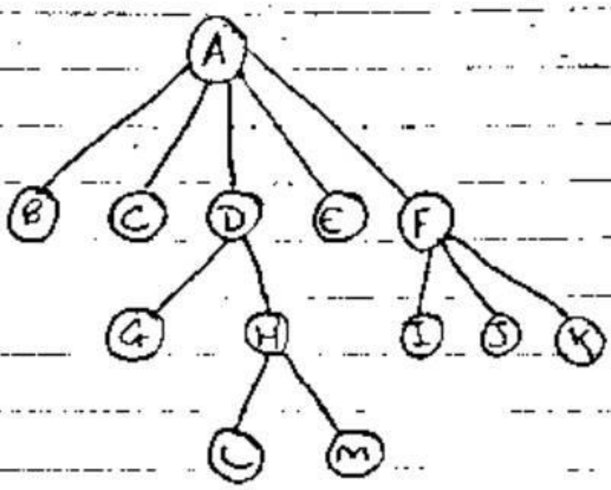
Trees

Tree:

A Tree is a finite set of one or more data items (node). such that there is a special data item which is called the root of the tree and its remaining data items are partition into number of mutually exclusive (independent) subsets, Each of which itself is a tree and they are called subtree.



In computer science, a tree is an abstract model of a higher hierarchical order structure that consist of nodes with a parent-child relationship. A tree has a following characteristics:

- Non-linear data structure
- Combines advantages of an ordered array and linked list.
- Searching is as fast as in ordered array
- Insertion and deletion are as fast as in linked list.

Application of Tree:

Tree can be implemented in different applications. Some of them are:

Hierarchy ~~and~~ of an organization; array
Directory structure of a file store
Structure of arithmetic expressions

Tree Terminologies:

Node: It is an element in tree data type.

- A node that points to one or more other nodes is the parent of those nodes and the nodes pointed to are the children. ~~Every node except~~

- Every node except the root has exactly one parent.
- Nodes with no children are leaves node.
- Nodes with children are internal node.
- Nodes with same parents are siblings.

Root: It is special design data item in a tree

- Root is a first in the hierarchical arrangement of data items in a tree.

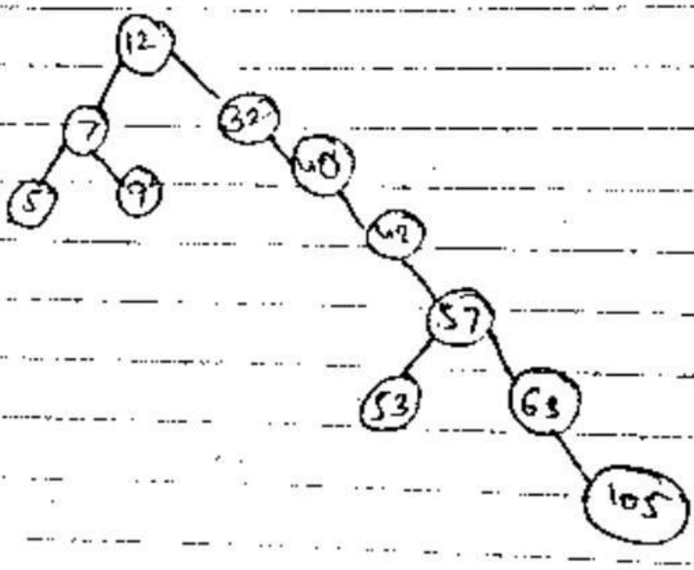
Descendents of a Node:

- The descendents of a node consists of its children and their children and so on.

Ancestors of a node
- Ancestors of node consists of the parent nodes and the parent of some ancestor of that node.

Binary Search Tree :

12, 7, 9, 5, 32, 40, 42, 57, 53, 63, 105



Edge : Connecting lines between two nodes

Degree of node : It is a number of the sub-trees of a node in a given tree.

Terminal node : Node with degree zero is terminal node (leaf, external node).

Path: It is a sequence of consective edges from source node to destination node. There is a single unique path from the root to any node. Length of a path is equal to number of edges travelled.

Node's height: A nodes height is equal to the max. path length from that node to a leaf node. A leaf node has height 0.

