Introduction to Algorithm

- In mathematics and computer science, an algorithm is a finite sequence of well-defined, computer-implementable instructions, typically to solve a class of problems or to perform a computation. Algorithms are unambiguous specifications for performing calculation, data processing, automated reasoning, and other tasks.
- As an effective method, an algorithm can be expressed within a finite amount of space and time, and in a well-defined formal language for calculating a function. Starting from an initial state and initial input, the instructions describe a computation that, when executed, proceeds through a finite number of well-defined successive states, eventually producing "output" and terminating at a final ending state.
- Will accept Zero or more input, but generate at least one output.

Algorithm Development Cycle

- Problem Definition: Understand Problem
- Constraints & Conditions: Understand constraints is any
- Design Strategies (Algorithmic Strategy)
- Express & Develop the algo
- Validation (Dry run)
- Analysis (Space and Time analysis)
- Coding
- Testing & Debugging
- Installation
- Maintenance

Need for Analysis

- We do analysis of algorithm to do a performance comparison between different algorithm to figure out which one is best possible option.
- Following are the parameters which can be considered while analysis of an algorithm
    - Time
    - Space
    - Bandwidth
    - Register
    - Battery power
- Out of all time is the most important Criteria for analysis of algorithm

Types of Analysis

- **Experimental or Apostrium analysis**: Means analysis of algorithm after it is converted to code. Implement both the algorithms and run the two programs on your computer for different inputs and see which one takes less time.
    - **Advantage:** Exact values no rough
    - **Disadvantage**: final result instead of depending only algorithm depends on many other factors like background software & hardware, programming language, even the temperature of the room.
- **Apriori Analysis or Independent analysis**: we do analysis using asymptomatic notations and mathematical tools of only algorithm, i.e. before converting it into program of a particular programming language. In Asymptotic Analysis, we evaluate the performance of an algorithm in terms of input size (we don't measure the actual running time). We calculate, how does the time (or space) taken by an algorithm increases with the input size. Asymptotic Analysis is not perfect, but that's the best way available for analyzing algorithms. Also, in Asymptotic analysis, we always talk about input sizes larger than a constant value. It might be possible that those large inputs are never given to your software and an algorithm which is asymptotically slower, always performs better for your particular situation. So, you may end up choosing an algorithm that is Asymptotically slower but faster for your software.
    - **Advantage**: Uniform result depends only on algorithm
    - **Disadvantage**: Estimated or approximate value no accurate and precise value.

Order of Magnitude

- Order of magnitude of a statement is number of times the fundamental operations involves in the statement.
- It is a two-step process
    - Find the number of fundamental operations
    - Finding the number of times, the fundamental operation executes.

Methods of Analysis

- **Worst Case Analysis**: In the worst-case analysis, we calculate upper bound on running time of an algorithm. We must know the case that causes maximum number of operations to be executed.
- It is that i/p class for which the algo takes maximum time.
- e.g. quick sort takes maximum time on a sorted i/p array.

- **Best Case Analysis**: In the best-case analysis, we calculate lower bound on running time of an algorithm. We must know the case that causes minimum number of operations to be executed.
- It is that i/p class for which the algo takes minimum time.
- Guaranteeing a lower bound on an algorithm doesn't provide any information specially in time analysis as in the worst case.

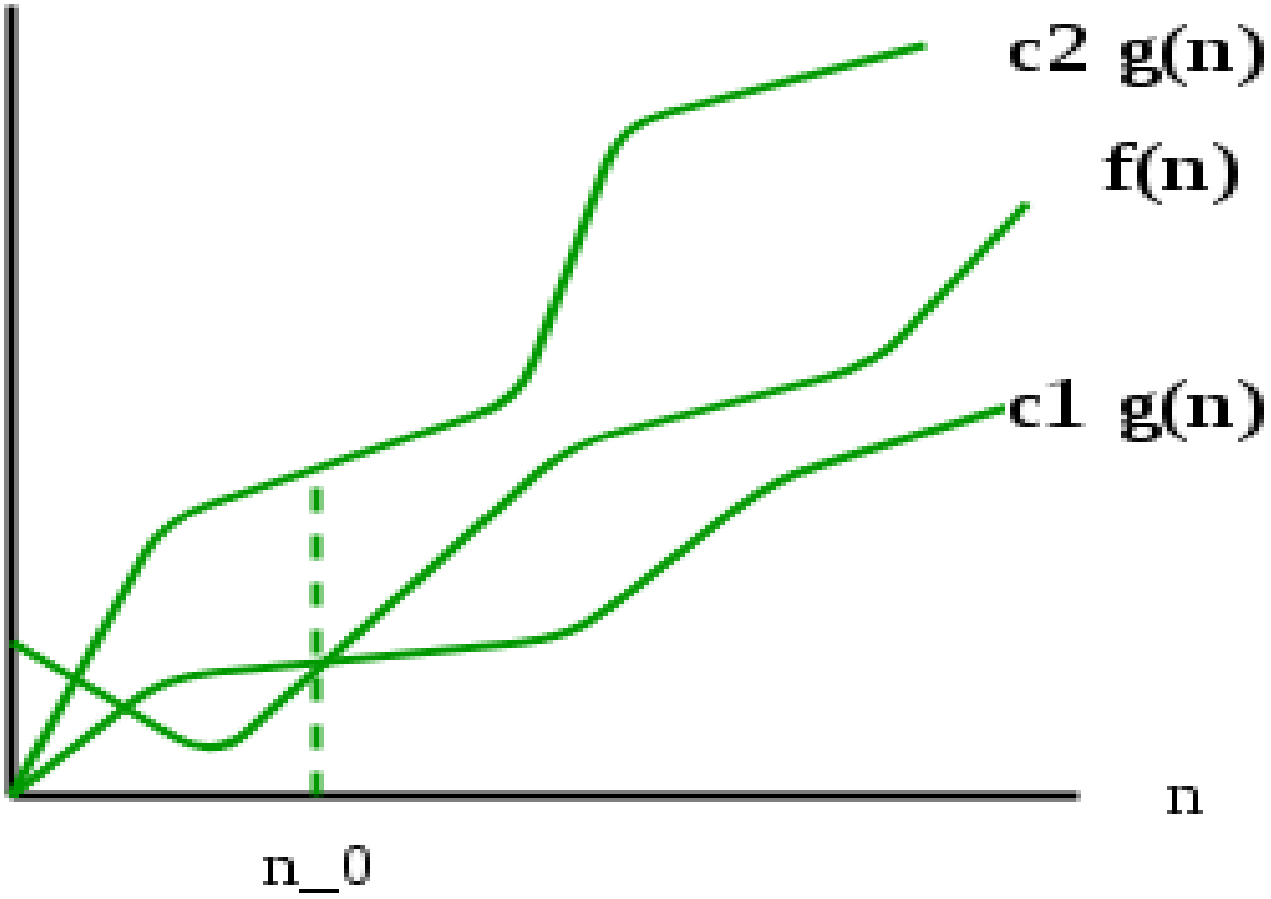
- **Average Case Analysis**: In average case analysis, we take all possible inputs and calculate computing time for all of the inputs. Sum all the calculated values and divide the sum by total number of inputs. We must know (or predict) distribution of cases.
- Identification of all i/p class I_1, I_2, I_3…I_k. Determine the probability that the algo take the i/p from the respective i/p class (P_1, P_2, P_3...P_k). Determine the time taken by the algo for each input class based on the order of magnitude (T_1, T_2, T_3, ….., T_k).
- A(n) = $\sum_{i=1}^{k} Pi * Ti$
- The average case analysis is not easy to do in most of the practical cases and it is rarely done. In the average case analysis, we must know (or predict) the mathematical distribution of all possible inputs.

Asymptotic Notations

- Asymptotic notations are Abstract notation for describing the behavior of algorithm and determine the rate of growth of a function.
- We have discussed Asymptotic Analysis, and Worst, Average and Best Cases of Algorithms. The main idea of asymptotic analysis is to have a measure of efficiency of algorithms that doesn't depend on machine specific constants, and doesn't require algorithms to be implemented and time taken by programs to be compared. Asymptotic notations are mathematical tools to represent time complexity of algorithms for asymptotic analysis. The following 3 asymptotic notations are mostly used to represent time complexity of algorithms.

Theta Notation

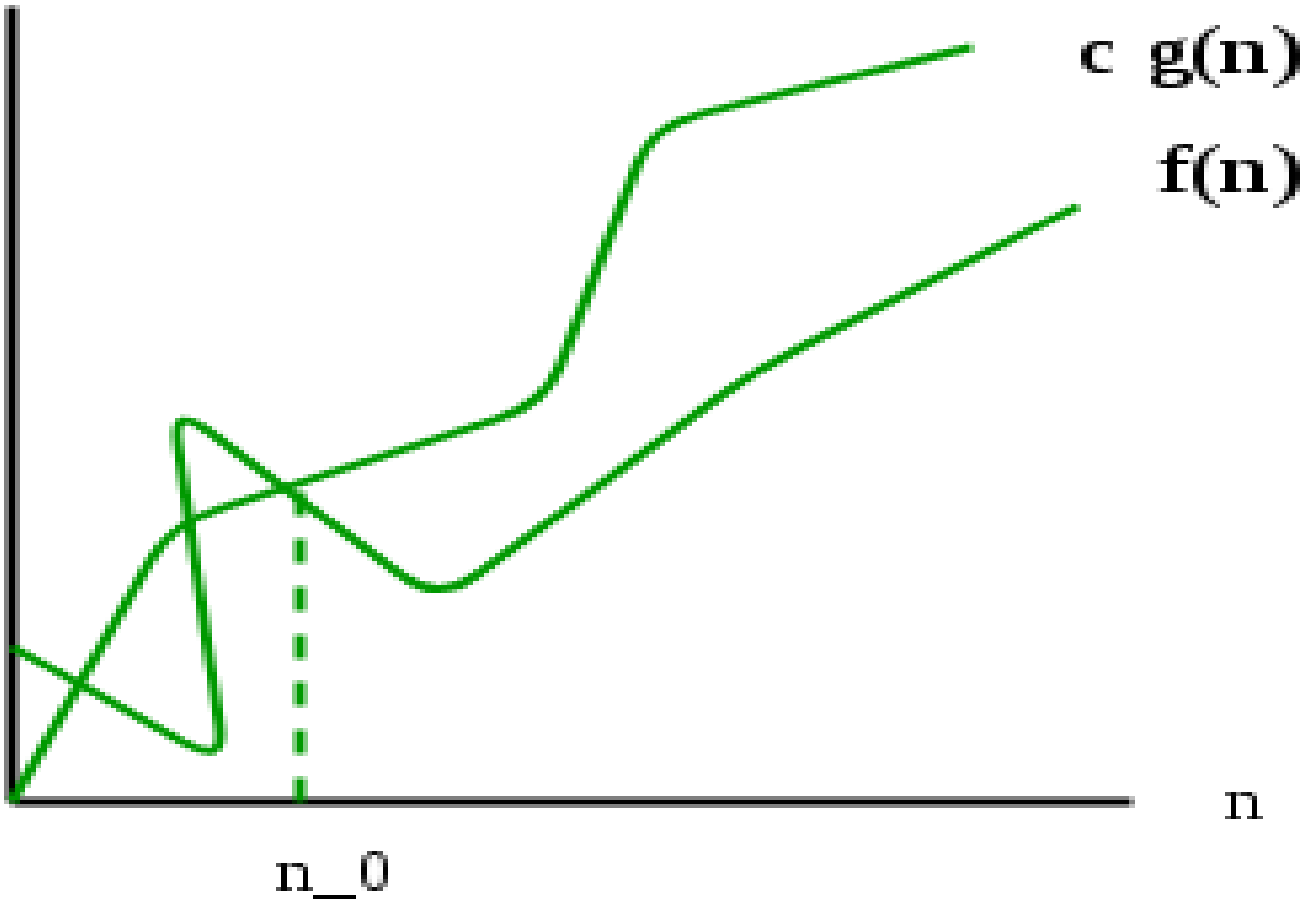
- **Θ Notation:** The theta notation bounds a function from above and below, so it defines exact asymptotic behaviour.
- For a given function g(n), we denote Θ(g(n)) is following set of functions.
- Θ(g(n)) = {f(n): there exist positive constants C_1, C_2 and n_0 such that $0 <= C_1*g(n) <= f(n) <= C_2*g(n)$ for all $n >= n_0$}
- The above definition means, if f(n) is theta of g(n), then the value f(n) is always between $C_1*g(n)$ and $C_2*g(n)$ for large values of n ($n >= n_0$).
- The definition of theta also requires that f(n) must be non-negative for values of n greater than n_0.



f(n) = theta(g(n))

Big O Notation

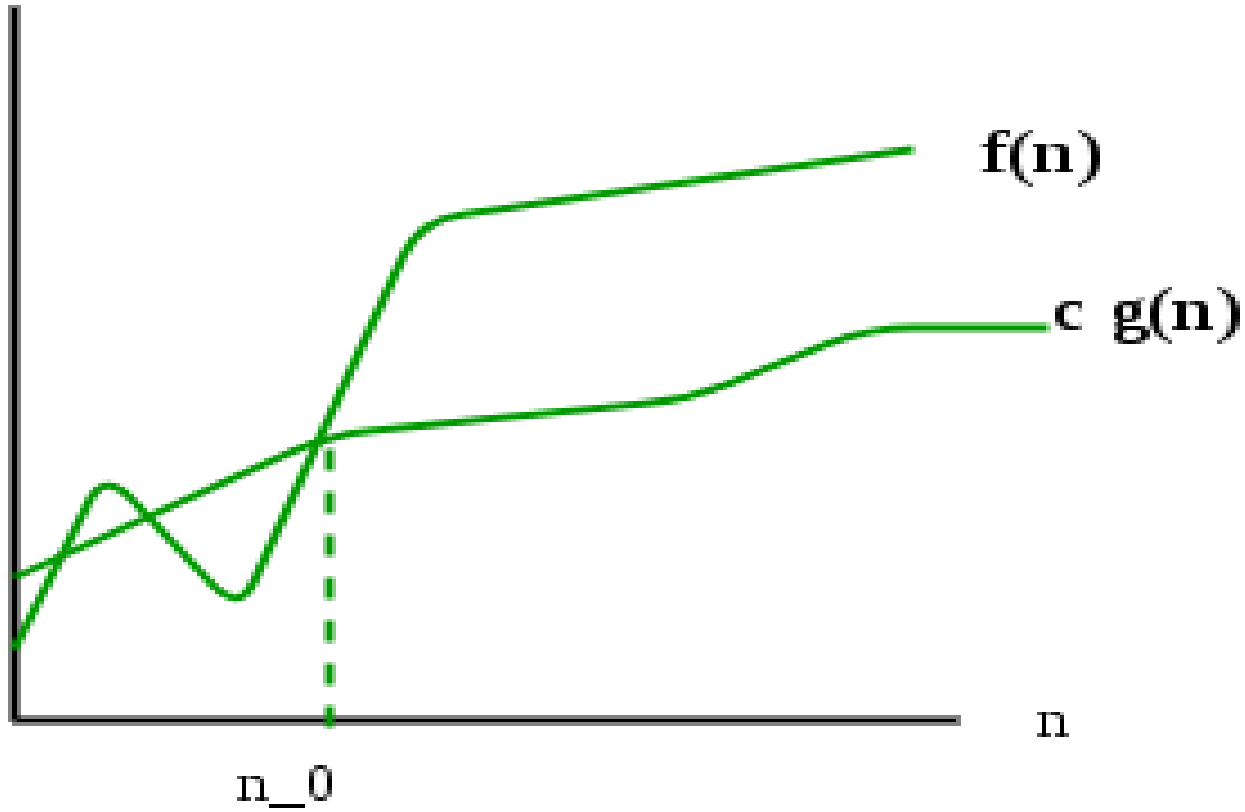
- The Big O notation defines an upper bound of an algorithm, it bounds a function only from above.
- The Big O notation is useful when we only have upper bound on time complexity of an algorithm.
- Many times, we easily find an upper bound by simply looking at the algorithm.
- $O(g(n)) = \{f(n)$: there exist positive constants C and N_0 such that $0 <= f(n) <= C*G(n)$ for all $N >= N_0\}$



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

Ω Notation

- Just as Big O notation provides an asymptotic upper bound on a function, Ω notation provides an asymptotic lower bound.
- Ω Notation can be useful when we have lower bound on time complexity of an algorithm.
- As discussed, the best-case performance of an algorithm is generally not useful in time analysis, the Omega notation is the least used notation among all three.
- For a given function g(n), we denote by Ω(g(n)) the set of functions.
- Ω (g(n)) = {f(n): there exist positive constants c and n_0 such that 0 <= c*g(n) <= f(n) for all n >= n_0}.



f(n) = Omega(g(n))

Small notations

- Everything is same as big notations just, just we take strictly increasing or monotonically increasing case and equal case is not allowed.
- Analogy of asymptomatic notation with real numbers
    - f(n) is O((g(n)) a <= b
    - f(n) is Ω (g(n)) a >= b
    - f(n) is Θ (g(n)) a = b
    - f(n) is o(g(n)) a < b
    - f(n) is ω(g(n)) a > b

Properties of Asymptotic notations

- Reflexivity:
    - f(n) = O(f(n))
    - f(n) = Ω(f(n))
    - f(n) = Θ(f(n))
- Symmetry:
    - f(n) = Θ(g(n)) if and only if g(n) = Θ(f(n))
- Transitivity:
    - f(n) = O(g(n)) and g(n) = O(h(n)) ⇒ f(n) = O(h(n))
    - f(n) = Θ(g(n)) and g(n) = Θ(h(n)) ⇒ f(n) = Θ(h(n))
    - f(n) = Ω(g(n)) and g(n) = Ω(h(n)) ⇒ f(n) = Ω(h(n))
    - f(n) = o(g(n)) and g(n) = o(h(n)) ⇒ f(n) = o(h(n))
    - f(n) = ω(g(n)) and g(n) = ω(h(n)) ⇒ f(n) = ω(h(n))
- Transpose Symmetry:

f(n) = O(g(n)) if and only if g(n) = Ω(f(n))

- if f(n) = O (g(n))
    - a f(n) = O(g(n))
- if f(n) is O(g(n)) and p(n) is O(q(n))
    - f(n) + p(n) is O(max (g(n), q(n)))
    - f(n) * p(n) is O(max (g(n) . q(n)))

Q Let w(n) and A(n) denote respectively, the worst case and average case running time of an algorithm executed on an input of size n. which of the following is ALWAYS TRUE? (Gate-2012) (1- Marks)

(A) A(n) = Ω(W(n)) (B) A(n) = Θ(W(n))

(C) A(n) = O(W(n)) (D) A(n) = o(W(n))

ANSWER C

Consider the following two functions:

$$g_1(n) = \begin{cases} n^3 \text{ for } 0 \leq n \leq 10,000 \\ n^2 \text{ for } n \geq 10,000 \end{cases}$$

$$g_2(n) = \begin{cases} n \text{ for } 0 \leq n \leq 100 \\ n^3 \text{ for } n > 100 \end{cases}$$

Which of the following is true?

A. $g_1(n)$ is $O(g_2(n))$

B. $g_1(n)$ is $O(n^3)$

C. $g_2(n)$ is $O(g_1(n))$

D. $g_2(n)$ is $O(n)$

(Gate-1994) (2 Marks)

Q Consider the following three claims
I $(n + k)^m = n^m$, where k and m are constants
II $2^{n+1} = 0(2^n)$
III $2^{2n+1} = 0(2^n)$
Which of these claims are correct? (GATE-2003) (1 Marks)
(A) I and II **(B)** I and III **(C)** II and III **(D)** I, II and III
Answer: (A)

Q Which of the following is not $O(n^2)$?
(A) $(15^{10}) * n + 12099$ **(B)** $n^{1.98}$ **(C)** $n^3 / (\text{sqrt}(n))$ **(D)** $(2^{20}) * n$
Answer: (C)

Q Let f(n) = n and $g(n) = n^{(1+\sin n)}$, where n is a positive integer. Which of the following statements is/are correct? (Gate-2015) (2 Marks)
I. f(n) = O(g(n)) II. f(n) = Ω(g(n))
(A) Only I **(B)** Only II **(C)** Both I and II **(D)** Neither I nor II
Answer: (D)

Which of the following is false?

