Design Analysis Of Algorithms

UNIT - 1

Introduction To Algorithm

1. Define algorithm?

An Algorithm is a finite sequence of instructions, each of which has a clear meaning and can be performed with a finite amount of effort in a finite length of time.

2. List and explain characteristics of algorithm?

1. Input: There are zero or more quantities, which are externally supplied.

2. Output: At least one quantity is produced..

3. Definiteness: Each instruction must be clear and unambiguous.

4. Finiteness: If we trace out the instructions of an algorithm, then for all cases the algorithm will terminate after a finite number of steps;

5. Effectiveness: Every instruction must be basic so that it can be carried out by a person using only pencil and paper.

3. Define debugging?

Debugging is the process of executing programs on sample data sets to determine whether faulty results occur.

4. Define pseudocode?

Pseudo code is compact and informal high level description of Program.

5. Write algorithm to find and return the sum of given numbers in array?

```
Algorithm Sum(a, n) {
        s := 0.0;
        for i := 1 to n do
                s := s+ a[i];
        return s;
}
```

6. Define space complexity?

The space complexity of an algorithm is the amount of memory it needs to run to completion of Algorithm.

Space for simple variable and fixed size variable is known as **aggregate**.

7. Define time complexity?

The time complexity of an algorithm is the amount of computer time it needs to run to completion of Algorithm.

8. Explain asymptotic notations?

The value of the function increase or decrease as the n value increase. The asymptotic behavior of a function is the study of how the value of a function varies for large value of n of where n is the size of the input.

- **Worst-case:**

  f (n) defined by the maximum number of steps taken on any instance of size n.

- **Best-case:**

  f (n) defined by the minimum number of steps taken on any instance of size n.

- **Average case:**

  f (n) defined by the average number of steps taken on any instance of size n.

UNIT - 2

Design And Analysis Of Algorithms

1. Define recurrence relation?

A recurrence relation is an equation that defines a sequence based on a rule that gives the next term as a function of the previous term(s).

2. Define asymptotic notation and list its types?

Asymptotic notation is a notation, which is used to take meaningful statement about the efficiency of a program.

Types -

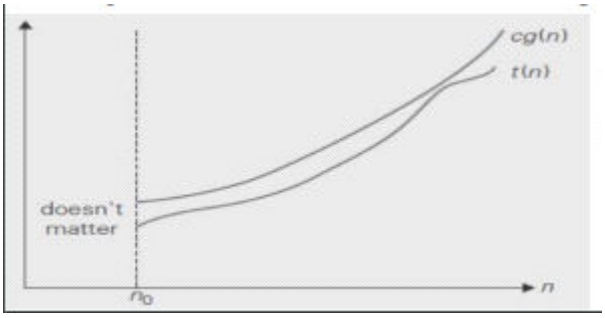
- O - Big oh notation
- Ω - Big omega notation
- Θ - Big theta notation

3. Explain the asymptotic notations?

1. O - Big oh notation :

Big-O notation represents the upper bound of the running time of an algorithm. It gives the worst-case complexity of an algorithm.

It specifies the upper bound of a function.The maximum time required by an algorithm or the worst-case time complexity.



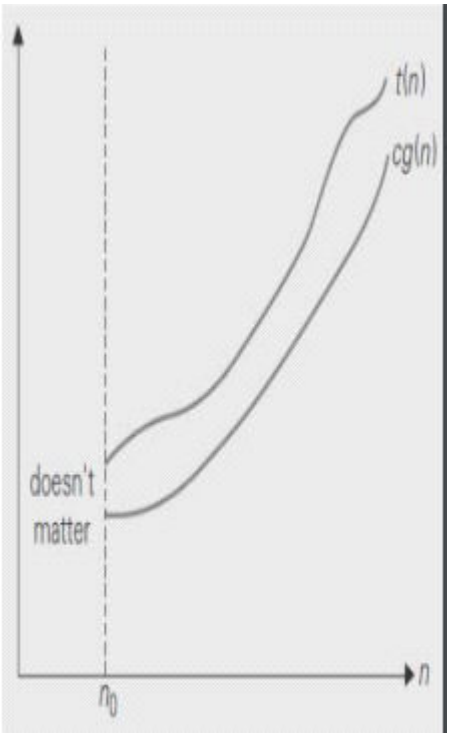
A function t(n) is said to be in O(g(n)), if t(n) is bounded above by some constant multiple of g(n) for all large n, i.e if there exist some positive constant c and some nonnegative integer n0 such that,

$$t(n) <= c.g(n) \text{ for all } n >= n0$$

- Where t(n) and g(n) are non-negative functions defined on the set of natural numbers.
- O = Asymptotic upper bound = Usefull for worst-case analysis = Loose bound

2. Ω - Big omega notation :

- Omega notation represents the lower bound of the running time of an algorithm. It provides the best case complexity of an algorithm.
- The execution time serves as a lower bound on the algorithm's time complexity.
- It is defined as the condition that allows an algorithm to complete statement execution in the shortest amount of time.