Trees height: Height of a tree is height of root.

Node's depth: A node's depth is equal to the path length from the root to that node. The root has depth 0.

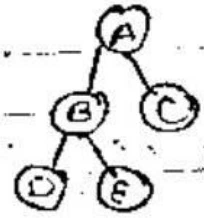
Depth of tree: Depth of tree is maximum level of any leaf in a tree. This is equal to length of longest path in a tree.

level of a node: The level of node is zero if it is root. otherwise it is one more than its parent.

Binary Tree:

A binary tree is a finite set of a elements ie. either empty or is partition into three disjoint sub-set. The first sub-set contain single element for the root of the tree and other two sub-sets are themself. Binary tree called left and right sub-tree of the original tree.

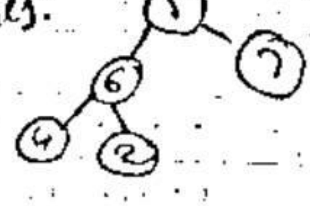
In binary tree each node has at must two children.



Strictly Binary tree:

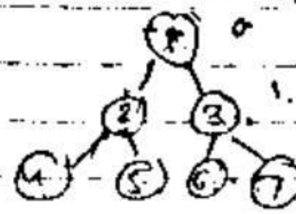
If every internal node in a binary tree has non-empty left and right sub-trees then the such tree is called strictly binary tree.

A strictly binary tree with 'n' leafs always contains 2n-1 nodes.



Complete binary tree:

Complete binary tree of depth 'd' is the strictly binary tree all of whose leaves are at level d.



0 $2^0 = 1$

1. $2^1 = 2$

2 $2^2 = 4$

It contains exactly 2^d nodes at each level d. The tree of depth d contains 2^d nodes.

$2^d = 2^2$
$= 4 - 1$

Properties:

- Every internal node has three children.
- Binary tree with depth d will have 2^d nodes.
- Binary tree with depth d will have $2^d - 1$ internal node (leaf).
- No. of external node = No. of internal nodes +1

Almost Complete Binary Tree:

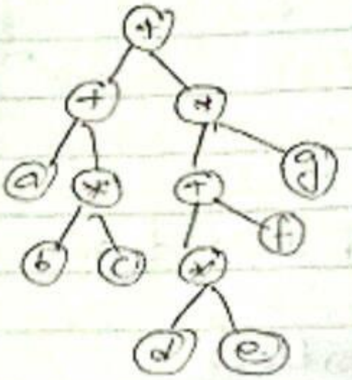
ACBT of depth 'd' is an almost completely binary tree: i,

① If any node (nd) at level less than d-1 has two children.

② For any node (nd) in the tree with a right descendent at level d, nd must have a left child, but every left descendent of nd is either a leaf at level d or has two children.

Expression Tree:

An exp. tree is a strictly binary tree in which leaf node contain operand and non-leaf node contain operator. Root node contain the operator i.e. applied to result of left & right sub-trees.



An expression tree for the exp:

$(a+b \times c) + ((d \times e)+f) \times g)$

An exp. tree is built up from simple operand and operators of an expression by placing the simple operands as the leaves and the operators as the internal nodes.

Tree Traversal / Tree representation:

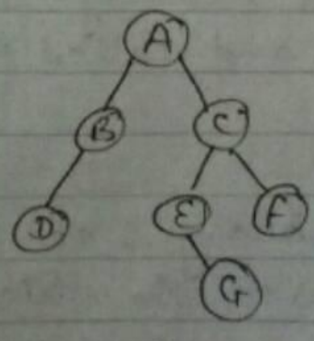
A binary tree can be represented by array implementation or by link list implementation. When tree is represented with array, traversal becomes from front to end and from end to front. But generally binary tree is representation with link implementation. In this representation, binary trees are non-linear data structures. Hence traversal can be in different orders.