A. $100n \log n = O(\frac{n \log n}{100})$

B. $\sqrt{\log n} = O(\log \log n)$

C. If $0 < x < y$ then $n^x = O(n^y)$

D. $2^n \neq O(nk)$

(Gate-1996) (1 Marks)
Ans: b, d

Q Two alternative packages A and B are available for processing a database having 10^k records. Package A requires $0.0001n^2$ time units and package B requires $10n\log_{10}n$ time units to process n records. What is the smallest value of k for which package B will be preferred over A? (Gate-2010) (1 Marks)

a) 12 b) 10 c) 6 d) 5

ANSWER 6

Q Let $f(n) = n^2 Logn$ and $g(n) = n (logn)^{10}$ be two positive functions of n. Which of the following statements is correct? (Gate-2001) (1 Marks)

(A) f(n) = O(g(n)) and g(n) != O(f(n)) (B) f(n) != O(g(n)) and g(n) = O(f(n))

(C) f(n) = O(g(n)) but g(n) = O(f(n)) (D) f(n) != O(g(n)) but g(n) != O(f(n))

Ans: b

Q Consider the following functions:

$f(n) = 2^n$ g(n) = n! $h(n) = n^{logn}$

Which of the following statements about the asymptotic behaviour of f(n), g(n), and h(n) is true? (Gate-2008) (2 Marks)

(A) f(n) = O(g(n)); g(n) = O(h(n)) (B) f(n) = (g(n)); g(n) = O(h(n))

(C) g(n) = O(f(n)); h(n) = O(f(n)) (D) h(n) = O(f(n)); g(n) = Ω (f(n))

Answer: (D)

Q Consider the following functions

$$f(n) = 3n^{\sqrt{n}}$$

$$g(n) = 2^{\sqrt{n}\log_2 n}$$

$$h(n) = n!$$

Which of the following is true? (GATE-2000) (2 Marks)
(a) h(n) is 0(f(n)) (b) h(n) is 0(g(n)) (c) g(n) is not 0(f(n)) (d) f(n) is 0(g(n))
Answer: (D)

Q Which of the given options provides the increasing order of asymptotic complexity of functions f1, f2, f3 and f4? (Gate-2011) (2 Marks)
$f_1(n) = 2^n$ $f_2(n) = n^{(3/2)}$ $f_3(n) = n\text{Log}n$ $f_4(n) = n^{(\text{Log}n)}$
(A) f3, f2, f4, f1 **(B)** f3, f2, f1, f4 **(C)** f2, f3, f1, f4 **(D)** f2, f3, f4, f1
Answer: (A)

Q The increasing order of following functions in terms of asymptotic complexity is (Gate-2015) (2 Marks)

$$\begin{aligned} f_1(n) &= n^{0.999999}\log n \\ f_2(n) &= 10000000n \\ f_3(n) &= 1.000001^n \\ f_4(n) &= n^2 \end{aligned}$$

(A) f1(n); f4(n); f2(n); f3(n) **(B)** f1(n); f2(n); f3(n); f4(n);
(C) f2(n); f1(n); f4(n); f3(n) **(D)** f1(n); f2(n); f4(n); f3(n)
Answer: (D)

Q Consider the following functions from positives integers to real numbers
10, $\sqrt{n}$, n, $\log_2 n$, 100/n.
The CORRECT arrangement of the above functions in increasing order of asymptotic

complexity is: (GATE-2017) (1 Marks)

(A) $\log_2 n$, 100/n, 10, √n, n

(B) 100/n, 10, $\log_2 n$, √n, n

(C) 10, 100/n ,√n, $\log_2 n$, n

(D) 100/n, $\log_2 n$, 10 ,√n, n

Answer: (B)

Q Arrange the following functions in increasing asymptotic order: (GATE-2008) (1 Marks)

a) $n^{1/3}$ b) e^n c) $n^{7/4}$ d) $n \log^9 n$ e) 1.0000001^n

(A) A, D, C, E, B

(B) D, A, C, E, B

(C) A, C, D, E, B

(D) A, C, D, B, E

Ans: a

Q Let f(n), g(n) and h(n) be functions defined for positive inter such that

f(n) = O(g(n)), g(n) ≠ O(f(n)), g(n) = O(h(n)), and h(n) = O(g(n)).

Which one of the following statements is FALSE? (Gate-2004) (2 Marks)

(A) f(n) + g(n) = O(h(n)) + h(n))

(B) f(n) = O(h(n))

(C) h(n) ≠ O(f(n))

(D) f(n)h(n) ≠ O(g(n)h(n))

Ans: d

Master Theorem

- In the analysis of algorithms, the master theorem for divide-and-conquer recurrences provides an asymptotic analysis (using Big O notation) for recurrence relations of types that occur in the analysis of many divide and conquer algorithms.
- The approach was first presented by Jon Bentley, Dorothea Haken, and James B. Saxe in 1980, where it was described as a "unifying method" for solving such recurrences. The name "master theorem" was popularized by the widely used algorithms textbook Introduction to Algorithms by Cormen, Leiserson, Rivest, and Stein.
- $T(n) = a\ T(n/b) + f(n)$
- The above equation divides the problem into 'a' number of subproblems recursively, a >= 1
- Each subproblem being of size n/b. the subproblems (of size less than k) that do not recurse. (b > 1)
- where f(n) is the time to create the subproblems and combine their results in the above procedure.
- The master theorem allows many recurrence relations of this form to be converted to Θ-notation directly, without doing an expansion of the recursive relation.
- The master theorem often yields asymptotically tight bounds to some recurrences from divide and conquer algorithms that partition an input into smaller subproblems of equal sizes, solve the subproblems recursively, and then combine the subproblem solutions to give a solution to the original problem.

Case 1

- $f(n) = O(n^{\log_b a} - \epsilon)$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$

Q $T(n) = 4T(n/2) + n$

Q $T(n) = 9T(n/3) + n$

Q $T(n) = 7T(n/2) + n^2$

Q $T(n) = 8T(n/2) + n^2$

Case 2

- $f(n) = \Theta (n^{\log_b a})$, then $T(n) = \Theta (n^{\log_b a} \lg n)$

Q $T(n) = T(2n/3) + 1$

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

Case 3

- $f(n) = \Omega (n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $a\, f(n/b) <= c\, f(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta (f(n))$

Q $T(n) = T(n/3) + n$

Q $T(n) = 3T(n/4) + n \lg n$

- Note that the three cases do not cover all the possibilities for f(n), and recurrence relations cannot be solved by master theorem.
- Q $T(n) = 2T(n/2) + n \lg n$

Q Which one of the following correctly determines the solution of the recurrence relation with T(1) = 1? (Gate-2014) (1 Marks)
T(n) = 2T(n/2) + Logn
a) Θ(n) b) Θ(nLogn) c) Θ(n*n) d) Θ(log n)
ANSWER A

Q Suppose T(n) = 2T (n/2) + n, T(0) = T(1) = 1
Which one of the following is FALSE? (Gate-2005) (2 Marks)
(A) $T(n) = O(n^2)$ **(B)** $T(n) = \theta(n \log n)$
(C) $T(n) = \Omega(n^2)$ **(D)** $T(n) = O(n \log n)$
Ans C

Q The recurrence equation
T(1) = 1
T(n) = 2T(n - 1) + n, n ≥ 2
evaluates to (Gate-2004) (2 Marks)
(A) $2^{n+1} - n - 2$ **(B)** $2^n - n$ **(C)** $2^{n+1} - 2n - 2$ **(D)** $2^n + n$
Answer: (A)

Q Solve the recurrence equations: (Gate-1987) (2 Marks)
T(n)=T(n−1) + n
T(1)=1

Q The solution to the recurrence equation T(2k) = 3 T(2k-1) + 1, T (1) = 1, is: (Gate-2002) (2 Marks)
(A) 2k **(B)** $(3^{k+1} - 1)/2$ **(C)** $3^{\log_2 k}$ **(D)** $2^{\log_3 k}$
Answer: (B)

Q The running time of an algorithm is represented by the following recurrence relation:

if n <= 3 then T(n) = n
else T(n) = T(n/3) + cn
Which one of the following represents the time complexity of the algorithm?
(A) (n) (B) (n log n) (C) (n^2) (D) (n^2log n)
Answer: (A)

Q Let T(n) be a function defined by the recurrence

T(n) = 2T(n/2) + √n for n ≥ 2 and T(1) = 1
Which of the following statements is TRUE? (Gate-2005) (2 Marks)
(A) T(n) = θ(log n) **(B)** T(n) = θ(√n)
(C) T(n) = θ(n) **(D)** T(n) = θ(n log n)
Ans: c

Q Consider the recurrence function

$$T(n) = \begin{cases} 2T(\sqrt{n}) + 1, & n > 2 \\ 2, & 0 < n \leq 2 \end{cases}$$

Then T(n) in terms of Θ notation is (Gate-2017) (2 Marks)

a) Θ(loglogn) b) Θ(logn) c) Θ(√n) d) Θ(n)

Ans: b

Q Consider the following recurrence:

T(n)=2T(√n)+1, T(1)=1

Which one of the following is true? (Gate-2006) (2 Marks)

a) T(n)=Θ(loglogn) b) T(n)=Θ(logn)

c) T(n)=Θ(√n) d) T(n)=Θ(n)

Ans: b

Q Consider the following recurrence relation

$T(1) = 1$

$T(n + 1) = T(n) + \lfloor \sqrt{n+1} \rfloor$ for all $n \geq 1$

The value of $T(m^2)$ for $m \geq 1$ is (Gate-2003) (2 Marks)

(A) $(m/6)(21m - 39) + 4$ **(B)** $(m/6)(4m^2 - 3m + 5)$

(C) $(m/2)(m^{2.5} - 11m + 20) - 5$ **(D)** $(m/6)(5m^3 - 34m^2 + 137m - 104) + (5/6)$

Answer: (B)

Q The minimum number of comparisons required to find the minimum and the maximum of 100 numbers is _____.(Gate-2014) (1 Marks)

(A) 147.1 to 148.1 **(B)** 145.1 to 146.1 **(C)** 140 to 146 **(D)** 140 to 148

Answer: (A)

Q An array of n numbers is given, where n is an even number. The maximum as well as the minimum of these n numbers needs to be determined. Which of the following is TRUE about the number of comparisons needed? (Gate-2007) (2 Marks)

a) At least 2n - c comparisons, for some constant c, are needed.

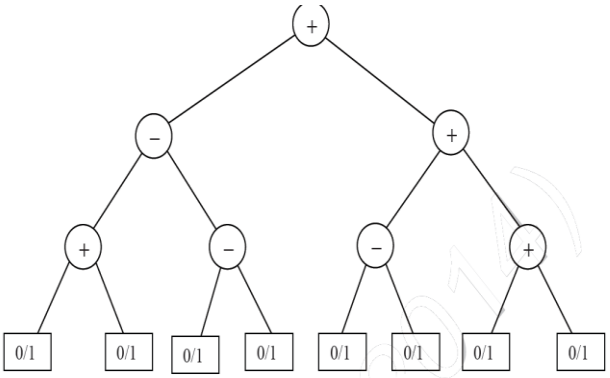
b) At most 1.5n - 2 comparisons are needed.

c) At least nlog2n comparisons are needed.

d) None of the above

Answer B

Q Consider the expression tree shown. Each leaf represents a numerical value, which can either be 0 or 1. Over all possible choices of the values at the leaves, the maximum possible value of the expression represented by the tree is ___. (Gate-2014) (2 Marks)



(A) 4 **(B)** 6 **(C)** 8 **(D)** 10

Answer: (B)

Q There are 25 horses among which you need to find out the fastest 3 horses. You can conduct race among at most 5 to find out their relative speed. At no point you can find out the actual speed of the horse in a race. Find out how many races are required to get the top 3 horses.

(A) 5 **(B)** 7 **(C)** 8 **(D)** 9

Answer: (B)

Explanation: Divide the horses in 5 groups and run 5 races. Take toppers of 5 races and run 6th race more race, the topper of this race will be the fastest horse among all 25. Now run the 7th race among following horses.

1) Second and third fastest horses of 6th race

1) Take the second and third fastest horses of the group which belong to the topper of 6th race.

2) Take the second fastest horse of the group which belongs to second fastest horse of 6th race.

The fastest and second fastest of 7th race are the 2nd and 3rd fastest among all 25.

Q An unordered list contains n distinct elements. The number of comparisons to find an element in this list that is neither maximum nor minimum is (Gate-2015) (1 Marks)

(A) Θ(nlogn) **(B)** Θ(n) **(C)** Θ(logn) **(D)** Θ(1)

Answer: (D)

Explanation: We only need to consider any 3 elements and compare them. So the number of comparisons is constants, that makes time complexity as Θ(1)

Q The minimum number of comparisons required to determine if an integer appears more than n/2 times in a sorted array of n integers is (Gate-2008) (2 Marks)

(A) Θ(n) **(B)** Θ(logn) **(C)** Θ(log*n) **(D)** Θ(1)
Answer: (B)

Q Consider a complete binary tree where the left and the right subtrees of the root are max-heaps. The lower bound for the number of operations to convert the tree to a heap is (Gate-2015) (1 Marks)
(A) Ω(logn) **(B)** Ω(n) **(C)** Ω(nlogn) **(D)** $\Omega(n^2)$
Answer: (A)

Q The recurrence relation capturing the optimal execution time of the Towers of Hanoi problem with n discs is (Gate-2012) (1 Marks)
(A) T(n) = 2T(n − 2) + 2 **(B)** T(n) = 2T(n − 1) + n
(C) T(n) = 2T(n/2) + 1 **(D)** T(n) = 2T(n − 1) + 1
Answer: (D)

Q In the worst case, the number of comparisons needed to search a single linked list of length n for a given element is (Gate-2002) (1 Marks)
a) logn b) n/2 c) $\log_2 n-1$ d) n

Q The concatenation of two lists is to be performed in O(1) time. Which of the following implementations of a list should be used? (Gate-1997) (1 Marks)
(A) singly linked list **(B)** doubly linked list
(C) circular doubly linked list **(D)** array implementation of lists
Answer: (C)

Q Suppose there are ⌈ log n ⌉ sorted lists of ⌊ n/log n ⌋ elements each. The time complexity of producing a sorted list of all these elements is (Gate-2005) (2 Marks)
(Hint : Use a heap data structure)
(A) O(n log log n) **(B)** θ(n log n) **(C)** Ω(n log n) **(D)** Ω(n3/2)
Answer: (A)

Q Let A be an array of 31 numbers consisting of a sequence of 0's followed by a sequence of 1's. The problem is to find the smallest index i such that A[i] is 1 by probing the minimum number of locations in A. The worst-case number of probes performed by an optimal algorithm is_____. (Gate-2017) (2 Marks)
Answer: (5)

Q Match the algorithms with their time complexities: (Gate-2017) (1 Marks)

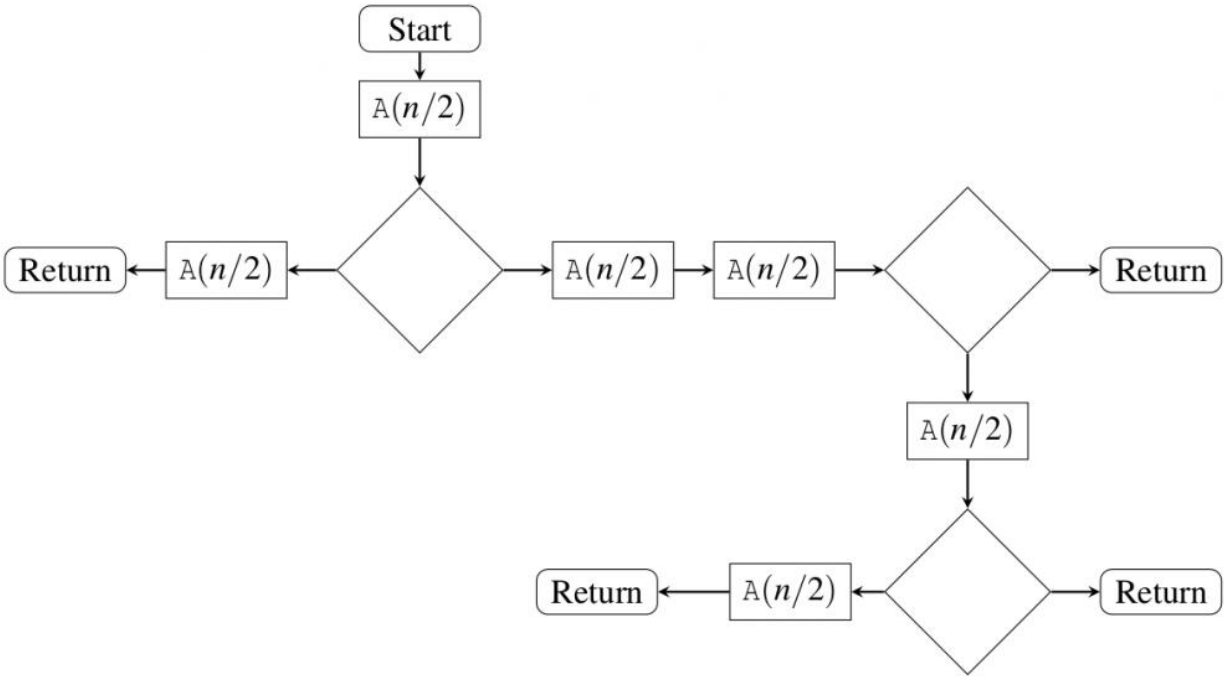
Algorithm	Time complexity
(P) Towers of Hanoi with n disks	(i) $\Theta(n^2)$
(Q) Binary search given n sorted numbers	(ii) $\Theta(n \log n)$
(R) Heap sort given n numbers at the worst case	(iii) $\Theta(2^n)$
(S) Addition of two $n \times n$ matrices	(iv) $\Theta(\log n)$

a) P-> (iii), Q -> (iv), R -> (i), S -> (ii)
b) P-> (iv), Q -> (iii), R -> (i), S -> (ii)
c) P-> (iii), Q -> (iv), R -> (ii), S -> (i)
d) P-> (iv), Q -> (iii), R -> (ii), S -> (i)

Q N items are stored in a sorted doubly linked list. For a delete operation, a pointer is provided to the record to be deleted. For a decrease-key operation, a pointer is provided to the record on which the operation is to be performed. An algorithm performs the following operations on the list in this order:
Θ(N) delete, O(log N) insert, O(log N) find, and Θ(N) decrease-key
What is the time complexity of all these operations put together (Gate-2016) (1 Marks)
(A) $O(Log^2N)$
(B) O(N)
(C) $O(N^2)$
(D) $\Theta(N^2 \text{ Log } N)$
Answer: (C)

Q The given diagram shows the flowchart for a recursive function A(n). Assume that all statements, except for the recursive calls, have O(1) time complexity. If the worst-case time complexity of this function is $O(n^{\alpha})$, then the least possible value (accurate up to two decimal positions) of α is _____ (Gate-2016) (2 Marks)

Flowchart for Recursive Function A(n)



Q An algorithm performs $(\log N)^{1/2}$ find operations, N insert operations, $(\log N)^{1/2}$ delete operations, and $(\log N)^{1/2}$ decrease-key operations on a set of data items with keys drawn from a linearly ordered set. For a delete operation, a pointer is provided to the record that must be deleted. For the decrease-key operation, a pointer is provided to the record that has its key decreased. Which one of the following data structures is the most suited for the algorithm to use, if the goal is to achieve the best total asymptotic complexity considering all the operations? (Gate-2015) (2 Marks)

(1) Unsorted array
(2) Min-heap
(3) Sorted array
(4) Sorted doubly linked list

ANSWER A

Q The time complexity of computing the transitive closure of a binary relation on a set of n elements is known to be (Gate-2005) (1 Marks)

(A) O (n) **(B)** O (n log n) **(C)** $O(n^{3/2})$ **(D)** $O(n^3)$

Answer: (D)

Q The tightest lower bound on the number of comparisons, in the worst case, for comparison-based sorting is of the order of (Gate-2004) (1 Marks)

(A) n **(B)** n^2 **(C)** n log n **(D)** $n \log^2 n$
Answer: (C)

Q The cube root of a natural number n is defined as the largest natural number m such that $m^3 \leq n$. The complexity of computing the cube root of n (n is represented in binary notation) is: (Gate-2003) (2 Marks)
(A) O(n) but not $O(n^{0.5})$
(B) $O(n^{0.5})$ but not $O((\log n)^k)$ for any constant $k > 0$
(C) $O((\log n)^k)$ for some constant $k > 0$, but not O $((\log \log n)^m)$ for any constant $m > 0$
(D) $O((\log \log n)^m)$ for some constant $k > 0.5$, but not $O((\log \log n)^{0.5})$
Answer: (C)

Q Consider the following algorithm for searching for a given number x in an unsorted array A[1……n] having n distinct values:

1. Choose an i uniformly at random from 1..... n;

2. If A[i] = x then Stop else Goto 1;

Assuming that x is present in A, what is the expected number of comparisons made by the algorithm before it terminates? (Gate-2002) (2 Marks)
(A) n **(B)** n – 1 **(C)** 2n **(D)** n/2
Answer: (A)

Q Given two arrays of numbers a1, a2, a3,…an and b1, b2, .. bn where each number is 0 or 1, the fastest algorithm to find the largest span(i, j) such that ai + ai+1, ….aj = bi, bi+1, .. bj. or report that there is not such span,(Gate-2006) (2 Marks)
(A) Takes $O(n^3)$ and $\Omega(2^n)$ time if hashing is permitted
(B) Takes $O(n^3)$ and $\Omega(n^{2.5})$ time in the key comparison model
(C) Takes $\theta(n)$ time and space
(D) Takes $O(\sqrt{n})$ time only if the sum of the 2n elements is an even number
Answer: (C)

Q The minimum number of arithmetic operations required to evaluate the polynomial $P(X) = X^5 + 4X^3 + 6X + 5$ for a given value of X using only one temporary variable.
(A) 6 **(B)** 7 **(C)** 8 **(D)** 9
Answer: (B)

Q Which of the following is the best possible time complexity to get Nth Fibonacci number with O(1) extra space

(A) Time complexity T(n) is T(n-1) + T(n-2) which is exponential **(B)** O(n)

(C) O(Logn) **(D)** O(n^2)

Answer: (C)

Explanation: The best possible time complexity is O(Logn).

