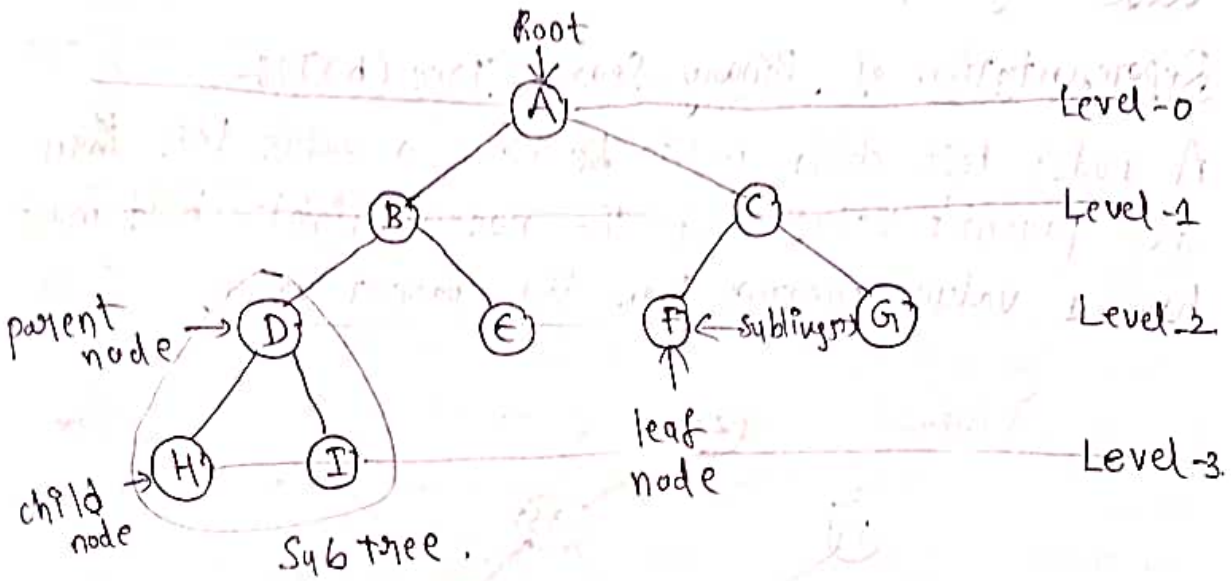
UNIT-V :-

Trees and Graphs

Trees :-

Tree represents the nodes connected by edges. Binary tree is a special data structure used for data storage purpose. A binary tree has a special condition that each node can have a maximum of two childrens. Searching is very easy in binary tree.



Important Terms :-

Path :- The sequence of nodes along the edges of a tree.

Root :- The node at the top of the tree is called root. There is only one root per tree, and one path from root node to any node.

Parent :- Any node except the root node has one edge upward to a node called parent.

child :- The node below a given node connected by its edge downward is called its child node.

Leaf:- the node which does not have any child is called the leaf node.

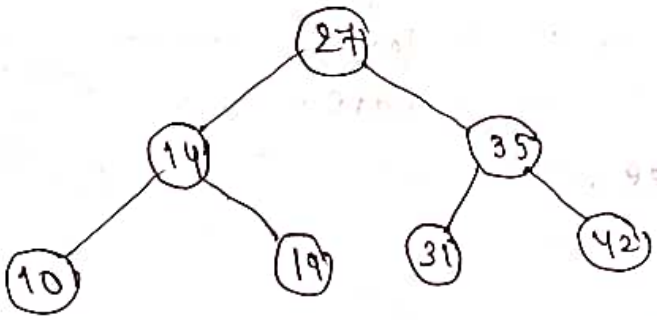
Traversing:- Visiting all nodes in a specific (in particular) order.

Levels:- Level of a node represents the generation of a node. If the root node is at level 0, then its next child node is at level 1, its grand child is at level 2, and so on.

Key:- Key represents the value of a node.

Representation of Binary Search Tree (BST):-

A node's left child must have a value less than its parent's value and the node's right child must have a value greater than its parent value.



