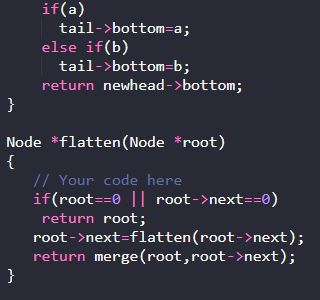
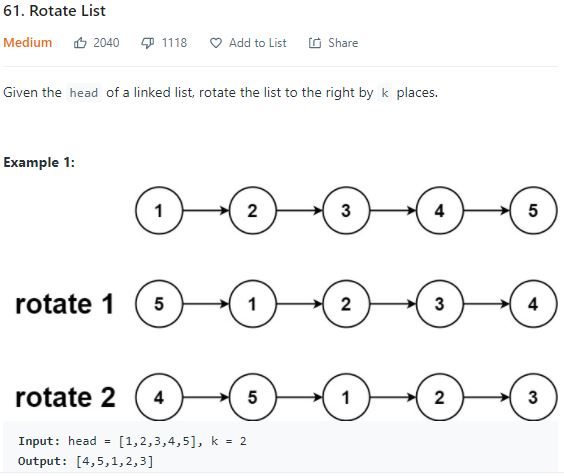
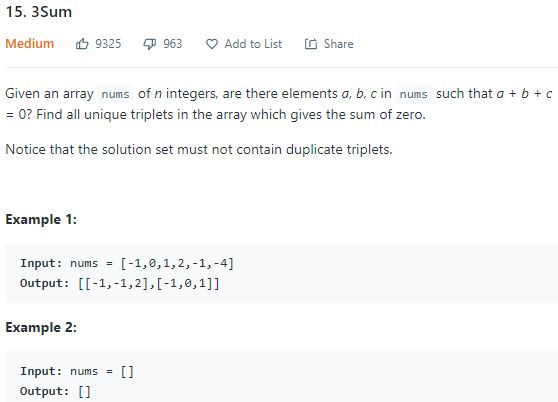
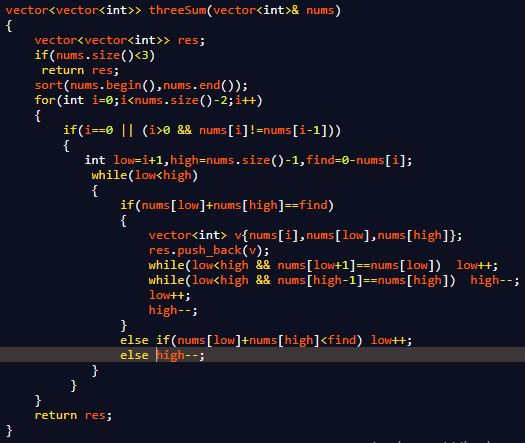