Sorting

- **T**he p**ro**cess of arranging data items (numeric or char) in a specific order either increasing or decreasing order is called sorting. It is an important process which is used in a number of applications as many times we require sorted data.
- There are number of approaches available for sorting and some parameter based on which we judge the performance of these algorithm.
- Time complexity: - That is the time required to sort the entire given input sequence.
- Space complexity: - Internal sorting and external sorting
- Internal sorting means when a sorting algorithm does not require any additional memory apart from the space acquired by the problem.
- While on the other and some sorting algo requires additional space called external sorting algorithm.
- Stable or unstable: - When the input sequence contains copies then we check whether the output sequence preserve the order of the respective copies or not?
- Even studying all the criterion, we understand that the different sorting algo have different advantages in different scenarios therefor we must understand the actual logic of the algorithm and then can solve problems based on them.

Selection Sort

```
Selection sort (A, n)
{
      for k ←1 to n
      {
            min = A[k]
            Loc = k
            for j ←k+1 to n
            {
                  if(min > A[j])
                  {
                        min = A[j]
                        Loc = j
                  }
            }
            swap(A[k],A[Loc])
      }
}
```

Q Which one of the following is the tightest upper bound that represents the number of swaps required to sort n numbers using selection sort? (Gate-2013) (1 Marks)

A) O(log n) B) O(n) C) O(n log n) D) $O(n^2)$

Q What is the number of swaps required to sort n elements using selection sort, in the worst case? (Gate-2009) (1 Marks)

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

Bubble Sort

```
Bubble sort (A, n)
{
      for k ←1 to n-1
      {
            ptr = 1
            while(ptr <= n-k)
            {
                  if(A[ptr] > data[ptr+1])
                  {
                        swap(A[ptr],A[ptr+1])
                  }
                  ptr = ptr+1
            }
      }
}
```

Q What is the best time complexity of bubble sort?

(A) N^2 **(B)** NlogN **(C)** N **(D)** $N(\log N)^2$

Insertion Sort

```
Insertion sort (A, n)
{
    for j ←2 to n
    {
        key = A[j]
        i = j - 1
        while(i>0 and A[i] > key)
        {
            A[i+1] = A[i]
            i = i-1
        }
        A[i+1]=key
    }
}
```

Q What is the worst-case time complexity of insertion sort where position of the data to be inserted is calculated using binary search?

(A) N **(B)** NlogN **(C)** N^2 **(D)** N(logN)^2

Merge Sort

```
Merge_Sort(A, p, r)
{
        if(p < r)
        {
                q ← ⌊ (p + r)/2 ⌋
                Merge_Sort (A, p, q)
                Merge_Sort (A, q + 1, r)
                Merge_Sort (A, p, q, r)
        }
}

Merge (A, p, q, r)
{
        n1 ← q – p + 1
        n2 ← r – q
        Create array L [1…….. n1+1] and R [1…….. n2+1]
        for i ← 1 to n1
                do L[i] = A [p + i - 1]
        for j ← 1 to n2
                do R[j] = A [j + q]
        L [n1+1] ← ∞
        R [n2+1] ← ∞
        i ← 1
        j ← 1
        for k ← p to r
        {
                if(L[i] <= R[j])
                {
                        A[k] = L[i]
                        i = i + 1
                }
                Else
                {
                        A[k] = R[j]
                        j = j + 1
                }
        }
}
```

Q Assume that a Merge_Sort algorithm in the worst case takes 30 seconds for an input of size 64. Which of the following most closely approximates the maximum input size of a problem that can be solved in 6 minutes? (Gate-2015) (1 Marks)

a) 256 b) 512 c) 1024 d) 2048

Q A list of n string, each of length n, is sorted into lexicographic order using the merge-sort algorithm. The worst case running time of this computation is (Gate-2012) (1 Marks)

(A) O(nlogn) (B) $O(n^2 \log n)$ (C) $O(n^2 + \log n)$ (D) $O(n^2)$

Q Suppose P, Q, R, S, T are sorted sequences having lengths 20, 24, 30, 35, 50 respectively. They are to be merged into a single sequence by merging together two sequences at a time. The number of comparisons that will be needed in the worst case by the optimal algorithm for doing this is ____. (Gate-2014) (1 Marks)

Q In a modified merge sort, the input array is splitted at a position one-third of the length(N) of the array. What is the worst-case time complexity of this merge sort?

(A) $N(\log_3 N)$ **(B)** $N(\log_{2/3} N)$ **(C)** $N(\log_{1/3} N)$ **(D)** $N(\log_{3/2} N)$

Heap Sort

```
Heap_Sort(A)
{
        Build_Max_heap(A)
        for i ← length[A] down to 2
        {
                do exchange (A[1] ←→A[i])
                Heap-size[A] ← Heap-size[A] – 1
                Max-Heapify(A,1)
        }
}

Build_Max_Heap(A)
{
        Heap-size[A] ← length[A]
        for i ← ⌊ length[A]/2 ⌋ down to 1
        {
                do Max-Heapify (A, i)
        }
}

Max-Heapify(A, i)
{
        L ← Left[i]
        R ← Right[i]
        if( L <= Heap_size[A] and A[L] > A[i])
                Largest ← L
        Else
                Largest ← i
        if(R <= Heap_size[A] and A[r] > A[Largest])
                Lergest ← R
        if(Largest != i)
        {
                Exchange( A[i] ←→A[Largest])
                Max-Heapify(A, Lergest)
        }
}
```

Q An operator delete(i) for a binary heap data structure is to be designed to delete the item in the i-th node. Assume that the heap is implemented in an array and i refers to the i-th index of the array. If the heap tree has depth d (number of edges on the path from the root to the farthest leaf), then what is the time complexity to re-fix the heap efficiently after the removal of the element? (Gate-2016) (1 Marks)

a) O(1)
b) O(d) but not O(1)
c) O(2d) but not O(d)
d) O(d2d) but not O(2d)

Q Consider a complete binary tree where the left and the right subtrees of the root are max-heaps. The lower bound for the number of operations to convert the tree to a heap is **(GATE-2015) (1 Marks)**

a) Ω(logn) **b)** Ω(n) **c)** Ω(nlogn) **d)** Ω(n2)

Q Consider the process of inserting an element into a Max Heap, where the Max Heap is represented by an array. Suppose we perform a binary search on the path from the new leaf to the root to find the position for the newly inserted element, the number of comparisons performed is: (Gate-2007) (1 Marks)

a) $\theta(\log_2 n)$ b) $\theta(\log_2 \log_2 n)$ c) $\theta(n)$ d) $\theta(n \log_2 n)$

Q Consider the following statements: (Gate-2019) (1 Marks)

- **I.** The smallest element in a max-heap is always at a leaf node.
- **II.** The second largest element in a max-heap is always a child of the root node.
- **III.** A max-heap can be constructed from a binary search tree in Θ(n) time.
- **IV.** A binary search tree can be constructed from a max-heap in Θ(n) time.

Which of the above statements is/are TRUE?

(A) II, III and IV **(B)** I, II and III **(C)** I, III and IV **(D)** I, II and IV

Q Which of the following is true about merge sort?

(A) Merge Sort works better than quick sort if data is accessed from slow sequential memory.

(B) Merge Sort is stable sort by nature

(C) Merge sort outperforms heap sort in most of the practical situations.

(D) All of the above.

Q An array of integers of size n can be converted into a heap by adjusting the heaps rooted at each internal node of the complete binary tree starting at the node $\lfloor (n-1)/2 \rfloor$, and doing this adjustment up to the root node (root node is at index 0) in the order $\lfloor (n-1)/2 \rfloor$, $\lfloor (n-3)/2 \rfloor$, ….., 0. The time required to construct a heap in this manner is (Gate-2014) (1 Marks)

(A) O(log n) **(B)** O(n) **(C)** O (n log log n) **(D)** O(n log n)

Quick Sort

```
Quick_Sort(A, p, r)
{
        if(p < r)
        {
                q ← partition (A, p, r)
                quick_Sort(A, p, q - 1)
                quick_Sort(A, q + 1, r)
        }
}

Partition (A, p, r)
{
        x ← A[r]
        i ← p – 1
        for j ← p to r – 1
        {
                if(A[j] <= x)
                {
                        i ← i + 1
                        Exchange( A[i] ←→A[j])
                }
        }
        Exchange( A[i + 1] ←→A[r])
        return i+1
}
```

Q You have an array of n elements. Suppose you implement quicksort by always choosing the central element of the array as the pivot. Then the tightest upper bound for the worst-case performance is (Gate-2014) (1 Marks)

a) $O(n^2)$ b) O(nLogn) c) Theta(nLogn) d) $O(n^3)$

Q An array of 25 distinct elements is to be sorted using quicksort. Assume that the pivot element is chosen uniformly at random. The probability that the pivot element gets placed in the worst possible location in the first round of partitioning (rounded off to 2 decimal places) is _____. (Gate-2019) (1 Marks)

Q Suppose we are sorting an array of eight integers using quicksort, and we have just finished the first partitioning with the array looking like this:
2 5 1 7 9 12 11 10
Which statement is correct?
(A) The pivot could be either the 7 or the 9
(B) The pivot could be the 7, but it is not the 9
(C) The pivot is not the 7, but it could be the 9
(D) Neither the 7 nor the 9 is the pivot.

Q What is recurrence for worst case of QuickSort and what is the time complexity in Worst case?
(A) Recurrence is T(n) = T(n-2) + O(n) and time complexity is O(n^2)
(B) Recurrence is T(n) = T(n-1) + O(n) and time complexity is O(n^2)
(C) Recurrence is T(n) = 2T(n/2) + O(n) and time complexity is O(nLogn)
(D) Recurrence is T(n) = T(n/10) + T(9n/10) + O(n) and time complexity is O(nLogn)

Q Suppose we have a O(n) time algorithm that finds median of an unsorted array.
Now consider a QuickSort implementation where we first find median using the above algorithm, then use median as pivot. What will be the worst case time complexity of this modified QuickSort.
(A) O(n^2 Logn) **(B)** O(n^2) **(C)** O(n Logn Logn) **(D)** O(nLogn)

Q In quick sort, for sorting n elements, the (n/4)th smallest element is selected as pivot using an O(n) time algorithm. What is the worst-case time complexity of the quick sort? (Gate-2009) (1 Marks)
(A) (n) (B) (nLogn) (C) (n^2) (D) (n^2 log n)

Q Consider the Quicksort algorithm. Suppose there is a procedure for finding a pivot element which splits the list into two sub-lists each of which contains at least one-fifth of the elements. Let T(n) be the number of comparisons required to sort n elements. Then (Gate-2008) (2 Marks)
(A) T(n) <= 2T(n/5) + n
(B) T(n) <= T(n/5) + T(4n/5) + n
(C) T(n) <= 2T(4n/5) + n
(D) T(n) <= 2T(n/2) + n

Q Which one of the following is the recurrence equation for the worst case time complexity of the Quicksort algorithm for sorting n(≥ 2) numbers? In the recurrence equations given in the options below, c is a constant. (Gate-2015) (1 Marks)
(1) T(n) = 2T(n/2) + cn
(2) T(n) = T(n – 1) + T(1) + cn
(3) T(n) = 2T(n – 1) + cn
(4) T(n) = T(n/2) + cn

Q The worst-case running times of Insertion sort, Merge sort and Quick sort, respectively, are: (Gate-2016) (1 Marks)

a) $\Theta(n \log n)$, $\Theta(n \log n)$ and $\Theta(n^2)$
b) $\Theta(n^2)$, $\Theta(n^2)$ and $\Theta(n \text{ Log } n)$
c) $\Theta(n^2)$, $\Theta(n \log n)$ and $\Theta(n \log n)$
d) $\Theta(n^2)$, $\Theta(n \log n)$ and $\Theta(n^2)$

Q Assume that the algorithms considered here sort the input sequences in ascending order. If the input is already in ascending order, which of the following are TRUE? (Gate-2016) (1 Marks)

I. Quicksort runs in $\Theta(n^2)$ time
II. Bubble sort runs in $\Theta(n^2)$ time
III. Merge_Sort runs in $\Theta(n)$ time
IV. Insertion sort runs in $\Theta(n)$ time

a) I and II only b) I and III only c) II and IV only d) I and IV only

Q Which of the following sorting algorithms has the lowest worst-case complexity?

(A) Merge Sort **(B)** Bubble Sort **(C)** Quick Sort **(D)** Selection Sort

Q Which of the following is not a stable sorting algorithm in its typical implementation.

(A) Insertion Sort **(B)** Merge Sort **(C)** Quick Sort **(D)** Bubble Sort

Q Which of the following sorting algorithms in its typical implementation gives best performance when applied on an array which is sorted or almost sorted (maximum 1 or two elements are misplaced).

(A) Quick Sort **(B)** Heap Sort **(C)** Merge Sort **(D)** Insertion Sort

Q Consider a situation where swap operation is very costly. Which of the following sorting algorithms should be preferred so that the number of swap operations are minimized in general?

(A) Heap Sort **(B)** Selection Sort **(C)** Insertion Sort **(D)** Merge Sort

Q You have to sort 1 GB of data with only 100 MB of available main memory. Which sorting technique will be most appropriate?

(A) Heap sort **(B)** Merge sort **(C)** Quick sort **(D)** Insertion sort

Q Which sorting algorithm will take least time when all elements of input array are identical? Consider typical implementations of sorting algorithms.

(A) Insertion Sort **(B)** Heap Sort **(C)** Merge Sort **(D)** Selection Sort

Q An element in an array X is called a leader if it is greater than all elements to the right of it in X. The best algorithm to find all leaders in an array (GATE-2006) (2 Marks)

(A) Solves it in linear time using a left to right pass of the array

(B) Solves it in linear time using a right to left pass of the array

(C) Solves it using divide and conquer in time theta(nlogn)

(D) Solves it in time theta(n^2)

Q The minimum number of comparisons required to determine if an integer appears more than n/2 times in a sorted array of n integers is (Gate-2008) (1 Marks)

(A) (n) (B) (logn) (C) (log*n) (D) (n)

Q Let *A* be an array of 31 numbers consisting of a sequence of 0's followed by a sequence of 1's. The problem is to find the smallest index *i* such that *A[i]* is 1 by probing the minimum number of locations in A. The *worst-case* number of probes performed by an *optimal* algorithm is _____. (Gate-2017) (1 Marks)

Q Which of the following operations is not O(1) for an array of sorted data. You may assume that array elements are distinct.

(A) Find the ith largest element **(B)** Delete an element

(C) Find the ith smallest element **(D)** All of the above

Q Consider the problem of searching an element x in an array 'arr[]' of size n. The problem can be solved in O(Logn) time if.

1) Array is sorted

2) Array is sorted and rotated by k. k is given to you and k <= n

3) Array is sorted and rotated by k. k is NOT given to you and k <= n

4) Array is not sorted

(A) 1 Only **(B)** 1 & 2 only **(C)** 1, 2 and 3 only **(D)** 1, 2, 3 and 4

Q Which sorting algorithms is most efficient to sort string consisting of ASCII characters?

(A) Quick sort **(B)** Heap sort **(C)** Merge sort **(D)** Counting sort

Q The tightest lower bound on the number of comparisons, in the worst case, for comparison-based sorting is of the order of

(A) N **(B)** N^2 **(C)** NlogN **(D)** $N(logN)^2$

Q Given an unsorted array. The array has this property that every element in array is at most k distance from its position in sorted array where k is a positive integer smaller than size of array. Which sorting algorithm can be easily modified for sorting this array and what is the obtainable time complexity?

(A) Insertion Sort with time complexity O(kn)
(B) Heap Sort with time complexity O(nLogk)
(C) Quick Sort with time complexity O(kLogk)
(D) Merge Sort with time complexity O(kLogk)

Q Given an array where numbers are in range from 1 to n^6, which sorting algorithm can be used to sort these number in linear time?

