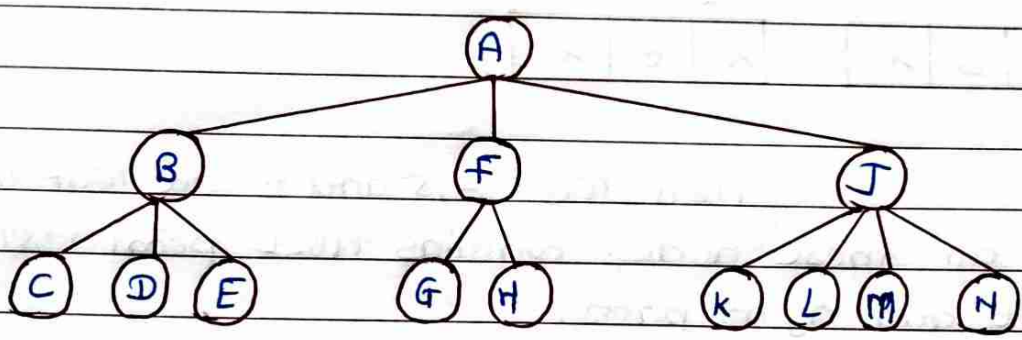
Types of Tree data structure :—

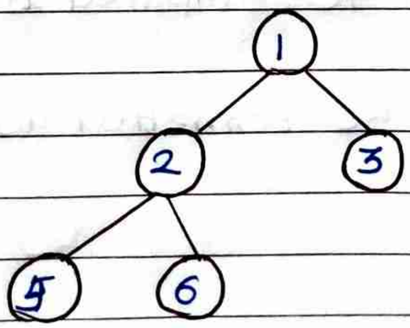
1). General Type :— In a general tree, a node can have either 0 or maximum n number of nodes. There is no restrictions imposed on the degree of node (number of nodes that a node can contain). The topmost node in a general tree is known as root node. The children of parent node are known as subtree.



There can be n number of subtrees in general tree. In general tree, subtrees are unordered as nodes in subtree cannot be ordered.

Every non empty tree has a downward edge, and these edges are connected to nodes known as child nodes. The nodes that have same parent are known as siblings.

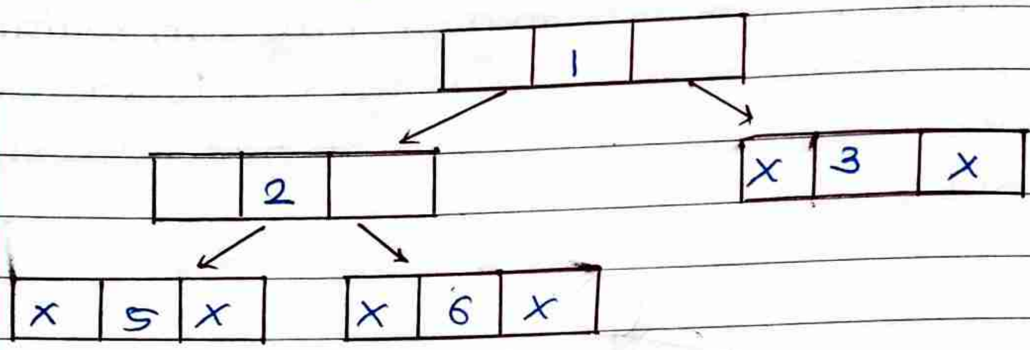
2). Binary Tree :— Binary tree means that the node can have maximum two children.



← given tree is a binary tree because
– each node contains atmost two children.

logical representation of above tree :-

In above tree, node 1 contains two pointers i.e left and right pointer pointing to left and right node respectively.



The nodes 3, 5 and 6 are leaf nodes, so all these nodes contains NULL pointer on both left and right parts.

