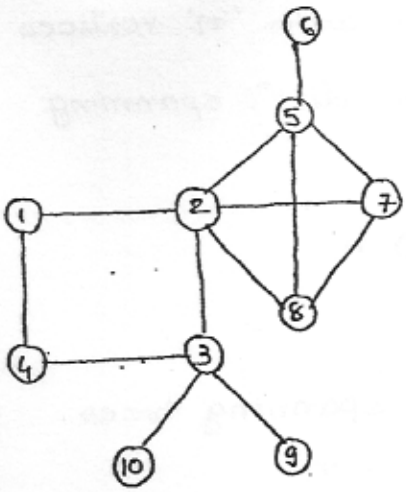
A point 'p' is called articulation point whose removal together all its edges will disconnect the graph into two or more nonempty components.

e.g.



⇒ Only 2 is articulation point.

Ex: Find articulation points:

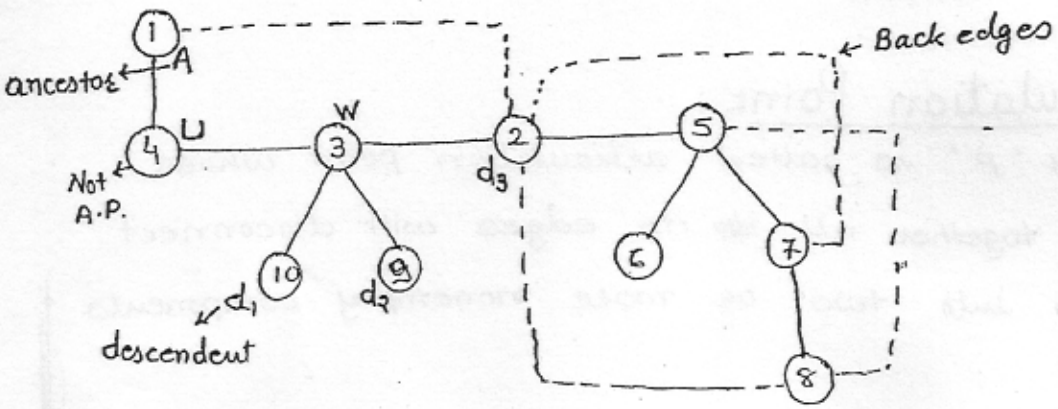


Ans: 2, 3 & 5 are articular points.

→ Bi-connected: It graph has no articular points then it is called as Bi-connected graph.

⇒ Find articulation point using DFS:

Consider above graph.



Only one connected component.

1). Root node of DFS tree not an articulation point if it contains only one child. So, node '1' is not an articulation point (A.P).

2). All leaf nodes of DFS tree are not an A.P. So, 6, 8, 9, 10 are not an A.P.

3). An internal node 'U' is said to be not an A.P. iff from every child 'W' of 'U' it is possible to reach an ancestor 'U' using the path made up of descendent of 'W' & back edges.

→ Back edges: Edges which are present in the graph but not present in DFS are called back edges. Shown by dotted lines.

→ Tree edges: Edges which are present in graph as well as in DFS tree are called tree edges. These are shown by thick lines.

Ex: In DFS of a graph with 'n' vertices 'k' edges are marked as Tree edges. The maximum no. of connected components.

(A) k (B) k+1 (C) n-k+1 (D) n-k

→ In the previous example n = 10 & tree edges = 9 & No. of connected component is = 1.

$\Rightarrow 10 - 9 = 1$

* Job Sequencing Problem:

Find maximum profit by processing below jobs.

Job	J_1	J_2	J_3	J_4
Dead lines	2	1	2	1
Profit	100	10	15	27

→ Let us consider n jobs ($J_1, J_2, \ldots J_n$)

→ Each job having deadline d_i & it we process the job within its deadline we

→ Only one job can be process at a time

→ only one CPU is avaible for processing all jobs.

→ CPU can take only one unit of time for processing any job.

→ All jobs arrived at same time.

→ Objective of Job sequencing (JS) is process as many job as possible within its deadline & generate maximum profit.

Shortcut: If there are n jobs, possible subsets are 2^n. Objective of J.S. is to find the subset which generates maximum profit.

- Arrange all jobs in decreasing order of their process their profit & then process in that order within its deadline.

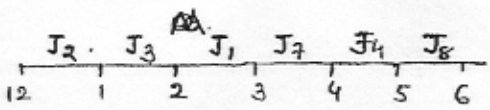
$J_1 \geq J_4 \geq J_3 \geq J_2$.

$\langle J_1, J_4 \rangle \Rightarrow 100 + 27 = 127$

Ex:

-	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8
deadline	6	5	6	6	3	4	4	5
Profit	10	8	9	12	3	6	11	13

$J_8 \geq J_4 \geq J_7 \geq J_1 \geq J_3 \geq J_2 \geq J_6 \geq J_5$



Max. Profit = 63

Ex: Linked que.

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

Q1: Are all tasks are completed?

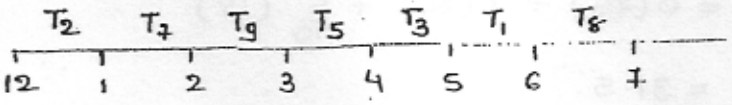
(A) All are completed (B) T_1 & T_6 are left out

(C) T_4 & T_6 are left out (D) T_1 & T_8 are left out.

$T_3 \geq T_9 \geq T_7 \geq T_2 \geq T_4 \geq T_5 \geq T_8 \geq T_1 \geq T_6$

Q2: What is the maximum profit?

(A) 144 (B) 147 (C) 150 (D) 152



* Analysis of Job Sequencing:

- To impliment J.S. we are use priority queue where priority are assigned to profits of job. so, the time complexity is $O(n \log n)$.

Ex: ~~Find maximum profit by placing~~

* Knap Sack Problem:

Find maximum profit by placing below objects into the knapsack.

objects	O_1	O_2	O_3
$(W_1\ W_2\ W_3)$	18	15	10
$(P_1\ P_2\ P_3)$	25	24	15

n=3 M = 20

Greedy about Weight $\Rightarrow$ Profit $= 0(25) + \frac{10}{15}(24) + 1(35)$

$= 31$

Greedy about profit $\Rightarrow$ Profit $= 1(25) + \frac{2}{15}(24) + 0(15)$

$= 28.5$.

Greedy about unit weight profit $\Rightarrow$

$$\frac{P_1}{W_1} = \frac{25}{18} = 1.3$$

$$\frac{P_2}{W_2} = \frac{24}{15} = 1.6$$

$$\frac{P_3}{W_3} = \frac{15}{10} = 1.5.$$

$\left.\begin{array}{l}\\\\\\\end{array}\right\}$ $\frac{P_2}{W_2} \geq \frac{P_3}{W_3} \geq \frac{P_1}{W_1}$

$\therefore$ Profit $= 0(25) + 1(24) + \frac{5}{10}(15)$

$= 31.5$

Shortcut: Arrange all unit weight profit into decreasing order & then process objects in that order without exceding knapsack size.