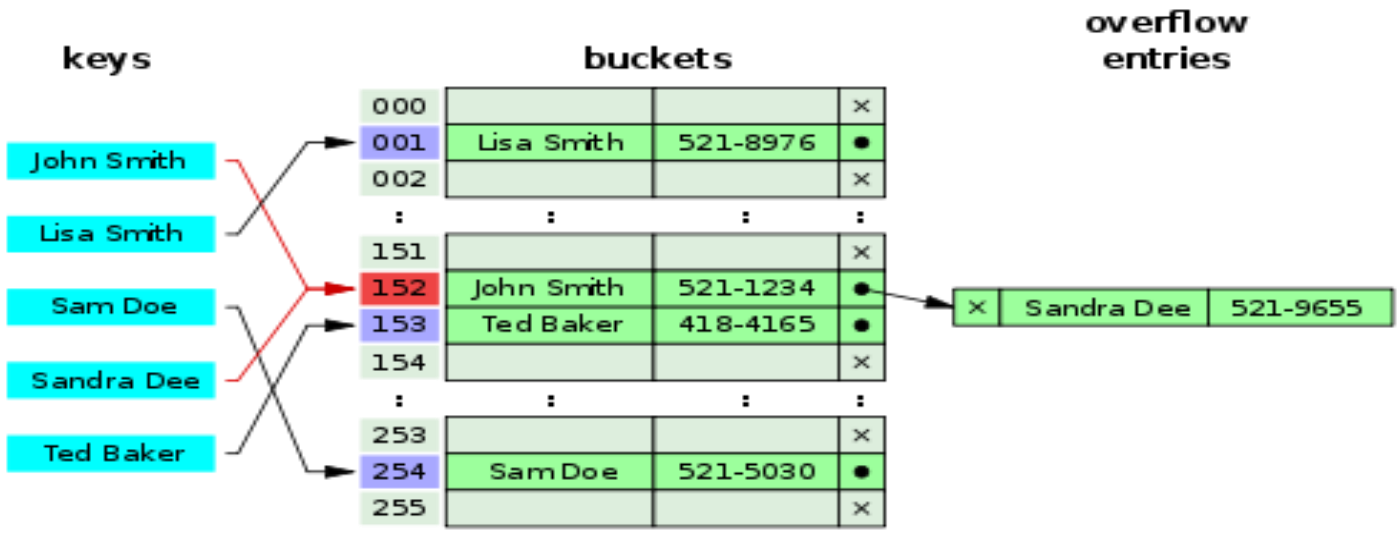
(A) Not possible to sort in linear time
(B) Radix Sort
(C) Counting Sort
(D) Quick Sort

Q An inversion in a an array A[] is a pair (A[i], A[j]) such that A[i] > A[j] and i < j. An array will have maximum number of inversions if it is:

(A) Sorted in increasing order
(B) Sorted in decreasing order
(C) Sorted in alternate fashion
(D) Both A and B

Introduction to hashing

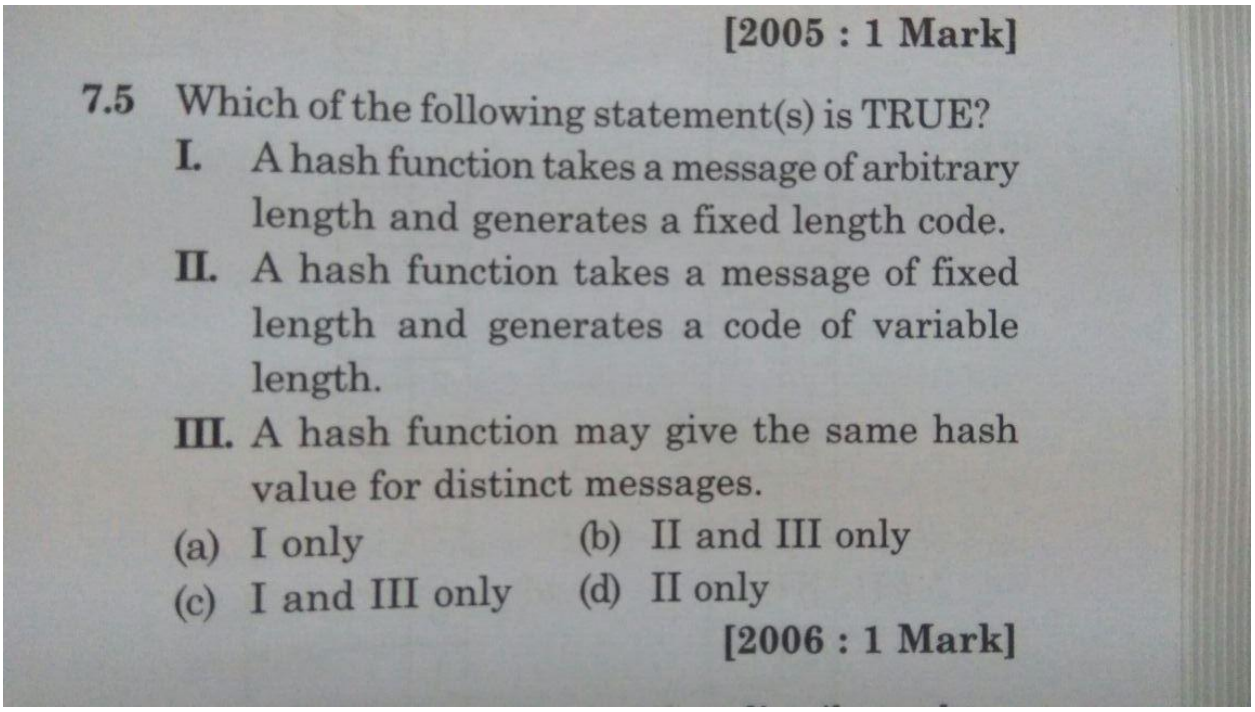
- Main idea of data structure is to help us store the data. But Most common operation on any data structure is not insert or delete but actually search, as even for insertion and deletion search is also required.
- In any of the data structure the search time first depends on the number of elements which data structure contains and then on type of structure. for e.g.
    - Unsorted array – O(n)
    - sorted array – O(logn)
    - link list – O(n)
    - BT – O(n)
    - BST – O(n)
    - AVL – O(logn)
- So hashing is a technique where search time is independent of the number of items in which we are searching a data value.
- The basic idea is to use the key itself to find the address in the memory to make searching easy. For e.g. *to use phone number, roll no, Aadhar card, voter id or any other key and convert it into a smaller* practical *number (*but it must be modified so a great deal of space is not wasted*) and uses the small number as index in a table called hash table.*
- The values are then stored in **hash table**, By using that key you can access the element in **O(1)** time.
- This conversion called hash function which is from the set of K keys into the set of memory location L.
    - H: K→L
- In simple terms, a hash function maps a big number or string to a small integer that can be used as index in hash table. An array that stores pointers to records corresponding to our search key. The remaining entries can be nil.



Most popular hash functions

- *Division-remainder method:* The size of the number of items in the table is estimated. That number is then used as a divisor into each original value or key to extract a quotient and a remainder. The remainder is the hashed value. (Since this method is liable to produce a number of collisions, any search mechanism would have to be able to recognize a collision and offer an alternate search mechanism.)
    - *H(K) = K(mod m)*
    - *H(K) = K(mod m) + 1*
- Collision: - It is possible that two different set of keys K_1 and K_2 will yield the same hash address. This situation is called collision. The technique to resolve collision is called collision resolution.

- Characteristics of good hash function
    - Easy to compute and understand
    - Efficiently computable- It must take less time to compute
    - Should uniformly distribute the keys (Each table position equally likely for each key) and should not result in clustering.
    - Must have low collision rate
- **Note**: Irrespective of how good a hash function is, collisions are bound to occur. Therefore, to maintain the performance of a hash table, it is important to manage collisions through various collision resolution techniques.



[2005 : 1 Mark]

7.5 Which of the following statement(s) is TRUE?

I. A hash function takes a message of arbitrary length and generates a fixed length code.

II. A hash function takes a message of fixed length and generates a code of variable length.

III. A hash function may give the same hash value for distinct messages.

(a) I only (b) II and III only

(c) I and III only (d) II only

[2006 : 1 Mark]

Q Given the following input (4322, 1334, 1471, 9679, 1989, 6171, 6173, 4199) and the hash function x mod 10, which of the following statements are true? (Gate-2004) (1 Marks)

i. 9679, 1989, 4199 hash to the same value

ii. 1471, 6171 has to the same value

iii. All elements hash to the same value

iv. Each element hashes to a different value

(A) i only **(B)** ii only **(C)** i and ii only **(D)** iii or iv

Collision resolution technique

- **Open Addressing/closed hashing** - In Open Addressing, all elements are stored in the hash table itself. i.e. collision is resolved by **probing** or searching through alternate locations in the Hash table itself in a particular *sequence.*
- When searching for an element, we one by one examine table slots until the desired element is found or it is clear that the element is not in the table. So, at any point, size of table must be greater than or equal to total number of keys.
- It is of three types linear probing, quadratic probing, double hashing

- Performance of Open Addressing: Like Chaining, performance of hashing can be evaluated under the assumption that each key is equally likely to be hashed to any slot of table (simple uniform hashing)
    - m = Number of slots in hash table
    - n = Number of keys to be inserted in hash table
    - Load factor $\alpha = n/m$ (< 1)
    - Expected time to search/insert/delete $< 1/(1 - \alpha)$
    - So Search, Insert and Delete take $(1/(1 - \alpha))$ time

Q Given a hash table T with 25 slots that stores 2000 elements, the load factor α for T is _____(Gate-2015) (1 Marks)

(A) 80 **(B)** 0.0125 **(C)** 8000 **(D)** 1.25

Q Consider a hash function that distributes keys uniformly. The hash table size is 20. After hashing of how many keys will the probability that any new key hashed collides with an existing one exceed 0.5. (Gate-2007) (2 Marks)

(A) 5 **(B)** 6 **(C)** 7 **(D)** 10

Linear probing

- Searches the table sequentially starting at the position given by the hash function, until finding a cell with a matching key or an empty cell.
- It takes constant expected time per search, insertion, or deletion when implemented using a random hash function.
- Using linear probing, dictionary operations can be implemented in constant expected time. In other words, insert, remove and search operations can be implemented in O(1), as long as the load factor of the hash table is a constant strictly less than one.
    - Insert(k): Keep probing until an empty slot is found. Once an empty slot is found, insert k.
    - Search(k): Keep probing until slot's key doesn't become equal to k or an empty slot is reached.
    - Delete(k): ***Delete operation is interesting***. If we simply delete a key, then search may fail. So slots of deleted keys are marked specially as "deleted". Insert can insert an item in a deleted slot, but search doesn't stop at a deleted slot.
- Advantage
    - The most popular implementation on standard hardware uses linear probing, which is both fast and simple.
    - Linear probing can provide high performance because of its good locality of reference.
- Disadvantage
    - Is more sensitive to the quality of its hash function than some other collision resolution schemes.
    - its performance degrades more quickly at high load factors because of primary clustering, a tendency for one collision to cause more nearby collisions. Basic operations take more time. etc
    - Additionally, achieving good performance with this method requires a higher-quality hash function than for some other collision resolution schemes.

Q Consider a hash table of size seven, with starting index zero, and a hash function (7x+3) mod 4. Assuming the hash table is initially empty, which of the following is the contents of the table when the sequence 1, 3, 8, 10 is inserted into the table using closed hashing? Here "__" denotes an empty location in the table. **(NET-JULY-2018)**

(1) 3, 10, 1, 8, __ , __ , __

(2) 1, 3, 8, 10, __ , __ , __

(3) 1, __ , 3, __ , 8, __ , 10

(4) 3, 10, __ , __ , 8, __ , _

Q A hash table contains 10 buckets and uses linear probing to resolve collision. The key values are integers and the hash function used is key%10, if the values 43 165 62 123 142 are inserted in the table, in what location would the key value 142 be inserted? (Gate-2005) (1 Marks)

A) 2 b) 3 c) 4 d)

Q The keys 12, 18, 13, 2, 3, 23, 5 and 15 are inserted into an initially empty hash table of length 10 using open addressing with hash function h(k) = k mod 10 and linear probing. What is the resultant hash table? (Gate-2009) (2 Marks)

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

(A)

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

(B)

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

(C)

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

(D)

Q consider a hash table of size 11 that uses open addressing with linear probing. Let h(k)= k mod 11 be the hash function used. A sequence of records with keys 43 36 92 87 11 4 71 13 14 is inserted into an initially empty hash table, indexing is from 0, what is the index of the bin into which the last record is inserted? (Gate-2008) (2 Marks)

a) 3 b) 4 c) 6 d) 7

Q Consider a hash table of size seven, with starting index zero, and a hash function (3x + 4)mod7. Assuming the hash table is initially empty, which of the following is the contents of the table when the sequence 1, 3, 8, 10 is inserted into the table using closed hashing? Note that '_' denotes an empty location in the table. (Gate-2007) (2 Marks)

(A) 8, _, _, _, _, _, 10

(B) 1, 8, 10, _, _, _, 3

(C) 1, _, _, _, _, _,3

(D) 1, 10, 8, _, _, _, 3

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

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

Which one of the following choices gives a possible order in which the key values could have been inserted in the table? (Gate-2010) (2 Marks)

(A) 46, 42, 34, 52, 23, 33 **(B)** 34, 42, 23, 52, 33, 46

(C) 46, 34, 42, 23, 52, 33 **(D)** 42, 46, 33, 23, 34, 52

Q How many different insertion sequences of the key values using the same hash function and linear probing will result in the hash table shown above? (Gate-2010) (2 Marks)

(A) 10 **(B)** 20 **(C)** 30 **(D)** 40

Q A hash table with ten buckets with one slot per bucket is shown in the following figure. The symbols S1S1 to S7S7 initially entered using a hashing function with linear probing. The maximum number of comparisons needed in searching an item that is not present is (Gate-1989) (2 Marks)

- **Primary clustering** is one of two major failure modes of open addressing based hash tables, especially those using linear probing. For instance, in linear probing, a record involved in a collision is always moved to the next available hash table cell subsequent to the position given by its hash function, creating a contiguous cluster of occupied hash table cells. Whenever another record is hashed to anywhere within the cluster, it grows in size by one cell.

- A related phenomenon, **secondary clustering**, occurs more generally with open addressing modes including linear probing and quadratic probing in which the probe sequence is independent of the key.
- In this phenomenon, a low-quality hash function may cause many keys to hash to the same location, after which they all follow the same probe sequence or are placed in the same hash chain as each other, causing them to have slow access times.

Quadratic probing

- Quadratic probing operates by taking the original hash index and adding successive values of an arbitrary quadratic polynomial until an open slot is found. for e.g. $i + 1^2$, $i + 2^2$, $i + 3^2$, $i + 4^2$, $i + 5^2$, $i + 6^2$, ……
- **Advantage**
    - Quadratic probing can be a more efficient algorithm in a closed hashing table, since it better avoids the clustering problem that can occur with linear probing, although it is not immune.
    - It also provides good memory caching because it preserves some locality of reference; however, linear probing has greater locality and, thus, better cache performance.
- **Disadvantage**
    - Quadratic probing lies between the two in terms of cache performance and clustering.

Double Hashing

- **Advantage**
  - Exhibits virtually no clustering
  - Double hashing has poor cache performance but no clustering. Double hashing requires more computation time as two hash functions need to be computed.
- **Disadvantage**
  - Double hashing can also require more computation than other forms of probing.
  - While double hashing has poor cache performance
- Conclusion
  - A critical influence on performance of an open addressing hash table is the *load factor*; that is, the proportion of the slots in the array that are used.
  - As the load factor increases towards 100%, the number of probes that may be required to find or insert a given key rises dramatically.
  - Once the table becomes full, probing algorithms may even fail to terminate. Even with good hash functions, load factors are normally limited to 80%.
  - A poor hash function can exhibit poor performance even at very low load factors by generating significant clustering, especially with the simplest linear addressing method. Generally typical load factors with most open addressing methods are 50%

Chaining

- The idea is to make each cell of hash table point to a linked list of records that have same hash function value.
    - **Advantage: -** Chaining is simple
    - **Disadvantage: - B**ut requires additional memory outside the table.

S.No.	Separate Chaining	Open Addressing
1.	Chaining is Simpler to implement.	Open Addressing requires more computation.
2.	In chaining, Hash table never fills up, we can always add more elements to chain.	In open addressing, table may become full.
3.	Chaining is Less sensitive to the hash function or load factors.	Open addressing requires extra care for to avoid clustering and load factor.
4.	Chaining is mostly used when it is unknown how many and how frequently keys may be inserted or deleted.	Open addressing is used when the frequency and number of keys is known.
5.	Cache performance of chaining is not good as keys are stored using linked list.	Open addressing provides better cache performance as everything is stored in the same table.
6.	Wastage of Space (Some Parts of hash table in chaining are never used).	In Open addressing, a slot can be used even if an input doesn't map to it.
7.	Chaining uses extra space for links.	No links in Open addressing

Q An advantage of chained hash table (external hashing) over the open addressing scheme is (Gate-1996) (1 Marks)
(A) Worst case complexity of search operations is less
(B) Space used is less
(C) Deletion is easier
(D) None of the above

Q Consider a hash table with 9 slots. The hash function is *h(k) = k mod 9*. The collisions are resolved by chaining. The following 9 keys are inserted in the order: 5, 28, 19, 15, 20, 33, 12, 17, 10. The maximum, minimum, and average chain lengths in the hash table, respectively, are (Gate-2014) (2-Marks)

(A) 3, 0, and 1 **(B)** 3, 3, and 3 **(C)** 4, 0, and 1 **(D)** 3, 0, and 2

Q Consider a hash table with 100 slots. Collisions are resolved using chaining. Assuming simple uniform hashing, what is the probability that the first 3 slots are unfilled after the first 3 insertions? (Gate-2014) (2 Marks)

(A) $(97 \times 97 \times 97)/100^3$

(B) $(99 \times 98 \times 97)/100^3$

(C) $(97 \times 96 \times 95)/100^3$

(D) $(97 \times 96 \times 95)/(3! \times 100^3)$

Q Which one of the following hash functions on integers will distribute keys most uniformly over 10 buckets numbered 0 to 9 for i ranging from 0 to 2020? (Gate-2015) (2-Marks)

(A) $h(i) = i^2 \bmod 10$

(B) $h(i) = i^3 \bmod 10$

(C) $h(i) = (11 * i^2) \bmod 10$

(D) $h(i) = (12 * i) \bmod 10$

Q Consider a relational table r with sufficient number of records, having attributes A1, A2..., An and let 1 <= p <= n. Two queries Q1 and Q2 are given below. (cs-2011)

$$Q1: \pi_{A1...A_n}\left(\sigma_{A_p=C}(r)\right) \text{ where c is a const}$$

$$Q2: \pi_{A1...A_n}\left(\sigma_{C_1 \le A_p \le C_2}(r)\right) \text{ where } c_1 \text{ and } c_2 \text{ are constants}$$

The database can be configured to do ordered indexing on Ap or hashing on Ap. Which of the following statements is TRUE?

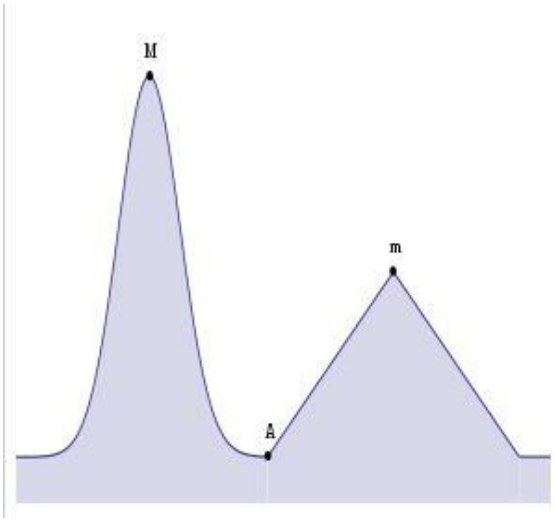
(A) Ordered indexing will always outperform hashing for both queries

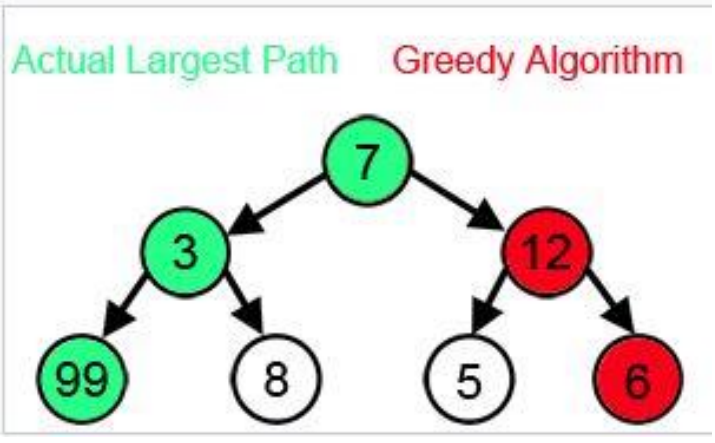
(B) Hashing will always outperform ordered indexing for both queries

(C) Hashing will outperform ordered indexing on Q1, but not on Q2

(D) Hashing will outperform ordered indexing on Q2, but not on Q1.