BST Basic Operations:-

Insert:- Inserts an element in a tree

Search:- Searches an element in a tree.

Preorder-Traversal:- Traverses a tree in pre-order manner.

Inorder Traversal:- Traverses a tree in inorder manner

Postorder Traversal:- Traverses a tree in post-order manner.

Insert Operation:-

The very first insertion creates the tree. Whenever an element is to be inserted, first locate its proper location. Start searching from the root node, if the data is less than the key value, search for the empty location in the left subtree and insert the data. Otherwise, search for the empty location in the right subtree and insert the data.

Algorithm:-

```
If root is NULL
    create root node
    return.
If root exist
    compare the data with root node data
    while until insertion position is located
        If data > root node data
            goto right subtree
        else
            goto left subtree.
        end while.
    insert data.
end if.
```

Search Operation:-

Whenever an element is to be searched, start searching from the root node, if the data < the key value, search for the element in left subtree. Otherwise, search for the element in right subtree. Follow the same algorithm for each node.

Algorithm:-

If root data == search data
    return Root.
else
    while data not found.
        if data > node data.
            goto right subtree.
        else
            goto left subtree.
        If data found
            return node
    end while.
    return data not found.
end if.

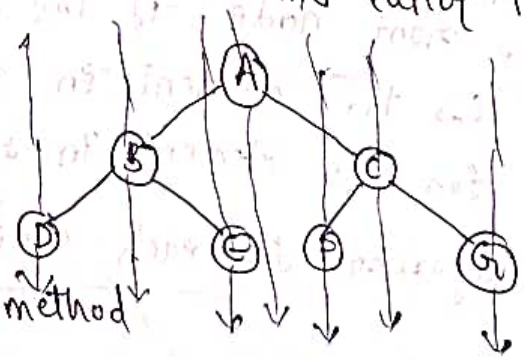
Tree Traversal Techniques :-

Traversal is a process to visit all the nodes of a tree and may print their values too. All nodes are connected via edges (links). There are three ways we used to traverse a tree.

(i) In-order Traversal
(ii) Pre-order Traversal
(iii) Post-order Traversal.

In-Order Traversal:-

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

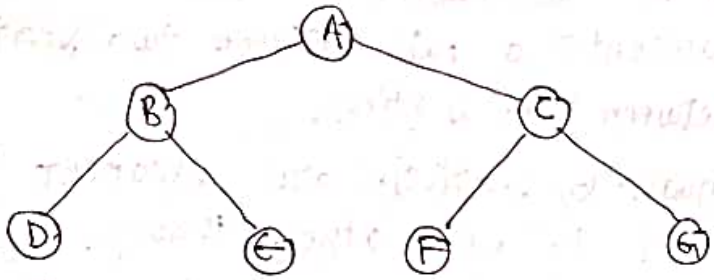


⇒ Free Fall method

⇒ D B E A F C G = output of in-order traversal.

pre-order Traversal :-

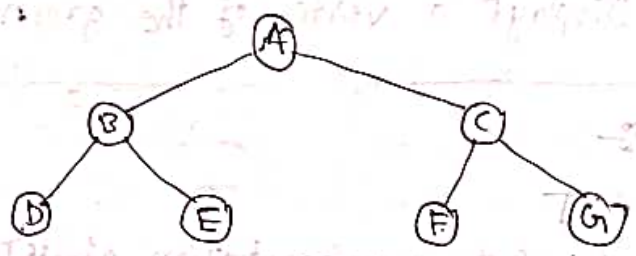
In this traversal method, the root node is visited first, then the left subtree and finally the right subtree.



⇒ A B D E C F G = o/p of pre-order traversal.

Post-order traversal :-

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

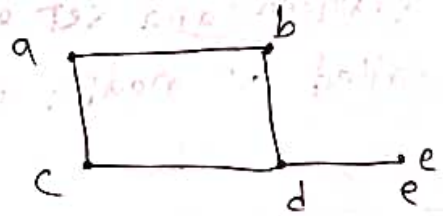


⇒ D E B F G C A

Graph :-

It is a ADT. It is pictorial representation of set of objects. where some pairs of objects are connected by links.