→ Since we are using priority queue, where priority are assign to unit weigh profit.

So, time complexity $= O(n \log n)$

Note: Let 'x' denote decision on i^{th} object then $0 \leq x_i \leq 1$ (Fractional knapsack problem).

* Optimal Merge Problem:

Let us consider 3 tiles F_1, F_2 & F_3 with their corresponding record length 30, 10 & 20 respectively.

Since

But

objec

to ti

move

Shortcu

of th

select

record

merg

→ H

F_1

(3

F_1 F_2

F_2 F_3

F_3 F_1

I_1

40

F_1

(30)

In (A)

In (B)

In (C)

. Since $n = 3$, we can arrange in $n! \Rightarrow 3!$

$\Rightarrow 6$ ways.

But mearging can be done in $\frac{n!}{2}$ ways $\Rightarrow \frac{3!}{2} = 3$ ways.

Objective of OMP is out of $\frac{n!}{2}$ way, we have to find the weight with least no. of record movement.

Shortcut: Arrange all tiles in the increasing order of their record length and in each iteration select two tiles which are having least no. of records and merge them into single tile.

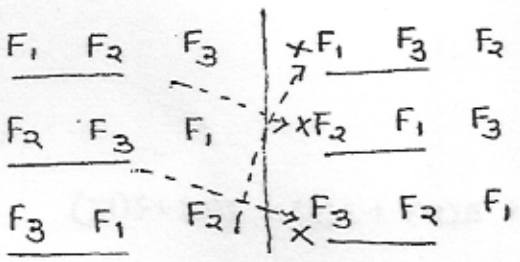
Continue the process until all tiles are merged.

→ Here, $n = 3$

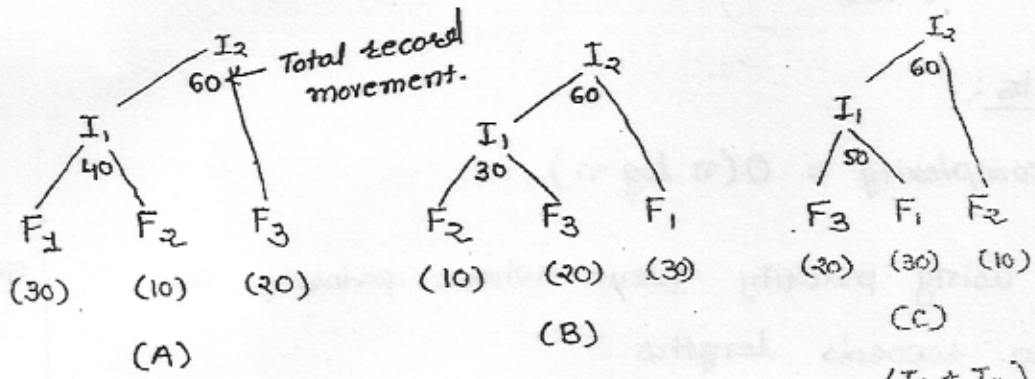
F_1 F_2 F_3

(30) (10) (20)

$3! = 6$ ways



$\Rightarrow \frac{n!}{2} = \frac{3!}{2} = 3$ ways.



In (A) Total no. of record movements = 100

In (B) " " " " " = 90

In (C) " " " " " = 110

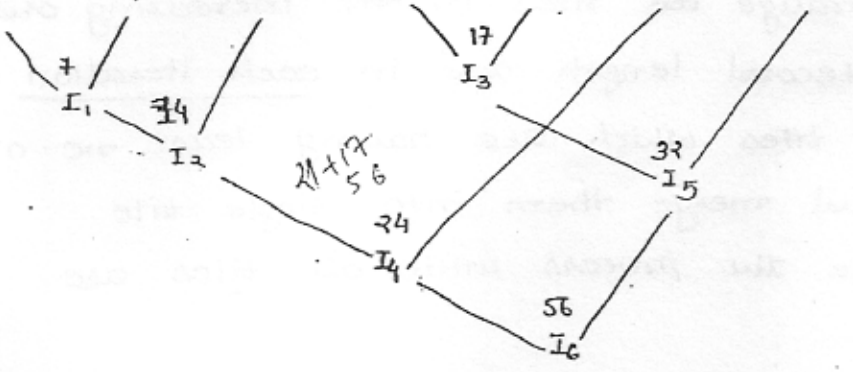
Ex: Find least no. of record movement.

F_1	F_2	F_3	F_4	F_5	F_6	F_7
4	8	15	3	7	9	10

(A) 150 (B) 171 (C) 169 (D) 170.

$F_4 \leq F_1 \leq F_5 \leq F_2 \leq F_6 \leq F_7 \leq F_3$

3 4 7 8 9 10 15



Note: Total no. of record movements = $\sum_{i=1}^{n} d_i q_i$

where, d_i = distance from root to i^{th} file.

q_i = Length of i^{th} file.

For above.

Total no. of record movements

$= 4(3) + 4(4) + 3(7) + 3(8) + 3(9) + 2(10) + 2(15)$

$= 150$

⇒ Analysis:

- Time complexity = $O(n \log n)$.

- We are using priority queue where priority are assign to records lengths.

Step-1:

* Huf

→ Obje
no.
whic
poin

→ Thes
- we
So,

* Huffman Encoding:

→ Objective of H.E. is encode letters with least no. of bits. Let us consider a Gmail application which contains letter a, b, c, d, e with corresponding frequency 10, 20, 4, 15, 6 respectively.