Greedy Algorithm

- A **greedy algorithm** is any algorithm that follows the problem-solving heuristic of making the locally optimal choice at each stage with the intent of finding a global optimum. In many problems, a greedy strategy does not usually produce an optimal solution, but nonetheless a greedy heuristic may yield locally optimal solutions that approximate a globally optimal solution in a reasonable amount of time.
- For example, a greedy strategy for the traveling salesman problem (which is of a high computational complexity) is the following heuristic: "At each step of the journey, visit the nearest unvisited city." This heuristic does not intend to find a best solution, but it terminates in a reasonable number of steps; finding an optimal solution to such a complex problem typically requires unreasonably many steps.
- We can make whatever choice seems best at the moment and then solve the subproblems that arise later. The choice made by a greedy algorithm may depend on choices made so far, but not on future choices or all the solutions to the subproblem. It iteratively makes one greedy choice after another, reducing each given problem into a smaller one. In other words, a greedy algorithm never reconsiders its choices. This is the main difference from dynamic programming, which is exhaustive and is guaranteed to find the solution. After every stage, dynamic programming makes decisions based on all the decisions made in the previous stage, and may reconsider the previous stage's algorithmic path to solution.
- 

- 

Huffman coding

- In computer science and information theory, a **Huffman code** is a particular type of optimal prefix code that is commonly used for lossless data compression.
- The process of finding or using such a code proceeds by means of **Huffman coding**, an algorithm developed by David A. Huffman while he was a Sc.D. student at MIT, and published in the 1952 paper "A Method for the Construction of Minimum-Redundancy Codes".
- The output from Huffman's algorithm can be viewed as a variable-length code table for encoding a source symbol (such as a character in a file).
- The algorithm derives this table from the estimated probability or frequency of occurrence (*weight*) for each possible value of the source symbol.
- Huffman's method can be efficiently implemented, finding a code in time linear to the number of input weights if these weights are sorted. However, although optimal among methods encoding symbols separately, Huffman coding is not always optimal among all compression methods.

Q Consider the following character with probability and generate Huffman tree, find Huffman code for each character, find the number of bits required per character?

Character	Probability
M_1	.12
M_2	.04
M_3	.44
M_4	.17
M_5	.23

Q A message is made up entirely of characters from the set *X = {P, Q, R, S, T}*. The table of probabilities for each of the characters is shown below:

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

If a message of 100 characters over *X* is encoded using Huffman coding, then the expected length of the encoded message in bits is _____. **(Gate-2017) (2 Marks)**

ANSWER 225

Q Suppose the letters a, b, c, d, e, f has probabilities 1/2, 1/4, 1/8, 1/16, 1/32, 1/32 respectively. Which of the following is the Huffman code for the letter a, b, c, d, e, f? (Gate - 2007) (2 Marks)

(A) 0, 10, 110, 1110, 11110, 11111
(B) 11, 10, 011, 010, 001, 000
(C) 11, 10, 01, 001, 0001, 0000
(D) 110, 100, 010, 000, 001, 111

Answer: (A)

Q The characters a to h have the set of frequencies based on the first 8 Fibonacci numbers as follows
a : 1, b : 1, c : 2, d : 3, e : 5, f : 8, g : 13, h : 21

A Huffman code is used to represent the characters. What is the sequence of characters corresponding to the following code? (Gate-2006) (2 Marks)
110111100111010

(A) fdheg **(B)** ecgdf **(C)** dchfg **(D)** fehdg

Answer: (A)

Q Suppose the letters a, b, c, d, e, f has probabilities 1/2, 1/4, 1/8, 1/16, 1/32, 1/32 respectively. What is the average length of Huffman codes? (Gate - 2007) (2 Marks)

(A) 3 **(B)** 2.1875 **(C)** 2.25 **(D)** 1.19375

Answer: (D)

Q A networking company uses a compression technique to encode the message before transmitting over the network. Suppose the message contains the following characters with their frequency:

character	Frequency
a	5
b	9
c	12
d	13
e	16
f	45

If the compression technique used is Huffman Coding, how many bits will be saved in the message?

(A) 224 **(B)** 800 **(C)** 576 **(D)** 324

Answer: (C)

Q In question #2, which of the following represents the word *"dead"*?

(A) 1011111100101 **(B)** 0100000011010 **(C)** Both A and B **(D)** None of these

Answer: (C)

Q What is the time complexity of Huffman Coding?

(A) O(N) **(B)** O(NlogN) **(C)** $O(N(\log N)^2)$ **(D)** $O(N^2)$

Answer: (B)

File sorting (Optimal merge pattern)

- **Optimal merge pattern** is a pattern that relates to the merging of two or more sorted files in a single sorted file by two-way merge, with minimum number of record movements.
- Size of solution space is n!

Q Given a set of 8 files from F1 to F8 with following number of pages find the minimum number of record movements to merge them into single file?

F1	F2	F3	F4	F5	F6	F7	F8
18	3	15	12	10	11	7	9

Q Suppose P, Q, R, S, T are sorted sequences having lengths 20, 24, 30, 35, 50 respectively. They are to be merged into a single sequence by merging together two sequences at a time. The number of comparisons that will be needed in the worst case by the optimal algorithm for doing this is ____. (Gate-2014) (2 Marks)

Answer: (358)

Knap Sack Problem

- The **knapsack problem** or **rucksack problem** is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items.
- The problem often arises in resource allocation where there are financial constraints and is studied in fields such as combinatorics, computer science, complexity theory, cryptography and applied mathematics.
- The knapsack problem has been studied for more than a century, with early works dating as far back as 1897. The name "knapsack problem" dates back to the early works of mathematician Tobias Dantzig (1884–1956).
- Knap Sack problem can be studied in two versions fractional Knap Sack and 0/1 Knap Sack, here we will be discussing Fractional Knap Sack and then 0/1 Knap Sack will be solved in another algorithmic Paradigm.
- More formally there are n number of objects O_1, O_2, O_3…O_n, each object has a weight associated with its W_i, and a profit associated with it P_i, we can take x_i fraction of the object ranging from $0 <= x_i <= 1$
- Weight Condition $\sum_{i=1}^{n} Wi \,.\, Xi$ <= M
- Profit $\sum_{i=1}^{n} Pi \,.\, Xi$

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

Object	O_1	O_2	O_3
Weight	18	15	10
Profit	25	24	15

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

Item number	Weight (in Kgs)	Value (in Rupees)
1	10	60
2	7	28
3	4	20
4	2	24

The task is to pick a subset of these items such that their total weight is no more than 11 Kgs and their total value is maximized. Moreover, no item may be split. The total value of items picked by an optimal algorithm is denoted by V_{opt}. A greedy algorithm sorts the items by their value-to-weight ratios in descending order and packs them greedily, starting from the first item in the ordered list. The total value of items picked by the greedy algorithm is denoted by V_{greedy}. The value of $V_{opt} - V_{greedy}$ is _____ . (Gate-2018) (2 Marks)

(ANSWER- 16)

Job scheduling problem

- In job scheduling problem we have single CPU with Non-Primitive Scheduling and n-jobs. with each job we assume arrival time is 0, burst time of each job requirement is 1, and each job is associated with a deadline D_i and a profit P_i.
- Select a Subset of 'n' jobs, such that, the jobs in the subset can be completed with in the deadline and generate Max profit.

Q if we have for task T_1, T_2, T_3, T_4, having Deadline $D_1 = 2$, $D_2 = 1$, $D_3 = 2$, $D_4 = 1$, and profit $P_1 = 100$, $P_2 = 10$, $P_3 = 15$, $P_4 = 27$, find the maximum profit possible?

Q We are given 9 tasks T1, T2…. T9. The execution of each task requires one unit of time. We can execute one task at a time. Each task Ti has a profit Pi and a deadline di Profit Pi is earned if the task is completed before the end of the dith unit of time.

Task	T1	T2	T3	T4	T5	T6	T7	T8	T9
Profit	15	20	30	18	18	10	23	16	25
Deadline	7	2	5	3	4	5	2	7	3

Are all tasks completed in the schedule that gives maximum profit? (Gate-2005) (2 Marks)

(A) All tasks are completed **(B)** T1 and T6 are left out
(C) T1 and T8 are left out **(D)** T4 and T6 are left out
Answer: (D)

Q We are given 9 tasks T1, T2…. T9. The execution of each task requires one unit of time. We can execute one task at a time. Each task Ti has a profit Pi and a deadline di Profit Pi is earned if the task is completed before the end of the dith unit of time.

Task	T1	T2	T3	T4	T5	T6	T7	T8	T9
Profit	15	20	30	18	18	10	23	16	25
Deadline	7	2	5	3	4	5	2	7	3

What is the maximum profit earned? (Gate-2005) (2 Marks)

(A) 147 **(B)** 165 **(C)** 167 **(D)** 175
Answer: (A)

Q Consider n jobs J_1, J_2……J_n such that job J_i has execution time t_i and a non-negative integer weight w_i. The weighted mean completion time of the jobs is defined to be,

$$\frac{\sum_{i=1}^{n} w_i T_i}{\sum_{i=1}^{n} w_i},$$

where T_i is the completion time of job J_i. Assuming that there is only one processor available, in what order must the jobs be executed in order to minimize the weighted mean completion time of the jobs? (Gate-2007) (2 Marks)

(A) Non-decreasing order of t_i
(B) Non-increasing order of w_i
(C) Non-increasing order of $w_i t_i$
(D) None-increasing order of w_i/t_i

Answer: (D)

Q The following are the starting and ending times of activities A, B, C, D, E, F, G and H respectively in chronological order: "a_s b_s c_s a_e d_s c_e e_s f_s b_e d_e g_s e_e f_e h_s g_e h_e" Here, x_s denotes the starting time and x_e denotes the ending time of activity X. W need to schedule the activities in a set of rooms available to us. An activity can be scheduled in a room only if the room is reserved for the activity for its entire duration. What is the minimum number of rooms required? (Gate-2003) (2 Marks)

(A) 3 **(B)** 4 **(C)** 5 **(D)** 6

Answer: (B)

Dynamic Programming

- Dynamic programming is like the divide and conquer method, solve problems by combining the solutions to the subproblems.
- Divide and conquer algo partition the problem into independent subproblem, solve the subproblems recursively and then combine their solutions to solve the original problems.
- In contrast, dynamic programming is applicable when the subproblems are not independent, i.e. when subproblems share subsubproblems. A dynamic-programming algorithm solves every subsubproblems just one and then saves its answer in a table there by avoiding the work of recomputing the answer every time the subproblem is encountered.
- Dynamic programming is typically applied to optimization problems. In such case there can be many possible solutions. Each solution has a value, and we wish to find a solution with the optimal value (minimum, maximum).

- There are four steps of dynamic programming
    - Characterize the solution of an optimal solution.
    - Recursively define the value of an optimal solution.
    - Compute the value of an optimal solution in a bottom-up-fashion.
    - Construct an optimal solution from computed information.

Longest common subsequence

Let there are two sequence X and Y, we say that a sequence Z is a common subsequence of X and Y where

$X_m = \{x_1, x_2, x_3, \ldots\ldots\ldots x_m\}$

$Y_n = \{y_1, y_2, y_3, \ldots\ldots\ldots y_n\}$

we wish to find the maximum length of common of both X and Y, $Z = \{z_1, z_2, z_3, \ldots\ldots\ldots z_k\}$

STEP 1: Optimal Substructure

- if $x_m = y_n$
  - then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}
- if x_m != y_n, then z_k != x_m
  - Implies that Z_k is an LCS of X_{m-1} and Y_n
- if x_m != y_n, then z_k != y_n
  - Implies that Z_k is an LCS of X_m and Y_{n-1}

STEP 2: Recursive Solution

- Let us define C[i, j] to the length of L.C.S of sequence of X_i and Y_j, if either i=0 or j=0, so LCS has length 0.
- The optimal substructure of LCS sub program gives the recursive formula
  - C[i, j] = {0 if i=0 or j=0}
  - C[i, j] = {C[i-1, j-1] + 1 if i, j > 0 and $x_i = y_j$}
  - C[i, j] = {max(C[i, j-1], C[i-1, j] if i, j > 0 and x_i != y_j}

STEP 3: Computing the length of L.C.S

```
LCS-Length (x, y)
{
        m ← Length[x]
        n ← Length[y]
        for i ←1 to m
        {
                do C[i, 0] ← 0
        }
        for j ←0 to n
        {
                do C[0, j] ← 0
        }
        for i ←1 to m
        {
                for j ←1 to n
                {
                        if (xi = yj)
                        {
                                c[i, j] ← c[i-1, j-1] + 1
                                b[i, j] ← 'D_edge'
                        }
                        else if (C[i-1, j] >= C[i, j-1])
                        {
                                C[i, j] ← C[i-1, j]
                                b[i, j] ← 'V_edge'
                        }
                        else
                        {
                                C[i, j] ← C[i, j-1]
                                b[] ← 'H_edge'
                        }
                }
        }
        return b and C
}
```

```
Print_LCS (b, X, i, j)
{
        if i=0 or j=0
                return
        if (b[i, j] = 'D_edge')
        {
                then Print_LCS (b, X, i-1, j-1)
                Print Xi
        }
        Else if (b[i, j] = 'V_edge')
        {
                Print_LCS (b, X, i-1, j)
        }
        Else
        {
                Print_LCS (b, X, i, j-1)
        }
}
```

Q Consider two strings X = "A, B, C, B, D, A, B" and Y = "B, D, C, A, B, A". Find the longest common subsequence?

Q Consider two strings A = "qpqrr" and B = "pqprqrp". Let x be the length of the longest common subsequence (not necessarily contiguous) between A and B and let y be the number of such longest common subsequences between A and B. Then x + 10y = ___(Gate-2014) (2 Marks)
Answer: (34)

Q A sub-sequence of a given sequence is just the given sequence with some elements (possibly none or all) left out. We are given two sequences X[m] and Y[n] of lengths m and n respectively, with indexes of X and Y starting from 0.
We wish to find the length of the longest common sub-sequence(LCS) of X[m] and Y[n] as l(m, n), where an incomplete recursive definition for the function l(i, j) to compute the length of The LCS of X[m] and Y[n] is given below(Gate-2009) (2 Marks)
l(i,j) = 0, if either i=0 or j=0
= expr1, if i,j > 0 and X[i-1] = Y[j-1]
= expr2, if i,j > 0 and X[i-1] != Y[j-1]

A) expr1 ≡ l(i-1, j) + 1
(B) expr1 ≡ l(i, j-1)
(C) expr2 ≡ max(l(i-1, j), l(i, j-1))
(D) expr2 ≡ max(l(i-1,j-1),l(i, j))
Answer: (C)

Q Consider the data given in the previous question. The values of l(i, j) could be obtained by dynamic programming based on the correct recursive definition of l(i, j) of the form given above, using an array L[M, N], where M = m+1 and N =n+1, such that L[i, j] = l(i, j).
Which one of the following statements would be TRUE regarding the dynamic programming solution for the recursive definition of l(i, j)?
(A) All elements L should be initialized to 0 for the values of l(i, j) to be properly computed
(B) The values of l(i, j) may be computed in a row major order or column major order of L(M,N)
(C) The values of l(i, j) cannot be computed in either row major order or column major order of L(M,N)
(D) L[p, q] needs to be computed before L[r, s] if either p < r or q < s.
Answer: (B)

Matrix chain multiplication

In matrix-chain multiplication problem, we are not actually multiplying matrix. Our goal is only to determine an order for multiplying matrix that the lowest cost.

STEP 1: The structure of an optimal paranthesization

- Suppose there are A_i, A_{i+1}……, A_j matrix to be multiplied
- Now let split the product chain A_i,……, A_K ,A_{K+1}……, A_j
- Let A_{ij} where i<=j for the matrix that results from evaluation of the product A_i,……, A_K ,A_{K+1}……, A_j
- if i<=j then any paranthesization of product A_i,……, A_K ,A_{K+1}……, A_j must split the product between A_K & A_{K+1} in the range i <= k< j for the value of k. we first compute the matrix A_i,……, A_K ,A_{K+1}……, A_j. then multiplying them to produce the final product A_i,……, A_j
- The cost of this paranthesization is the cost of computing the matrix A_i * A_K and A_{K+1} to A_j and the cost of multiplying them together.
- Suppose that the optimal paranthesization of A_i,……, A_j splits the product between A_k and A_{k+1} then the paranthesization of prefix sub chain. A_1….A_k with in this optimal paranthesization of A_i….A_j must be optimal paranthesization of A_1 to A_k. A similar observation holds for the paranthesization of sub chain A_{k+1} to A_j

STEP 2: Recursive Solution

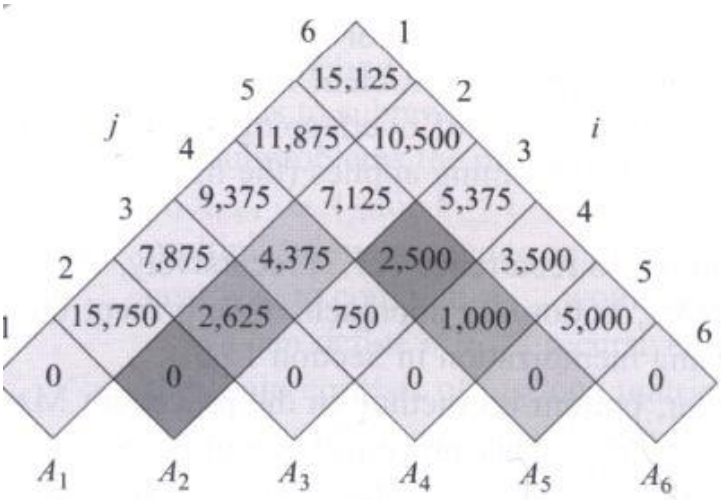
- We define the cost of an optimal solution recursively in terms of the optimal solution to subproblems.
- Let m[i, j] be the minimum number of scalar product needed to compute the matrix $A_{i...j}$
- Let us assume that the optimal paranthesization splits the product A_i, A_{i+1}……, A_j between A_k and A_{k+1} . where i <= k < j. Then m[i, j] is equal to the minimum cost for computing the sub products A_i,……, A_K and A_{K+1}……, A_j plus the cost of multiplying these two matrix together.
- Each matrix Ai is P_{i-1} * P_i we see that commuting the matrix product A_i,……, A_K ,A_{K+1}……, A_j takes P_{i-1} P_k P_j scalar multiplication.
- m[i, j] = { 0 if(i=j)}
- m[i, j] = { min[m[i, k] + m[k+1, j] + P_{i-1} P_k P_j] if(i < j)}

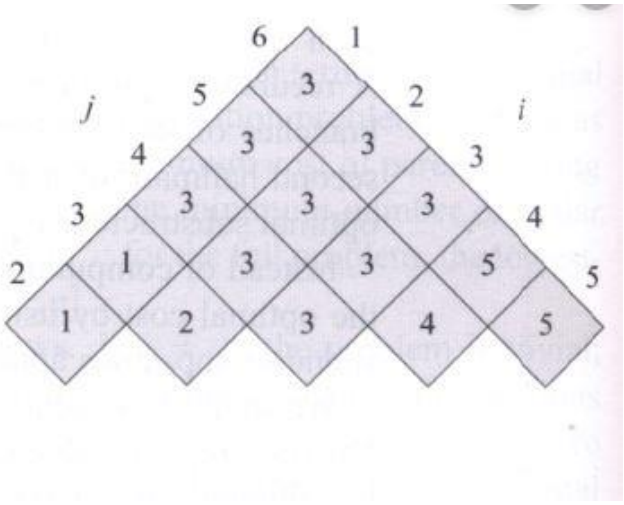
STEP 3: Computing the Optimal Cost

STEP 4: Constructing an optimal Solution

```
Print-Optimal Parenthesis (S, i, j)
{
        if (i=j)
        {
                then print 'Ai'
        }
        Else
        {
                print "("
                Print-Optimal Parenthesis (S, i, S [i, j])
                Print-Optimal Parenthesis (S, S [i, j] + 1, j)
                print ')'
        }
}
```

Q Let A_1, A_2, A_3, A_4 , A_5 and A_6 be four matrices of dimensions 30 x 35, 35 x 15, 15 x 5, 15 x 10, 10 x 20, and 20 x 25, respectively. The minimum number of scalar multiplications required to find the product $A_1A_2A_3A_4A_5A_6$using the basic matrix multiplication method is?