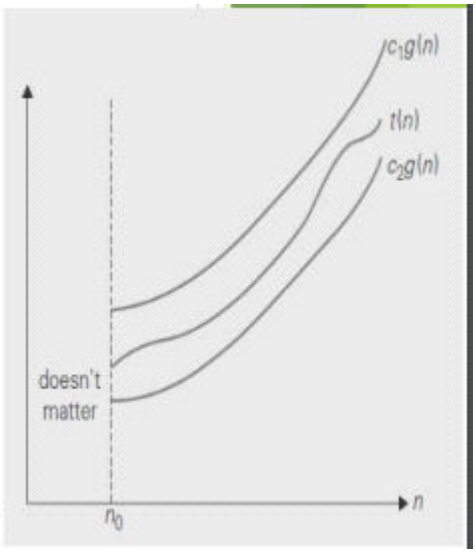
A function t(n) is said to be in **Ω**(g(n)), if t(n) is bounded below by some constant multiple of g(n) for all large n, i.e if there exist some positive constant c and some nonnegative integer n0 such that,

$$t(n) >= c.g(n) \text{ for all } n >= n0$$

- Where t(n) and g(n) are non-negative functions defined on the set of natural numbers.
- **Ω**= Asymptotic upper bound = Usefull for best-case analysis = Loose bound

3. Θ - Big theta notation :

Theta notation represents the upper and the lower bound of the running time of an algorithm, it is used for analyzing the average-case complexity of an algorithm.



A function t(n) is said to be in **Θ**(g(n)), if t(n) is bounded both above and below by some constant multiple of g(n) for all large n, i.e if there exist some positive constant c1 and c2 and some nonnegative integer n0 such that,

c2g(n) <= t(n) <= c1g(n) for all n >= n0

- Where t(n) and g(n) are non-negative functions defined on the set of natural numbers.
- **Ω**= Asymptotic tight bound = Usefull for average-case analysis

4. Define recursive algorithm?

A recursive algorithm calls itself with smaller input values and returns the result for the current input by carrying out basic operations on the returned value for the smaller input.

5. List the basic asymptotic efficiency classess?

1	constant	Outside best-case, few examples
$\log n$	logarithmic	Algorithms that decrease by a constant
n	linear	Algorithms that scan an n-sized list
$n \log n$	$n \log n$	Algorithms that divide and conquer, e.g. quicksort
n^2	quadratic	Typically two embedded loops
n^3	cubic	Typically three embedded loops
2^n	exponential	Algorithms that generate all subsets of an n-element list
$n!$	factorial	Algorithms that generate all permutations of an n-element list

6. Give general plan for non-recursive algorithms?

- Decide on parameter indicating an input's size.
- Identify the algorithm's basic operation
- Checking the no. of times basic operation executed depends on size of input. If it depends on some additional property, then best, worst, avg. cases need to be investigated
- Set up sum expressing the no. of times the basic operation is executed. (establishing order of growth).

7. Give general plan for recursive algorithms?

- Decide on parameter indicating an input's size.
- Identify the algorithm's basic operation
- Checking the no. of times basic operation executed depends on size of input. If it depends on some additional property, then best, worst, avg. cases need to be investigated

- Set up the recurrence relation, with an appropriate initial condition, for the number of times the basic operation is executed
- Solve recurrence (establishing order of growth)

8. Write algorithm to find the number of binary digits in the binary representation of a positive decimal integer?

Algorithm Binary(n)

//Input: A positive decimal integer n

//Output: The number of binary digits in n's binary representation

count ←1

while n > 1 do

count ← count + 1

n←[n/2]

return count

9. Define algorithm validation?

The process of measuring the effectiveness of the algorithm before actually making program or code from it, in order to know whether the algorithm is correct for valid input is known as algorithm validation.

