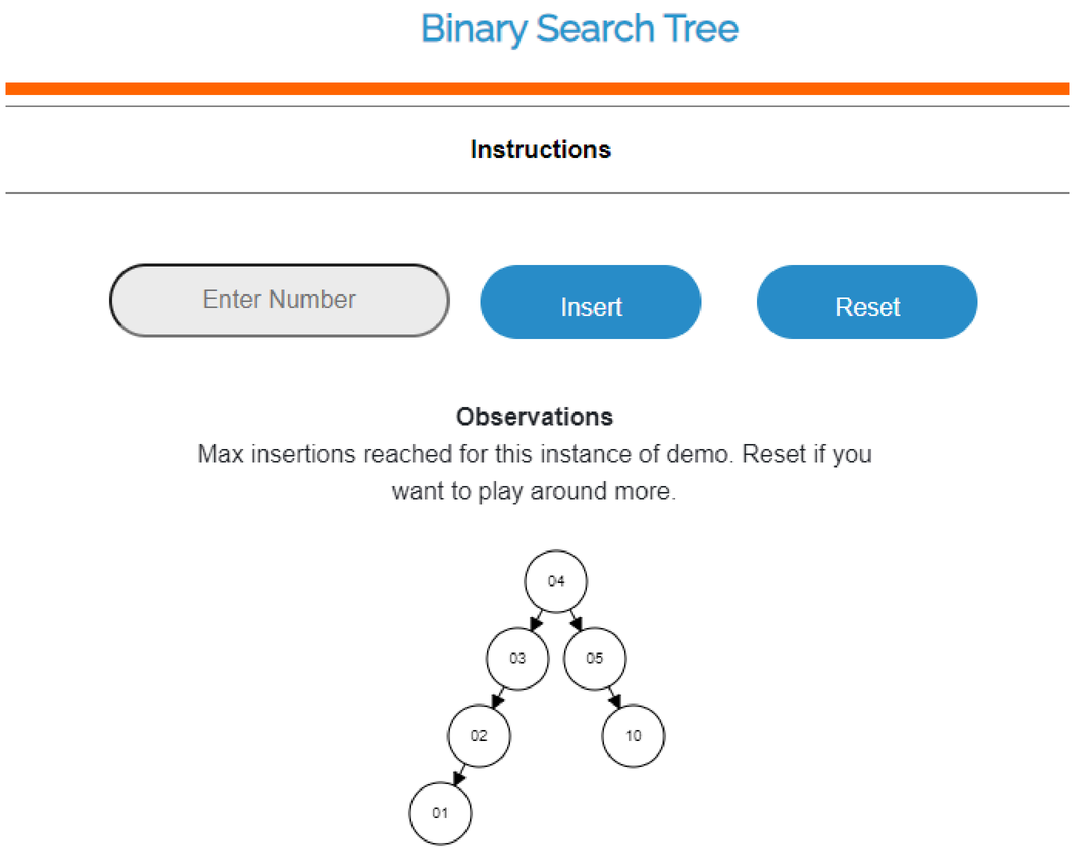
Week 7-LAB B

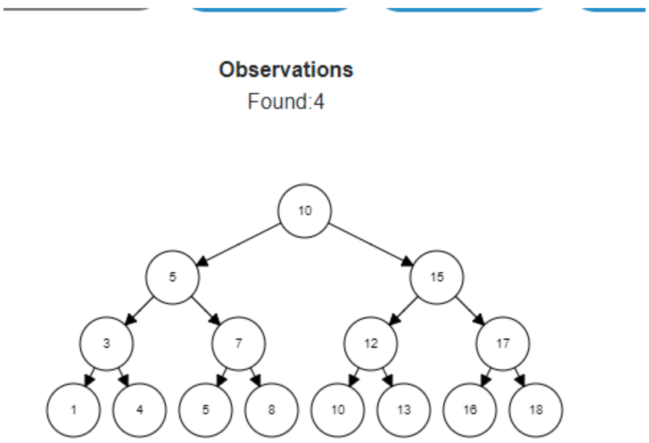


Binary Search Tree

Instructions

Observations

Max insertions reached for this instance of demo. Reset if you want to play around more.