Q Let A_1, A_2, A_3, and A_4 be four matrices of dimensions 10 x 5, 5 x 20, 20 x 10, and 10 x 5, respectively. The minimum number of scalar multiplications required to find the product $A_1A_2A_3A_4$ using the basic matrix multiplication method is (Gate-2016) (2 Marks)
Ans: 1500

Q Assume that multiplying a matrix G_1 of dimension p × q with another matrix G_2 of dimension q × r requires pqr scalar multiplications. Computing the product of n matrices $G_1G_2G_3$ ….. G_n can be done by parenthesizing in different ways. Define G_iG_{i+1} as an **explicitly computed pair** for a given paranthesization if they are directly multiplied. For example, in the matrix multiplication chain $G_1G_2G_3G_4G_5G_6$ using paranthesization $(G_1(G_2G_3))(G_4(G_5G_6))$, G_2G_3 and G_5G_6 are only explicitly computed pairs.
Consider a matrix multiplication chain $F_1F_2F_3F_4F_5$, where matrices F_1, F_2, F_3, F_4 and F_5 are of dimensions 2×25,25×3,3×16,16×1 and 1×1000, respectively. In the paranthesization of $F_1F_2F_3F_4F_5$ that minimizes the total number of scalar multiplications, the explicitly computed pairs is/are (Gate-2018) (2 Marks)

(A) F_1F_2 and F_3F_4 only
(B) F_2F_3 only
(C) F_3F_4 only
(D) F_1F_2 and F_4F_5 only

Answer: (C)

Q Four matrices M1, M2, M3 and M4 of dimensions pxq, qxr, rxs and sxt respectively can be multiplied is several ways with different number of total scalar multiplications. For example,

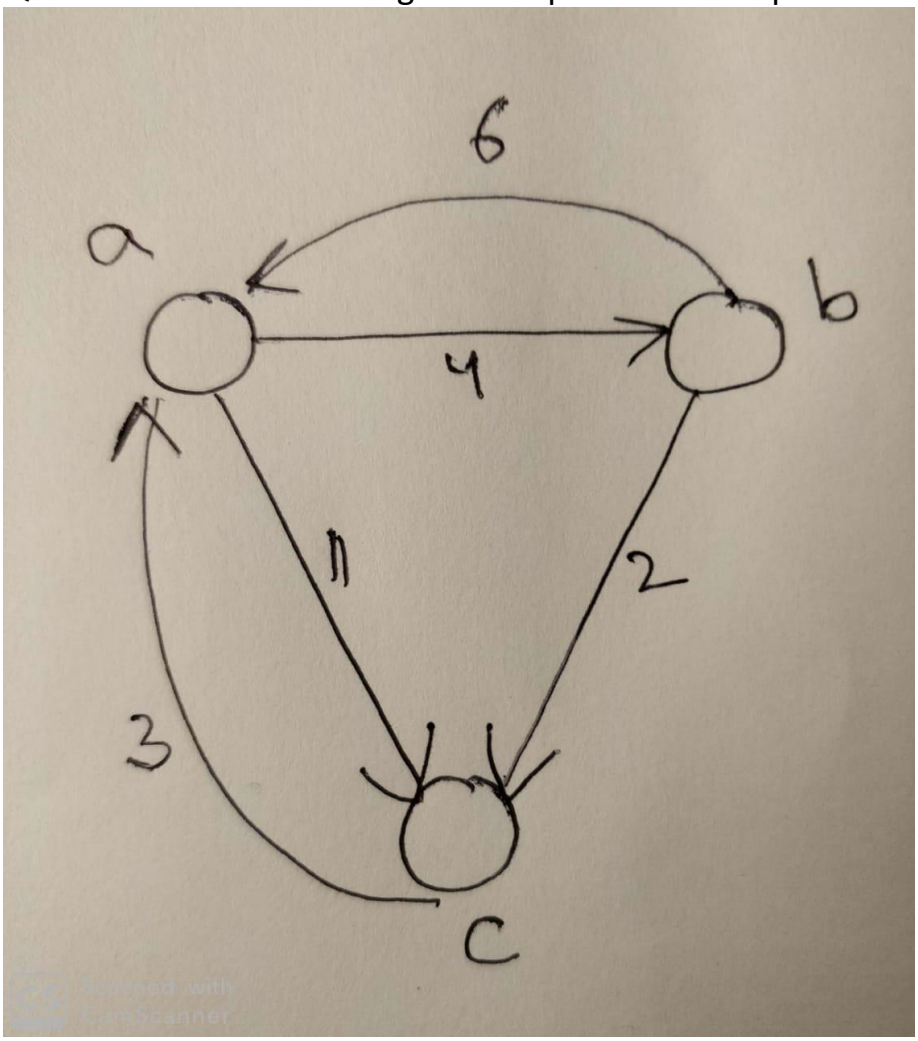
when multiplied as ((M1 X M2) X (M3 X M4)), the total number of multiplications is pqr + rst + prt. When multiplied as (((M1 X M2) X M3) X M4), the total number of scalar multiplications is pqr + prs + pst. If p = 10, q = 100, r = 20, s = 5 and t = 80, then the number of scalar multiplications needed is:

a) 248000 b) 44000 c) 19000 d) 25000

ANSWER C

Floyd warshall problem

Q Consider a Directed weighted Graph and find all pair shortest path?



$D^0 =$

	a	b	c
a			
b			
c			

$\prod^0 =$

	a	b	c
a			
b			
c			

$D^a =$

	a	b	c
a			
b			
c			

$\prod^a =$

	a	b	c
a			
b			
c			

$D^b = \begin{array}{c} \\ a \\ b \\ c \end{array} \begin{array}{ccc} a & b & c \\ & & \\ & & \\ & & \end{array}$ $\Pi^b = \begin{array}{c} \\ a \\ b \\ c \end{array} \begin{array}{ccc} a & b & c \\ & & \\ & & \\ & & \end{array}$

$D^c = \begin{array}{c} \\ a \\ b \\ c \end{array} \begin{array}{ccc} a & b & c \\ & & \\ & & \\ & & \end{array}$ $\Pi^c = \begin{array}{c} \\ a \\ b \\ c \end{array} \begin{array}{ccc} a & b & c \\ & & \\ & & \\ & & \end{array}$

Q The Floyd-Warshall algorithm for all-pair shortest paths computation is based on: (Gate-2016) (1 Marks)
(A) Greedy paradigm.
(B) Divide-and-Conquer paradigm.
(C) Dynamic Programming paradigm.
(D) neither Greedy nor Divide-and-Conquer nor Dynamic Programming paradigm.
Answer: (C)

Q Consider the weighted undirected graph with 4 vertices, where the weight of edge {i, j} g is given by the entry W_{ij} in the matrix W

$$W = \begin{bmatrix} 0 & 2 & 8 & 5 \\ 2 & 0 & 5 & 8 \\ 8 & 5 & 0 & x \\ 5 & 8 & x & 0 \end{bmatrix}$$

The largest possible integer value of x, for which at least one shortest path between some pair of vertices will contain the edge with weight x is _____ **(Gate-2016) (2 Marks)**
Answer: (12)

Q Let G (V, E) be a directed graph with n vertices. A path from vi to vj in G is sequence of vertices (v_i, v_{i+1}, ……., v_j) such that (v_k, v_{k+1}) ∈ E for all k in i through j – 1. A simple path is a path in which no vertex appears more than once.
Let A be an n*n array initialized as follow

$$A[j,k] = \begin{cases} 1 \text{ if } (j,k) \in E \\ 0 \text{ otherwise} \end{cases}$$

Consider the following algorithm.

```
for i = 1 to n
  for j = 1 to n
    for k = 1 to n
      A [j, k] = max (A[j, k] (A[j, i] + A [i, k]);
```

Which of the following statements is necessarily true for all j and k after terminal of the above algorithm? (Gate-2003) (2 Marks)

(A) $A[j, k] \leq n$

(B) If $A[j, k] \geq n - 1$, then G has a Hamiltonian cycle

(C) If there exists a path from j to k, A[j, k] contains the longest path lens from j to k

(D) If there exists a path from j to k, every simple path from j to k contain most A[j, k] edges

Answer: (D)

Sum of subset problem

- Given a set of non-negative integers, and a value *sum*, determine if there is a subset of the given set with sum equal to given *sum*.

Q Consider a set of non-negative integer S = {2, 3, 7, 8, 10}, find if there is a sub set of S with sum equal to 14?

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2															
3															
7															
8															
10															

Q The subset-sum problem is defined as follows. Given a set of n positive integers, S = {a_1, a_2 ,a_3 ,…,a_n} and positive integer W, is there a subset of S whose elements sum to W? A dynamic program for solving this problem uses a 2-dimensional Boolean array X, with n rows and W+1 column. X[i, j],1 <= i <= n, 0 <= j <= W, is TRUE if and only if there is a subset of {a1 ,a2 ,...,ai} whose elements sum to j. Which of the following is valid for 2 <= i <= n and ai <= j <= W? (Gate-2019) (2 Marks)

(A) X[i, j] = X[i – 1, j] V X[i, j -ai] **(B)** X[i, j] = X[i – 1, j] V X[i – 1, j – ai]

(C) X[i, j] = X[i – 1, j] V X[i, j – ai] **(D)** X[i, j] = X[i – 1, j] V X[i -1, j – ai]

Answer: (B)

Q In the above question, which entry of the array X, if TRUE, implies that there is a subset whose elements sum to W?

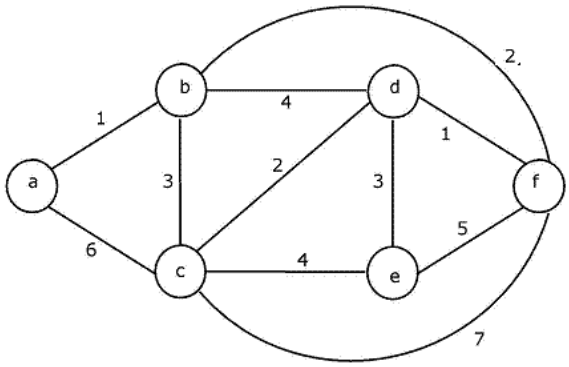
(A) X[1, W] **(B)** X[n ,0] **(C)** X[n, W] **(D)** X[n -1, n]

Answer: (C)

Kruskal

```
Minimum_Spanning_Tree (G, w)
{
    A ← ϕ
    For each vertex v ϵ V(G)
    {
        do Make_Set(v)
    }
    Sort the edges of E into non-decreasing order by weight w
    for each edge (u, v) ϵ E, then in non-decreasing order by
    weights
    {
        if (Find_Set(u) = Find_Set(v))
        {
                A ← A U {(u, v)}
                UNION (u, v)
        }
    }
    Return A
}
```

Q Consider the following graph:



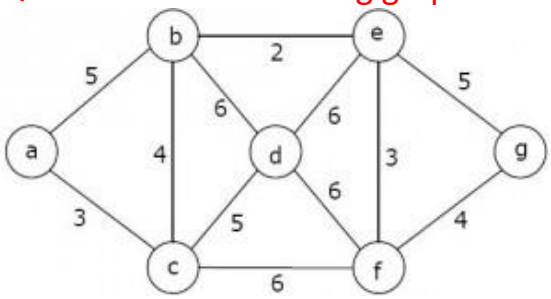
Which one of the following cannot be the sequence of edges added, in that order, to a minimum spanning tree using Kruskal's algorithm? (Gate-2006) (2 Marks)

(A) (a—b),(d—f),(b—f),(d—c),(d—e) **(B)** (a—b),(d—f),(d—c),(b—f),(d—e)

(C) (d—f),(a—b),(d—c),(b—f),(d—e) **(D)** (d—f),(a—b),(b—f),(d—e),(d—c)

Answer: (D)

Q Consider the following graph:



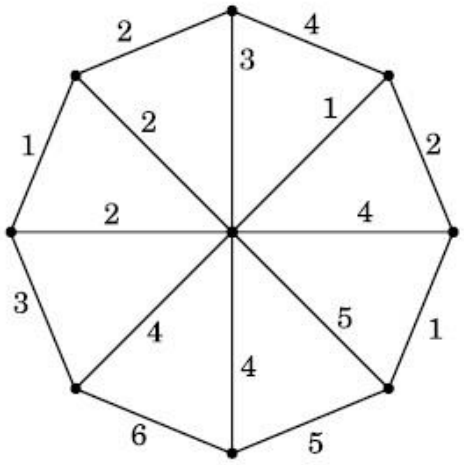
Which one of the following is NOT the sequence of edges added to the minimum spanning tree using Kruskal's algorithm? (Gate-2009)(2 ?Marks)

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

(C) (b, e),(a, c),(e, f),(b, c),(f, g),(c, d) **(D)** (b, e),(e, f),(b, c),(a, c),(f, g),(c, d)

Answer: (D)

Q Consider the graph shown below (NET-DEC-2018)



Use Kruskal's algorithm to find the minimum spanning tree of the graph. The weight of this minimum spanning tree is

a) 17 b) 14 c) 16 d) 13

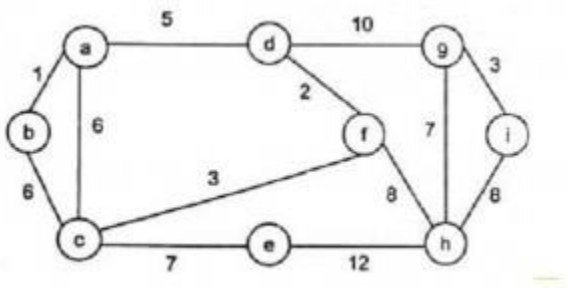
Prim's

Prim's algorithm: -

- Try to take an edge with minimum weight
- Try to expand you tree with minimum cost neighbour.
- Never form a cycle, keep repeating the process until all the nodes are included

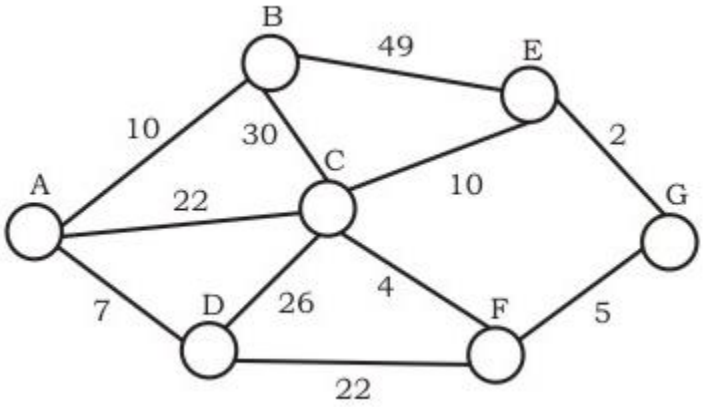
```
Minimum_Spanning_Tree (G, W, R)
{
    for each u ϵ V(G)
    {
        key[u] ← ∞
        ∏[u] ← NIL
    }
    Key[r] ← 0
    Q ←V[G]
    While (Q != ϕ)
    {
        u ← Extract-Min(Q)
        For each v ϵ adj[u]
        {
            if (v ϵ Q and w(u, v) < key[v])
            {
                ∏[v] ← u
                key[u] ← w(u, v)
            }
        }
    }
}
```

Q For the undirected, weighted graph given below, which of the following sequences of edges represents a correct execution of Prim's algorithm to construct a Minimum Spanning Tree? (Gate-2008) (2 Marks)



(A) (a, b), (d, f), (f, c), (g, i), (d, a), (g, h), (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)
Answer: (C)

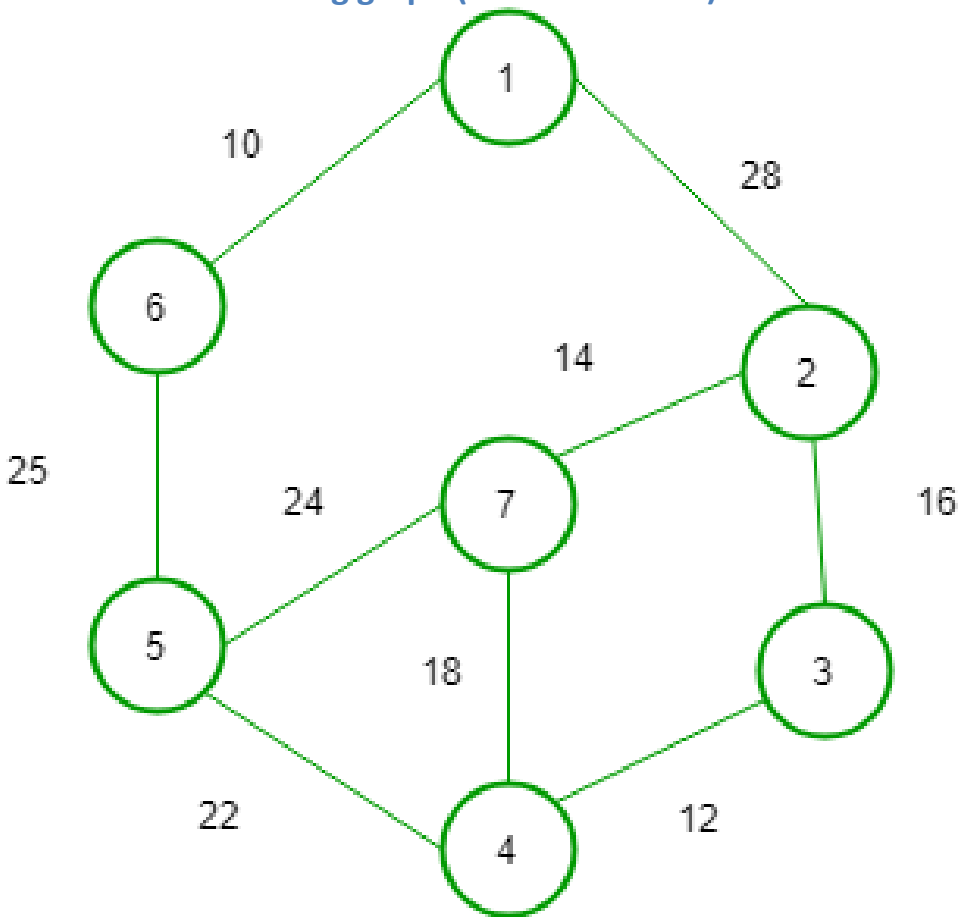
Q Consider the undirected graph below:



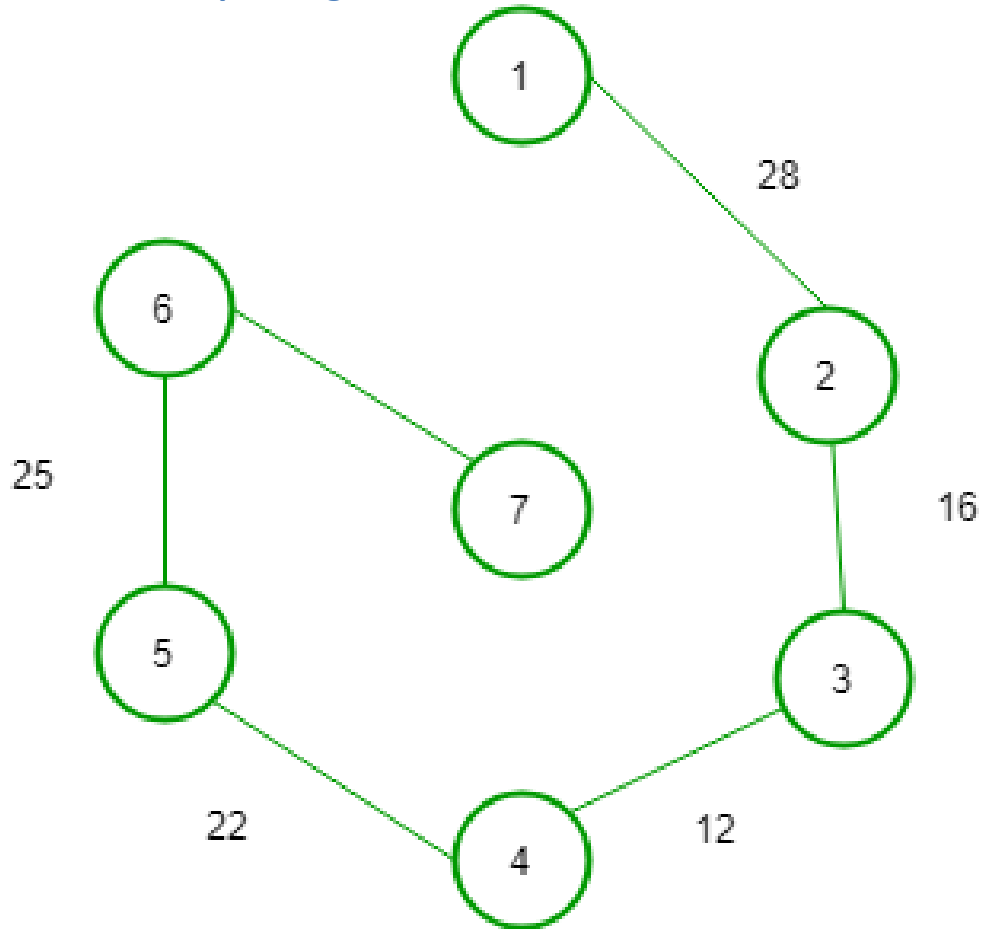
Using Prim's algorithm to construct a minimum spanning tree starting with node A, which one of the following sequences of edges represents a possible order in which the edges would be added to construct the minimum spanning tree? (Gate-2004) (2 Marks)
(A) (E, G), (C, F), (F, G), (A, D), (A, B), (A, C)
(B) (A, D), (A, B), (A, C), (C, F), (G, E), (F, G)
(C) (A, B), (A, D), (D, F), (F, G), (G, E), (F, C)
(D) (A, D), (A, B), (D, F), (F, C), (F, G), (G, E)
Answer: (D)

General

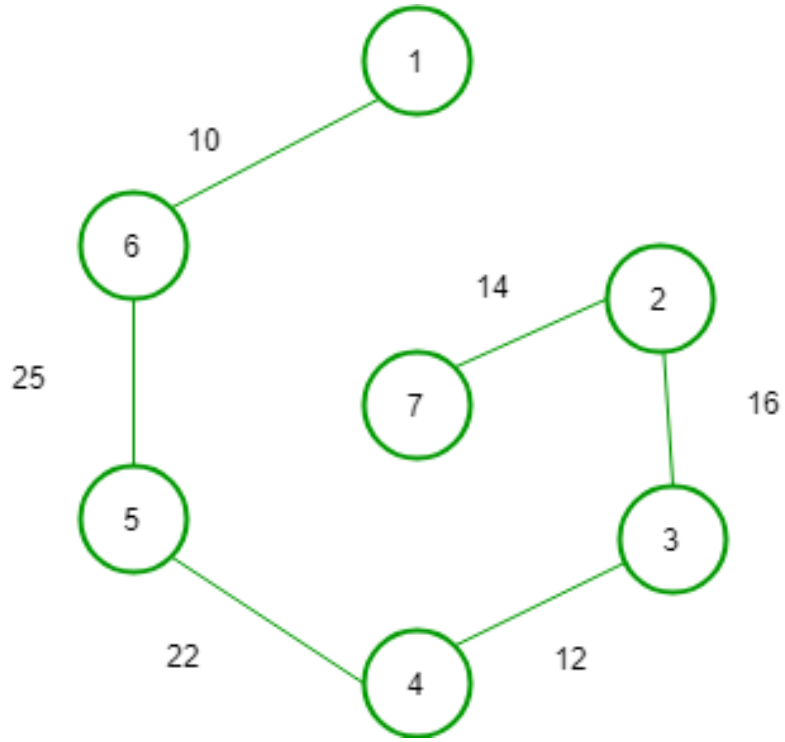
Q Consider the following graph (NET-JUNE-2015)



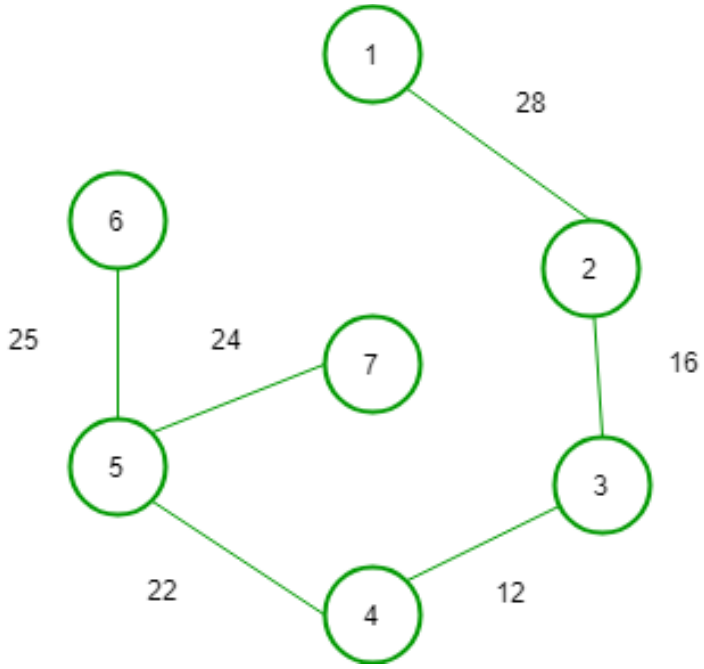
Its minimum cost spanning tree is -



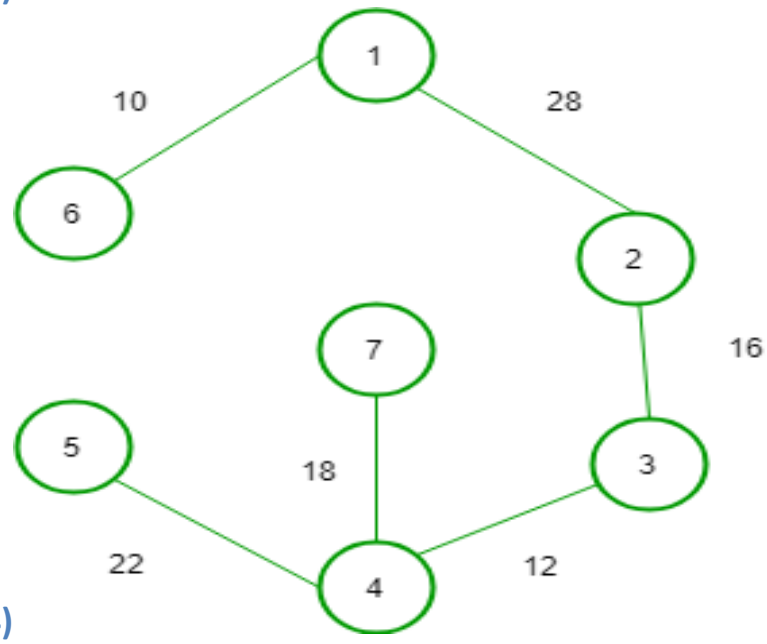
(1)


1
10
6
14
2
25
7
16
5
3
22
12
4

(2)

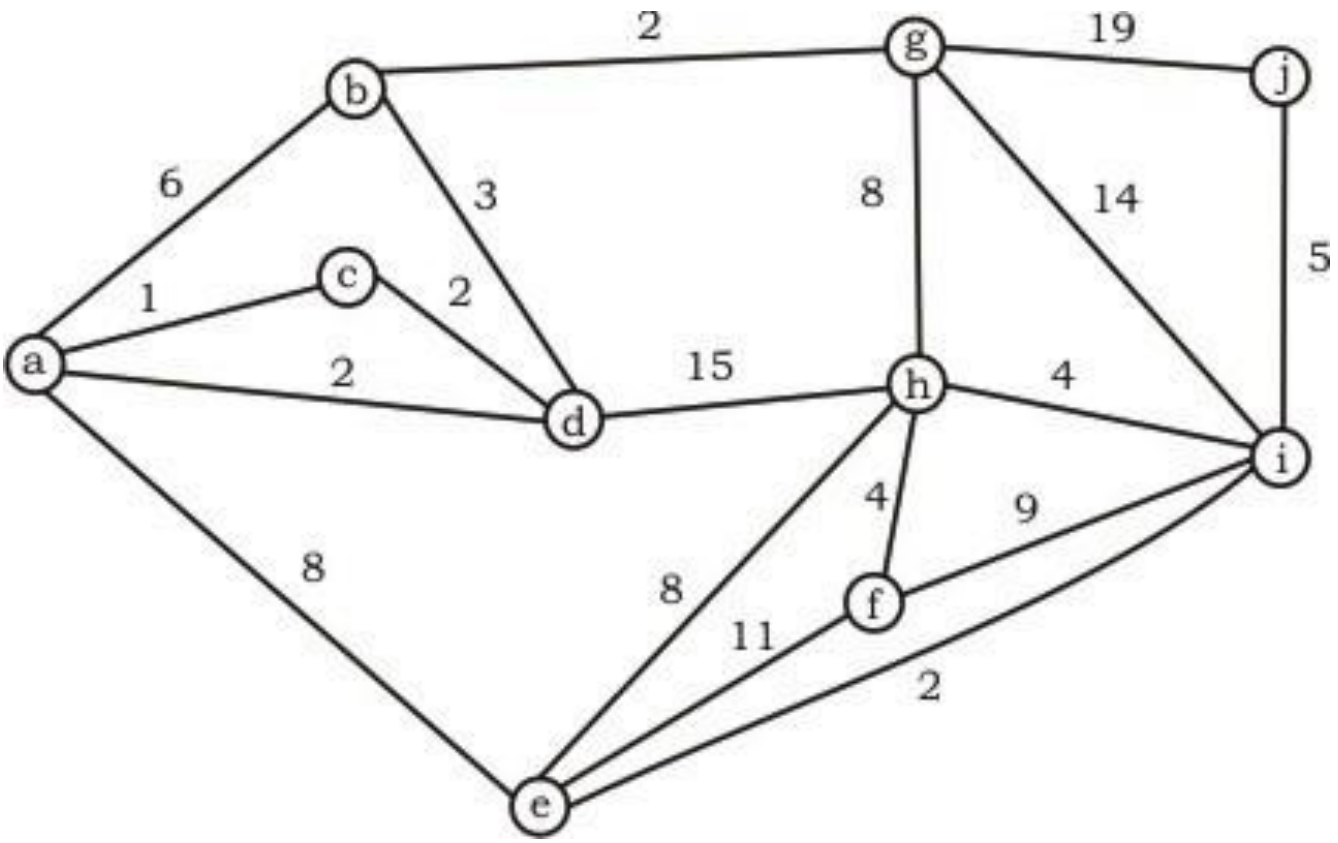

1
28
6
2
25
24
7
16
5
3
22
12
4

(3)


1
10
28
6
2
7
16
5
18
3
22
12
4

(4)

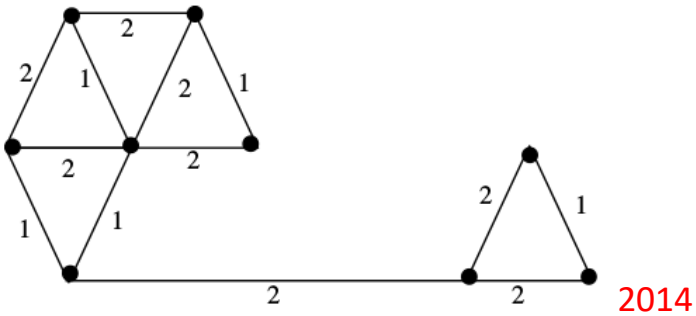
Q What is the weight of a minimum spanning tree of the following graph? (Gate-2003) (2 Marks)



(A) 29 **(B)** 31 **(C)** 38 **(D)** 41

Answer: (B)

Q The number of distinct minimum spanning trees for the weighted graph below is ____ (GATE-2017) (2 Marks)



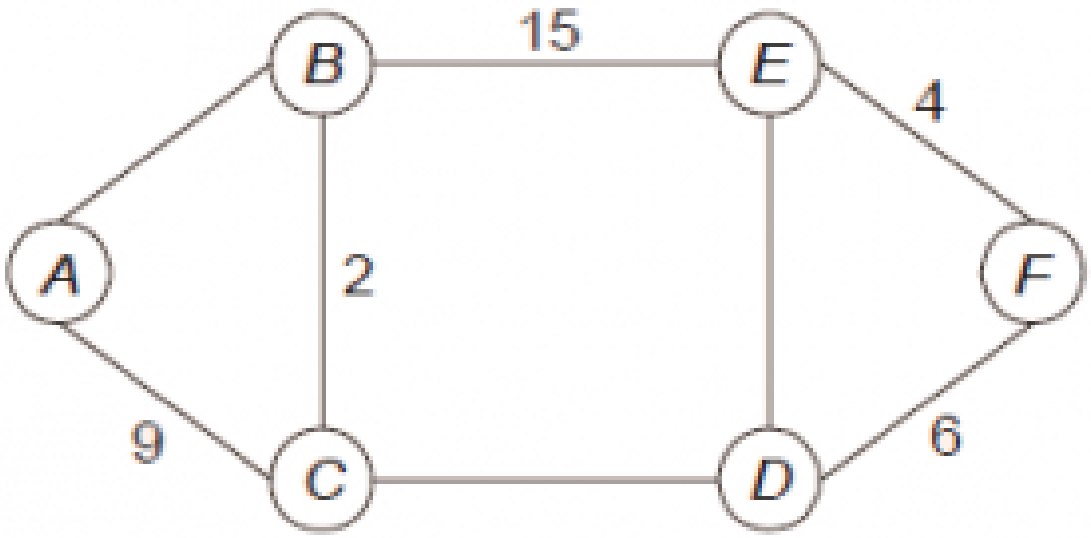
2014

Answer: (6)

Q Let G be connected undirected graph of 100 vertices and 300 edges. The weight of a minimum spanning tree of G is 500. When the weight of each edge of G is increased by five, the weight of a minimum spanning tree becomes _____. (GATE-2015) (2 Marks)

Answer: (995)

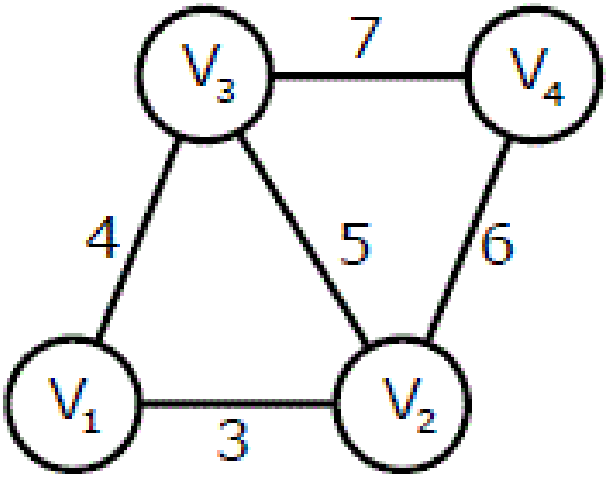
Q The graph shown below 8 edges with distinct integer edge weights. The minimum spanning tree (MST) is of weight 36 and contains the edges: {(A, C), (B, C), (B, E), (E, F), (D, F)}. The edge weights of only those edges which are in the MST are given in the figure shown below. The minimum possible sum of weights of all 8 edges of this graph is _____. (Gate - 2015) (2 Marks)



Answer: (69)

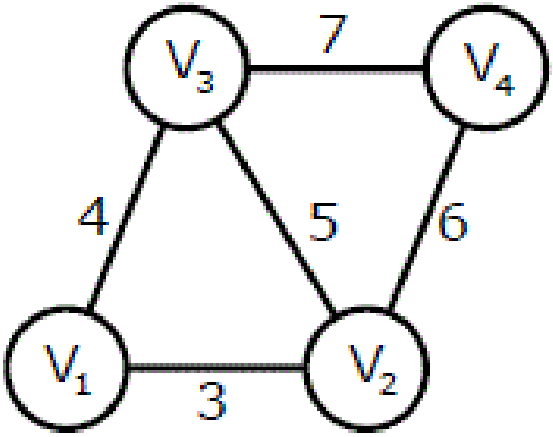
Q An undirected graph G has n nodes. Its adjacency matrix is given by an n × n square matrix whose (i) diagonal elements are 0‘s and (ii) non-diagonal elements are 1‘s. which one of the following is TRUE? (Gate-2005) (2 Marks)
(A) Graph G has no minimum spanning tree (MST)
(B) Graph G has a unique MST of cost n-1
(C) Graph G has multiple distinct MSTs, each of cost n-1
(D) Graph G has multiple spanning trees of different costs
Answer: (C)

Q An undirected graph G(V, E) contains n (n > 2) nodes named v1 , v2 ,….vn. Two nodes vi , vj are connected if and only if 0 < |i – j| <= 2. Each edge (vi, vj) is assigned a weight i + j. A sample graph with n = 4 is shown below. What will be the cost of the minimum spanning tree (MST) of such a graph with n nodes? (GATE - 2011) (2 Marks)



(A) 1/12(11n^2 – 5n) **(B)** n^2 – n + 1 **(C)** 6n – 11 **(D)** 2n + 1
Answer: (B)

Q The length of the path from v5 to v6 in the MST of previous question with n = 10 is (GATE - 2011) (2 Marks)



(A) 11 **(B)** 25 **(C)** 31 **(D)** 41
Answer: (C)

Q Consider a complete undirected graph with vertex set {0, 1, 2, 3, 4}. Entry Wij in the matrix W below is the weight of the edge {i, j}. What is the minimum possible weight of a spanning tree T in this graph such that vertex 0 is a leaf node in the tree T? (GATE - 2010) (2 Marks)

$$W = \begin{pmatrix} 0 & 1 & 8 & 1 & 4 \\ 1 & 0 & 12 & 4 & 9 \\ 8 & 12 & 0 & 7 & 3 \\ 1 & 4 & 7 & 0 & 2 \\ 4 & 9 & 3 & 2 & 0 \end{pmatrix}$$

(A) 7 **(B)** 8 **(C)** 9 **(D)** 10
Answer: (D)

Q In the graph given in above question, what is the minimum possible weight of a path P from vertex 1 to vertex 2 in this graph such that P contains at most 3 edges? (GATE - 2010) (2 Marks)
(A) 7 **(B)** 8 **(C)** 9 **(D)** 10
Answer: (B)

Q Let G be a weighted connected undirected graph with distinct positive edge weights. If every edge weight is increased by the same value, then which of the following statements is/are TRUE? (Gate-2016) (2 Marks)
P: Minimum spanning tree of G does not change
Q: Shortest path between any pair of vertices does not change
(A) P only (B) Q only (C) Neither P nor Q (D) Both P and Q
Answer: (A)

Q G = (V, E) is an undirected simple graph in which each edge has a distinct weight, and e is a particular edge of G. Which of the following statements about the minimum spanning trees (MSTs) of G is/are TRUE (Gate-2016) (2 Marks)
I. If e is the lightest edge of some cycle in G, then every MST of G includes e
II. If e is the heaviest edge of some cycle in G, then every MST of G excludes e
(A) I only (B) II only (C) both I and II (D) neither I nor II
Answer: (B)

Q Consider a weighted complete graph G on the vertex set $\{v_1, v_2, \ldots..v_n\}$ such that the weight of the edge (v_i, v_j) is $2|i-j|$. The weight of a minimum spanning tree of G is: (GATE - 2006)
(A) n — 1 **(B)** 2n — 2 **(C)** ${}^nC_2$ **(D)** 2
Answer: (B)
Explanation: Weight of the minimum spanning tree

$= 2|2-1| + 2|3-2| + 2|4-3| + 2|5-4| \ldots + 2|n-(n-1)|$
$= 2n - 2$

Q Let G be a complete undirected graph on 4 vertices, having 6 edges with weights being 1, 2, 3, 4, 5, and 6. The maximum possible weight that a minimum weight spanning tree of G can have is. (Gate-2016) (2 Marks)
Answer: (7)