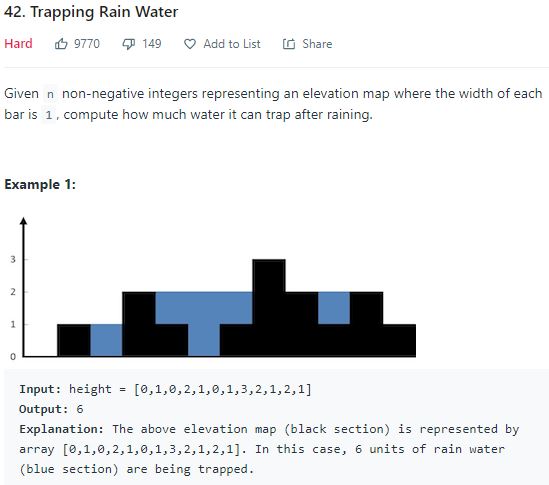
****

****

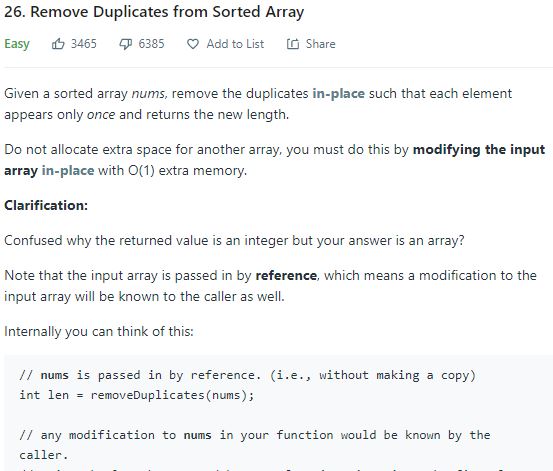
****

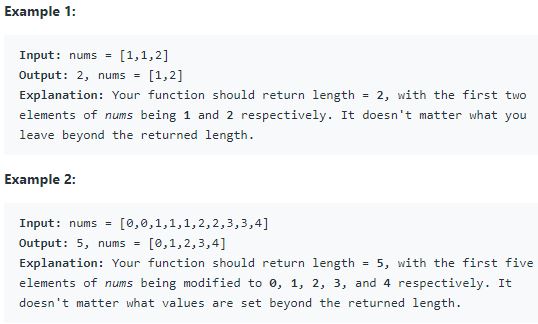
****

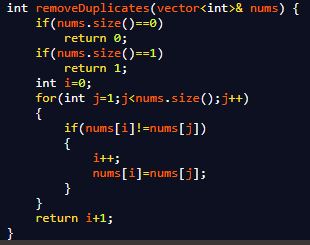
****

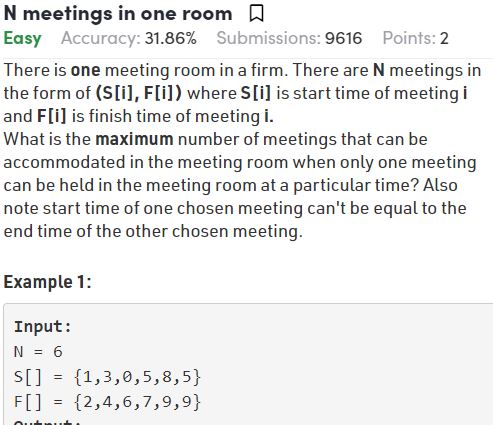
****

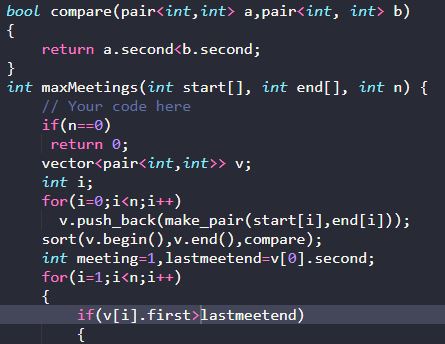
****

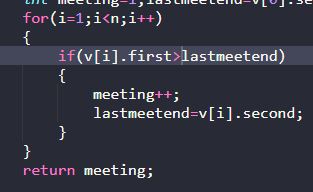
****

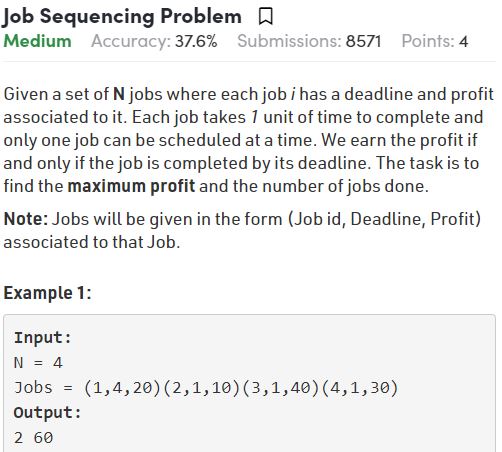
****

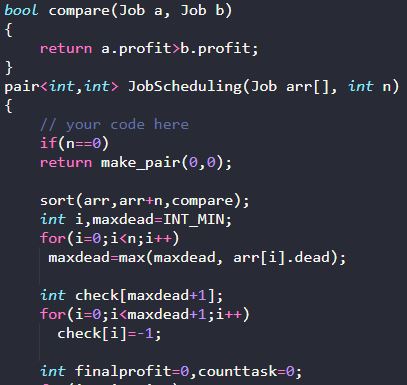
****

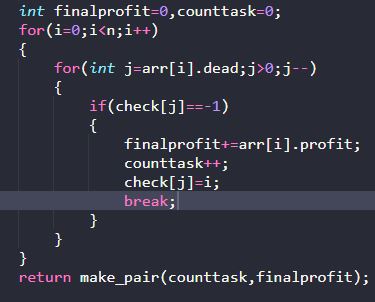
****

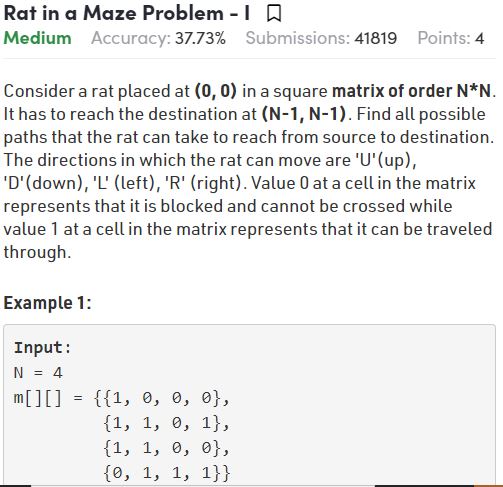
****

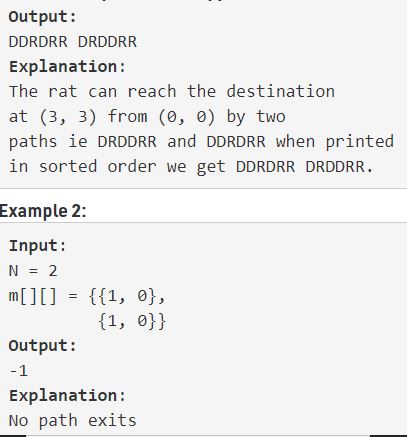
****

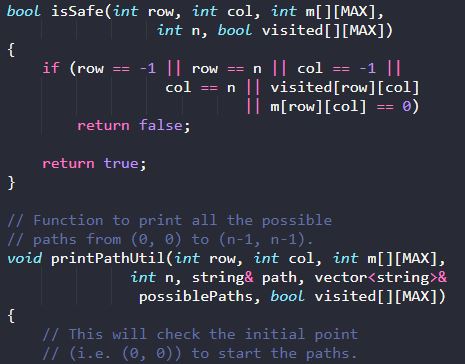
****

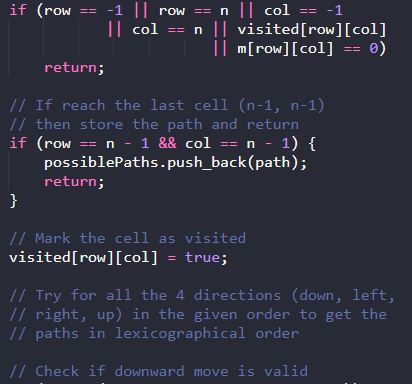
****

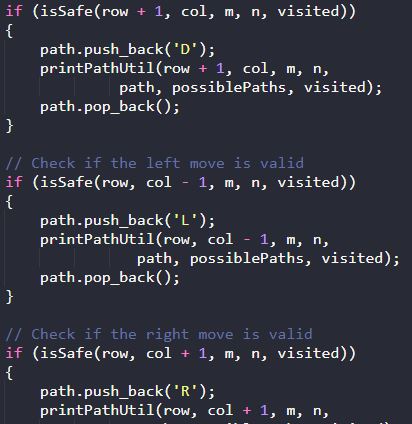
****

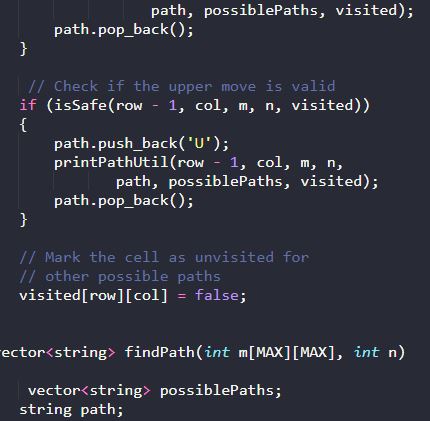
****

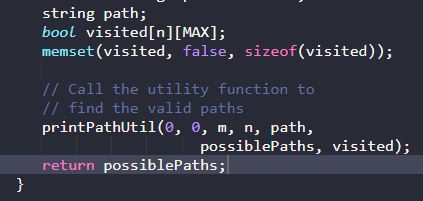
****

****

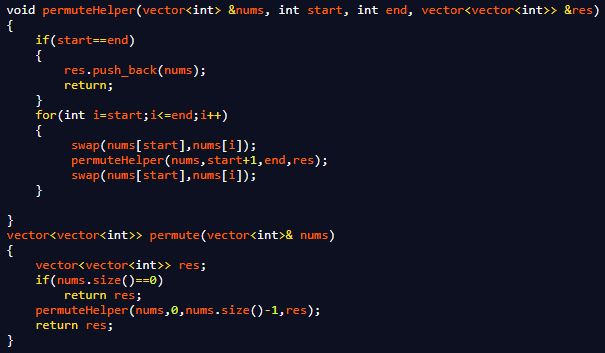
****

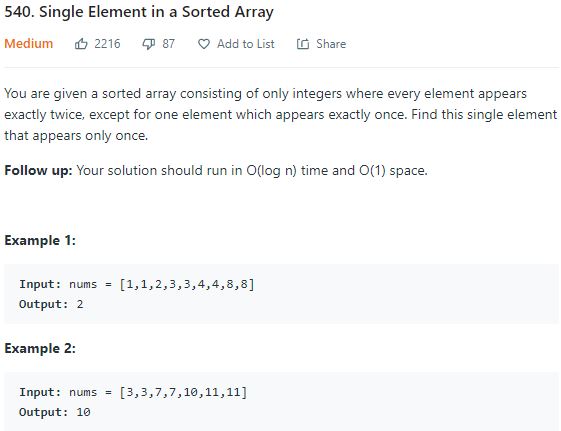
****

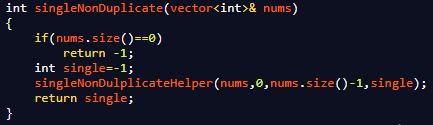
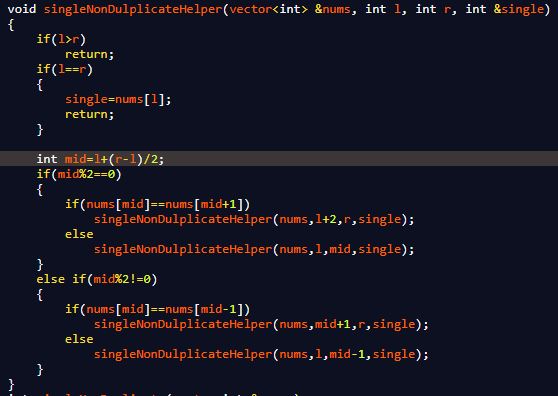
****

