15B17CI371 – Data Structures Lab
ODD 2024
Week 7-LAB B
Practice Lab

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)**

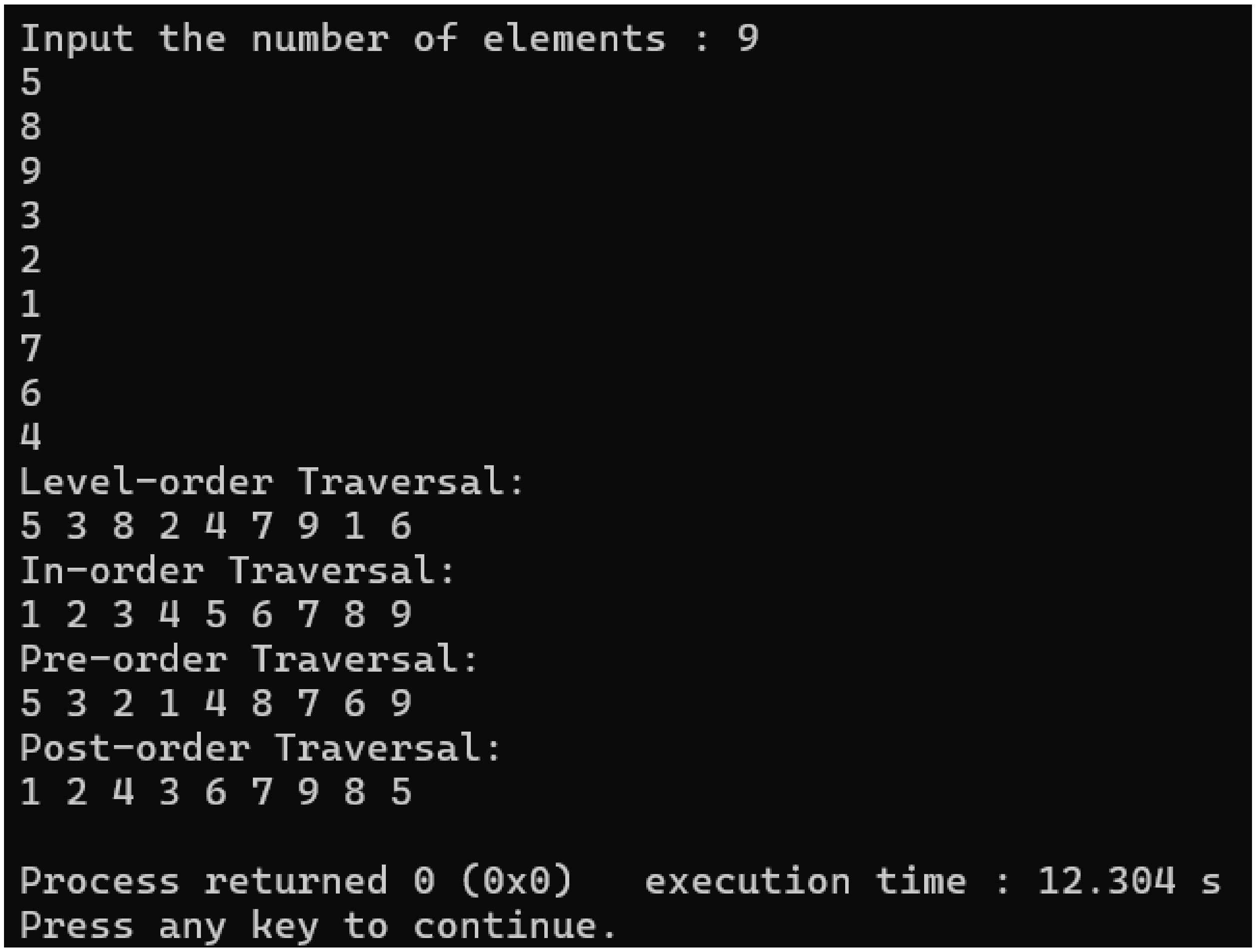
```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;
struct Node {
   int data;
   Node* left;
   Node* right;
   Node(int x) : data(x),left(nullptr),right(nullptr) {}
};
Node* insert(Node* root,int value)
{
   if(root==nullptr)
      return new Node(value);
   else if(value<root->data)
      root->left=insert(root->left,value);
   else
      root->right=insert(root->right,value);
   return root;
}
void inOrderTraversal(Node* root)
{
   if(root!=nullptr)
   {
      inOrderTraversal(root->left);
      cout<<root->data<<" ";
      inOrderTraversal(root->right);
   }
}
void preOrderTraversal(Node* root)
{
   if(root!=nullptr)
   {
      cout<<root->data<<" ";
      preOrderTraversal(root->left);
      preOrderTraversal(root->right);
```

```
    }
}
void postOrderTraversal(Node* root)
{
    if(root!=nullptr)
    {
        postOrderTraversal(root->left);
        postOrderTraversal(root->right);
        cout<<root->data<<" ";
    }
}
void levelOrderTraversal(Node* root)
{
    if(root==nullptr)
        return;
    queue<Node*> q;
    q.push(root);
    while(!q.empty())
    {
        Node* current=q.front();
        q.pop();
        cout<<current->data<<" ";
        if(current->left!=nullptr)
            q.push(current->left);
        if(current->right!=nullptr)
            q.push(current->right);
    }
}

int main() {
    vector<int> values;
    int n,val;
    cout<<"Input the number of elements : ";
    cin>>n;
    for(int i=0;i<n;i++)
    {
        cin>>val;
        values.push_back(val);
    }
    Node* root=nullptr;
    for(int value : values)
        root=insert(root,value);
    cout<<"Level-order Traversal:\n";
    levelOrderTraversal(root);
    cout<<"\n";
    cout<<"In-order Traversal:\n";
    inOrderTraversal(root);
    cout<<"\n";
```

```
   cout<<"Pre-order Traversal:\n";
   preOrderTraversal(root);
   cout<<"\n";
   cout<<"Post-order Traversal:\n";
   postOrderTraversal(root);
   cout<<"\n";
}
```