Properties of Binary tree :-

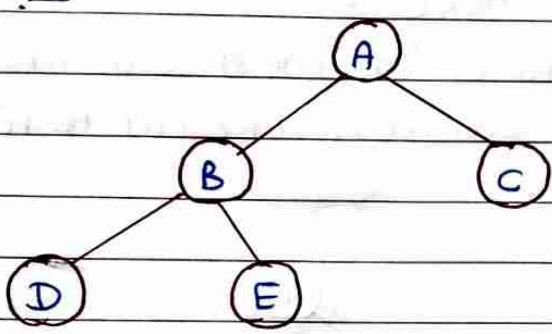
- At each level of i, the maximum number of nodes is 2^i.
- The height of tree is longest path from root node to leaf node. In general, maximum number of nodes possible at height is $(2^0 + 2^1 + 2^2 + \ldots 2^h)$
- The minimum number of nodes possible at height h is equal to h+1.
- If number of nodes is minimum, then height of tree would be maximum.
- minimum height can be computed as:
  $h = \log_2(n+1) - 1$
- maximum height can be computed as:
  $h = n-1$.

Types of Binary tree :-

1). full / proper / strict Binary tree :-

If each node contains either 0 or two children. The tree in which each node must contain 2 children except left nodes.

Example :-



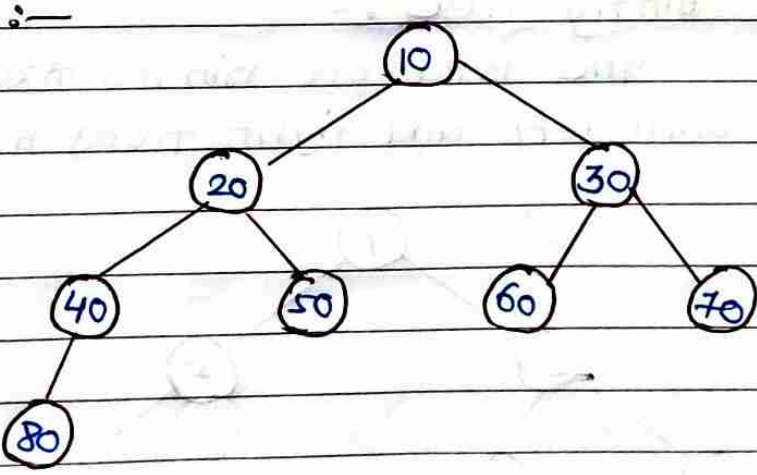
Properties :-

- maximum number of nodes : $2^{h+1} - 1$.
- minimum number of nodes : $2 * h - 1$
- minimum height $\log_2 (n+1) - 1$
- maximum height $h = \frac{n+1}{2}$

2). complete Binary Tree :-

The tree in which all nodes are completely filled except the last level. In complete Binary tree nodes should be added from left.

Example :-



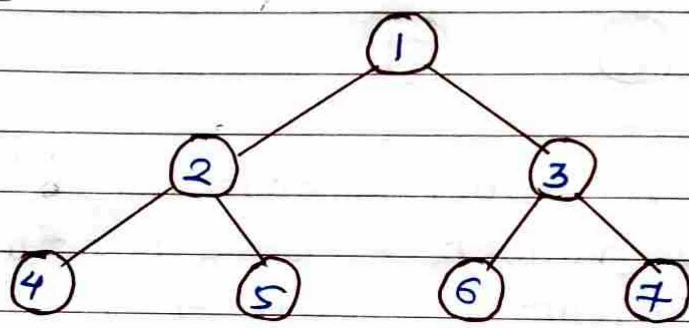
properties :-

- maximum number of nodes $\rightarrow 2^{h+1}-1$.
- minimum number of nodes $\rightarrow 2^{h}$
- minimum height $\rightarrow \log_2(n+1)-1$.

3). Perfect Binary tree :-

A tree in which all the internal nodes have 2 children. and all leaf nodes are at the same level.