There are three popular methods of traversal. They are:

1. pre-order Traversal
2. In-order Traversal
3. post-order Traversal

1. Algorithm (module) of a non-empty binary tree traversal in pre-order:

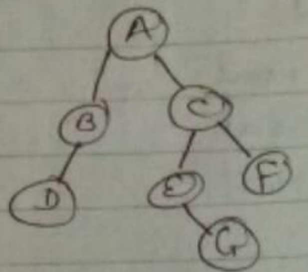
- Visit the root node
- Traverse the left sub-tree in pre-order
- Traverse the right sub-tree in pre-order



Output: ABDCEGF

Algorithm (module) of a non-empty binary tree traversal in in-order:

- Traverse the left sub-tree in inorder
- Visit the root
- Traverse the right sub-tree in inorder.



Output : DBAEGCF

Post-Order traversal : similar as L, R, Root

Balanced Binary Trees:

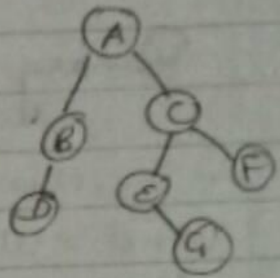
It is the binary tree ~~in~~ which is balanced such that search process become efficient. A tree can be balanced in term of its height called ~~or~~ height balance tree or weight and called weight balance tree.

Height balance tree: A binary tree is called height balance binary tree in which the height of two subtrees of every node never differs by more than one. An AVL tree is a height balance tree named after ~~as~~ a Russian scientist AVL tree is a advanced binary searched tree.

In-order : L, Root, R
Pre-order : Root, L, R
Post-order : L, R, root

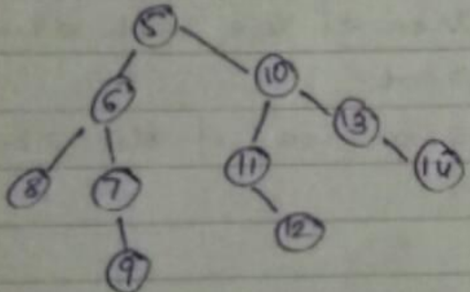
Module for post-order traversal

1. Traverse the left sub tree in postorder.
2. Traverse the right sub-tree in postorder.
3. Visit the root.



The output of postorder traversal in given tree is DBGEFCA

Q.

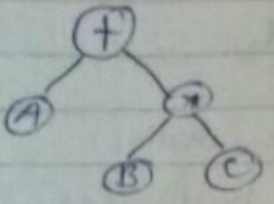


→ Ans :

pre-order : 5 6 8 7 9 10 11 12 13 14
In-order : 8 6 7 9 5 11 12 10 13 14
Post-order : 8 9 7 6 12 11 14 13 10 5

Expression Tree Traversal

Tree Traversal technique is used for traversing an expression tree to obtain an expressions infix, pre-fix and post-fix notation. These can be obtained by inorder traversal, pre-order and post order traversal respectively.



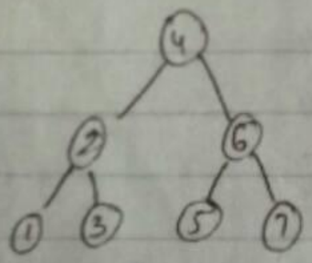
prefix expression : +A*BC
Infix expression : A+B*C
Postfixexpression : ABC*+

Binary Search Tree:

A binary search tree is a binary tree ie. either empty or every node contains a key value and satisfies the following condition:

- All keys in left sub-tree of the root are smaller than the key in the root.
- All keys in right sub-tree of the root are greater than the key in the root.
- The left and right sub-tree of the root are again binary search tree.

eg: 1 2 3 4 5 6 7



Make a BST from given elements;

Q. 12, 7, 9, 5 , 32, 40, 42, 57, 53, 63, 105

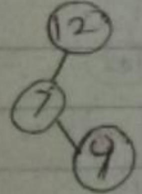
AVL tree is a advanced binary searched tree.