Observations

Found:4

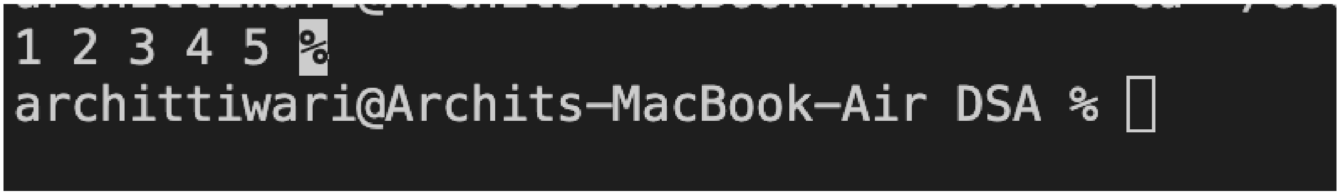
1. Given an array, create a BST. Display in the following manner:

   i. Breadth-first traversal (level-order traversal)

   ii. Depth-First traversal (In-, pre-, post-order traversals)

Ans:

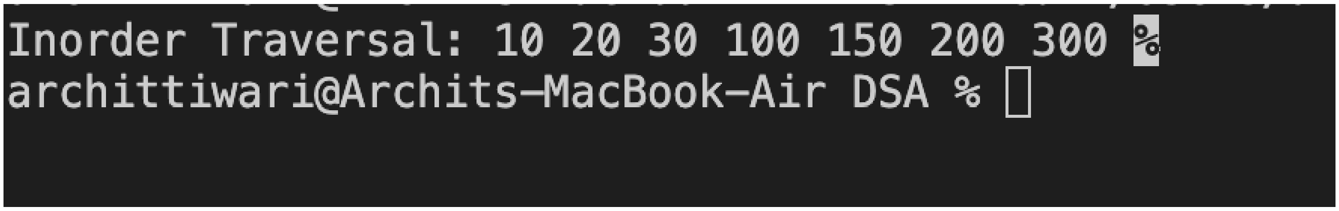
```
#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node *left, *right;
    Node(int value) {
        data = value;
        left = NULL;
        right = NULL;
    }
};

void printCurrentLevel(Node* root, int level);
int height(Node* node);


void printLevelOrder(Node* root) {
    int h = height(root);
    for (int i = 1; i <= h; i++)
        printCurrentLevel(root, i);
}

void printCurrentLevel(Node* root, int level) {
    if (root == NULL)
        return;
    if (level == 1)
        cout << root->data << " ";
    else if (level > 1) {
        printCurrentLevel(root->left, level - 1);
        printCurrentLevel(root->right, level - 1);
    }
}


int height(Node* node) {
    if (node == NULL)
        return 0;
    else {
        int lheight = height(node->left);
        int rheight = height(node->right);
        return (lheight > rheight) ? (lheight + 1) :
                                    (rheight + 1);
    }
}

int main() {
    Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);
    printLevelOrder(root);
    return 0;
}
```



```
1 2 3 4 5 %
archittiwari@Archits-MacBook-Air DSA % 
```