Graph is a pair of sets (vertices, Edges).



Vertices = {a, b, c, d, e}

Edges = {ab, ac, bd, cd, de}

Graph Data structure:-

Important Terms:-

Vertex:- Each node of the graph is represented as a vertex.

Edge:- Edge represents a path between two vertices or a line between two vertices.

Adjacency:- Two nodes or vertices are adjacent if they are connected to each other through an edge.

Path:- Path represents a sequence of edges b/w the two vertices.

Basic Operations:-

Add vertex:- Adds a vertex to the graph

Add Edge:- Adds an edge between the two vertices of the graph.

Display Vertex:- Displays a vertex of the graph.

What is Graph?

(i) Graph is a ADT

(ii) Graph is a pictorial representation of set of objects where some pairs of objects are connected by links.

(iii) Graph is used to implement the undirected graph and directed graph concepts from mathematics.

(iv) Applications

→ Graphs are used to represent the networks.

→ It is used in social networks like Facebook, Linked In etc.

(v) Graph is a set of vertices and set of edges.

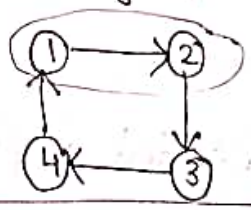
(vi) The vertices also called as nodes. Edges also called as links.

There are two types of graphs.

(i) Directed Graph (ii) Undirected Graph.

Directed Graph:-

The graph contains ordered pair of vertices is said to be directed graph.

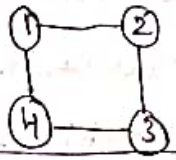


(1) ⇒ is called start vertex

(2) ⇒ is called end vertex.

Undirect Graph:-

The graph contains unordered pair of vertices is said to be undirected graph.



Adjacency Matrix:-

(i) It is a way to represent a graph.

(ii) Graph is represented using square matrix.

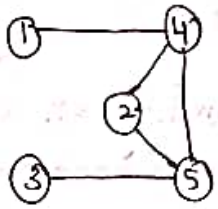
(iii) Graphs can be divided into two categories.

→ Sparse Graph
It contains less number of edges.

→ Dense Graph.
It contains number of edges.

(iv) Adjacency matrix is best for dense graph.

⇒



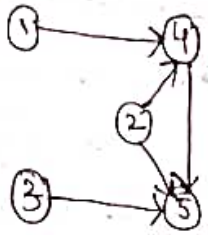
undirected Graph

	1	2	3	4	5
1	0	0	0	1	
2	0	0	0	1	1
3	0	0	0	0	1
4	1	1	0	0	1
5	0	1	1	1	0

Adjacency Matrix.

(v) Adjacency matrix of an undirected graph is always a symmetric.

⇒



Directed Graph

	1	2	3	4	5
1	0	0	0	1	0
2	0	0	0	1	1
3	0	0	0	0	1
4	0	0	0	0	1
5	0	0	0	0	0

Adjacency matrix.

(vi) Adjacency matrix of an directed graph is never symmetric.

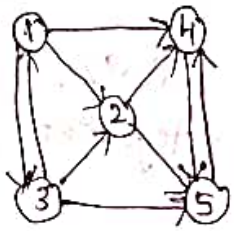
Disadvantages of Adjacency Matrix:-

(i) It consumes huge amount of memory for storing big graphs. It req's huge efforts for adding & removing a vertex.

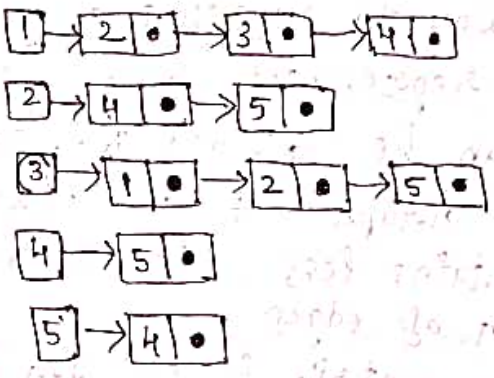
(ii) Adjacency list is another representation of graphs.