→ Therefor there are 5 letters to encode each letter we need atleast 3 - ~~letters~~ bits,

So, total no. of bits required = (10+20+4+5+6) x 3
= 165

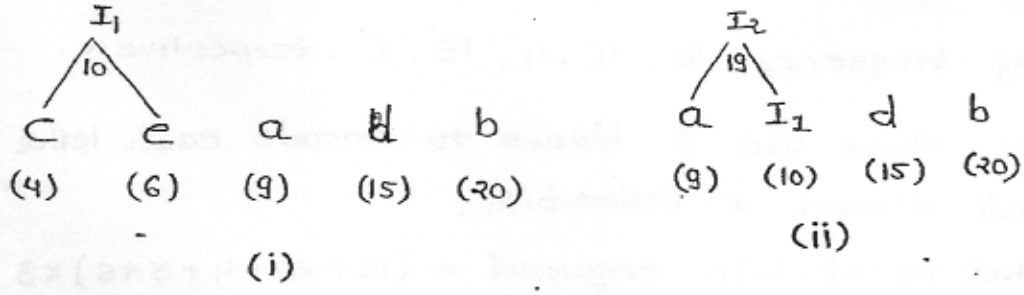
No. of bits.	No. of letters to be encoded.
1	0 – a 1 – b
2	00 – a 01 – d 10 – c 11 – b
3	000 – 001 – 010 0[illegible] – a 100 – e 101 – h 110 – c 111 – d

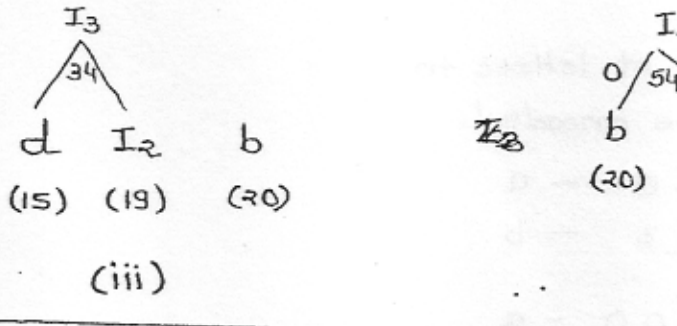
10)+2(15)

use

Step-1: Arrange all letters in the increasing order of their frequencies & then in each iteration construct binary tree by assigning 1st least frequency letter as left child & 2nd least freq^n. letter as a right child.

Step-2: After constructing binary tree from root to leaf path every left branch is assign with 0 & right branch is assign with 1.





	d_i	q_i	
a	1 1 0	= 3 x 9	= 27
b	0	= 1 x 20	= 20
c	1 1 1 0	= 4 x 4	= 16
d	1 0	= 2 x 15	= 30
e	1 1 1 1	= 4 x 6	= 24
			117 bits.

Note:

→ Total no. of bits required $= \sum_{i=1}^{n} d_i q_i$

Where, d_i = distance from root to i^{th} letter

q_i = freq^n. of i^{th} letter.

→ In

Eff

* Mul

For

I

9
7
3
2
1

It we

I

Since

minim

metho

mult

i.e.

Greedy

decisio

Dynamic

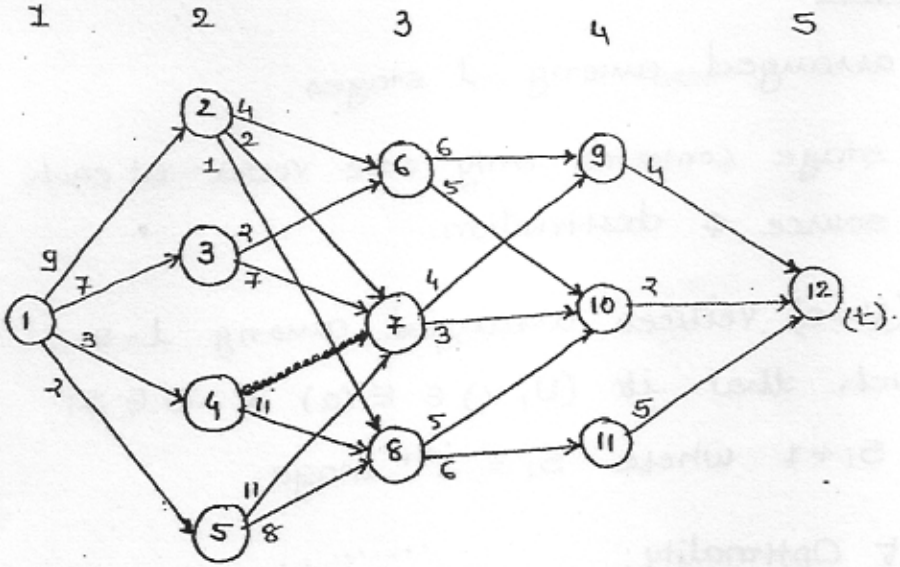
decisio

to → In H.E. more frequently occuring letter is
with encoded with least no. of bits.

Dynamic Programming.

* Multistage Graph:

Find shortest path distance cost from the vertex 1 to vertex 12.



If we apply greedy method we get path

1 - 5 - 8 - 10 - 12 = cost = 17.

Since these exists another path whose cost is minimum then the greedy method cost. So, greedy method fails to evaluate shortest path cost in multistage graph.

i.e. 1 - 2 - 7 - 10 - 12 = 16 ← least then greed.

Greedy Method: It always gives stage wise optimal decision which looks good at that movement only.

Dynamic Program: Its provide series of optimal decisions which look good for future purpose.

Multistage Graph:

~~If~~ 'V' vertices arranged among 'l' stages.

1st & last vertices stage contains only 1 vertex at each

Remaining (V-2) vertices arranged among l-2 stages such that

Multistage graph:

1. 'V' vertices arranged among l stages.
2. 1st & last stage contains only one vertex at each known as source & destination.
3. Remaining (V-2) vertices arranged among l-2 stages such that if $(U, V) \in E(a)$ & $U \in S_i$ then $V \in S_{i+1}$ where S_i = i^{th} stage.

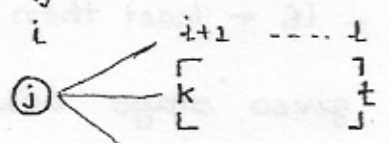
* Principle Of Optimality:

"Whatever initial state decision you are, remaining sequence of decision must consitute an optimal solution with regard to starting state decision."

$C(i,j)$ = Cost of an edge b/w 'i' and 'j'

$COST(i,j)$ = Minimum cost of path from vertex 'j' in stage 'i' to the destination 't'.

Stage ↵
Vertex ↵



$$COST(i,j) = \text{Min}\{C(j,k) + COST(i+1, k)\}$$

where, $\langle j,k \rangle \in E(G)$

A^0	1
1	0
2	6
3	∞

From the previous graph:

$$\text{cost}(1,1) = \min \begin{cases} c(1,2) + \text{cost}(2,2) \\ c(1,3) + \text{cost}(2,3) \\ c(1,4) + \text{cost}(2,4) \\ c(1,5) + \text{cost}(2,5) \end{cases} = \text{cost} = 16 \text{ (Minimum sum).}$$

$$\text{cost}(2,2) = \min \begin{cases} c(2,6) + \text{cost}(3,6) \\ c(2,7) + \text{cost}(3,7) \\ c(2,8) + \text{cost}(3,8) \end{cases} = \text{cost} = 7$$

$$\text{cost}(2,3) = \min \begin{cases} c(3,6) + \text{cost}(3,6) \\ c(3,7) + \text{cost}(3,7) \end{cases} = 9$$

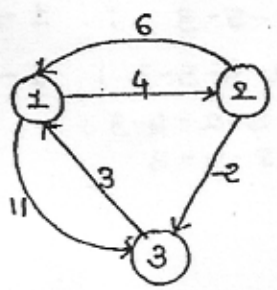
$$\text{cost}(2,4) = \min \{ c(4,8) + \text{cost}(3,8) \} = 18$$

$$\text{cost}(2,5) = \min \begin{cases} c(5,7) + \text{cost}(3,7) \\ c(5,8) + \text{cost}(3,8) \end{cases} = 15$$

→ Time complexity of multistage graph using adjacency list is $O(V+E)$.