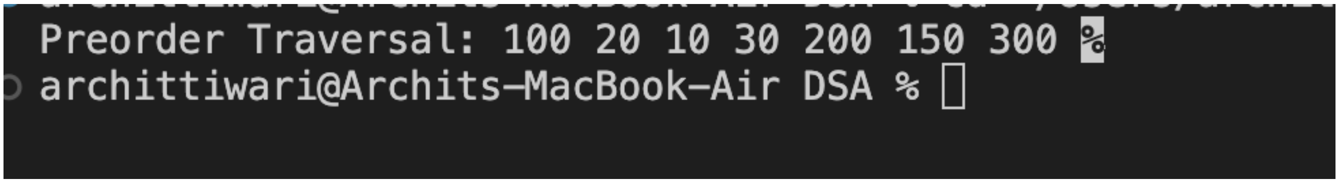
```
#include<iostream>

using namespace std;
class Node {
public:
    int data;
    Node* left;
    Node* right;
    Node(int v)
    {
        this->data = v;
        this->left = this->right = NULL;
    }
};

void printInorder(Node* node)
{
    if (node == NULL)
        return;

    printInorder(node->left);
    cout << node->data << " ";
    printInorder(node->right);
}
```

```
int main()
{

    Node* root = new Node(100);
    root->left = new Node(20);
    root->right = new Node(200);
    root->left->left = new Node(10);
    root->left->right = new Node(30);
    root->right->left = new Node(150);
    root->right->right = new Node(300);

    cout << "Inorder Traversal: ";
    printInorder(root);
    return 0;
}
```



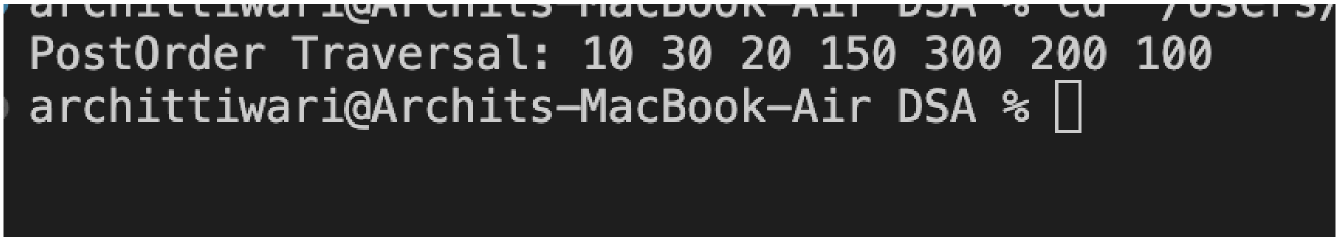
```
Inorder Traversal: 10 20 30 100 150 200 300 %
archittiwari@Archits-MacBook-Air DSA % 
```

```
#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* left;
    Node* right;
    Node(int v)
    {
        this->data = v;
        this->left = this->right = NULL;
    }
};


void printPreOrder(Node* node)
{
    if (node == NULL)
        return;

    cout << node->data << " ";
    printPreOrder(node->left);
    printPreOrder(node->right);
}


int main()
{
    Node* root = new Node(100);
    root->left = new Node(20);
    root->right = new Node(200);
    root->left->left = new Node(10);
    root->left->right = new Node(30);
    root->right->left = new Node(150);
    root->right->right = new Node(300);
    cout << "Preorder Traversal: ";
    printPreOrder(root);
    return 0;
}
```



```
Preorder Traversal: 100 20 10 30 200 150 300 %
archittiwari@Archits-MacBook-Air DSA % 
```

```
#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* left;
    Node* right;
    Node(int v)
    {
        this->data = v;
        this->left = this->right = NULL;
    }
};

void printPostOrder(Node* node)
{
    if (node == NULL)
        return;
    printPostOrder(node->left);
    printPostOrder(node->right);
    cout << node->data << " ";
}

int main()
{
    Node* root = new Node(100);
    root->left = new Node(20);
    root->right = new Node(200);
    root->left->left = new Node(10);
    root->left->right = new Node(30);
    root->right->left = new Node(150);
    root->right->right = new Node(300);
    cout << "PostOrder Traversal: ";
    printPostOrder(root);
    cout << "\n";

    return 0;
}
```