(iii) Adjacency list requires less amount of memory.

(iv) In adjacency list, an array of linked list is used.



Directed Graph



Adjacency list

Disadvantages of Adjacency List:-

(i) It is not easy for adding & removing an edge.

Graph Traversal Techniques:-

(i) It is also known as graph search.

(ii) Graph traversal means visiting each and exactly one node.

(iii) There are two techniques used in graph traversal.

→ DFS = Depth First Search

→ BFS = Breadth First Search.

DFS :-

(i) DFS is used for traversing a finite graph.
(ii) DFS uses stack.

Algorithm :-

(i) Start by putting any one of the graph's vertices at the top of the stack.
(ii) Take the top item of the stack and add it to the visited list.
(iii) Create a list of what vertex'ces adjacent nodes add ones which aren't in the visited list to the top of the stack.
(iv) Keep repeating step-2 and step-3 until the stack is empty.

BFS :-

(i) BFS is used for traversing a finite graph.
(ii) BFS uses queue

Algorithm :-

(i) Start by putting any one of the graph's vertices at the top/front of the queue.
(ii) Take the front item of the queue and add it to the visited list.
(iii) Create a list of what vertices adjacent nodes add ones which aren't in the visited list to the back of the queue.
(iv) Keep repeating steps 2 and 3 until the queue is empty.

Linear search:-

(i) It is a very basic and simple search algorithm. In Linear search, we search an element or value by traversing from the starting, till the desired element is found.

(ii) Linear search is applied on unsorted or unordered list, when there are fewer elements.

Linear Search Algorithm:-

(i). It is used for unsorted and unordered small list of elements.

(ii) It has time complexity O(n) [order of n], The time is linearly depends on the number of elements.

(iii) It has a very simple implementation.

Binary Search:-

Binary search is used with sorted array or list.

(i) We start by comparing the element is to be searched with the middle of the element in array.

(ii) If we get a match return the middle element.

(iii) If we do not get a match, check whether the element is greater or lesser than the middle element.

(iv) If the element is greater than the middle element then go to right side, search for that element.

(v) If the element is lesser than the middle element then go to left side, search for that element.

Binary search is useful when there are large number of elements in an array and they are sorted.

Binary Search Algorith:-

(i) It is useful for searching large sorted arrays.

(ii) The time complexity is O(logn) [order of logn].

(iii) It has a simple implementation.

Implementation of Linear Search:-

(i) Traverse the array by using a for loop.
(ii) In every iteration, compare the target value with the current value of the array
   → If the value match, return the current value.
   → If the values do not match, move to next value.
(iii) If no match is found return -1.

Implementation of Binary Search Algorithm:-

(i) start with the middle element.
   → If the target value match, return the middle element.
   → If the target value do not match, check whether the element is greater or lesser than the middle element.
(ii) If the target value is greater then pick the elements to the right of the middle element (step-1)
(iii) If the target value is lesser then pick the elements to the left of the middle element (step-1)
(iv) When the match is found, return the value.
(v) If do not match, return -1.

Sorting Techniques:-

Sorting is nothing but arranging the data in ascending or descending order.

Different Sorting Algorithms:-

Sorts are categorized into 2-types.
(i) internal sort → main memory
(ii) External sort → temporary memory or auxiliary memory

① Bubble sort ② Selection sort ③ Insertion Sort
④ Quick sort ⑤ Merge Sort ⑥ Heap sort.
⑦ Radix Sort ⑧ Shell sort.

Bubble Sort Algorithm:-

Bubble sort is a simple algorithm. It compares all the elements one by one.

Bubble sort will start by comparing the first element of the array with the 2nd element, if 1st element > the 2nd element, it will swap both the elements and then to compare the 2nd and 3rd element and so on.

If we have total n elements, then repeat this process for n-1 times.

