**Two methods of traversal :-**

1. BFS (Level order traversal) - Queue
2. DFS (a. Inorder b. Preorder c. Postorder) – Stack

***It is evident from above points that extra space required for Level order traversal is likely to be more when tree is more balanced and extra space for Depth First Traversal is likely to be more when tree is less balanced.***

**The most important points is, BFS starts visiting nodes from root while DFS starts visiting nodes from leaves. So if our problem is to search something that is more likely to closer to root, we would prefer BFS. And if the target node is close to a leaf, we would prefer DFS.**

If height = n, then total number of nodes in a binary tree = 2^(n+1)-1

Divided in two parts to improve readability:- Part I - Trees + Few BST functions Part II – mainly BST

**Part I**

#include <iostream>

using namespace std;

struct node

{int data;

node\* left;

node\* right;

};

////////////////////////////////////////////////////

int count=0;

int size\_my(node\* root) // Pre-order

{if (root==NULL)

{return count;}

else

{count++;//cout<<"asdf"<<count;

count=size\_my(root->left);

count=size\_my(root->right);

}

return count;

}

///////////////////////////////////////////////////

int size(node\* root) // In-order

{if (root==NULL)

{return 0;}

else

{return (size(root->left)+1+size(root->right));

}

}

node\* newNode(int data) // Helper function to

{node\* ptr=new node; // new node

ptr->data=data;

ptr->left=NULL;

ptr->right=NULL;

return ptr;}

int max\_depth(node\* root) // No. of nodes in the longest branch

{if (root==NULL)

{return 0;}

else

{int l=max\_depth(root->left);

int r=max\_depth(root->right);

if (l>r)

{return l+1;}

else

{return r+1;}

}

}

int maxm=-1; // Recursive

int max\_val(node\* root) // Max val of a binary search tree

{if (root->right==NULL)

{maxm=root->data;

cout<<"The maximum value of tree = ";

return maxm;}

else

{return max\_val(root->right);

}

}

int min\_val(node\* root) // Min val of a binary search tree

{if (root==NULL) // Iterative

{cout<<"The tree is empty";

return (-1);}

else

{

node\* curr=root;

while(curr->left!=NULL)

{curr=curr->left;}

cout<<"The minimum value of tree = ";

return(curr->data);

}

}

void print\_inorder(node\* root) // Recursive inorder printing of

{if (root==NULL) return; // tree elements

else {print\_inorder(root->left);

cout<<root->data<<" ";

print\_inorder(root->right);}

}

void print\_preorder(node\* root) // Recursive preorder printing of

{if (root==NULL) // tree elements

{return;}

else {cout<<root->data<<" ";

print\_preorder(root->left);

print\_preorder(root->right);}

}

void print\_postorder(node\* root) // Recursive postorder printing of

{if (root==NULL) return; // tree elements

else {print\_postorder(root->left);

print\_postorder(root->right);

cout<<root->data<<" ";}

}

void print\_onelevel(node\* root,int level) // BFS - Level by level

{if (root==NULL) // Level 0, 1, 2, and 3 for height = 3

{return;}

if (level==0)

{cout<<root->data<<" ";}

else if (level>0)

{print\_onelevel(root->left,level-1);

print\_onelevel(root->right,level-1);}

}

void print\_BFS(node\* root)

{int height = max\_depth(root)-1; // Height is one less than no. of nodes

for (int i=0;i<=height;i++) // in the longest branch

{print\_onelevel(root,i);

}

}

int main()

{

node\* root=new node;

root=newNode(6);

root->left=newNode(4);

root->right=newNode(8);

root->right->right=newNode(9);

root->right->left=newNode(7);

root->left->right=newNode(5);

root->left->left=newNode(2);

root->left->left->right=newNode(3);

root->left->left->left=newNode(1);

root->left->left->left->left=newNode(0);

cout<<"The size of the tree is = "<<size(root)<<endl;

cout<<max\_val(root)<<endl;

cout<<min\_val(root)<<endl;

cout<<"The level by level print (BFS) ";

print\_BFS(root);

cout<<endl;

cout<<"Inorder print ";

print\_inorder(root);

cout<<endl;

cout<<"Preorder print ";

print\_preorder(root);

cout<<endl;

cout<<"Postorder print ";

print\_postorder(root);

cout<<endl;

return 0;

}