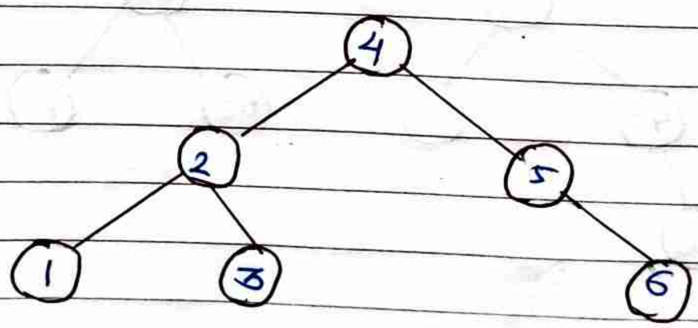
Example :-



Note :- All the perfect Binary trees are complete binary trees as well as the full Binary trees as But, vice versa is not true, all complete binary trees and full binary trees are the perfect Binary trees

4). Balanced Binary Tree :-

The balanced binary tree is a tree in which both left and right trees by almost 1.



above tree is balance : diff betn left Subtree & right S.T. is zero

Binary Tree implementation :-

```
struct node
{
  int data;
  struct node *left, *right;
}
```

Tree Traversal :-

The process of visiting nodes is called as tree traversal.

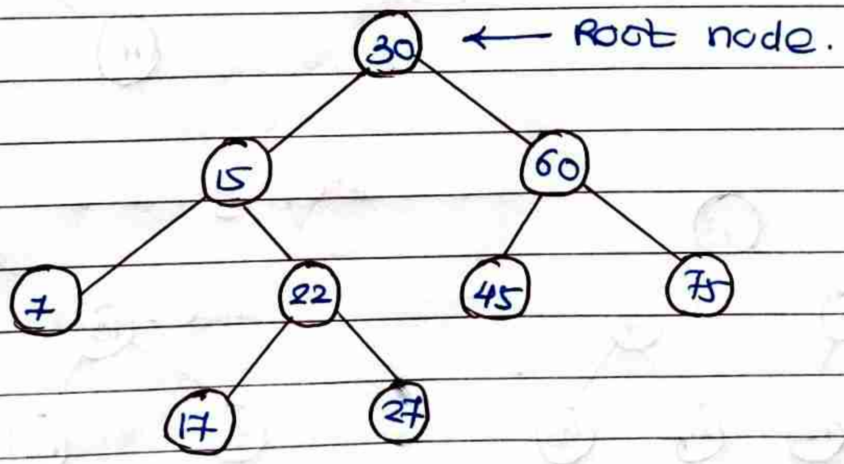
There are three types of traversals used to visit a node :

1). Inorder Traversal
2). preorder Traversal
3). postorder Traversal

3). Binary Search Tree :-

defn :- Binary search tree can be defined as a class of binary Trees, in which a nodes are arranged in a specific order. also called as ordered Binary Tree.

- similarly value of all nodes in right subtree is greater than or equal to value of root.



Example : create binary search tree using following data elements :-

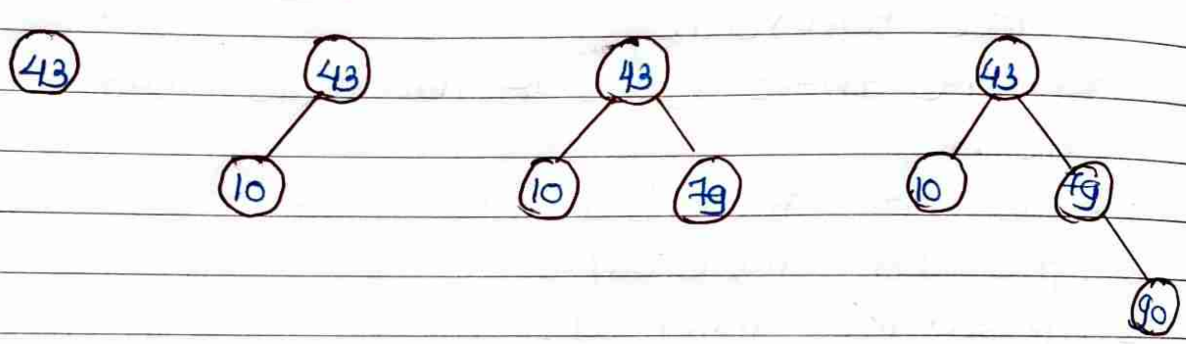
43, 10, 79, 90, 12, 54, 11, 9, 50.

