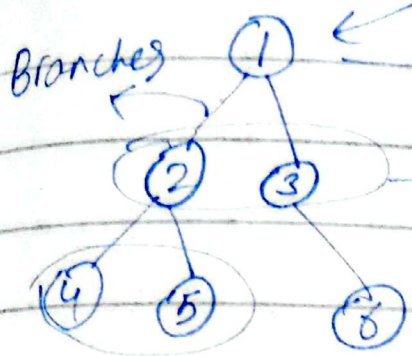
Binary Trees

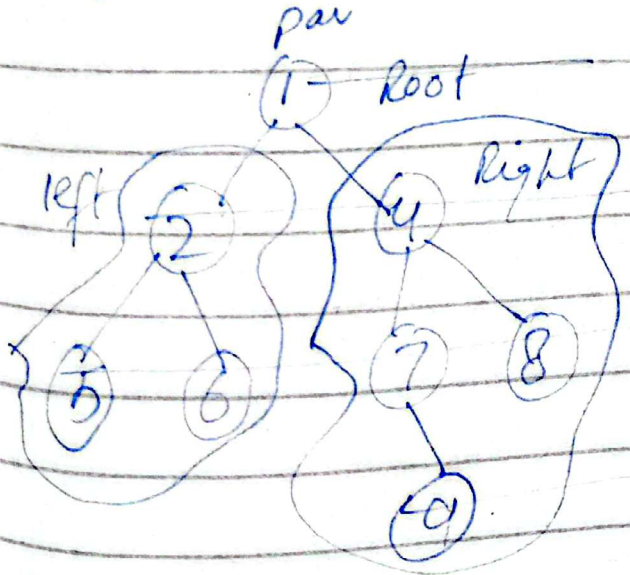
↳ (Heirarichal Data structure)
- ↳ Like File System.
- ↳ Family Tree.

Generic Tree :-



Binary Tree

Each Node → at max 2 children



→ Leaf Node
- No Children
- Last Nodes (5) (6) (9) (8)

→ Branches
Connection medium b/w two nodes

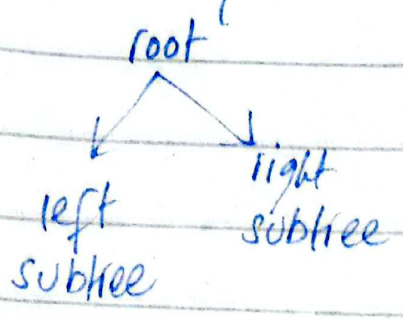
→ Siblings
(2) and (4)

→ Levels
4 levels in given tree

→ Height
h = levels of tree

→ Subtree
Small part of tree

Binary Tree with Recursion

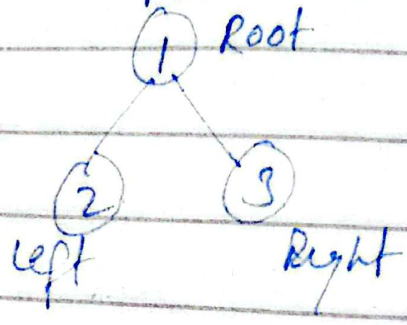



```
solve(root){
solve[left subtree]
solve[right subtree]
Calc the sol for the root
}
```

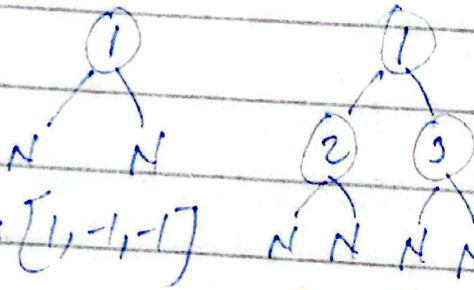
Build a Binary Tree

Preorder Sequence = [1,2,-1,-1,3,4,-1,-1,5,-1,-1]

```
Node {
int data
Node* left
Node* right
Node(val){
  data=val
  left=right=NULL
}
```

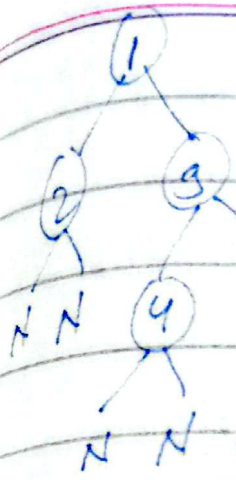


In tree, we have root, left, right



Tree can be built using the pre order seq

So, from the given sequence, we can built the Binary Tree as:



[1,2,-1,-1,3,4,-1,-1,5,-1,-1]

[T.C = O(n)]

```
static idx = -1
Node* Buildtree(preord[]){
  idx++;
  if(pre[idx] == -1) return NULL;
  Node *root = newNode(pre[idx])
  root->left = buildtree(preorder)->left
  root->right = buildtree(preorder)->Right
  return root;
}
```

Traversal Algos

① Preorder Traversal (Recursion Based)

Using the root Node
root, left, right (pattern)
Root Node first that is why Preorder.

- Travel
- Min
- Max
- Count

```
void Preorder(root){
  if(root == NULL){
    return;
  cout << root->data << " ";
  Preorder(root->left)
  Preorder(root->right)
}
```

T.C = O(n)

② Inorder Traversal (Recursion Based)

(left, Root, Right) Pattern.
Start from the left subtree.

```
Inorder(root){
  if(root==NULL){
        return;
  Inorder(root->left)
  cout<<root->data;
  Inorder(root->right)
}
```
TC = O(n)

③ Postorder Traversal (Recursion Based)

left, right, root (pattern)

Start from the left and Right Subtree and Root in the end.

```
postorder(root->left)
postorder(root->right)
root->data.
```
TC = O(n)

④ Level order (Iterative)

- Using loops
- Using level
- Using Queues.

[1|2|3| ...]

Printing 1, 2, 3 ---

3 types: DFS and BFS

DFS	BFS
Depth First Search (Explore Branch by branch)	Breadth First Search (Explore layer by layer)

Steps

① Q.push(root)

② while(Q.size()>0){

```
  Node curr = Q.front();
      Q.pop()
  cout<<curr->data;
  If (curr->left != NULL){
        Q.push[curr->left]
  If (curr->right != NULL){
        Q.push(curr->right)
```
T.C = O(n)

Printing Nodes level by level

→ Adding extra NULL variable

① Q.push(root)
Q.push(NULL)

② while(Qsize > 0){
Node* curr = Q.front()
Q.pop()
if (curr == NULL){
if (!Q.empty()){
cout<<endl;
Q.push(NULL)
continue;
else
break;

NULL
Next line
Break
Q ≠ Empty