Weight balance tree: The weight of a tree is defined as the number of external node in the tree. If the ratio of the weight of the left sub-tree of every node to the weight of the right sub-tree rooted at the same node is between some fraction of A and 1-A. The tree is called weight balance tree of ratio A. (A = number of leaves node)

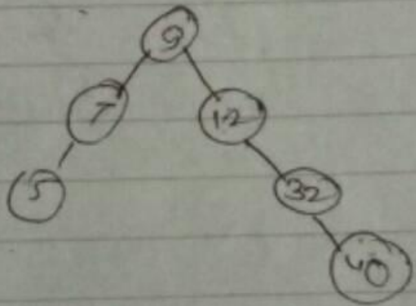
Construct AVL trees from given data:

(1) 12, 7, 9, 5, 32, 40, 42, 57, 53, 63, 105

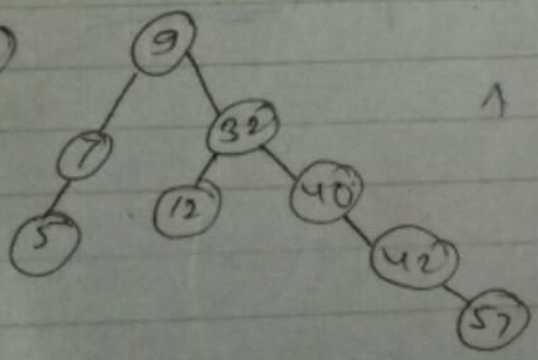
(a)



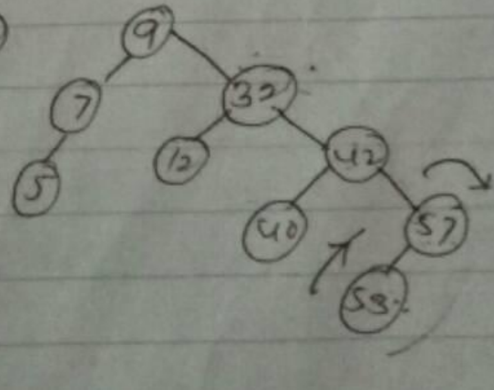
(b)



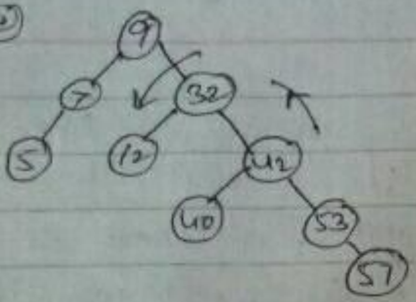
(c)



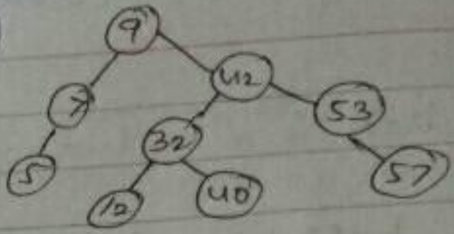
(d)



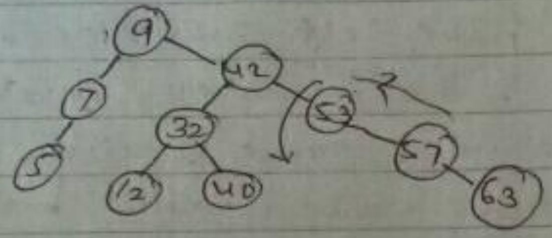
(e)



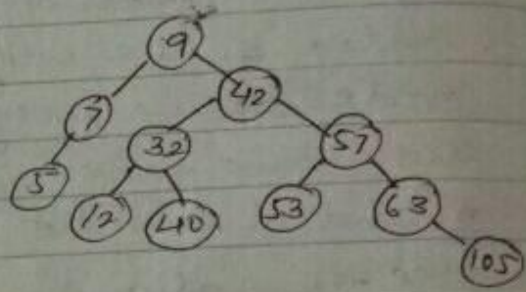
(f)



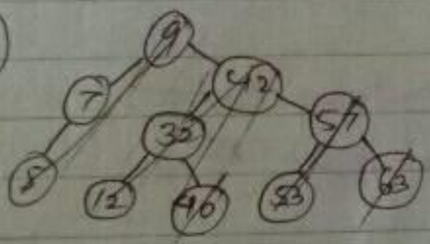
(g)



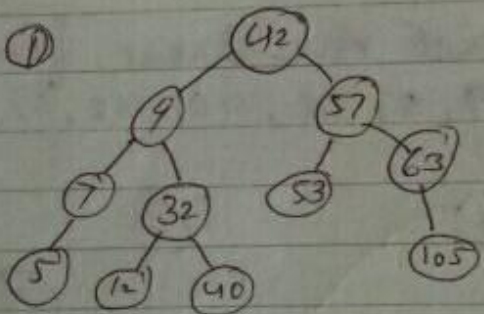
(h)