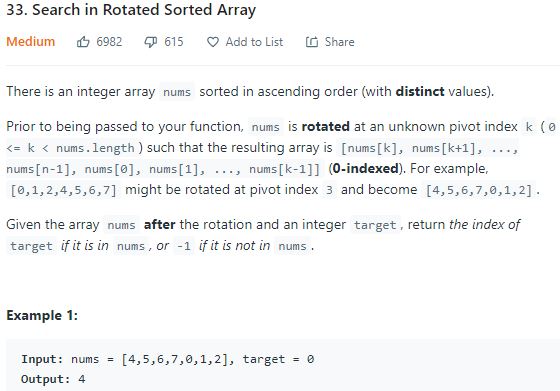
****

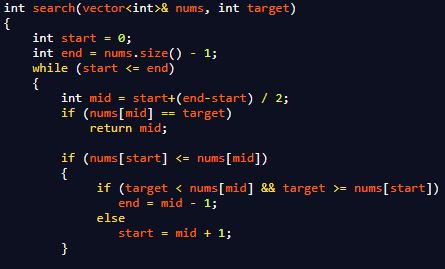
****

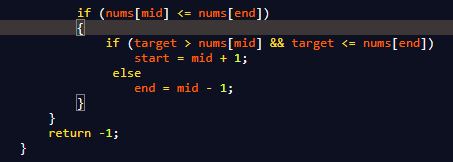
****

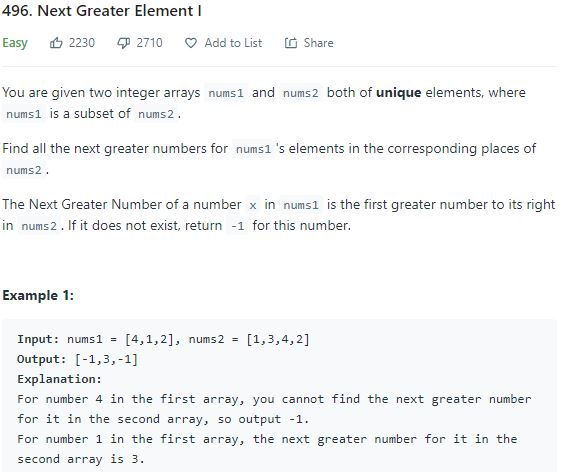
****

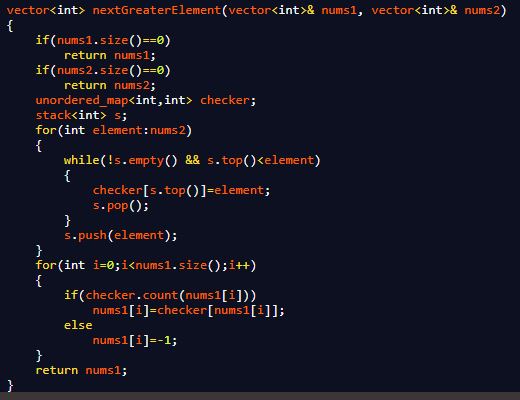
****

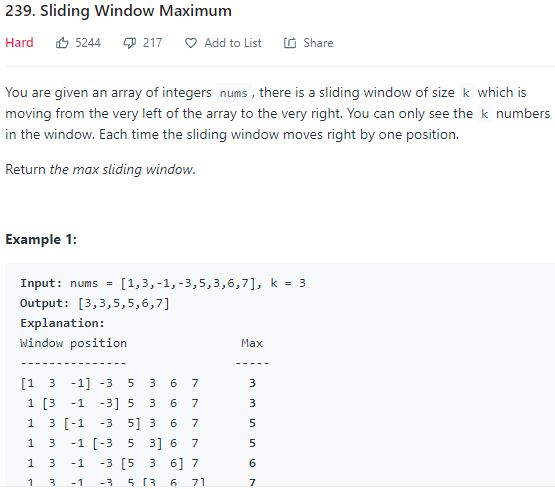
****

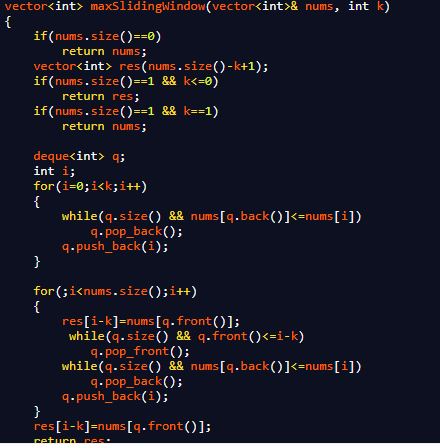
****

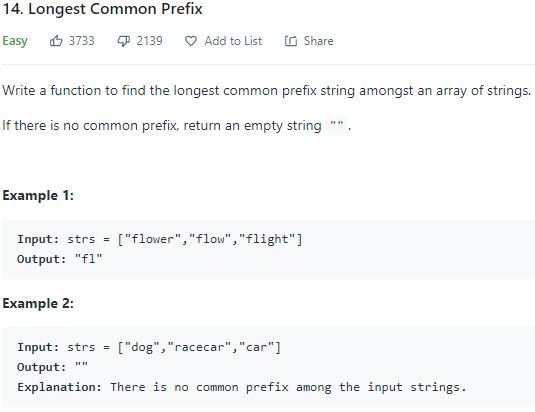
****

****

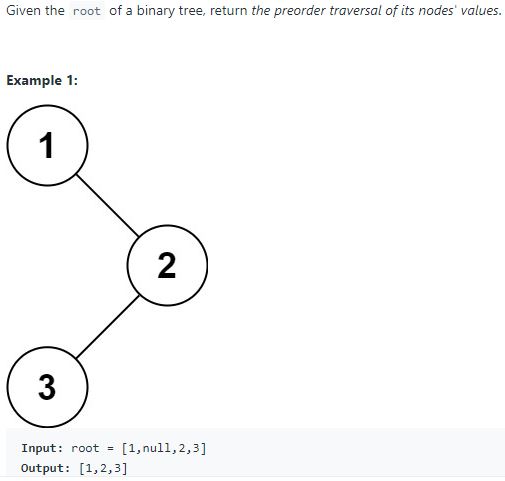
****

****

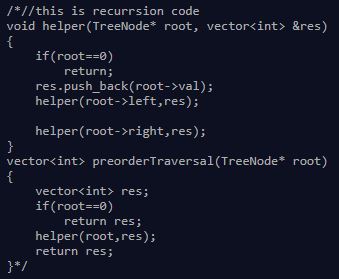
****

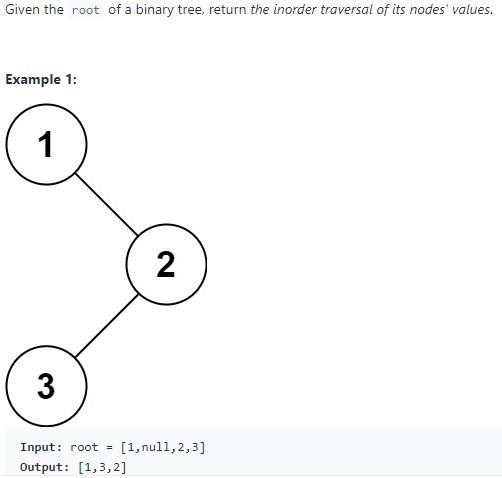
****

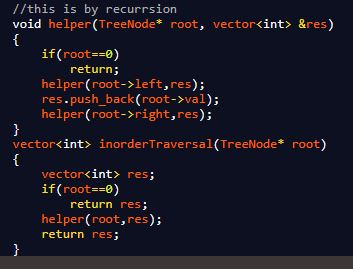
****

****

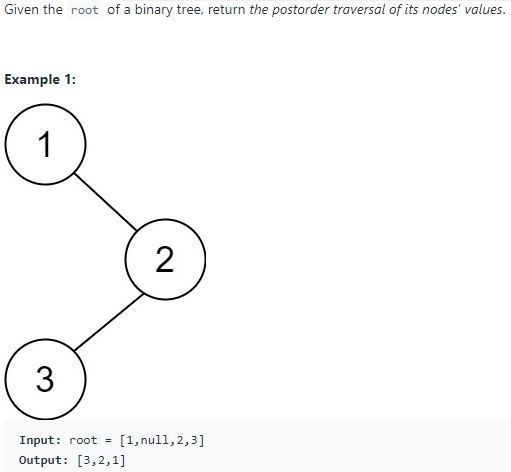
****

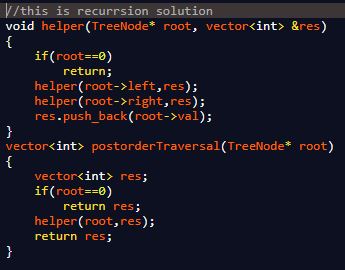
****

****

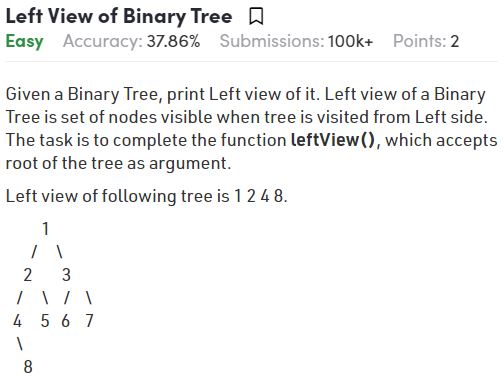
****

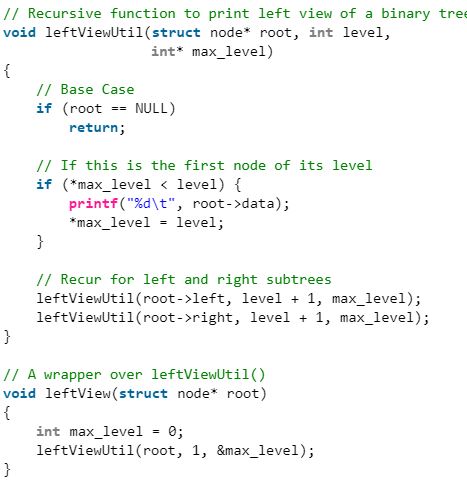
****

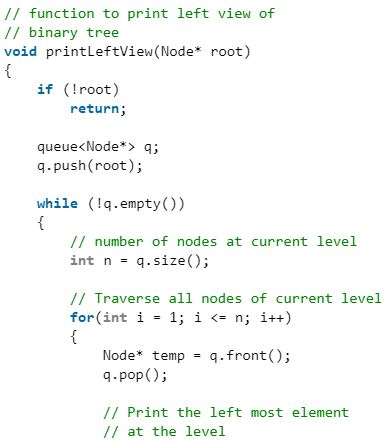
****

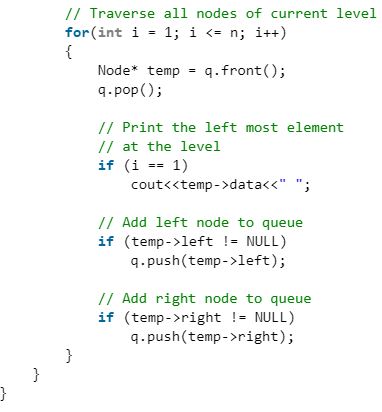
****

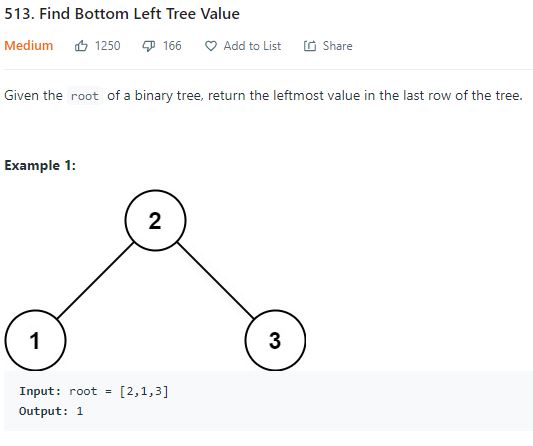
****

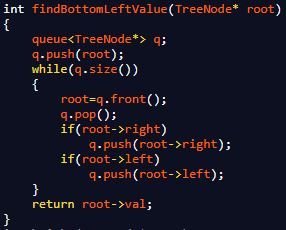
****

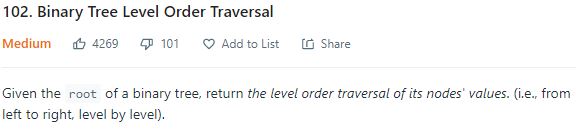
****

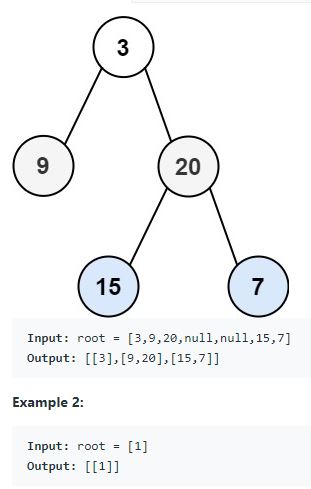
****

****

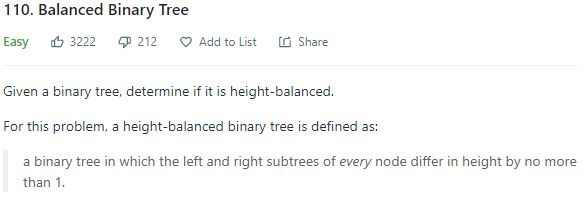
****

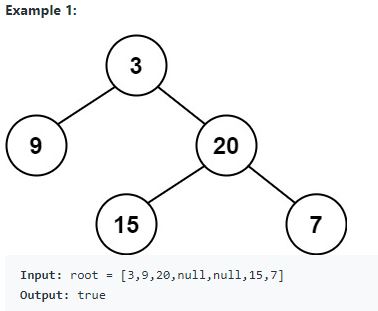