* All pairs Shortest Path:

Find shortest path distance bet^n every pair of vertices.



A^0	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

A^1	1	2	3
1	0	4	11
2	6	0	2
3	3	7	0

A^2	1	2	3
1	0	4	6
2	6	0	2
3	3	7	0

A^3	1	2	3
1	0	4	6
2	5	0	2
3	3	7	0

$A^1(2,3) = \min\{2 \xrightarrow{6} 1 \xrightarrow{11} 3 \;,\; 2 \xrightarrow{2} 3\} = 2$

$A^1(3,2) = \min\{3 \xrightarrow{3} 1 \xrightarrow{4} 2 \;,\; 3 \xrightarrow{\infty} 2\} = 7$

$A^2(1,3) = \min\{1 \xrightarrow{4} 2 \xrightarrow{2} 3 \;,\; 1 \xrightarrow{11} 3\} = 6$

$A^2(3,1) = \min\{3 \xrightarrow{\infty} 2 \to 1 \;,\; 3 \xrightarrow{3} 1\} = 3$

$A^3(1,2) = \min\{1 \xrightarrow{11} 3 \xrightarrow{\infty} 2 \;,\; 1 \xrightarrow{4} 4\} = 4$

$A^3(2,1) = \min\{2 \xrightarrow{2} 3 \xrightarrow{3} 1 \;,\; 2 \xrightarrow{6} 6\} = 5$

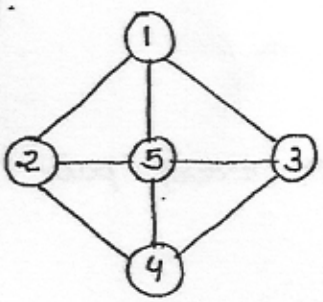
→ *Time Complexity :*

Since we are using $(n+1)$ vertices & each matrix contain n^2 elements. so time complexity is bounded by $O(n^3)$.

→ *Notation :*

$A^k(i,j)$ = shortest path distance bet^n. 'i' and 'j' in the graph whose highest intermediate vertex is 'k'.

e.g.



$$A^k(i,j) = \min\{A^{k-1}(i,k) + A^{k-1}(k,j),\; A^{k-1}(i,j)\}$$

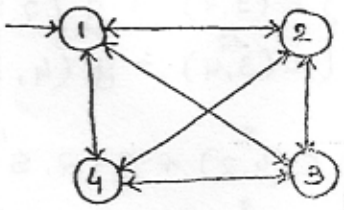
$$A^5(1,3) = \min \begin{cases} 1-5-3 \;: \\ 1-2-5-3 \;; & 1-2-4-3 \\ 1-2-4-5-3 \;; & 1-3 \\ 1-5-2-4-3 \;: \\ 1-5-4-3 \;, \end{cases}$$

* *Travelling Salesman (TSP) :*

Find minimum cost of tour from city ①, visiting all cities available in the graph and return to home town V_0.

e.g.

	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0



case-1: (1) ⟶ (visit) ⟶ (1)

case-2: (start from any city) — (visit) — (1)

→ If we apply greedy method we get path.

(1) — (2) — (3) — (4) — (1) = 39/-

Since there exists another path whose cost is minimum then greedy method cost that is.

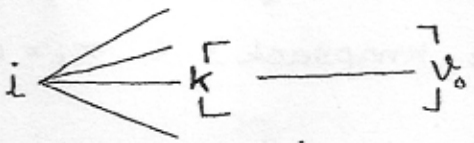
(1) — (2) — (4) — (3) — (1) = 35/-

So, greedy method fails to evaluate TSP.

→ **Notation:**

$g(i, S)$ = Min. cost of tour from city 'i' visiting all cities available in 'S' and return to starting (Home) city V_0

($i = V_0$ (case-1) & $i \neq V_0$ (case-2))



$$g(i, S) = \min_{\langle i,k \rangle \in E(G)} \{c(i,k) + g(k, S-\{k\})\}$$

e.g.

$$g(1, S=\{2,3,4\}) = \min \begin{cases} c(1,2) + g(2, S=\{3,4\}) \\ c(1,3) + g(3, S=\{2,4\}) \\ c(1,4) + g(4, S=\{2,3\}) \end{cases} = 35/-$$

($c(1,2)=10$, $g(2,S=\{3,4\})=25$; $c(1,3)=15$, $g(3,S=\{2,4\})=25$; $c(1,4)=20$, $g(4,S=\{2,3\})=23$)

$$g(2, S=\{3,4\}) = \min \begin{cases} c(2,3) + g(3, S=\{4\}) \\ c(2,4) + g(4, S=\{3\}) \end{cases} = 25/-$$

($c(2,3)=9$, $g(3,S=\{4\})=20$; $c(2,4)=10$, $g(4,S=\{3\})=15$)

$$g(3, S=\{2,4\}) = \min \begin{cases} c(3,2) + g(2, S=\{4\}) \\ c(3,4) + g(4, S=\{2\}) \end{cases} = 25/-$$

(13, 18, 12, 13)

$$g(4, S=\{2,3\}) = \min \begin{cases} c(4,2) + g(2, S=\{3\}) \\ c(4,3) + g(3, S=\{2\}) \end{cases} = 23/-$$

(8, 15, 9, 18)

3 — 2 — 1 (13, 5)

* 0/1 Knap Sack :

Find maximum profit by placing below objects into knapsack.

Objects	O_1	O_2	O_3
W_1 W_2 W_3	2	3	4
P_1 P_2 P_3	1	2	5

$M = 6$

→ Let us consider 'n' objects $O_1, O_2, \ldots O_n$ with their corresponding weight $w_1, w_2, \ldots w_n$ & profits $P_1, P_2, \ldots P_n$ respectively.

→ Let x_i denote decision on i^{th} object. then $x_i = 1$ (if we place it in to the knapsack) & $x_i = 0$ (if we ignore it).

→ Since each object takes '2' decisions, and there are 'n' objects,
So total no. of decisions are 2^n decision.

→ Time complexity 0/1 knapsack problem bounded by $O(2^{n/2})$ (approximatly).