Implementation of Bubble Sort Algorithm:-

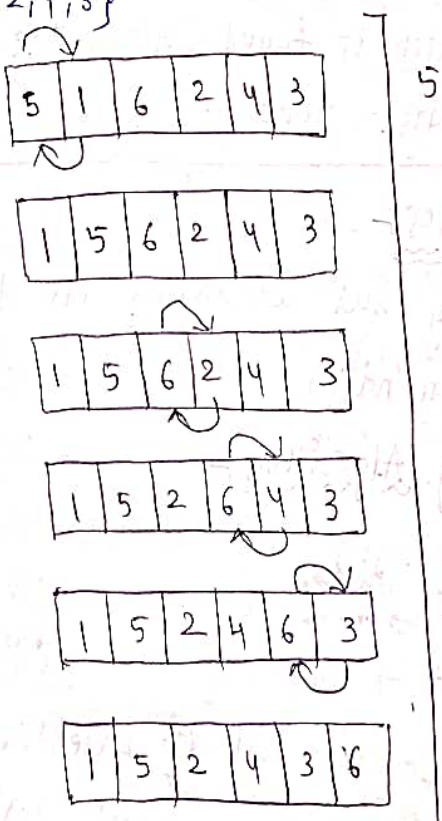
(i) Starting with the first element, compare the current element with the next element of the array.

(ii) If the current element is > the next element, the swap the two elements.

(iii) If the current element is < the next element, move to the next element, repeat step-1 & 2 & 3.

Pictorial Representation of Bubble Sort Algorithm:-

⇒ {5, 1, 6, 2, 4, 3}



After all the iterations, the array gets sorted.

(i) The time complexity of Bubble sort algorithm is $\frac{n(n-1)}{2}$ (or) $O(n^2)$.

(ii) The space complexity of Bubble sort is $O(1)$.

(iii) Following are the time & space complexity for the bubble sort algorithm.

→ Worst Case : $O(n^2)$

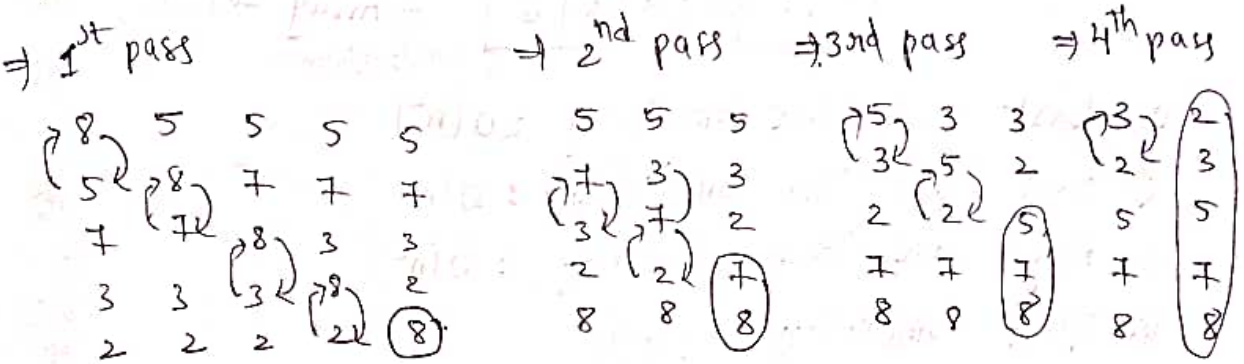
→ Best Case : $O(n)$

→ Avg Time Complexity: $O(n^2)$

→ Space Complexity : $O(1)$.

Eg: 8 5 7 3 2

⇒



⇒ Number of passes = $n-1$

Number of comparisions = $\frac{n(n-1)}{2} = O(n^2)$.

Time Complexity = $O(n^2)$.

Number of swaps = $\frac{n(n-1)}{2} = O(n^2)$.

Insertion Sort Algorithm :-

(i) It is efficient for smaller data sets, but very inefficient for larger lists.