****

****

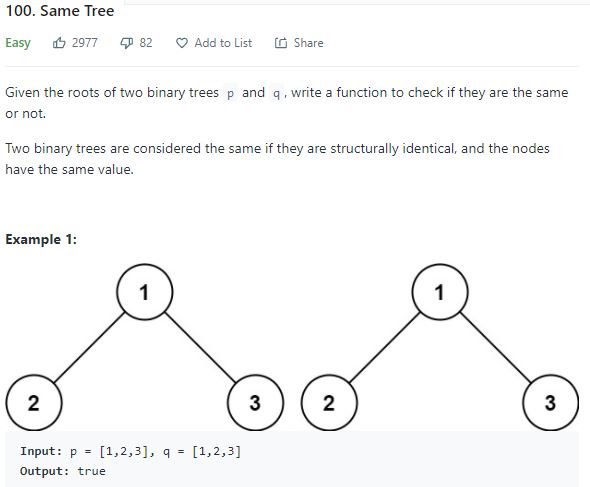
****

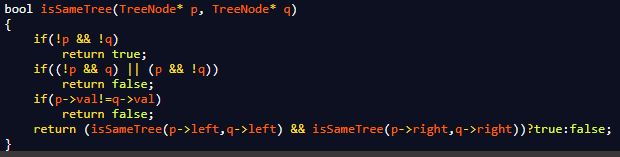
****

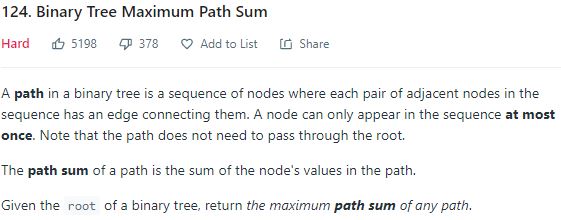
****

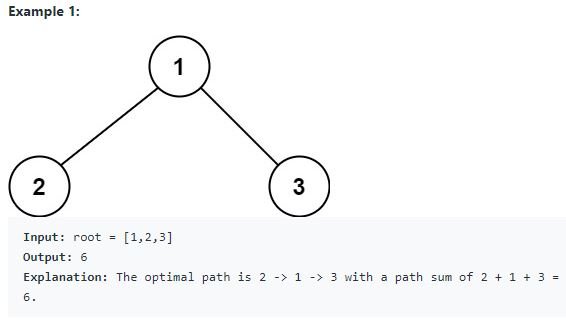
****

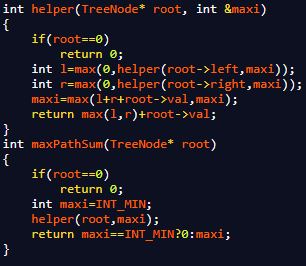
****

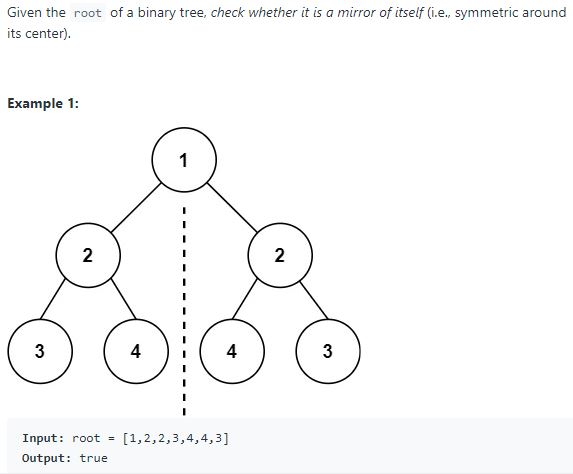
****

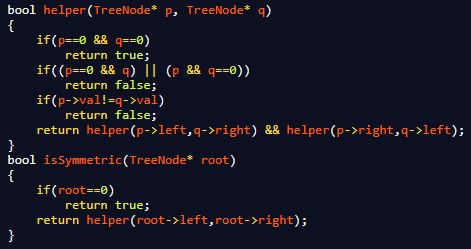
****

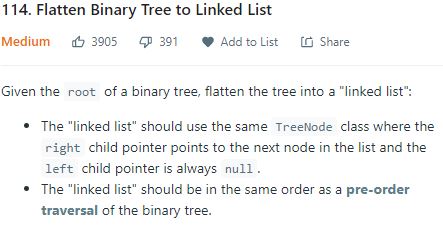
****

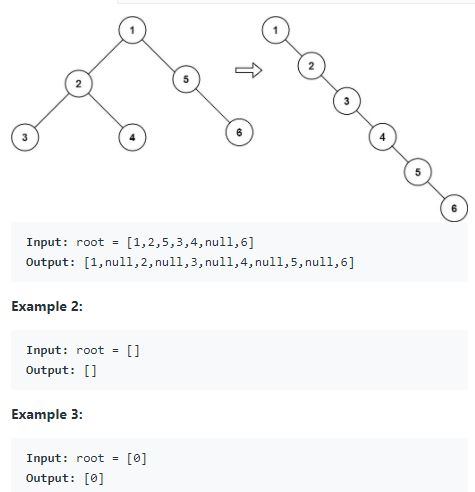
****

****

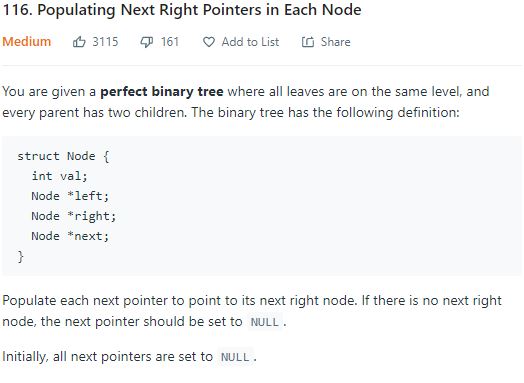
****

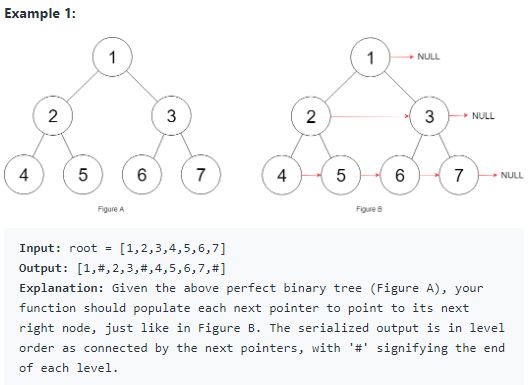
****

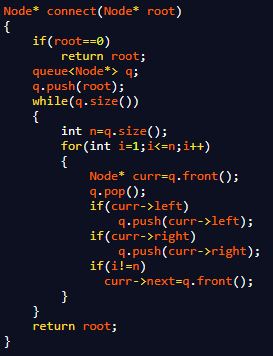
****

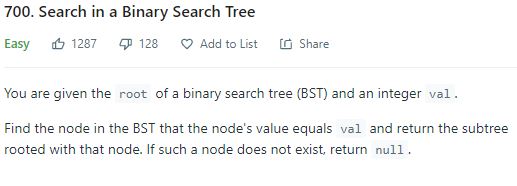
****

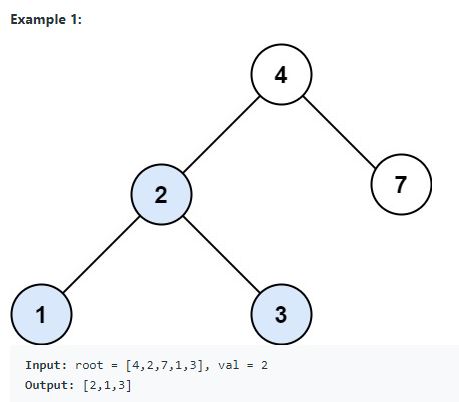
****

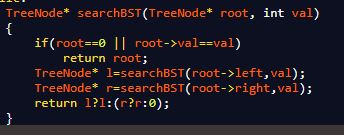
****

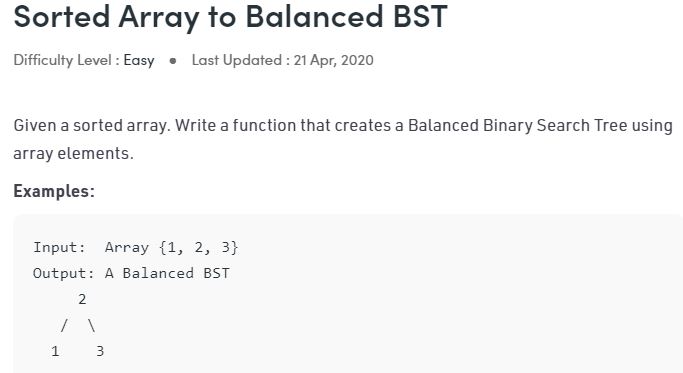
****

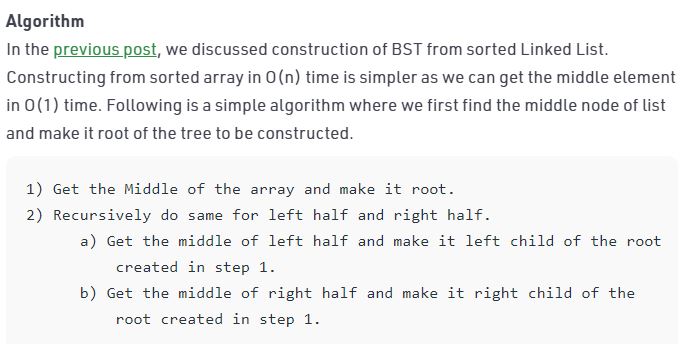
****

****

****

****

****

****

// C++ program to print BST in given range

#include<bits/stdc++.h>

using namespace std;

/\* A Binary Tree node \*/

class TNode

{

    public:

    int data;

    TNode\* left;

    TNode\* right;

};

TNode\* newNode(int data);

/\* A function that constructs Balanced

Binary Search Tree from a sorted array \*/

TNode\* sortedArrayToBST(int arr[],

                        int start, int end)

{

    /\* Base Case \*/

    if (start > end)

    return NULL;

    /\* Get the middle element and make it root \*/

    int mid = (start + end)/2;

    TNode \*root = newNode(arr[mid]);

    /\* Recursively construct the left subtree

    and make it left child of root \*/

    root->left = sortedArrayToBST(arr, start,

                                    mid - 1);

    /\* Recursively construct the right subtree

    and make it right child of root \*/

    root->right = sortedArrayToBST(arr, mid + 1, end);

    return root;

}

/\* Helper function that allocates a new node

with the given data and NULL left and right

pointers. \*/

TNode\* newNode(int data)

{

    TNode\* node = new TNode();

    node->data = data;

    node->left = NULL;

    node->right = NULL;

    return node;

}

/\* A utility function to print

preorder traversal of BST \*/

void preOrder(TNode\* node)

{

    if (node == NULL)

        return;

    cout << node->data << " ";

    preOrder(node->left);

    preOrder(node->right);