→ Notation :

$t_n(M)$ = Maximum profit obtained by filling knapsack with the objects, which are

selected from n-objects.

$$t_n(M) = \max\{t_{n-1}(M) + 0,\ t_{n-1}(M - w_i) + P_i\}$$

Ex: The subset-sum problem is defined as follows. For given set of n (+ve integers) (object) $\{a_1, a_2, \ldots a_n\}$, Is there any subset whose element sum to 'M'? To solve this problem, Dynamic program uses a Two-dimensional boolean array x[i, j], where $1 \le i \le n$, $0 \le j \le W$.

x[i,j] is true iff there is a subset of $\{a_1, a_2, \ldots a_i\}$ whose element sum to 'j'.

Q1: Which of the following is true?

(A) $x[i,j] = x[i, j-a_i] \vee x[i-1, j]$

(B) $x[i,j] = x[i-1, j-a_i] \vee x[i-1, j-a_i]$

(C) $x[i,j] = x[i, j-a_i] \wedge x[i-1, j]$

(D) $x[i,j] = x[i-1, j-a_i] \wedge x[i-1, j-a_i]$

Q2: Which entry of the following is true implies that there is a subset whose element sum to W?

(A) $x[n, W]$ (object size, knapsack size) (B) $x[n-1, n]$

(C) $x[n, 0]$ (D) $x[1, W]$

Ans-1: $t_n(M) = \max\{t_{n-1}(M) + 0,\ t_{n-1}(M - w_i) + P_i\}$

$0 \in (\blacktriangle V)$

So, (C) & (D) is not correct.

Sorting Technique (Continue...)

* Bubble Sort:

Sort the following 90, 30, 10, 20, 70

	Before	After
i = 1 (Pass 1)		
j = 1	90 30 10 20 70	30 90 10 20 70
j = 2	30 90 10 20 70	30 10 90 20 70
j = 3	30 10 90 20 70	30 10 20 90 70
j = 4	30 10 20 90 70	30 10 20 70 90
i = 2 (Pass 2)		
j = 1	30 10 20 70	10 30 20 70
j = 2	10 30 20 70	10 20 30 70
j = 3	10 20 30 70	10 20 30 | 70
⋮	.....	.....

No. of iteration	no. of comparision	No. of elements sorted.
1	n-1	1
2	n-2	1
⋮	⋮	⋮
n-1	1	2
		n

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

$$\text{No. of comparision} = \frac{n(n-1)}{2}$$

$$= n^2 - n = n^2$$

$\frac{n^2}{2} \rightarrow$

Since

analy

No.

Tota

Time

* Sel

→ In

ele

* Se

→ In

sm

So

i=1

$\frac{n^2}{2}$ → if (a[j] > a[j+1])

```
{
  swap (a[j], a[j+1])
}
```

```
Swap (a[j], a[j+1])
{
  temp = a[j];
  a[j] = a[j+1];
  a[j+1] = temp;
}
```

(3)

Since $n^2/2$ comparisons. By using probabilistic analysis we can take $1/2$ (comparisions) are true.

No. of swapping = $1/2 (n^2/2)$

= $n^2/4$

Total time for swap = $3 * n^2/4$

Time complexity for bubble sort = Time reqd. for compar.
+ Time reqd for swap.

$= \frac{n^2}{2} + (3 * n^2/4)$

$= O(n^2)$

* Sort

→ In bubble sort each iteration replace largest element in its proper place.

* Selection Sort:

→ In selection sort in each iteration we place smallest element in its proper position.

Sort the following 90, 30, 10, 20, 70.

Before

i = 1 (pass 1)

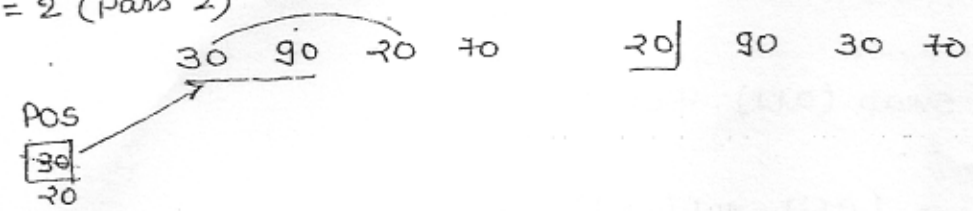
90 30 10 20 70

Pos
90
30
10

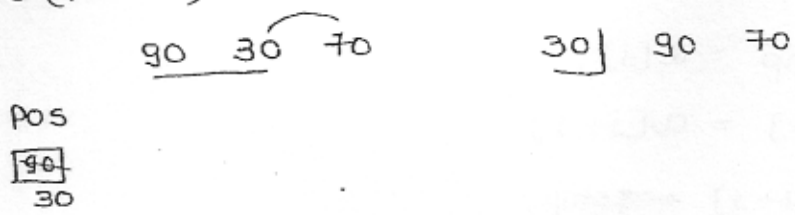
After.

10 | 30 90 20 70

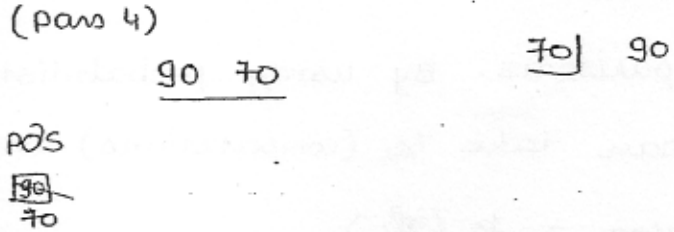
i = 2 (pass 2)



i = 3 (pass 3)



i = 4 (pass 4)



No. of iteration	No. of Comparision	No. of elements Sorted	No. of swaps
1	n - 1	1	1
2.	n - 2	1	1
⋮	⋮	⋮	⋮
n-1	1	2	1
	n(n-1)	n	n-1

No. of comparision = $\frac{n(n-1)}{2}$

$= \frac{n^2}{2} - \frac{n}{2} = \frac{n^2}{2}$

Time complexity of S.S. = Time req^d. for comparision + Time req^d. for swapping.

$= \frac{n^2}{2} + 3 * (n-1)$

$= O(n^2)$

* NP

In probler

* NP - Completeness

In theory of computation there are two types of problem. (i) P- class & (ii) NP - class.