Q An undirected graph G (V, E) contains n (n > 2) nodes named $v_1, v_2 \ldots, v_n$. Two nodes vi and Vj are connected if and only if $0 < |i-j| \leq 2$. Each edge (vi, vj) is assigned a weight i + j. The cost of the minimum spanning tree of such a graph with 10 nodes is: (NET-NOV-2017)
(1) 88 (2) 91 (3) 49 (4) 21

Q Consider a weighted complete graph G on the vertex set $\{v_1, v_2, \ldots v_n\}$ such that the weight of the edge (v_i, v_j) is $4|i-j|$. The weight of minimum cost spanning tree of G is: (NET-JULY-2016)
(1) $4n^2$ (2) n (3) 4n – 4 (4) 2n – 2
Ans C

Q Let G be an undirected connected graph with distinct edge weight. Let e_{max} be the edge with maximum weight and e_{min} the edge with minimum weight. Which of the following statements is false? (Gate-2000) (2 Marks) (NET-NOV-2017)
(A) Every minimum spanning tree of G must contain e_{min}
(B) If e_{max} is in a minimum spanning tree, then its removal must disconnect G
(C) No minimum spanning tree contains e_{max}
(D) G has a unique minimum spanning tree
Answer: (C)

Q Let G be a weighted graph with edge weights greater than one and G'be the graph constructed by squaring the weights of edges in G. Let T and T' be the minimum spanning trees of G and G', respectively, with total weights t and t'. Which of the following statements is TRUE? (Gate-2012) (2 Marks)
(A) T' = T with total weight $t' = t^2$ **(B)** T' = T with total weight $t' < t^2$
(C) T' != T but total weight $t' = t^2$ **(D)** None of the above
Answer: (D)

Q Let s and t be two vertices in a undirected graph G(V, E) having distinct positive edge weights. Let [X, Y] be a partition of V such that $s \in X$ and $t \in Y$. Consider the edge e having the minimum weight amongst all those edges that have one vertex in X and one vertex in Y, The edge e must definitely belong to: (Gate-2005) (2 Marks)
(A) the minimum weighted spanning tree of G

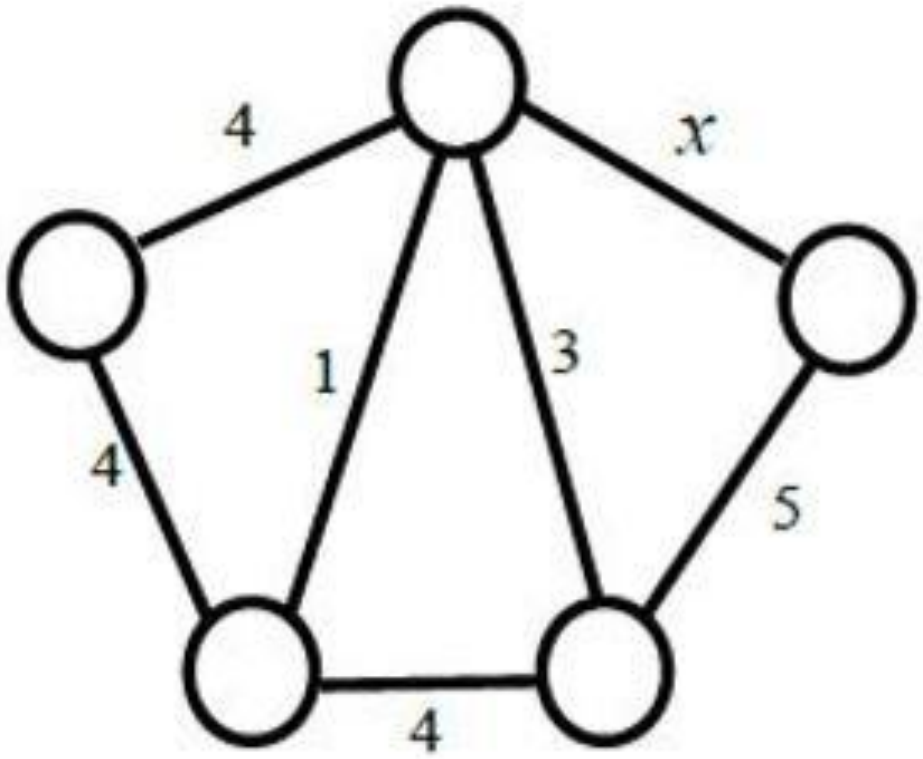
(B) the weighted shortest path from s to t
(C) each path from s to t
(D) the weighted longest path from s to t
Answer: (A)

Q Let w be the minimum weight among all edge weights in an undirected connected graph. Let e be a specific edge of weight w . Which of the following is FALSE? (Gate-2007) (2 Marks)
(A) There is a minimum spanning tree containing e.
(B) If e is not in a minimum spanning tree T, then in the cycle formed by adding e to T, all edges have the same weight.
(C) Every minimum spanning tree has an edge of weight w .
(D) e is present in every minimum spanning tree.
Answer: (D)

Q Let G be a weighted undirected graph and e be an edge with maximum weight in G. Suppose there is a minimum weight spanning tree in G containing the edge e. Which of the following statements is always TRUE? (Gate-2005)(2 Marks)
(A) There exists a cut set in G having all edges of maximum weight.
(B) There exists a cycle in G having all edges of maximum weight
(C) Edge e cannot be contained in a cycle.
(D) All edges in G have the same weight
Answer: (A)

Q Let G be any connection, weighted, undirected graph:
I. G has a unique minimum spanning tree if no two edges of G have the same weight.
II. G has a unique minimum spanning tree if, for every cut G, there is a unique minimum weight edge crossing the cut.
Which of the above two statements is/are TRUE? (Gate-2019) (2 Marks)
a) Neither I nor II b) I only c) II only d) Both I and II
Ans: C

Q Consider the following undirected graph



Choose a value for x that will maximize the number of minimum weights spanning trees (MWSTs) of G. The number of MWSTs of G for this value of x is _____. (Gate-2018) (2 Marks)
Ans: 4

Q Let G = (V, E) be any connected undirected edge-weighted graph. The weights of the edges in E are positive any distinct. Consider the following statements: (Gate-2017) (2 Marks)
I. Minimum Spanning Tree of G is always unique.
II. Shortest path between any two vertices of G is always unique.
Which of the above statements is/are necessarily true?
a) I only b) II only c) both I and II d) neither I and II
ANSWER A

Q What is the largest integer m such that every simple connected graph with n vertices and n edges contains at least m different spanning trees? (Gate-2007) (2 Marks)
(A) 1 **(B)** 2 **(C)** 3 **(D)** n
Answer: (C)

Q Which of the following is true about Kruskal and Prim MST algorithms? Assume that Prim is implemented for adjacency list representation using Binary Heap and Kruskal is implemented using union by rank.
(A) Worst case time complexity of both algorithms is same.

(B) Worst case time complexity of Kruskal is better than Prim
(C) Worst case time complexity of Prim is better than Kruskal
Answer: (A)

Q Which of the following algorithms can be used to most efficiently determine the presence of a cycle in a given graph?
(A) Depth First Search
(B) Breadth First Search
(C) Prim's Minimum Spanning Tree Algorithm
(D) Kruskal' Minimum Spanning Tree Algorithm
Answer: (A)

Q If all the edge weights of an undirected graph are positive, then any subset of edges that connects all the vertices and has minimum total weight is a **(GATE-2006) (1 Marks)**
(A) Hamiltonian cycle
(B) grid
(C) hypercube
(D) tree
Answer: (D)

Single Source Shortest Path

- In graph theory, the **shortest path problem** is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized.

Dijkstra's algorithm (or Dijkstra's Shortest Path First algorithm, SPF algorithm)

- Is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later.
- One morning I was shopping in Amsterdam with my young fiancée, and tired, we sat down on the café terrace to drink a cup of coffee and I was just thinking about whether I could do this, and I then designed the algorithm for the shortest path.
- As I said, it was a twenty-minute invention. In fact, it was published in ’59, three years later. The publication is still readable, it is, in fact, quite nice.
- One of the reasons that it is so nice was that I designed it without pencil and paper. I learned later that one of the advantages of designing without pencil and paper is that you are almost forced to avoid all avoidable complexities.
- Eventually that algorithm became, to my great amazement, one of the cornerstones of my fame.
- A widely used application of shortest path algorithm is network routing protocols, most notably IS-IS (Intermediate System to Intermediate System) and Open Shortest Path First (OSPF).
- Dijkstra's algorithm uses a data structure for storing and querying partial solutions sorted by distance from the start. The original algorithm uses a min-priority queue and runs in time $O(V^2)$(where V is the number of nodes)

```
Dijkstra algorithm (G, W, S)
{
    initialize-Single-source (G, S)
    S ← ϕ
    Q ← V[G]
    While (Q != ϕ)
    {
        u ← extract-min (Q)
        S ← S U {u}
        for each vertex v ϵ adj(u)
        {
            relax (u, v, w)
        }
    }
}
```

```
Initialize_Single_Source (G, S)
{
    for each vertex v ϵ V[G]
    {
        d[v] ← ∞
        ∏[v] ← NIL
    }
    d[S] ← 0
}
```

```
Relax (u, v, w)
{
    if(d[v] ← d[u] + w (u, v))
    {
        d[v] ← d[u] + w (u, v)
        ∏[v] ← u
    }
}
```

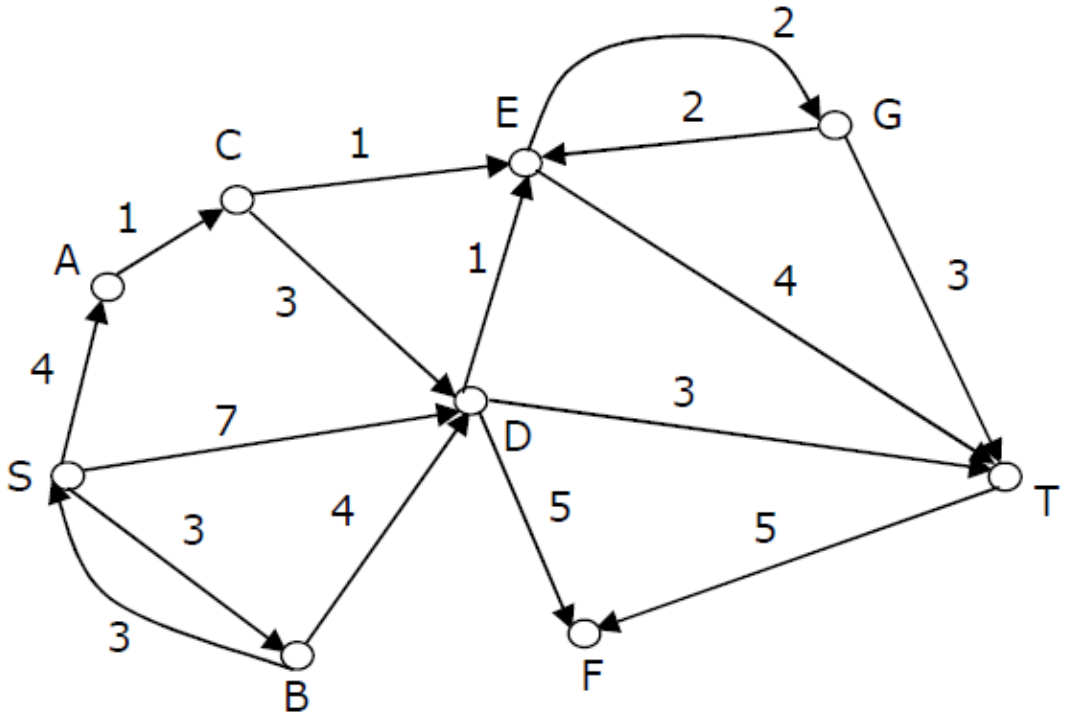
- Guarantee to find optimal solution in a connected graph with positive weights.
- Can fail on graph with negative weights.

Q Which of the following algorithms solves the single-source shortest paths? (NET-JULY-2018)

(1) Prim's algorithm
(2) Floyd - Warshall algorithm
(3) Johnson's algorithm
(4) Dijkstra's algorithm

Ans: 4

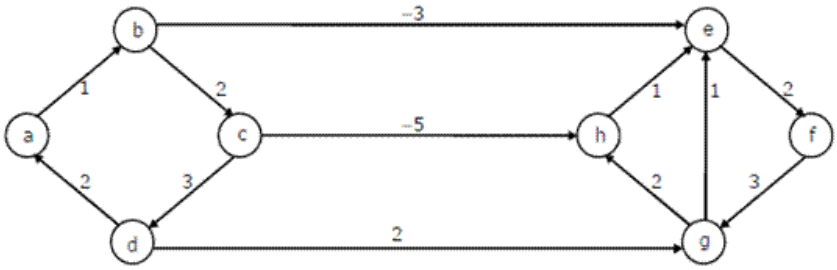
Q Consider the directed graph shown in the figure below. There are multiple shortest paths between vertices S and T. Which one will be reported by Dijkstra's shortest path algorithm? Assume that, in any iteration, the shortest path to a (Gate-2012) (2 Marks)



a) SDT b) SBDT c) SACDT d) SACET

Ans: d

Q Dijkstra's single source shortest path algorithm when run from vertex a in the below graph, computes the correct shortest path distance to (Gate-2008) (2 Marks)

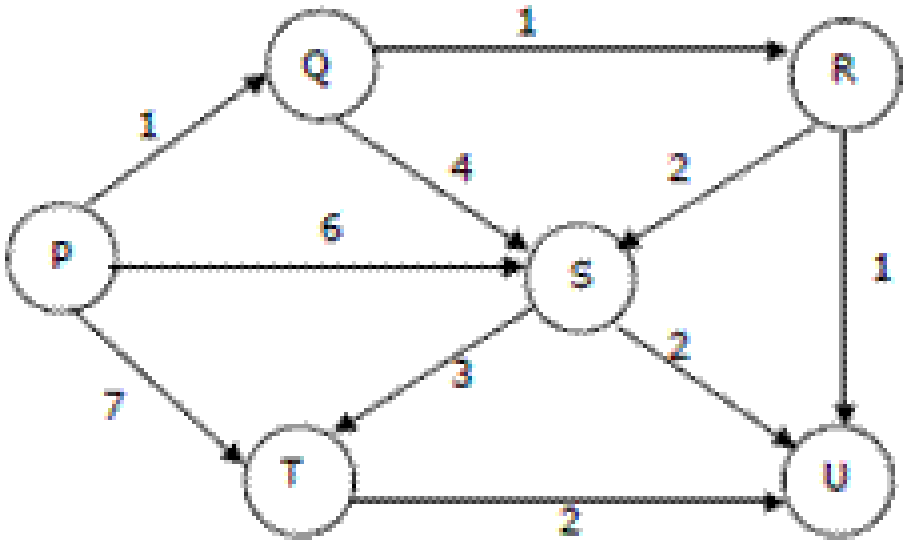


(A) only vertex a **(B)** only vertices a, e, f, g, h

(C) only vertices a, b, c, d **(D)** all the vertices

Ans: d

Q Suppose we run Dijkstra's single source shortest-path algorithm on the following edge weighted directed graph with vertex P as the source. In what order do the nodes get included into the set of vertices for which the shortest path distances are finalized? (GATE - 2004) (2 Marks)



(A) P, Q, R, S, T, U
(B) P, Q, R, U, S, T
(C) P, Q, R, U, T, S
(D) P, Q, T, R, U, S
Ans: b

Q To implement Dijkstra's shortest path algorithm on unweighted graphs so that it runs in linear time, the data structure to be used is: (Gate-2006) (1 Marks)
(A) Queue **(B)** Stack **(C)** Heap **(D)** B-Tree
Ans: a

Q Let G(V, E) an undirected graph with positive edge weights. Dijkstra's single-source shortest path algorithm can be implemented using the binary heap data structure with time complexity: (Gate-2005) (2 Marks)
(A) O(| V |2)
(B) O (| E | + | V | log | V |)
(C) O (| V | log | V |)
(D) O ((| E | + | V |) log | V |)
Ans: d

Bellman–Ford algorithm

- The **Bellman–Ford algorithm** is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph.
- It is slower than Dijkstra's algorithm for the same problem, but more versatile, as it is capable of handling graphs in which some of the edge weights are negative numbers.
- The algorithm was first proposed by Alfonso Shimbel (1955), but is instead named after Richard Bellman and Lester Ford Jr., who published it in 1958 and 1956, respectively.

```
Bellman_ford (G, W, S)
{
    initialize-Single-Source (G, S)
    for i ← 1 to |V(G)| - 1
    {
        for each edge (u, v) ϵ E(G)
        {
            Relax(u, v, w)
        }
    }
    for each edge (u, v) ϵ E(G)
    {
        if(d[v] ← d[u] + w (u, v))
            {
                Return false
            }
    }
}
```

```
Initialize_Single_Source (G, S)
{
    for each vertex v ϵ V[G]
    {
        d[v] ← ∞
        ∏[v] ← NIL
    }
    d[S] ← 0
}
```

```
Relax (u, v, w)
{
    if(d[v] ← d[u] + w (u, v))
    {
        d[v] ← d[u] + w (u, v)
        ∏[v] ← u

    }
}
```

Q Which of the following statement(s) is/are correct regarding Bellman-Ford shortest path algorithm? (Gate-2009) (1 Marks)

P: Always finds a negative weighted cycle, if one exists.

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

a) P only b) Q only c) Both P and Q d) Neither P nor Q

Q What is the time complexity of Bellman-Ford single-source shortest path algorithm on a complete graph of n vertices? (Gate-2013) (1 Marks)

a) $\Theta(n^3)$ **b)** $\Theta(n^2)$ **c)** $\Theta(n^2 \log n)$ **d)** $\Theta(n^3 \log n)$

Ans: a

Q An all-pairs shortest-paths problem is efficiently solved using: (NET-JUNE-2018)

a) Dijkstra' algorithm
b) Bellman-Ford algorithm
c) Kruskal algorithm
d) Floyd-Warshall algorithm

Q Consider the following table

Algorithms	Design Paradigms
(P) Kruskal	(i) Divide and Conquer
(Q) Quicksort	(ii) Greedy
(R) Floyd-Warshall	(iii) Dynamic Programming

Match the algorithm to design paradigms they are based on: (Gate-2017) (2 Marks)

a) P-(ii), Q-(iii), R-(i)
b) P-(iii), Q-(i), R-(ii)
c) P-(ii), Q-(i), R-(iii)
d) P-(i), Q-(ii), R-(iii)

Ans: c

Q Match the following (Gate-2015) (2 Marks)

List-I	List-II
A. Prim's algorithm for minimum spanning tree	1. Backtracking
B. Floyd-Warshall algorithm for all pairs shortest paths	2. Greed method
C. Merge sort	3. Dynamic programming
D. Hamiltonian circuit	4. Divide and conquer

Codes:

	A	B	C	D
a)	3	2	4	1
b)	1	2	4	3
c)	2	3	4	1
d)	2	1	3	4

Ans: c

Q Consider a weighted undirected graph with positive edge weights and let (u, v) be an edge in the graph. It is known that the shortest path from source vertex s to u has weight 53 and shortest path from s to v has weight 65. Which statement is always true? (NET-JUNE-2012)

(A) Weight (u, v) < 12
(B) Weight (u, v) = 12
(C) Weight (u, v) > 12
(D) Weight (u, v) > 12

Q In an unweighted, undirected connected graph, the shortest path from a node S to every other node is computed most efficiently, in terms of time complexity by (Gate-2007) (2 Marks)

(A) Dijkstra's algorithm starting from S.
(B) Warshall's algorithm
(C) Performing a DFS starting from S.
(D) Performing a BFS starting from S.

Ans: d

Q Consider the tree arcs of a BFS traversal from a source node W in an unweighted, connected, undirected graph. The tree T formed by the tree arcs is a data structure for computing. (Gate-2014) (2 Marks)

a) the shortest path between every pair of vertices.

b) the shortest path from W to every vertex in the graph.

c) the shortest paths from W to only those nodes that are leaves of T.

d) the longest path in the graph

Ans: b