→ 1). Insert 43 into tree as root of tree.

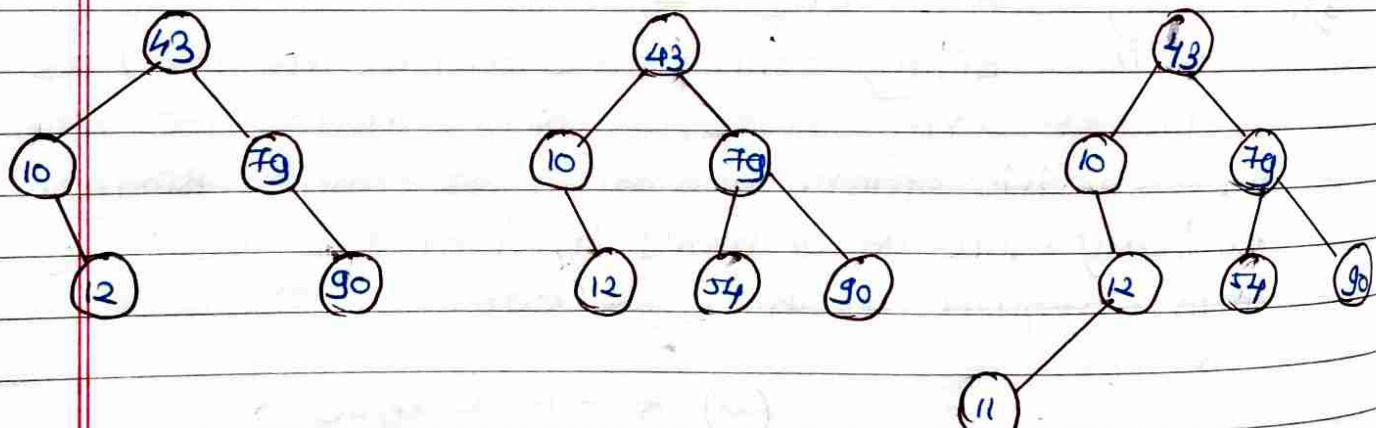
2). Read next element, if it is lesser root node element, insert it as root of left sub-tree.

3). otherwise, insert it as root of right of right subtree.

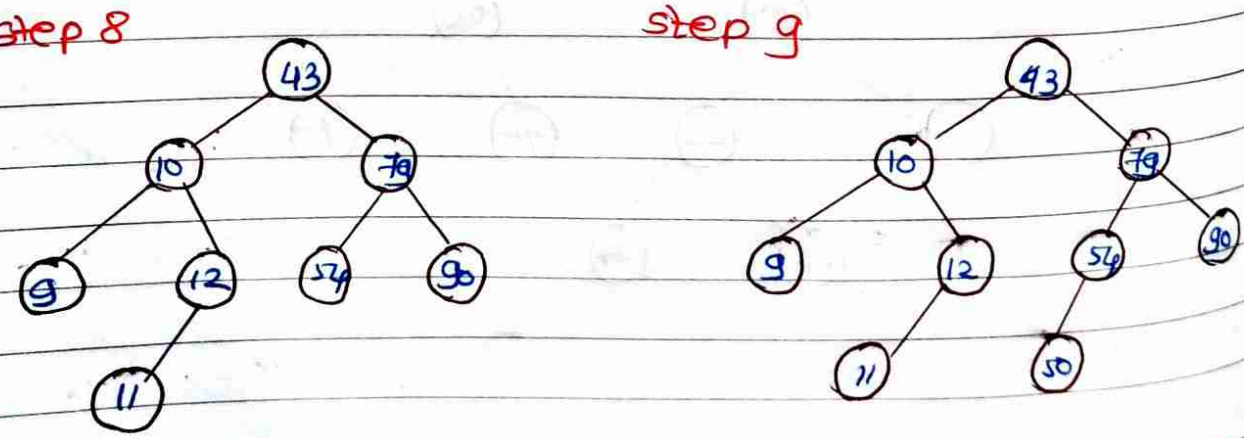
step 1 step 2 step 3 step 4



step 5 step 6 step 7



step 8 step 9



Operations on Binary Search Tree (BST) :-

Sr.No.	Operation	Description
1).	searching in BST	Finding location of some specific element in a Binary search Tree.
2).	Insertion in BST	Adding a new element to the binary search tree at appropriate location so that property of BST do not violate.
3).	Deletion in BST.	Deleting some specific node from a BST, However, tree there can be various cases in deletion depending upon number of children, node have.