(i)

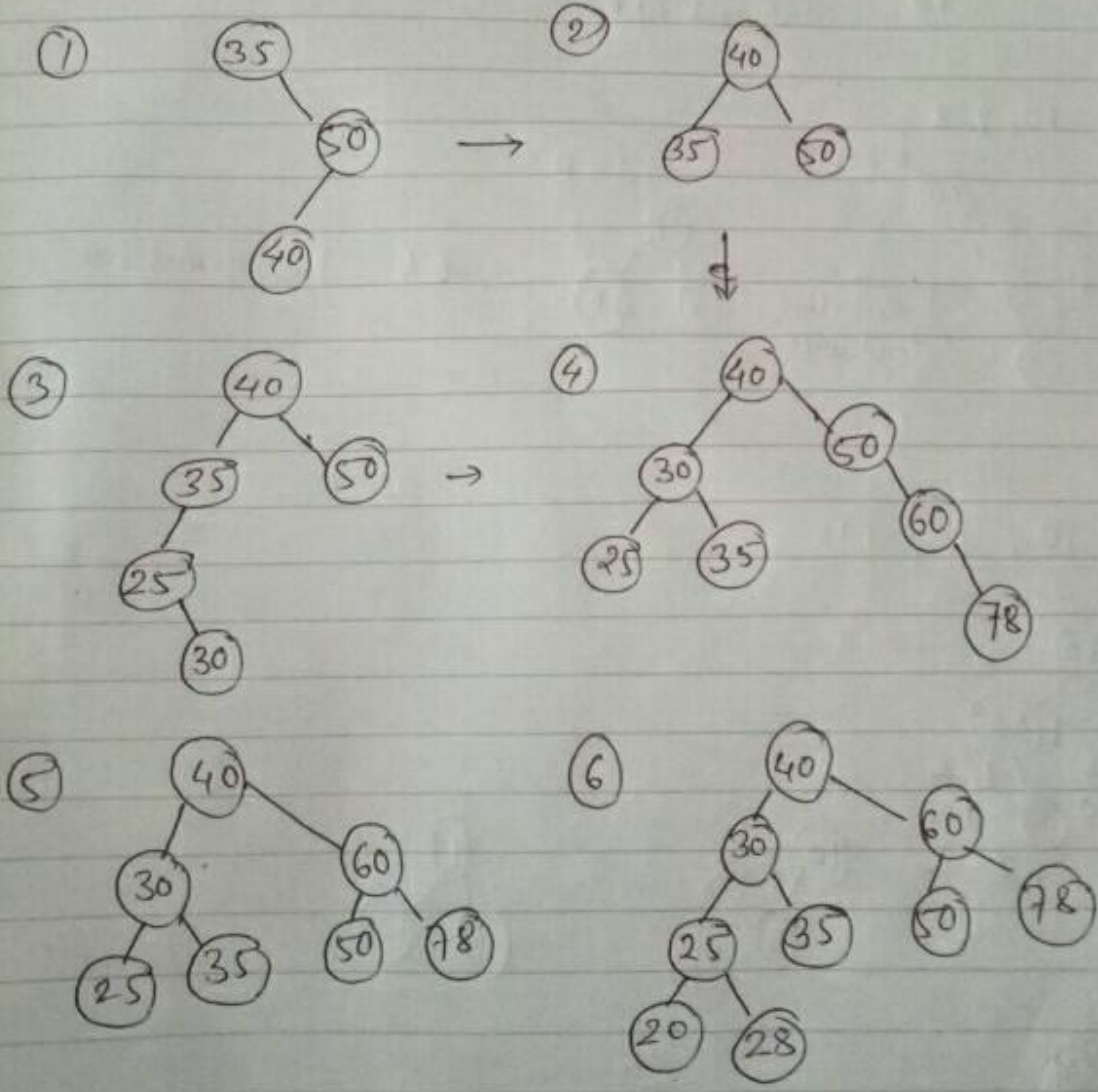


(j)



req. AVL tree

35, 50, 40, 25, 30, 60, 78, 20, 28



AVL Tree

B-Tree :