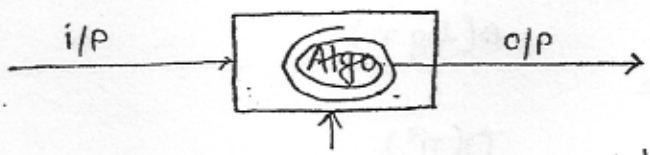
Algorithm	Time Complexity.
fact(n)	$O(n)$
fib(n)	$O(2^n)$
Hight of CBT	$O(\log_2 n)$
Binary Search	$O(\log_2 n)$
Seq^n. search	$O(n)$
Heap tree	$O(n \log n)$
Delete an element from heap	$O(\log n)$
Heap sort	$O(n \log n)$
Insert element into heap tree	$O(\log n)$
Bubble sort	$O(n^2)$
Selection sort	$O(n^2)$
Insertion sort	$O(n) \rightarrow O(n^2)$
Merge sort	$O(n \log n)$
Quick sort	$O(n \log n) \rightarrow O(n^2)$
Job sequencing, Knapsack, Huffman coding, Optimerg pattern	Priority Queue $O(n \log n)$
Prim's	$O(n^2)$ (using matrix)
	$O((V+E) \log V)$ (using heap)
Kruskals	$O(E \log E)$
DSSSP	$O(n^2)$ (using matrix)

DFS, BFS	$O(V+E)$	(∵ Adjacency list)
Finding connected component	$O(V+E)$	(∵ DFS)
Multistage graph	$O(V+E)$	(∵ Adjacency list)
All pairs shortest path	$O(n^3)$	(∵ (n+1) matrix)
0/1 knapsack	$O(2^{n/2})$	(approximatly)
TSP	$O(n^2 2^n)$	(approximatly.)

P-class:

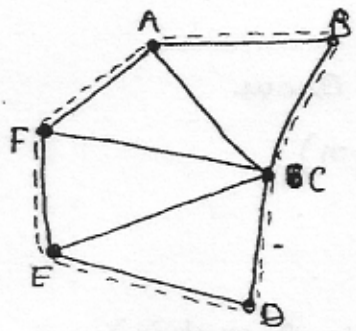
Problems which can be solved in polynomial time is called P-class problem.

Algorithm listed above accept 2, 26, 27 are comes under P-class problem.



→ **Hamiltonian Cycle:**

It hamiltonian cycle in a graph G is a cycle which spans all vertices of the graph.



→ **Hamiltonian Path:**

It is a path of length n-1 which spans all vertices of the graph. where n is no. of vertices.

NP-Cl
Po
using
NP-Cl
→ How
NP
1. Th
Ho
leng
2. Po
are
i.e.
→ Since
tim
Note:
prob
also
* Redu
Let
Problem
it is

$AB - BC - CD - DE - EF = \text{length}$
$= n - 1$

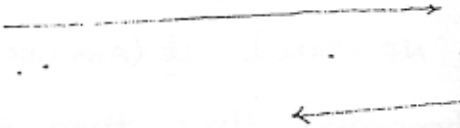
NP-class:

Problems which can be verify in polynomial time using non-deterministic tunning m/c are called NP-class problem

→ How do we prove wheather the problem belongs to NP-class or not.

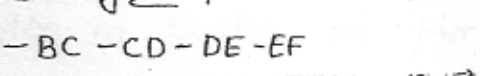
Prover — Verifier.

1. The above graph contain Hamiltonian path of length 5



How do you say?

2. Pass all edges which are forming H. path
i.e. AB − BC − CD − DE − EF



Now verifier varifies in $O(n-1) = O(n)$ and gives reply as 'YES'.

→ Since hamiltonian path problem verified in polynomial time. So, it is called as NP-class problem.

Note:

- problems which can be solved in polynomial time also verified in polynomial time. i.e. $P \subseteq NP$.

* Reducibility:

Let us consider two problems Π_1 & Π_2.

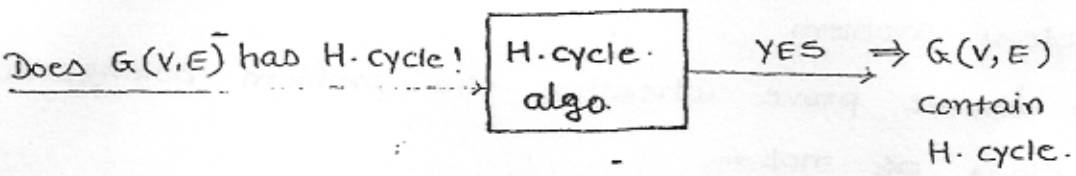
Problem Π_2 is said to be reducible to problem Π_1 if it is possible to solve Π_2 using algorithm of Π_1.

It is denoted by $\propto$.

e.g. $\pi_2 \propto \pi_1$

π_1 : Does G(V, E) has hamiltonian cycle !

π_2 : Does G(V, E) has hamiltonian path



Since H. path problem solved using algorithm of H. cycle, so we can say $\pi_2 \propto \pi_1$.

NP - Hard :

A problem π is called NP - hard if (Assume) it is possible to solve in polynomial time then every problem in NP is also solved in polynomial time.

or

Problems which can not be solved in polynomial times is called NP - hard problem.

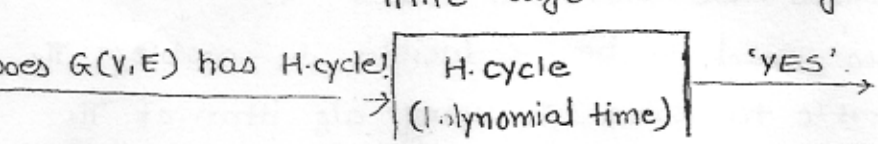
→ How do we prove that problem is NP-hard or not ?

Step-1: Assume that problem π can be solved in polynomial time.

Step-2: Using step - 1 if we can prove that there is a polynomial algo. for NP - hard problem then our assumption is false. So, π can not be solved in polynomial time ⇒ π is NP - hard.

e.g. Prove that H. cycle is NP - hard !

Step-1: Assume that there is a polynomial time algo for solving H. cycle.



'YES' means G(V,E) has hamiltonian cycle and it is solved in polynomial time.

Since we know that if G(V,E) has H cycle then it contains H. path.

∴ H. path problem solved in polynomial time. Which contradictory to that H. path can not be solved in polynomial time.

∴ Our assumption is wrong.

∴ There is no. polynomial time algo for H. cycle.

∴ H. cycle is NP – hard.

NP – Complete :

A problem 'π' is called NP-complete iff,

(i) $\pi \in NP$

(ii) π is NP hard.

Defn. (SAT)

SATISFIBILITY:-

(1) Let $x_1, x_2, \ldots x_n$ are boolean variables which takes either '0' or '1'

(2) Let $C_1, C_2, \ldots C_n$ are clauses which obtained by using literals x_i's

e.g. $C_1 = x_1 \vee x_2$

$C_2 = \bar{x_1} \vee x_2$

$C_3 = x_1 \vee \bar{x_2}$

(3) Let $F = C_1 \wedge C_2 \wedge C_3 \ldots \wedge C_n$ ---(*) is called boolean calculus formula. (CNF – Conjecutive normal formula)

Equation (*) is said to be in satisfibility iff there exists an assignment for each x_i's such that 'F' is true.

e.g. $F = C_1 \wedge C \wedge C_3$

$= (x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2)$

$= 1 \wedge 1 \wedge 1$

$= 1$ [∵ for $x_1 = 1$ & $x_2 = 1$ we have $F = 1$ So, F is satisfiability]

→ In 2-SAT problem each clause is obtain by using 2 literals & in 3-SAT problem each clause is obtain by using 3-literals.