}

// Driver Code

int main()

{

    int arr[] = {1, 2, 3, 4, 5, 6, 7};

    int n = sizeof(arr) / sizeof(arr[0]);

    /\* Convert List to BST \*/

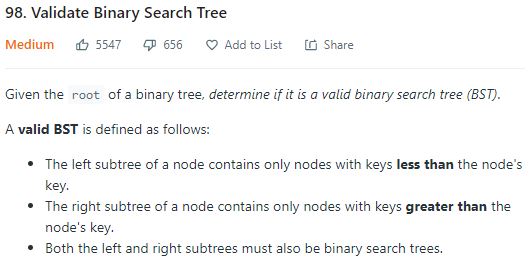
    TNode \*root = sortedArrayToBST(arr, 0, n-1);

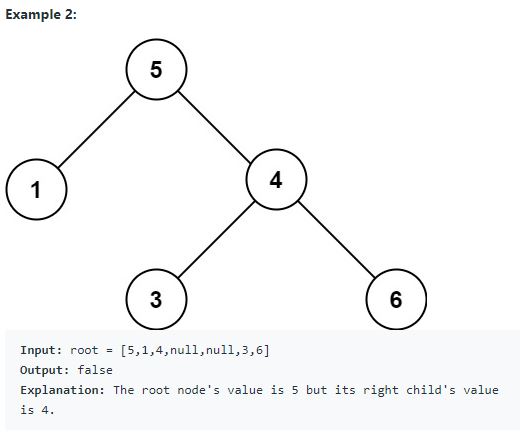
    cout << "PreOrder Traversal of constructed BST \n";

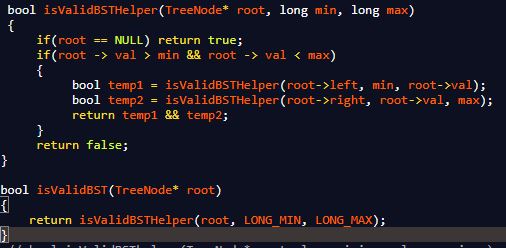
    preOrder(root);

    return 0;

}

****

****

****

Inorder predecessor and successor for a given key in BST

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium)
* Last Updated : 01 Mar, 2021

I recently encountered with a question in an interview at e-commerce company. The interviewer asked the following question:  
There is BST given with root node with key part as integer only. The structure of each node is as follows:

* C++
* Java

|  |
| --- |
| struct Node  {      int key;      struct Node \*left, \*right ;  }; |

You need to find the inorder successor and predecessor of a given key. In case the given key is not found in BST, then return the two values within which this key will lie.

[Recommended: Please solve it on “***PRACTICE***” first, before moving on to the solution.](https://practice.geeksforgeeks.org/problems/predecessor-and-successor/1)

Following is the algorithm to reach the desired result. Its a recursive method:

Input: root node, key

output: predecessor node, successor node

1. If root is NULL

then return

2. if key is found then

a. If its left subtree is not null

Then predecessor will be the right most

child of left subtree or left child itself.

b. If its right subtree is not null

The successor will be the left most child

of right subtree or right child itself.

return

3. If key is smaller then root node

set the successor as root

search recursively into left subtree

else

set the predecessor as root

search recursively into right subtree

Following is the implementation of the above algorithm:

* C++
* Java
* Python
* C#

|  |
| --- |
| // C++ program to find predecessor and successor in a BST  #include <iostream>  using namespace std;    // BST Node  struct Node  {      int key;      struct Node \*left, \*right;  };    // This function finds predecessor and successor of key in BST.  // It sets pre and suc as predecessor and successor respectively  void findPreSuc(Node\* root, Node\*& pre, Node\*& suc, int key)  {      // Base case      if (root == NULL)  return ;        // If key is present at root      if (root->key == key)      {          // the maximum value in left subtree is predecessor          if (root->left != NULL)          {              Node\* tmp = root->left;              while (tmp->right)                  tmp = tmp->right;              pre = tmp ;          }            // the minimum value in right subtree is successor          if (root->right != NULL)          {              Node\* tmp = root->right ;              while (tmp->left)                  tmp = tmp->left ;              suc = tmp ;          }          return ;      }        // If key is smaller than root's key, go to left subtree      if (root->key > key)      {          suc = root ;          findPreSuc(root->left, pre, suc, key) ;      }      else // go to right subtree      {          pre = root ;          findPreSuc(root->right, pre, suc, key) ;      }  }    // A utility function to create a new BST node  Node \*newNode(int item)  {      Node \*temp =  new Node;      temp->key = item;      temp->left = temp->right = NULL;      return temp;  }    /\* A utility function to insert a new node with given key in BST \*/  Node\* insert(Node\* node, int key)  {      if (node == NULL) return newNode(key);      if (key < node->key)          node->left  = insert(node->left, key);      else          node->right = insert(node->right, key);      return node;  }    // Driver program to test above function  int main()  {      int key = 65;    //Key to be searched in BST       /\* Let us create following BST                50             /     \            30      70           /  \    /  \         20   40  60   80 \*/      Node \*root = NULL;      root = insert(root, 50);      insert(root, 30);      insert(root, 20);      insert(root, 40);      insert(root, 70);      insert(root, 60);      insert(root, 80);          Node\* pre = NULL, \*suc = NULL;        findPreSuc(root, pre, suc, key);      if (pre != NULL)        cout << "Predecessor is " << pre->key << endl;      else        cout << "No Predecessor";        if (suc != NULL)        cout << "Successor is " << suc->key;      else        cout << "No Successor";      return 0;  } |

**Output:**

Predecessor is 60

Successor is 70

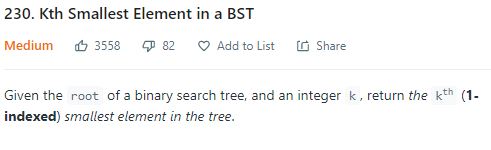
**Another Approach :**  
We can also find the inorder successor and inorder predecessor using inorder traversal . Check if the current node is smaller than the given key for predecessor and for successor, check if it is greater than the given key . If it is greater than the given key then, check if it is smaller than the already stored value in successor then, update it . At last, get the predecessor and successor stored in q(successor) and p(predecessor).

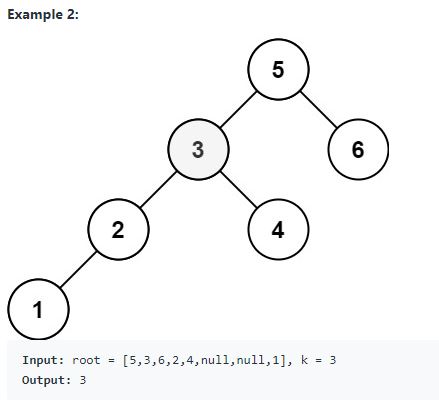
* C++
* Python3

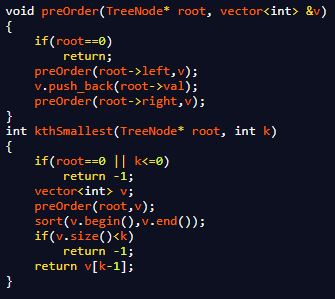
|  |
| --- |
| // CPP code for inorder succesor  // and predecessor of tree  #include<iostream>  #include<stdlib.h>    using namespace std;    struct Node  {      int data;      Node\* left,\*right;  };    // Function to return data  Node\* getnode(int info)  {      Node\* p = (Node\*)malloc(sizeof(Node));      p->data = info;      p->right = NULL;      p->left = NULL;      return p;  }    /\*  since inorder traversal results in  ascending order visit to node , we  can store the values of the largest  no which is smaller than a (predecessor)  and smallest no which is large than  a (succesor) using inorder traversal  \*/  void find\_p\_s(Node\* root,int a,                Node\*\* p, Node\*\* q)  {      // If root is null return      if(!root)          return ;        // traverse the left subtree      find\_p\_s(root->left, a, p, q);        // root data is greater than a      if(root&&root->data > a)      {            // q stores the node whose data is greater          // than a and is smaller than the previously          // stored data in \*q which is sucessor          if((!\*q) || (\*q) && (\*q)->data > root->data)                  \*q = root;      }        // if the root data is smaller than      // store it in p which is predecessor      else if(root && root->data < a)      {          \*p = root;      }        // traverse the right subtree      find\_p\_s(root->right, a, p, q);  }    // Driver code  int main()  {      Node\* root1 = getnode(50);      root1->left = getnode(20);      root1->right = getnode(60);      root1->left->left = getnode(10);      root1->left->right = getnode(30);      root1->right->left = getnode(55);      root1->right->right = getnode(70);      Node\* p = NULL, \*q = NULL;        find\_p\_s(root1, 55, &p, &q);        if(p)          cout << p->data;      if(q)          cout << " " << q->data;      return 0;  } |