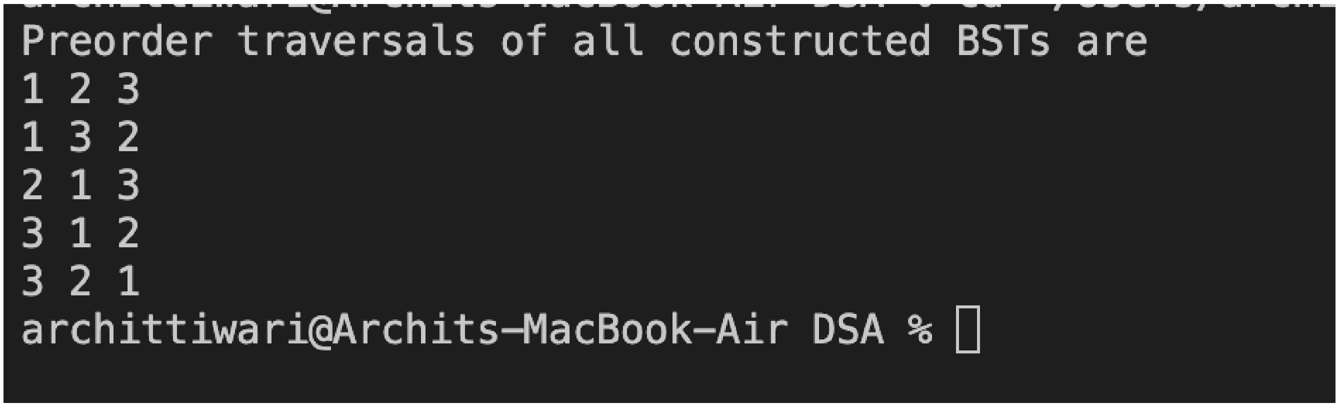


```
architiwari@Archits-MacBook-Air DSA % cd /Users/
PostOrder Traversal: 10 30 20 150 300 200 100
architiwari@Archits-MacBook-Air DSA % 
```

2. Given an integer n, return *all the structurally unique* BST'*s, which has exactly* n *nodes of unique values from* 1 *to* n. Return the answer in any order.
Input: n=3
Output: 5
Justification: [1,null,2,null],
[1,null,3,2],
[2,1,3],
[3,1,,null, null, 2],
[3,2,null, 1]

Ans:

```
#include <iostream>
#include<vector>
using namespace std;
struct node
{
        int key;
        struct node *left, *right;
};

struct node *newNode(int item)
{
        struct node *temp = new node;
        temp->key = item;
        temp->left = temp->right = NULL;
        return temp;
}

void preorder(struct node *root)
{
        if (root != NULL)
        {
                cout << root->key << " ";
                preorder(root->left);
                preorder(root->right);
        }
}

vector<struct node *> constructTrees(int start, int end)
{
        vector<struct node *> list;


        if (start > end)
        {
                list.push_back(NULL);
                return list;
        }


        for (int i = start; i <= end; i++)
        {
                vector<struct node *> leftSubtree = constructTrees(start, i - 1);
                vector<struct node *> rightSubtree = constructTrees(i + 1, end);
                for (int j = 0; j < leftSubtree.size(); j++)
                {
                        struct node* left = leftSubtree[j];
                        for (int k = 0; k < rightSubtree.size(); k++)
                        {
                                struct node * right = rightSubtree[k];
                                struct node * node = newNode(i);
                                node->left = left;
                                node->right = right;
                                list.push_back(node);
                        }
                }
        }
        return list;
}

int main()
{
	vector<struct node *> totalTreesFrom1toN = constructTrees(1, 3);
	cout << "Preorder traversals of all constructed BSTs are \n";
	for (int i = 0; i < totalTreesFrom1toN.size(); i++)
	{
		preorder(totalTreesFrom1toN[i]);
		cout << endl;
	}
	return 0;
}
```



```
Preorder traversals of all constructed BSTs are
1 2 3
1 3 2
2 1 3
3 1 2
3 2 1
archittiwari@Archits-MacBook-Air DSA % 
```

3. Definition of Lowest Common Ancestor (LCA): Let '*T*' be a rooted tree. The lowest common ancestor between two nodes '*n1*' and '*n2*' is defined as the lowest node in '*T*' that has both '*n1*' and '*n2*' as descendants (where we allow a node to be a descendant of itself). The LCA of '*n1*' and '*n2*' in '*T*' is the shared ancestor of '*n1*' and '*n2*' that is located farthest from the root [i.e., closest to '*n1*' and '*n2*'].