10. Write the algorithm to find the largest element in a given array?

ALGORITHM -

MaxElement(A[0..n − 1])

//Determines the value of the largest element in a given array

//Input: An array A[0..n − 1] of real numbers

//Output: The value of the largest element in A maxval ←A[0]

```
for i ← 1 to n − 1 do
        if A[i] > maxval
        maxval ← A[i]
return maxval
```

11. Write the algorithm to check whether all the elements in a given array are distinct?

Algorithm -

```
UniqueElements(A{0...n-1)
// Determine whether all the elements in a give are distinct
//Input - An array A[0..n-1]
//Output - Returns true if all the elements in A are distinct otherwise returns false.
for i <- 0 to n-2 do
        for j <- i + 1 to n-1 do
                if A[i] == A[j]
                        return false
return true
```

12. Write a recursive algorithm to find number of binary digits in a given decimal integer?

Algorithm -

```
BinRec(n)
//Input - A positive integer n
//Output - The number of binary digits in n's binary representation
if n == 1 return 1
```

else return BinRec([n/2] + 1)

UNIT - 3

Brute Force & Exhaustive Search

1. Define Brute Force approach? List its advantages.

Brute force is an intuitive, direct, and straight forward technique of problem-solving in which all the possible ways or all the possible solutions to a given problem are enumerated.

- Brute force is applicable to a very wide variety of problems.
- It is very useful for solving small size instances of a problem, even though it is inefficient.
- The brute-force approach yields reasonable algorithms of at least some practical value with no limitation on instance size for sorting, searching, and string matching.

2. Define selection sort and write a program to sort elements in array using selection sort?

Selection sort is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list.

Algorithm -

SelectionSort(A[0...n-1])

```
//Sorts a given array by selection sort
//Input - An array A[0...n-1] of orderable elements
//Output - Array A[0...n-1] sorted in nondescreasing order
for i <- 0 to n-2 do
        min <- i
        for j <- i +1 to n - 1 do
                if A[i] < A[min]
                min <- j
        swap A[i] and A[min]
```

3. Define bubble sort & write a program to sort elements in array using bubble sort?

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

Algorithm -

```
BubbleSort(A[0...n-1])
//Sort a given array by bubble sort
//Input - An array A[0...n-1] of orderable elements
//Output - Array A[0...n-1] sorted in nondecreasing order
for i <- 0 to n-2 do
        for j <- 0 to n-2-i do
                if A[j+1] < A[j]
                swap A[j] and A[j+1]
```

4. Define sequential search?

Sequential search is the algorithm that starts at one end and goes through each element of a list until the desired element is found, otherwise the search continues till the end of the data set.

5. Define exhaustive search?

Exhaustive Search is a brute-force algorithm that systematically enumerates all possible solutions to a problem and checks each one to see if it is a valid solution.

6. Explain the travelling salesman problem?

algorithm

7. Explain Knapsack problem?

algorithm

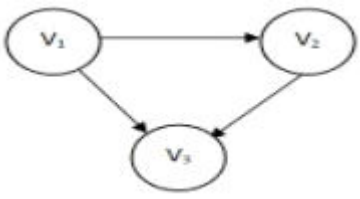
8. Define a graph? List its types.

A graph is a collection of nodes (also called vertices) and edges (also called ores or links) each connecting a pair of nodes.

Types -

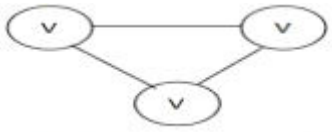
1. Directed Graph :

Directed graph is a graph which consists of directed edges where each edge in E is unidirectional.



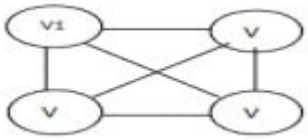
2. Undirected Graph :

An undirected graph is a graph, which consists of undirected edges.



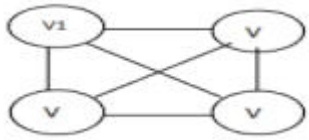
9. Define connected graph?

An directed graph is said to be connected if for every pair of distinct vertices V1 and V2 in V(G) there is a graph from Vi to Vj in G.



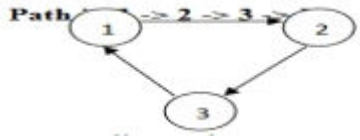
10. Define complete graph?

If an undirected graph of n vertices consists of n(n-1)/2 number of edges it is called as complete graph.



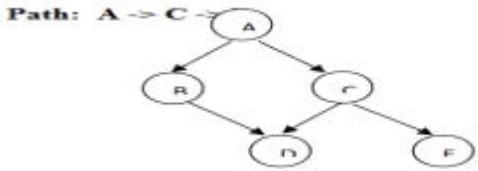
11. Define Cycle?