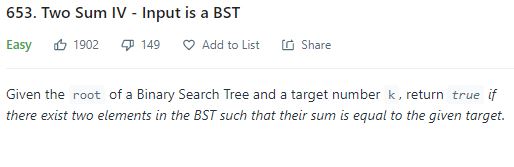
**Output :**

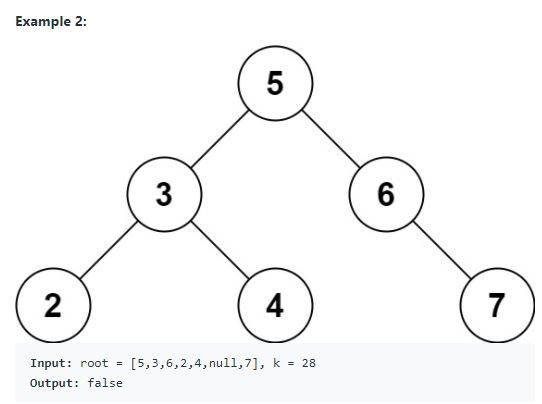
50 60

****

****

****

****

****

****

Largest BST in a Binary Tree | Set 2

* Difficulty Level : [Hard](https://www.geeksforgeeks.org/hard)
* Last Updated : 01 Mar, 2021

Given a Binary Tree, write a function that returns the size of the largest subtree which is also a Binary Search Tree (BST). If the complete Binary Tree is BST, then return the size of the whole tree.  
**Examples:** 

Input:

5

/ \

2 4

/ \

1 3

Output: 3

The following subtree is the

maximum size BST subtree

2

/ \

1 3

Input:

50

/ \

30 60

/ \ / \

5 20 45 70

/ \

65 80

Output: 5

The following subtree is the

maximum size BST subtree

60

/ \

45 70

/ \

65 80

[Recommended: Please solve it on “***PRACTICE***” first, before moving on to the solution.](https://practice.geeksforgeeks.org/problems/largest-bst/1)

We have discussed two methods in below post.   
[Find the largest BST subtree in a given Binary Tree | Set 1](https://www.geeksforgeeks.org/find-the-largest-subtree-in-a-tree-that-is-also-a-bst/)  
In this post, a different O(n) solution is discussed. This solution is simpler than the solutions discussed above and works in O(n) time.  
The idea is based on method 3 of [check if a binary tree is BST article](https://www.geeksforgeeks.org/a-program-to-check-if-a-binary-tree-is-bst-or-not/).   
A Tree is BST if following is true for every node x. 

1. The largest value in left subtree (of x) is smaller than value of x.
2. The smallest value in right subtree (of x) is greater than value of x.

We traverse tree in bottom up manner. For every traversed node, we return maximum and minimum values in subtree rooted with it. If any node follows above properties and size of 

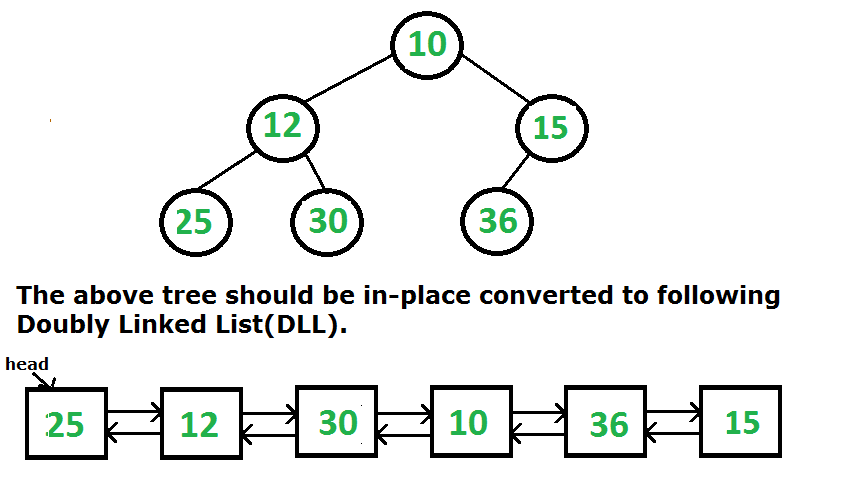
**C++**

|  |
| --- |
| // C++ program to find largest BST in a  // Binary Tree.  #include <bits/stdc++.h>  using namespace std;    /\* A binary tree node has data,  pointer to left child and a  pointer to right child \*/  struct Node  {      int data;      struct Node\* left;      struct Node\* right;  };    /\* Helper function that allocates a new  node with the given data and NULL left  and right pointers. \*/  struct Node\* newNode(int data)  {      struct Node\* node = new Node;      node->data = data;      node->left = node->right = NULL;        return(node);  }    // Information to be returned by every  // node in bottom up traversal.  struct Info  {      int sz; // Size of subtree      int max; // Min value in subtree      int min; // Max value in subtree      int ans; // Size of largest BST which      // is subtree of current node      bool isBST; // If subtree is BST  };    // Returns Information about subtree. The  // Information also includes size of largest  // subtree which is a BST.  Info largestBSTBT(Node\* root)  {      // Base cases : When tree is empty or it has      // one child.      if (root == NULL)          return {0, INT\_MIN, INT\_MAX, 0, true};      if (root->left == NULL && root->right == NULL)          return {1, root->data, root->data, 1, true};        // Recur for left subtree and right subtrees      Info l = largestBSTBT(root->left);      Info r = largestBSTBT(root->right);        // Create a return variable and initialize its      // size.      Info ret;      ret.sz = (1 + l.sz + r.sz);        // If whole tree rooted under current root is      // BST.      if (l.isBST && r.isBST && l.max < root->data &&              r.min > root->data)      {          ret.min = min(l.min, min(r.min, root->data));          ret.max = max(r.max, max(l.max, root->data));            // Update answer for tree rooted under          // current 'root'          ret.ans = ret.sz;          ret.isBST = true;            return ret;      }        // If whole tree is not BST, return maximum      // of left and right subtrees      ret.ans = max(l.ans, r.ans);      ret.isBST = false;        return ret;  }    /\* Driver program to test above functions\*/  int main()  {      /\* Let us construct the following Tree          60         /  \        65  70       /      50 \*/        struct Node \*root = newNode(60);      root->left = newNode(65);      root->right = newNode(70);      root->left->left = newNode(50);      printf(" Size of the largest BST is %d\n",             largestBSTBT(root).ans);      return 0;  } |

Convert a given Binary Tree to Doubly Linked List | Set 4

* Difficulty Level : [Hard](https://www.geeksforgeeks.org/hard)
* Last Updated : 28 Sep, 2020

Given a Binary Tree (BT), convert it to a Doubly Linked List(DLL) In-Place. The left and right pointers in nodes are to be used as previous and next pointers respectively in converted DLL. The order of nodes in DLL must be same as Inorder of the given Binary Tree. The first node of Inorder traversal (left most node in BT) must be head node of the DLL.