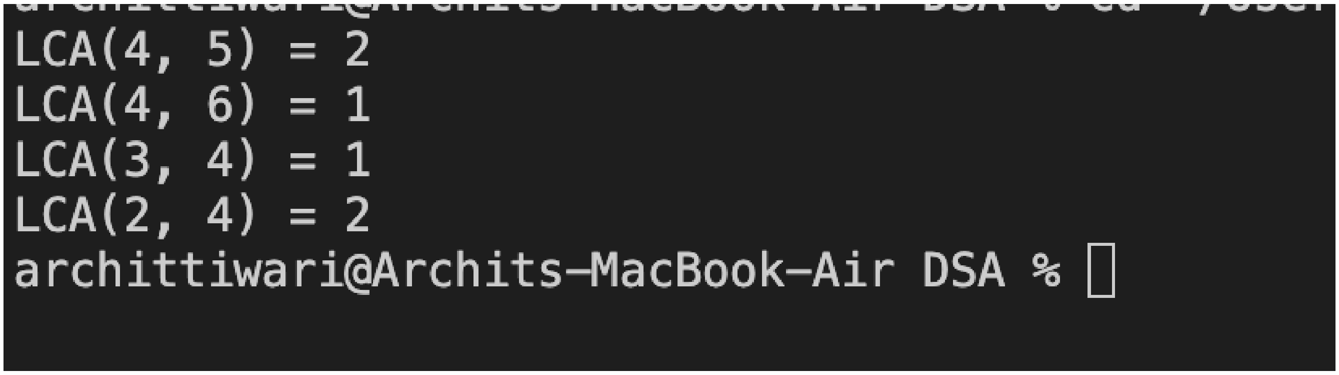
	Input: **root= [6,2,8,0,4,7,9,null, null, 3,5]** **p = 2, q = 8** **Output: 6** **Explanation: LCA of nodes 2, 8 is 6.**

Assumption: Node may be a descendant of itself.

Ans:

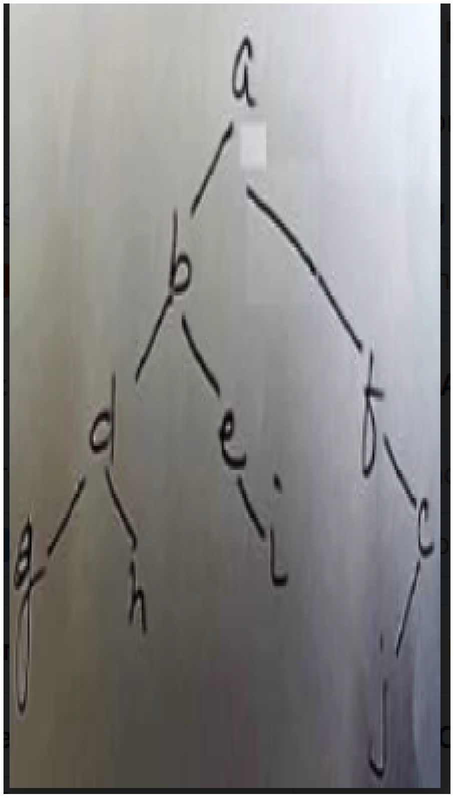
```
#include <iostream>
#include<vector>
using namespace std;

struct Node {
    int key;
    Node *left, *right;
    Node(int k) {
        key = k;
        left = NULL;
        right = NULL;
    }
};

bool findPath(Node* root, vector<int>& path, int k) {
    if (!root)
        return false;
    path.push_back(root->key);

    if (root->key == k)
        return true;


    if ((root->left && findPath(root->left, path, k)) ||
        (root->right && findPath(root->right, path, k)))
        return true;
    path.pop_back();
    return false;
}


int findLCA(Node* root, int n1, int n2) {

    vector<int> path1, path2;
    if (!findPath(root, path1, n1) ||
        !findPath(root, path2, n2))
        return -1;
    int i;
    for (i = 0; i < path1.size() && i < path2.size(); i++) {
        if (path1[i] != path2[i])
            break;
    }
    return path1[i - 1];
}

int main() {
    Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);
    root->right->left = new Node(6);
    root->right->right = new Node(7);

    cout << "LCA(4, 5) = " << findLCA(root, 4, 5) << endl;
    cout << "LCA(4, 6) = " << findLCA(root, 4, 6) << endl;
    cout << "LCA(3, 4) = " << findLCA(root, 3, 4) << endl;
    cout << "LCA(2, 4) = " << findLCA(root, 2, 4) << endl;

    return 0;
```