4). AVL Tree :- AVL Tree is invented by GM Adelson-Velsky and EM Landis in 1962. The tree is named as AVL in honour of its inventors.

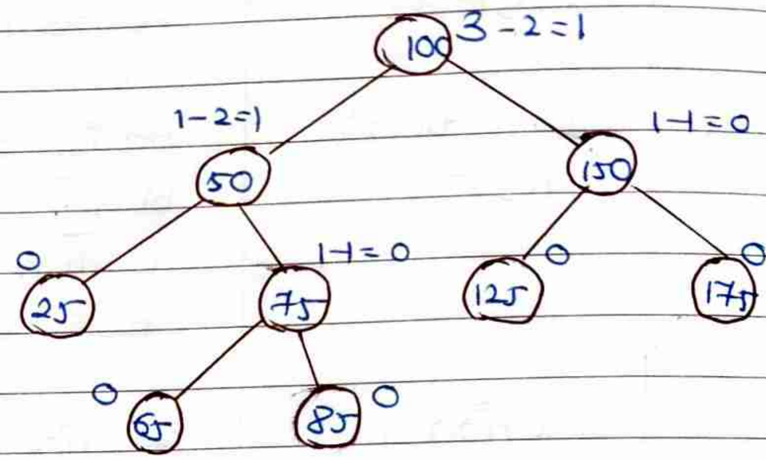
AVL tree is defined as height balanced binary search tree in which each node is associated with a balance factor which is calculated by subtracting the height of its R. subtree from its left subtree.

Balance factor (K) = height(left(K)) − height(right(K))

- If balance factor of any node is 1, it means that left sub-tree is one level higher than right subtree.

- If balance factor of any node is 0, it means that left sub-tree and right sub-tree contain equal height.
- If balance factor of any node is -1, it means that left sub-tree is one level lower than right subtree.

Example :-



Here we see that, balance factor associated with each node is between -1 and +1.

∴ It is an example of AVL tree.

complexity :-

Algorithm	Average case	Worst case
space	$O(n)$	$O(n)$
search	$O(\log n)$	$O(\log n)$
Insert	$O(\log n)$	$O(\log n)$
Delete	$O(\log n)$	$O(\log n)$

Why AVL Tree ? → AVL tree controls height of binary search tree by not letting it to be skewed The time taken by all operations in BST is $O(h)$ However it will be extended to $O(n)$ If BST became skewed (worst case). By limiting this height to log n, AVL tree imposes an upper bound on each operation to be O(logn), where n is number of nodes.

Operations on AVL tree :-

Sr. No	Operation	Description.
1).	Insertion	Insertion is performed in same way it performed in BST. However, it may lead to violation in the AVL tree property and so tree may need balancing. and tree can be balanced by rotation.
2).	Deletion	Deletion is also same way performed as BST It can be also disturb balance of tree, so various types of rotations are used to rebalance tree.

AVL Rotations :-
We perform rotations in AVL tree only in case if Balance Factor is other than -1, 0 and 1.
There are Basically four types of rotations which are as follows :

1). L-L rotation :- Inserted node is in the left subtree of left subtree of A.

2). R-R rotation :- Inserted node is in the right subtree of right subtree of A.

3). L-R rotation :- Inserted node is in right subtree of left subtree of A.