[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/TreeToList.png)

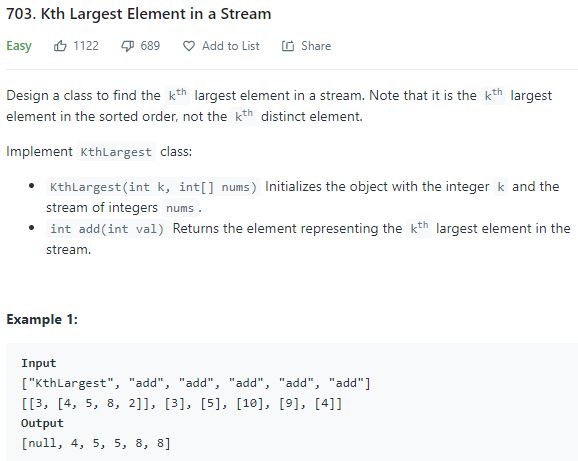
In the following implementation, we traverse the tree in inorder fashion. We add nodes at the beginning of current linked list and update head of the list using pointer to head pointer. Since we insert at the beginning, we need to process leaves in reverse order. For reverse order, we first traverse the right subtree before the left subtree. i.e. do a reverse inorder traversal.

## C++

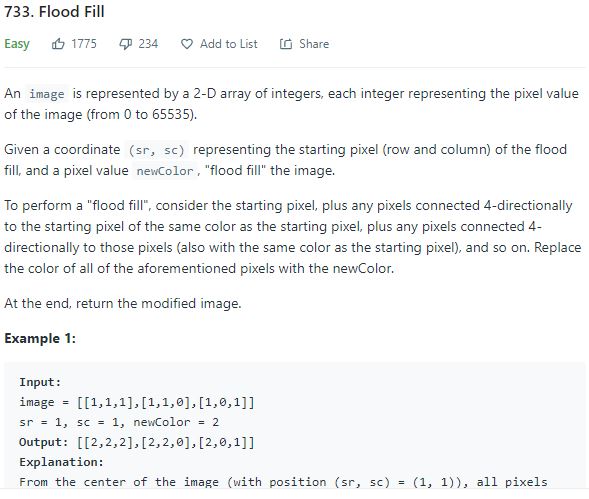
|  |
| --- |
| // C++ program to convert a given Binary Tree to Doubly Linked List  #include <bits/stdc++.h>    // Structure for tree and linked list  struct Node {      int data;      Node \*left, \*right;  };    // Utility function for allocating node for Binary  // Tree.  Node\* newNode(int data)  {      Node\* node = new Node;      node->data = data;      node->left = node->right = NULL;      return node;  }    // A simple recursive function to convert a given  // Binary tree to Doubly Linked List  // root    --> Root of Binary Tree  // head --> Pointer to head node of created doubly linked list  void BToDLL(Node\* root, Node\*& head)  {      // Base cases      if (root == NULL)          return;        // Recursively convert right subtree      BToDLL(root->right, head);        // insert root into DLL      root->right = head;        // Change left pointer of previous head      if (head != NULL)          head->left = root;        // Change head of Doubly linked list      head = root;        // Recursively convert left subtree      BToDLL(root->left, head);  }    // Utility function for printing double linked list.  void printList(Node\* head)  {      printf("Extracted Double Linked list is:\n");      while (head) {          printf("%d ", head->data);          head = head->right;      }  }    // Driver program to test above function  int main()  {      /\* Constructing below tree              5              / \              3     6          / \     \          1 4     8          / \     / \          0 2     7 9 \*/      Node\* root = newNode(5);      root->left = newNode(3);      root->right = newNode(6);      root->left->left = newNode(1);      root->left->right = newNode(4);      root->right->right = newNode(8);      root->left->left->left = newNode(0);      root->left->left->right = newNode(2);      root->right->right->left = newNode(7);      root->right->right->right = newNode(9);        Node\* head = NULL;      BToDLL(root, head);        printList(head);        return 0;  } |

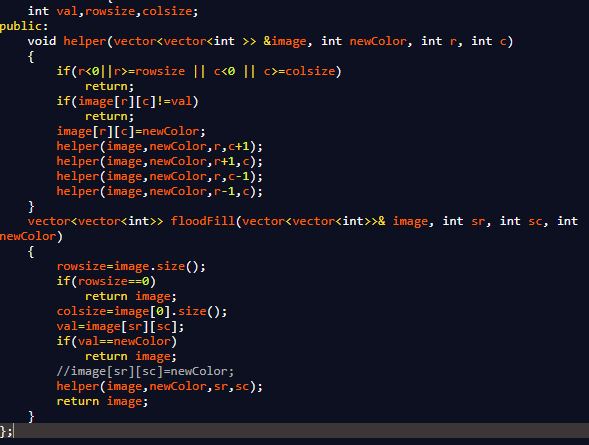
**Time Complexity:** O(n), as the solution does a single traversal of given Binary

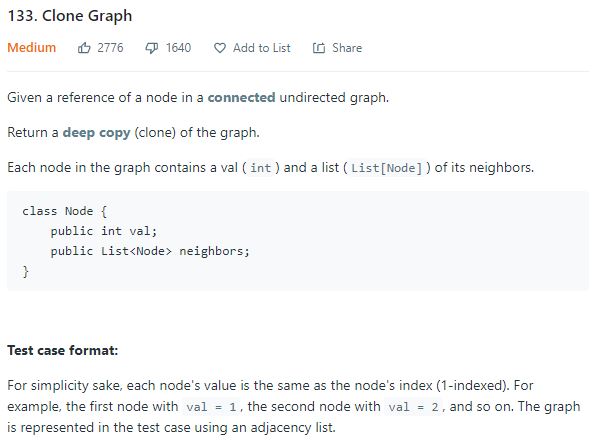
Tree

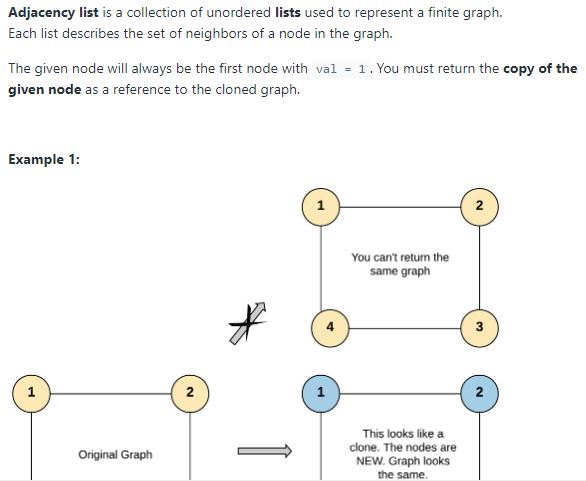
****

****

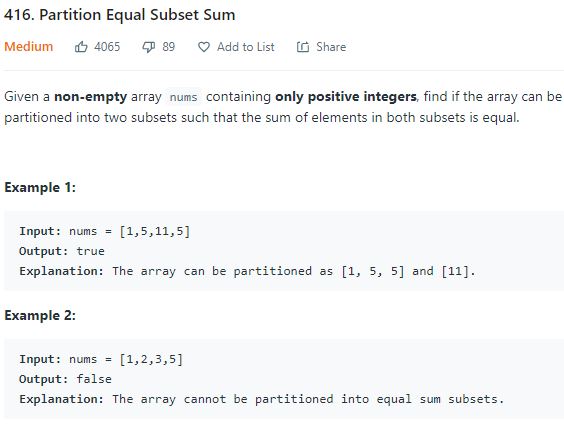
****

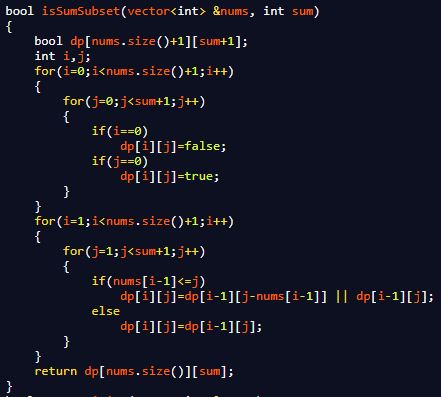
****

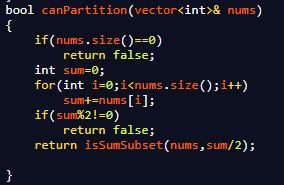
****

****

****

****

****

****