→ In SAT problem each clause is obtain by using n-literals.

* Cook's Theorem:

SAT is NP-complete.

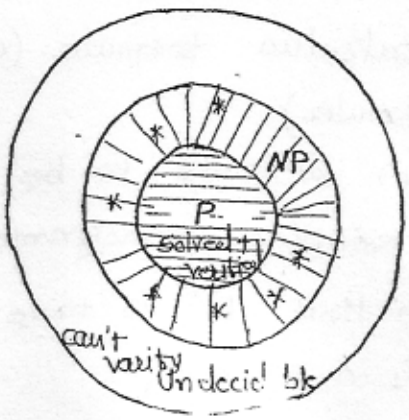
i.e. SAT ∈ NP and SAT is NP hard.

Proof:

1. Since we can verify SAT problem in polynomial time. So, SAT is in NP-class.
2. Since SAT problem each clause contains n-literals & each literal has two decisions then the total no of decision bounded by $O(2^n)$. So, it is NP-hard problem.

Note:

2-SAT problem can be solved in polynomial time so, it belongs to 'P' class. However, 3-SAT, 4 SAT & SAT problem can not be solved in polynomial time. so these are called as NP-hard problem.



Note:

→ Halting time in P Fun

(i) SAT (NP-H

(ii) Clique (NP

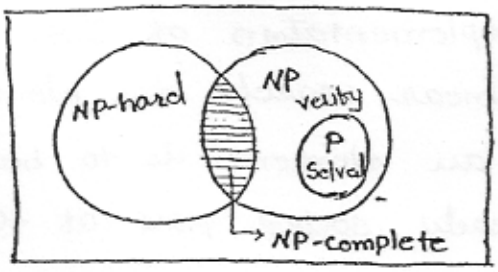
(iii) Indepe (NP

From

(i) NP

(ii) NO

2003 Ex: RAM ces tha pro sm to w



Note: Give an example for NP-hard problem which is not in NP

→ Halting problem can not be verified in polynomial time, so it is not in NP. Since it can be solved in polynomial time. So it is NP-hard.

Fundamentals of algo. by Horowitz – page 326. ↑

(i) SAT (NP-hard) $\leq_p$ Clique (Unknown) ⇒ Clique is NP-hard.

(ii) Clique (NP-hard) $\leq_p$ Independent set (Unknown) ⇒ Independent set is NP-hard

(iii) Independent set (NP-hard) $\leq_p$ Vertex Cover (Unknown) ⇒ Vertex cover is NP-hard.

From all the above we conclude that.

(i) NP-hard $\leq_p$ Unknown ⇒ Unknown is NP-hard.

(ii) NP-hard $\leq_p$ Unknown ⇒ NP complete.
(Unknown is in NP)

2003
Ex: RAM & SHYAM have been asked to show that a certain problem Π is NP complete. RAM shows that a polynomial time reduction from 3-SAT problem to Π.
SHYAM shows that a polynomial time reduction from Π to 3-SAT.
Which of the following is true?

2003
Ex: Usual $O(n^2)$ implementation of I.S. to sort an array uses linear search for identifying the position where an element is to be inserted into the already sorted part of the array. If instead we use binary search to identifying the position the worst case running time will be—

(A) Remains $O(n^2)$

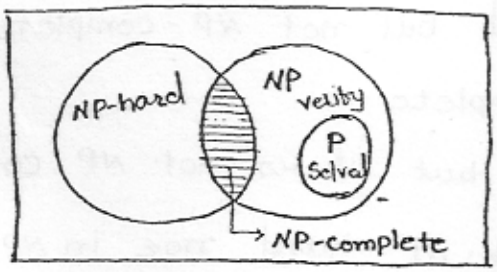
(B) Becomes $O(n \log n)$

(C) Becomes $O(n)$

(D) Becomes $O(n(\log n)^2)$

Seq^n. search $= O(n^2 + n)$

Binary search $= O(n^2 + \log n) = O(n^2)$



Note: Give an example for NP-hard problem which is not in NP

→ Halting problem can not be verified in polynomial time, so it is not in NP. Since it can be solved in polynomial time. So it is NP-hard.

Fundamentals of algo. by Horowitz – page 326. ↑

(i) SAT (NP-hard) $\leq_p$ Clique (Unknown) ⇒ Clique is NP-hard.

(ii) Clique (NP-hard) $\leq_p$ Independent set (Unknown) ⇒ Independent set is NP-hard

(iii) Independent set (NP-hard) $\leq_p$ Vertex Cover (Unknown) ⇒ Vertex cover is NP-hard.

From all the above we conclude that.

(i) NP-hard $\leq_p$ Unknown ⇒ Unknown is NP-hard.

(ii) NP-hard $\leq_p$ Unknown ⇒ NP complete.
(Unknown is in NP)

2003
Ex: RAM & SHYAM have been asked to show that a certain problem Π is NP complete. RAM shows that a polynomial time reduction from 3-SAT problem to Π.
SHYAM shows that a polynomial time reduction from Π to 3-SAT.
Which of the following is true?

(A) Π is NP-hard but not NP-complete.

(B) Π is NP-complete.

(C) Π is in NP but it is not NP complete.

(D) Π is neither NP-hard nor in NP.

→ RAM : $3\text{-SAT} \leq_p \Pi \Rightarrow \Pi$ is NP-hard.
(NP-hard) (Unknown)

SHYAM : $\Pi \leq_p 3\text{-SAT} \Rightarrow$ Wrong..
(Unknown) (NP-hard) [∵ NP-hard can't solve in P.T]

2006
Ex: Let 'S' be NP-complete problem. 'Q' & 'R' are two other problems not known to be in NP.

$Q \leq_p S$, $S \leq_p R$

Which of the following is true.

(A) R is NP-hard.

(B) R is NP-complete.

(C) Q is NP-hard.

(D) Q is NP-complete.

→ $Q \leq_p S$
(Unknown) (NP-complete)

$S \leq_p R \Rightarrow R$ is NP-complete.
(NP-complete) (Unknown)

2004
Ex: 2-SAT & 3-SAT problem.

Ans: P & NP-complete problem respectively

Ex: Let Π_A be a problem that belongs to class NP. Which of the following is true.

(A) There is no polynomial time algorithm for solving Π_A.