A B-Tree is balanced M-way tree with following properties :

1. Each node has maximum of M children, and a minimum of $\lceil M/2 \rceil$ children or any no. from 2 to maximum.

2. Each node has one fewer keys than children, with a maximum of M-1 keys.

3. Keys are arranged in a defined order within the node.

4. When the new key is to be inserted into a full node, the tree is split into two nodes and the key with the median value is inserted in the parent node. In case, the parent node (root) is full, a new root is created.

5. All leaves are on the same level ie. there is no empty sub-tree above the level of the leaves.

6. Order of B-tree is bound on the number of elements in each ~~other~~ node.

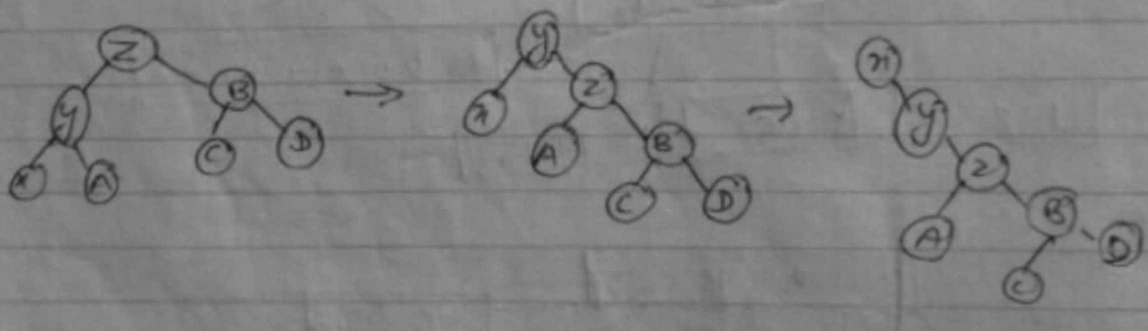
1. push, pull
2. push, push
3. Pull push, pull push

Splay Tree:

Splay Tree is a binary search tree with no explicit balance condition, in which a special operation called a splay is done after each search or insertion operation. Splaying at node x causes node x to become root of a binary search tree through a specific series of rotation as follows:

1. x has no grandparent (zig)
a. If x is left child of root y rotate (xy) R.
b. Else if x is right child of root y, then rotate (yx) L.

2. x is LL or RR grand child (zig-zig):
a. If x is left child of y & y is left child of z, then rotate at grand father (yz) R and then rotate at father (xy) R.



b. Else if x is right child of y and y is right child of z then rotate at grandfather (yz) L and then rotate at father (xy) L.
c. If x has not become the root then continues splaying at x.

3. x is LR or ~~and~~ RL grand child (zig-zag):

a. If x is right child of y and y is left child of z then rotate at father $(yx)L$ and then rotate at grand father at $(xz)R$.

b. Else if x is left child of y and y is right child of z then rotate at father $(yx)R$ and then rotate at grand father $(xz)L$.

c. If x has not become root then continue splaying x

Compare & Contrast between Splay trees and AVL trees:

① Both splay trees and AVL trees are BST with excellent performance guarantee but they differ in how they achieve those guarantee of performance. In an AVL tree the shape of tree is constrained (fixed/bounded) at all times such that the tree shape is balanced. Splay tree on the other hand, maintain efficient by reshaping the tree in response to look ups on it.

② If we need real time look ups the AVL tree likely to be better. However, Splay trees tend to be much faster on average. So, if we want to minimize the total run time of tree look ups the splay tree is likely to be better.

③ Splay tree support some operation such as splitting and merging very efficiently, while the corresponding AVL tree operation are less efficient.

④ Splay tree are more memory efficient than AVL tree because they donot need to store balance info. in the nodes.

However, AVL trees are more useful in multi-threaded environment with lot of looks up. Because looks up in an AVL tree can be done in parallel while they can't in splay tree.

⑤ Splay trees tend to be easier to implementation than AVL trees since the rotation logic is much easier.