}



```
archittiwari@Archits-MacBook-Air DSA % cd "/User
LCA(4, 5) = 2
LCA(4, 6) = 1
LCA(3, 4) = 1
LCA(2, 4) = 2
archittiwari@Archits-MacBook-Air DSA % 
```

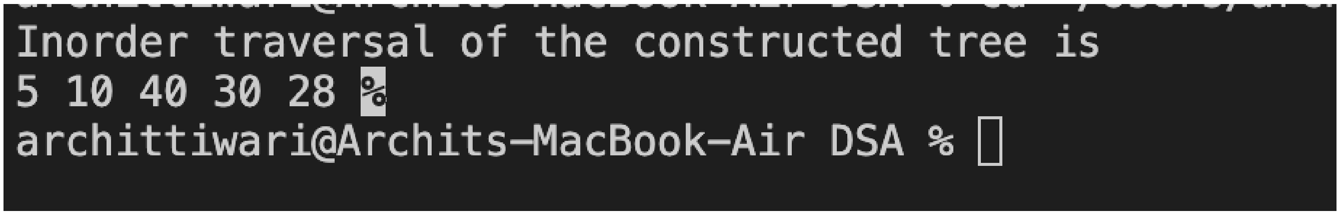
4. Given a list that depicts the in-order traversal of a binary tree, develop the binary search tree itself. Example: "gdhbeiafjc" is the in-order traversal for the BST:



Ans:

```
#include <iostream>
#include<vector>
using namespace std;
class node
{
        public:
        int data;
```

```
    node* left;
    node* right;
};


int max(int inorder[], int strt, int end);
node* newNode(int data);
node* buildTree (int inorder[], int start, int end)
{
    if (start > end)
        return NULL;
    int i = max (inorder, start, end);
    node *root = newNode(inorder[i]);
    if (start == end)
        return root;
    root->left = buildTree (inorder, start, i - 1);
    root->right = buildTree (inorder, i + 1, end);

    return root;
}
int max (int arr[], int strt, int end)
{
    int i, max = arr[strt], maxind = strt;
    for(i = strt + 1; i <= end; i++)
    {
        if(arr[i] > max)
        {
            max = arr[i];
            maxind = i;
        }
    }
    return maxind;
}


node* newNode (int data)
{
    node* Node = new node();
    Node->data = data;
    Node->left = NULL;
    Node->right = NULL;

    return Node;
}


void printInorder (node* node)
{
    if (node == NULL)
        return;
    printInorder (node->left);
    cout<<node->data<<" ";
    printInorder (node->right);
}

int main()
{

    int inorder[] = {5, 10, 40, 30, 28};
    int len = sizeof(inorder)/sizeof(inorder[0]);
    node *root = buildTree(inorder, 0, len - 1);
    cout << "Inorder traversal of the constructed tree is \n";
    printInorder(root);
    return 0;
}
```



```
Inorder traversal of the constructed tree is
5 10 40 30 28 %
archittiwari@Archits-MacBook-Air DSA % 
```