**Problem Name:  Increasing Order Search Tree**

**Topics:** Stack, Tree, DFS, BST, Binary tree

**Companies:** Facebook, Amazon, apple

**Level: Easy**

**Language:** C++

**Problem Statement:**

create a binary search tree by taking input and -1 as null, rearrange the tree in **in-order** so that the leftmost node in the tree is now the root of the tree, and every node has no left child and only one right child. Print the tree in inorder level

**Input Format:** The first line of input contains an integer n (no of total nodes in BST including null)

Next line contains n space-separated integer values.

**Output Format:** Print the Binary search tree in Inorder level

**Constraints:**

* The number of nodes in the given tree will be in the range [1, 100].
* 0 <= Node.val <= 1000

**Examples:**

**Brute force Solution:**

**Explanation:** Simply do a normal inorder traversal. But while doing keep on adding to the right of a moving pointer as required in question. Later return the first position of this moving pointer as is the root of the newly constructed tree. Here I have used head to store the first position of the moving pointer. (Initially I have used a dummy node, so final root is head->right)

**Code:**

#include <bits/stdc++.h>

using namespace std;

class Node{

public:

    int data;

    Node \*left, \*right;

    Node(int val){

        data = val;

        left = nullptr;

        right = nullptr;

    }

};

void InOrder(Node \*root)

{

    if (root == NULL)

        return;

    InOrder(root->left);

    cout << root->data << " ";

    InOrder(root->right);

}

Node \*moving,\*head;

void inorder(Node \*root){

    if(!root) return;

    inorder(root->left);

    moving->right= new Node(root->data);

    moving=moving->right;

    inorder(root->right);

}

Node\* increasingBST(Node\* root) {

    moving=new Node(0);

    head=moving;

    inorder(root);

    return head->right;

}

int main()

{

    int n;

    cin >> n;

    vector<int> v(n);

    for (int i = 0; i < n; i++)

        cin >> v[i];

    queue<Node \*> q;

    Node \*root = new Node(v[0]);

    q.push(root);

    int i = 1;

    while (i < n)

    {

        Node \*curr = q.front();

        q.pop();

        if (curr)

        {

            Node \*temp1, \*temp2;

            if (v[i] != -1)

                temp1 = new Node(v[i]);

            else

                temp1 = nullptr;

            curr->left = temp1;

            q.push(temp1);

            if (i + 1 < n)

            {

                if (v[i + 1] != -1)

                    temp2 = new Node(v[i + 1]);

                else

                    temp2 = nullptr;

                curr->right = temp2;

                q.push(temp2);

            }

            i += 2;

        }

    }

    Node\* result = increasingBST(root);

    InOrder(result);

    return 0;

}

**Time Complexity**: O(n)

**Space Complexity:** O(height)

**Optimized Solution:**

**Explanation:**

Recursively call function increasingBST(TreeNode root, TreeNode tail)  
tail is its next node in inorder,（the word next may be easier to understand, but it’s a keyword in python)

If root == null, the head will be tail, so we return tail directly.

we recursively call increasingBST(root.left, root),  
change left subtree into the linked list + current node.

we recursively call increasingBST(root.right, tail),  
change right subtree into the linked list + tail.

**Code:**

#include <bits/stdc++.h>

using namespace std;

class Node{

public:

    int data;

    Node \*left, \*right;

    Node(int val){

        data = val;

        left = nullptr;

        right = nullptr;

    }

};

void InOrder(Node \*root)

{

    if (root == NULL)

        return;

    InOrder(root->left);

    cout << root->data << " ";

    InOrder(root->right);

}

Node\* increasingBST(Node\* root, Node\* tail = NULL) {

        if (!root) return tail;

        Node\* res = increasingBST(root->left, root);

        root->left = NULL;

        root->right = increasingBST(root->right, tail);

        return res;

}

int main()

{

    int n;

    cin >> n;

    vector<int> v(n);

    for (int i = 0; i < n; i++)

        cin >> v[i];

    queue<Node \*> q;

    Node \*root = new Node(v[0]);

    q.push(root);

    int i = 1;

    while (i < n)

    {

        Node \*curr = q.front();

        q.pop();

        if (curr)

        {

            Node \*temp1, \*temp2;

            if (v[i] != -1)

                temp1 = new Node(v[i]);

            else

                temp1 = nullptr;

            curr->left = temp1;

            q.push(temp1);

            if (i + 1 < n)

            {

                if (v[i + 1] != -1)

                    temp2 = new Node(v[i + 1]);

                else

                    temp2 = nullptr;

                curr->right = temp2;

                q.push(temp2);

            }

            i += 2;

        }

    }

    Node\* result = increasingBST(root);

    InOrder(result);

    return 0;

}

**Time Complexity**: O(n)

**Space Complexity:** O(height)