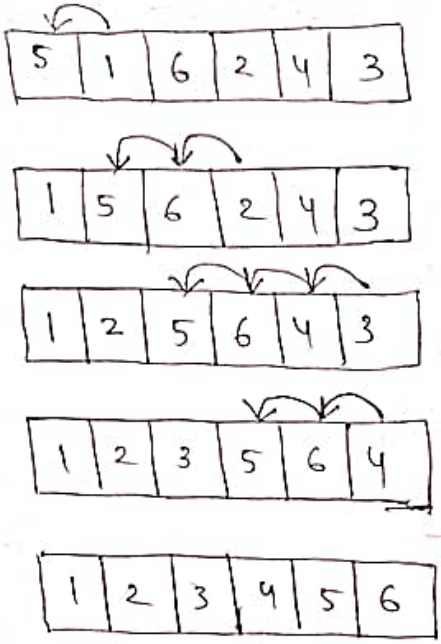
(ii) Insertion sort is adaptive and it is better than selection sort and bubble sort algorithms.

(iii) Its space complexity is less.

(iv) It is a stable sorting technique. It also req's single additional memory space like bubble sort.

Pictorial Representation of Insertion Sort:-

⇒ 5, 1, 6, 2, 4, 3.



⇒ Array Sorted.

(i) Worst Case Time Complexity : $O(n^2)$

(ii) Best Case Time Complexity : $O(n)$

(iii) Avg Case Time Complexity : $O(n^2)$.

(iv) Space Complexity : $O(1)$.

(v) It does not run using for loops it uses one while loop.

Selection Sort Algorithm:-

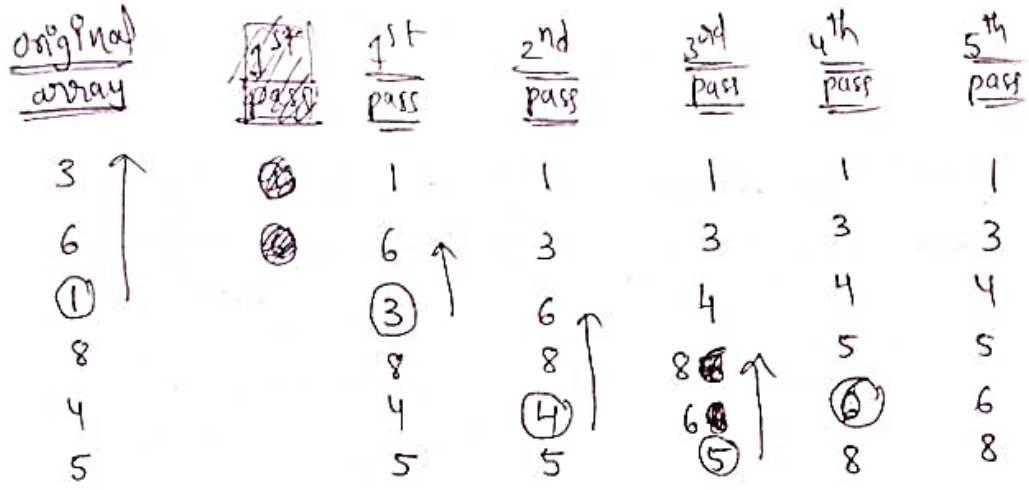
(i) Starting from the first element, we search the smallest element in the array and replace it in the first position.

(ii) And then move to 2nd position, find smallest element in sub array, replace it in 2nd position.

(iii) This is repeated untill the array is completely sorted.

Pictorial representation of selection start algorithm:-

⇒ 3, 6, 1, 8, 4, 5.



(i) Selection start requires two nested for loops.

(ii) Worst Case Time Complexity : $O(n^2)$ [Big-O]

(iii) Best Case Time Complexity : $O(n^2)$ [Bio-omega].

(iii) Avg Time Complexity : [Bio-theta]: $O(n^2)$

(iv) Space Complexity : $O(1)$.

Spanig Tree

Actual Parameters & Formal Parameters

Call-by-Value, & Call-by-Reference