A cycle in a graph is a path in which first and last vertex are the same.



12. Define Acyclic graph?

A directed graph is said to be acyclic when there is no cycle path in it. It is also called as DAG(Directed Acyclic Graph).



13. Explain Depth First Search(DFS)?

DFS is a recursive algorithm to search all the vertices of a tree data structure or a graph. The depth-first search (DFS) algorithm starts with the initial node of graph G and goes deeper until we find the goal node or the node with no children.

Algorithm -

```
DFS (G)
//Implements DFS traversal of a given graph
//Input : Graph G = { V, E}
//Output: DFS tree
//Mark each vertex in V with 0 as a mark of being “unvisited” count 0
for each vertex v in V do
        if v is marked with 0
        dfs(v)
        dfs(v)
count count + 1
mark v with count
for each vertex w in V adjacent to v do
        if w is marked with 0
        dfs(w)
```

14. Explain Breadth First Search(BFS)?

The Breadth First Search (BFS) algorithm is used to search a graph data structure for a node that meets a set of criteria. It starts at the root of the graph and visits all nodes at the current depth level before moving on to the nodes at the next depth level.

Applications of BFS -

- To check connectivity of a graph (number of times queue becomes empty tells the number of components in the graph)

- To check if a graph is acyclic. (no cross edges indicates no cycle)
- To find minimum edge path in a graph

Algorithm -

```
//implements BFS traversal of a given graph
//input: Graph G = { V, E}
//output: BFS tree/forest
//Mark each vertex in V with 0 as a mark of being “unvisited”
count 0
for each vertex v in V do
        if v is marked with 0
                bfs(v)
        bfs(v)
        count count + 1
```

UNIT - 4

Decrease And Conquer

Divide And Conquer

1. What is decrease and conquer? List its advantages and disadvantages

Decrease and conquer is a technique used to solve problems by reducing the size of the input data at each step of the solution process.

This technique is similar to divide-and-conquer, in that it breaks down a problem into smaller subproblems, but the difference is that in decrease-and-conquer, the size of the input data is reduced at each step.

Advantages -

- Simplicity
- Efficient algorithms
- Problem-specific

Disadvantages -

- Problem specific
- Implementation complexity

2. Whhat is insertion sort? Write algorithm and advantages

To sort an array of size N in ascending order iterate over the array and compare the current element (key) to its predecessor, if the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element

Algorithm -

```
Insertionsort(A[0 ... n-1]) {
        //Sorts a given array by insertion sort
        //Input - Array A[0 ... n-1]
        //Output - Sorted array A[0 ... n-1] in ascending order
        for i <- to n-1:
                v <- A[i]
                j <- i-1
                while j >= 0 AND A[j] > V do:
```

```
                A[j+1] <- A[j]
                j <- j - 1
            A[j+1] <- V
    }
```

Advantages -

- Simple implementation
- Efficient on small list of elements, on almost sorted list
- Running time is linear in best case
- It is a stable algorithm.
- It is a in-place algorithm.

Time complexity -

- **Worst case -** O(N^2)
- **Average case -** O(N^2)
- **Best case -** O(N)

3. Define topological sorting? Write algorithm and advantages

A **topological sort or topological ordering** of a directed graph is a linear ordering of its vertices in which u occurs before v in the ordering for every directed edge uv from vertex u to vertex v.

Algorithm -

- Create a stack to store the nodes.
- Initialize visited array of size N to keep the record of visited nodes.
- Run a loop from 0 till N :
    - if the node is not marked True in visited array then call the recursive function for topological sort and perform the following steps:

        - Mark the current node as True in the visited array.
    - Run a loop on all the nodes which has a directed edge to the current node
        - if the node is not marked True in the visited array:
        - Recursively call the topological sort function on the node
        - Push the current node in the stack.
        - Print all the elements in the stack.

Applications of topological sorting -

- Task scheduling
- Software dependency resolution
- Building makefiles
- Compiler optimization
- Dependency analysis

4. What is divide and conquer? Give general plan for divide and conquer

Divide and conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same (or related) type (divide), until these become simple enough to be solved directly (conquer).

General plan -

- A problem is divided into several subproblems of the same type, ideally of about equal size.
- The subproblems are solved (typically recursively, though sometimes a different algorithm is employed, especially when subproblems become small enough).
- If necessary, the solutions to the subproblems are combined to get a solution to the original problem.