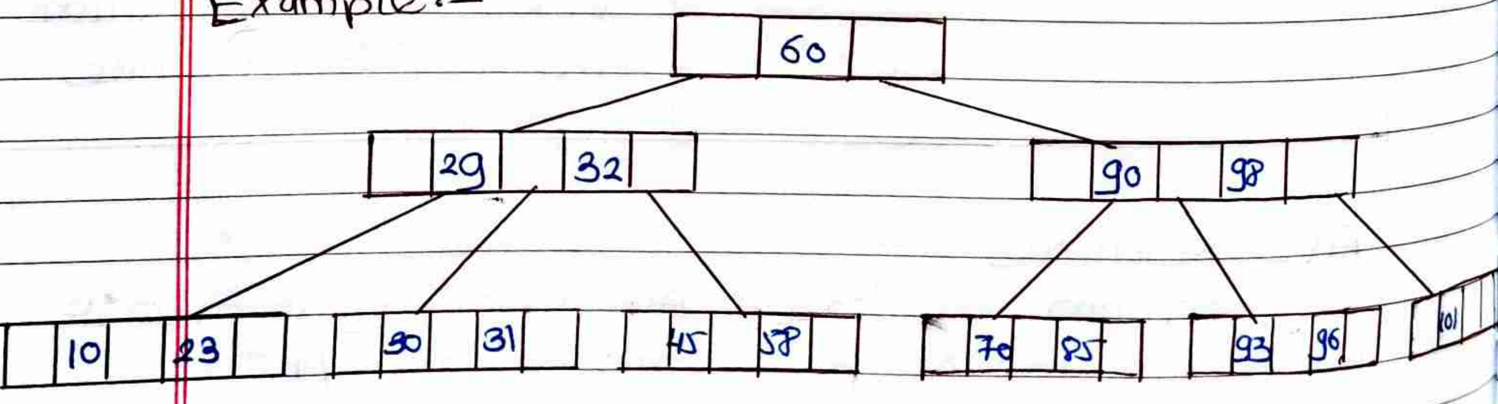
4). R-L Rotation :- Inserted node is in the left subtree of right subtree of A.

5). B Tree :- B Tree is specialized m-way tree that can be widely used for disk access. A B-tree of order m can have at most m-1 keys and m children.

Proporties :-

1. Every node in B-Tree contains at most m children
2. Every node in B-Tree except root node and leaf node contain at least m/2 children.
3. The root nodes must have at least 2 nodes.
4. All leaf nodes must be at the same level.

Example :-



Operations :-

1). searching :- The searching in B tree is similar to searching in Binary tree. For example, we search for an item 49 in following B Tree. The process will be:

(1). compare item 49 with root node 78. since 49 < 78 hence, move its left sub-tree.

(2). since, 40 < 49 < 56, traverse right subtree of 40.

(3). 49 > 45, move to right compare 49.

(4). match found, return.

Searching in B tree depends upon height of the tree. The search algorithm takes O(log n) time to search any element in B tree.

2). Inserting :- Insertion are done at leaf node level. The following algorithm needs to be followed in order to insert an item into B tree.

(1). Traverse B tree in order to find appropriate leaf node at which node can be inserted.

(2). If leaf node contains less than m-1 keys then insert element in increasing order.

(3). Else, if leaf node contains m-1 keys, then follow following steps:

- Insert new element in increasing order of elements.
- split node into two nodes at median.
- Push median element upto its parent node.
- If parent node also contain m-1 number of keys, then split it too by steps.

110

Application of B tree :-

- B tree is used to index data and provides fast access to actual data stored on disks since, the to value stored in a large database that is stored on a disk is a very time consuming process.

searching an un-indexed and unsorted database containing n key values needs O(n) running time.

6). B + Tree :-

B+tree is an extension of B tree which allows efficient insertion, deletion and search operations.

The leaf nodes of B+ tree are linked together in form of the singly linked list to make search queries more efficient.

Advantages of B+ tree :-

1). Records can be fetched in equal number of disk accesses.

2) Height of tree remains balanced and less as compare to B tree.

3) We can access data stored in B+ tree sequentially as well as directly.

4). Keys are used for indexing.

Graph :-

A graph can be defined as group of vertices and edges that are used to connect these vertices.

Defination :-

A graph G can be defined as an ordered set G(V, E) where V(G) represents set of vertices and E(G) represents set of edges.