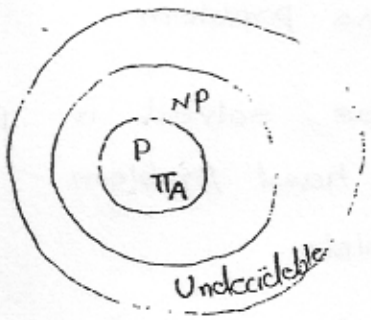
(B) If Π_A can be solved in deterministic...

polynomial time then P = NP.

(C) If π_A is NP-hard then it is NP-complete.

(D) π_A may be undecidable.

→



Ex: Which of the following is not NP-hard.

(A) Hamiltonian Cycle.

(B) 0/1 knapsack problem

(C) Graph coloring problem.

(D) Finding biconnected component.

No articulation point

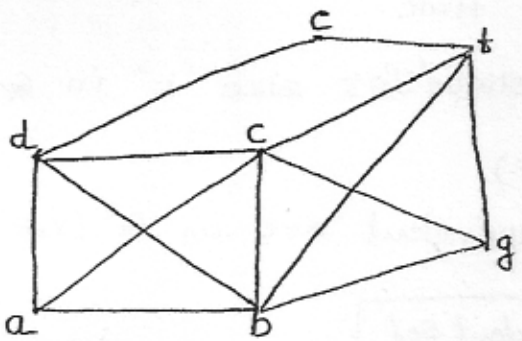
↓

Use DFS tree

Prove that independent set is NP-complete:

Defⁿ: (Clique):

A subset U of V in G (V, E) is said to be clique of size 'k' iff $\forall u_1, u_2 \in U$ there exists an edge in G(V, E).



$V = \{a, b, c, d, e, f, g\}$

$U = \{a, b, c, d\}$ → is a clique in size 4.

every vertex in U is connected.

Note:

1. Since clique problem verified in polynomial time, i.e. $O(k)$, where k is size of the clique.
   So, clique is NP class problem.

2. Since clique can not be solved in polynomial time, so it is called NP - hard problem.
   ∴ Clique is NP - complete.

* Independent Set:

A sub set U of V in $G(V, E)$ is said to be an independent set of size k iff $\forall u_1, u_2 \in U$ there does not exists an edge in $G(V, E)$.

Consider above graph.

$U = \{a, g, e\}$ → Independent of size 3.

Since independent set problem can be verified in in polynomial time i.e. $O(k)$, where k is size of independent set.
∴ So, it is belongs to NP - class.

Prove that Independent Set is NP - hard.

1. Assume that independent set problem can be solved in polynomial time.
2. Let us consider a clique 'U' of size 'k' in $G(V, E)$.
3. Now construct $G^c(V, E)$.
   i.e. 'U' is an independent set in $G^c(V, E)$.

Does $G^c(V, E)$ has a independent set U? → [Independent Set problem (Polynomial time)] → YES →

'YES' means independent set problem 'U' in G'(V,E) solved in polynomial time.

Clique 'U' in G(V,E) solved in polynomical time which is contradictory to clique is NP-hard.

∴ Our assumption is wrong.

∴ Independent set can not be solved in Polynomial time.

∴ Independent is NP-hard.

* Insertion Sort:

Sort the following element. 2, 18, 12, 14, 15, 3

Objective:

Insert an element among the sortest part of the array.

Best Case: If the array is already sorted, to insert an element it requires n comparisions.

Average Case: If the array is randomly distributed then to insert an element we have to perform to task.

Task-1: Arrange element in the increasing order using $O(n^2)$ time.

Task-2: Insert element amont sorted part of the array using $O(n)$ time.

$\therefore$ Total time $= O(n^2+n)$

$= O(n^2)$

Worst case: If the array elements are in the decreasing order, then it is called worst case.

To insert an element we have to perform to task which are described in average case.

* Merge Sort:

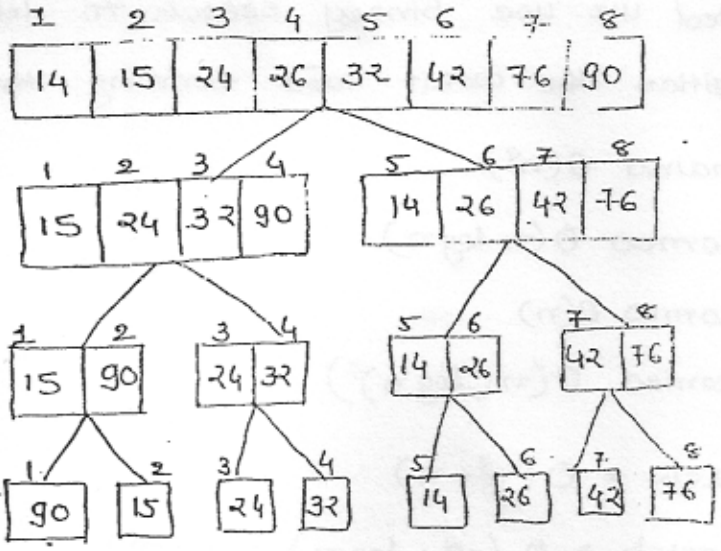
Sort the following array.

1	2	3	4	5	6	7	8
90	15	24	32	14	26	42	76

→ If the primary storage device (i.e. RAM) doesn't provide enough space to sort all element then

the array is splitted into two equal parts.
In such a case we use merge sort algorithm.

→ To implement merge sort we use auxiliary array of same size. So the space complexity $O(n)$.



$\Rightarrow h+1$

$\Rightarrow n(\log_2 n + 1)$

$= n \log_2 n + n$

$= O(n \log_2 n)$

$$T(n) = T(n/2) + T(n/2) + O(n)$$

↳ Time required for merging elements.

$$= 2T(n/2) + O(n)$$

$$= O(n \log n)$$

2007

Ex: Which of the following sorting algo. has lowest worst case time complexity.

(A) M.S $O(n \log n)$ (B) Q.S $O(n^2)$ (C) S.S $O(n^2)$ (D) B.S. $O(n^2)$

Ex: Which of the following sorting algo. needs min. no. of swapps.

(A) Q.S $n/2$ (B) S.S. 1 (C). I.S. (D). H.S. $\log n$

→ In I.S. there is no concept of swapping of data. Only movements of data is there.

2003
Ex: Usual $O(n^2)$ implementation of I.S. to sort an array uses linear search for identifying the position where an element is to be inserted into the already sorted part of the array. If instead we use binary search to identifying the position the worst case running time will be —

(A) Remains $O(n^2)$

(B) Becomes $O(n \log n)$

(C) Becomes $O(n)$

(D) Becomes $O(n (\log n)^2)$

Seq^n. search $= O(n^2 + n)$

Binary search $= O(n^2 + \log n) = O(n^2)$