5. Write an algorithm for finding the maximum and minimum?

Algorithm -

```
StraightMaxMin(a, n, max, min)
//Input: Array a[1..n]
//Output: Maximum (max) and minimum (min) values in the array
max := a[1]      // Set max to the first element
min := a[1]      // Set min to the first element
for i := 2 to n do:
    if (a[i] > max) then:
        max := a[i] // Update max if a[i] is greater
    else
        if (a[i] < min) then
            min := a[i] // Update min if a[i] is smaller
```

6. What is merge sort? Write its algorithm , advantages and disadvantages

Merge sort is defined as a sorting algorithm that SORT works by dividing an array into smaller subarrays, sorting each subarray, and then merging the sorted subarrays back together to form the final sorted array.

Algorithm -

```
MERGE_SORT(arr, low, end)
    if low < end
    set mid = (low + end)/2
    MERGE_SORT(arr, low, mid)
    MERGE_SORT(arr, mid + 1, end)
    MERGE (arr, low, mid, end)
end of if
```

```
END MERGE_SORT
```

Advantages of merge sort -

- Stability
- Guaranteed worst-case performance
- Parallelizable

Disadvantages of merge sort -

- Space complexity
- Not in place
- Not always optimal for small datasets

7. Define Quick sort? Write its algorithm and advantages

QuickSort is a sorting algorithm based on the Divide and Conquer algorithm that picks an element as a pivot and partitions the given array around the picked pivot by placing the pivot in its correct position in the sorted array.

Algorithm -

```
QUICKSORT (array A, start, end)
{
        if (start < end)
        {
                p = partition(A, start, end)
                QUICKSORT (A, start, p - 1)
                QUICKSORT (A, p + 1, end)
        }
}
PARTITION (array A, start, end) {
        pivot = A[end]
```

```
        i =    start-1
        for j = start to end -1 do{
            if (A[j] < pivot) {
                then i = i + 1
                swap A[i] with A[j]
        }}
        swap A[i+1] with A[end]
        return i+1
}
```

Time Complexity -

- Best case - O(n*logn)
- Average case - O(n*logn)
- Worst case - O(n2)

8. List advantages and disadvantages of divide and conquer?

Advantages -

- Parallelism - Divide and conquer algorithms tend to have a lot of inherent parallelism.
- Cache performance - Divide and conquer algorithms also tend to have good cache performance.

Disadvantages -

- Sometimes it can become more complicated than a basic iterative approach.
- Recursion is slow, which in some cases outweighs any advantages of this divide and conquer process.

9. Define binary search? Write its algorithm

Binary Search is defined as a searching algorithm used in a sorted array by repeatedly dividing the search interval in half. The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(log N).

Algorithm -

```
BinarySearch(A[0 .. n-1], K) {

//Implementation nonrecursive binary search

//Input - An array A[0 .. n-1] sorted in ascending order and a search key K

//Output - An index of the array's element that is equal to K or -1 if there is no such elements

i <- 0, l <- 0, r<-n-1;

while(i <= r) do:

        m <- (l+r)/2

        if k = A[m]

                return m

        else if K < A[m]

                r <- m-1

        else l <- m+1

return -1;
```

10. Define Traversal?

Traversal is a technique for visiting all of a tree's nodes and printing their values.

11. List and explain Types of Binary tree traversals ?

1. Inorder Tree Traversal -

The left subtree is visited first, followed by the root, and finally the right subtree in this traversal strategy.

Algorithm -

```
inorder(t){
        // t is a binary tree. Each node of t has three fields. lchild, data, and rchild.
        if( t != 0) then
                inorder(t -> lchild);
                inorder(t->data);
                inorder(t -> rchild);
}
```

2. Preorder Tree Traversal-

In this traversal method, the root node is visited first, then the left subtree, and finally the right subtree.

Algorithm -

```
preorder(t){
        // t is a binary tree. Each node of t has three fields. lchild, data, and rchild.
        if( t != 0) then
                preorder(t->data);
                preorder(t -> lchild);
                preorder(t -> rchild);
}
```

3. Postorder Tree Traversal -

The root node is visited last in this traversal method, hence the name. First, we traverse the left subtree, then the right subtree, and finally the root node.

Algorithm -

```
postorder(t){
        // t is a binary tree. Each node of t has three fields. lchild, data, and rchild.
```

```
        if( t != 0) then
                postorder(t -> lchild);
                postorder(t -> rchild);
                postorder(t->data);
}
```

12. Write algorithm to find the height of binary tree ?

Algorithm -

```
Height(t) {
        //Computes recursively the height of a binary tree
        //Input - A binary tree T
        //Output - The height of T
        if( t = 0) return -1;
        else return max(Height(T1), Height(Tk)) +1 ;
}
```

UNIT - 5

Greedy Techniques

1. Prims Algorithm ?

Prim's Algorithm is a greedy algorithm that is used to find the minimum spanning tree from a graph. Prim's algorithm finds the subset of edges that includes every vertex of the graph such that the sum of the weights of the edges can be minimized.

Prim's algorithm starts with the single node and explores all the adjacent nodes with all the connecting edges at every step. The edges with the minimal weights causing no cycles in the graph got selected.

How the prims algorithm works?

- First, we have to initialize an MST with the randomly chosen vertex.
- Now, we have to find all the edges that connect the tree in the above step with the new vertices. From the edges found, select the minimum edge and add it to the tree.
- Repeat step 2 until the minimum spanning tree is formed.

Applications of prim's algorithm are -

- Prim's algorithm can be used in network designing.
- It can be used to make network cycles.
- It can also be used to lay down electrical wiring cables.

Algorithm -

Step 1: Select a starting vertex

Step 2: Repeat Steps 3 and 4 until there are fringe vertices

Step 3: Select an edge 'e' connecting the tree vertex and fringe vertex that has minimum weight

Step 4: Add the selected edge and the vertex to the minimum spanning tree T

[END OF LOOP]

Step 5: EXIT

2. Kruskals Algorithm ?

Kruskal's Algorithm is used to find the minimum spanning tree for a connected weighted graph. The main target of the algorithm is to find the subset of edges by using which we can traverse every vertex of the graph.

It follows the greedy approach that finds an optimum solution at every stage instead of focusing on a global optimum.

How the Kruskal's algorithm works?

- First, sort all the edges from low weight to high.
- Now, take the edge with the lowest weight and add it to the spanning tree. If the edge to be added creates a cycle, then reject the edge.
- Continue to add the edges until we reach all vertices, and a minimum spanning tree is created.

Applications of Kruskal's algorithm are -

- Kruskal's algorithm can be used to layout electrical wiring among cities.
- It can be used to lay down LAN connections.

Algorithm -

Step 1: Create a forest F in such a way that every vertex of the graph is a separate tree.

Step 2: Create a set E that contains all the edges of the graph.

Step 3: Repeat Steps 4 and 5 while E is NOT EMPTY and F is not spanning

Step 4: Remove an edge from E with minimum weight

Step 5:

IF the edge obtained in Step 4 connects two different trees, then add it to the forest F (for combining two trees into one tree).

ELSE

Discard the edge

Step 6: END

3. Dijkstra's Algorithm ?

Dijkstra's Algorithm is a Graph algorithm that finds the shortest path from a source vertex to all other vertices in the Graph (single source shortest path).

It is a type of Greedy Algorithm that only works on Weighted Graphs having positive weights.

The time complexity of Dijkstra's Algorithm is O(V2) with the help of the adjacency matrix representation of the graph

How the Dijkstra algorithm works?

Step 1:

First, we will mark the source node with a current distance of 0 and set the rest of the nodes to INFINITY.

Step 2:

We will then set the unvisited node with the smallest current distance as the current node, suppose X.

Step 3:

For each neighbor N of the current node X: We will then add the current distance of X with the weight of the edge joining X-N. If it is smaller than the

current distance of N, set it as the new current distance of N.

Step 4:

We will then mark the current node X as visited.

Step 5:

We will repeat the process from 'Step 2' if there is any node unvisited left in the graph.

Algorithm or Pseudocode for Dijkstra -

```
function Dijkstra_Algorithm(Graph, source_node)
        for each node N in Graph:
                distance[N] = INFINITY
                previous[N] = NULL
                If N != source_node, add N to Priority Queue G
        distance[source_node] = 0
        while G is NOT empty:
                Q = node in G with the least distance[]
                mark Q visited
          for each unvisited neighbor node N of Q:
                                        temporary_distance = distance[Q] +
          distance_between(Q, N)
          if temporary_distance < distance[N]
                                distance[N] := temporary_distance
                                previous[N] := Q
    // returning the final list of distance
    return distance[], previous[]
```

4. Define spanning tree?

A spanning tree is the subgraph of an undirected connected graph.

5. Define minimum spanning tree?

Minimum spanning tree can be defined as the spanning tree in which the sum of the weights of the edge is minimum.