Output :



```
Input the number of elements : 9
5
8
9
3
2
1
7
6
4
Level-order Traversal:
5 3 8 2 4 7 9 1 6
In-order Traversal:
1 2 3 4 5 6 7 8 9
Pre-order Traversal:
5 3 2 1 4 8 7 6 9
Post-order Traversal:
1 2 4 3 6 7 9 8 5

Process returned 0 (0x0)   execution time : 12.304 s
Press any key to continue.
```

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.**

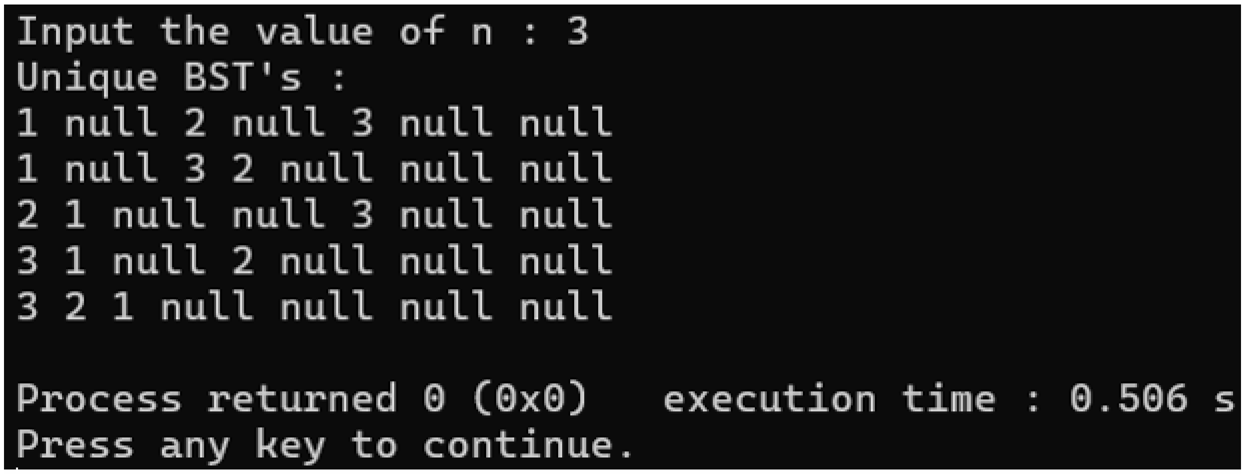
Input: n=3
Output: 5
Justification: [1,null,2,null],
[1,null,3,2],
[2,1,3],
[3,1,,null,null,2],
[3,2,null,1]

```
#include <iostream>
```

```
#include <vector>
using namespace std;
struct TreeNode
{
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x),left(NULL),right(NULL) {}
};
vector<TreeNode*> generateTrees(int start,int end)
{
    vector<TreeNode*> trees;
    if(start>end)
    {
        trees.push_back(nullptr);
        return trees;
    }
    for(int i=start; i<=end; i++)
    {
        vector<TreeNode*> leftTrees=generateTrees(start,i-1);
        vector<TreeNode*> rightTrees=generateTrees(i+1,end);
        for(TreeNode* left : leftTrees)
            for(TreeNode* right : rightTrees)
            {
                TreeNode* root=new TreeNode(i);
                root->left=left;
                root->right=right;
                trees.push_back(root);
            }
    }
    return trees;
}
vector<TreeNode*> generateTrees(int n)
{
    if(n==0) return {};
    return generateTrees(1,n);
}
void printPreorder(TreeNode* root)
{
    if(!root)
    {
        cout<<"null";
        return;
    }
    cout<<root->val<<" ";
    printPreorder(root->left);
    printPreorder(root->right);
}
```

```
int main()
{
   int n;
   cout<<"Input the value of n : ";
   cin>>n;
   vector<TreeNode*> result=generateTrees(n);
   for(TreeNode* tree:result)
   {
      printPreorder(tree);
      cout<<endl;
   }
}
```

Output :



```
Input the value of n : 3
Unique BST's :
1 null 2 null 3 null null
1 null 3 2 null null null
2 1 null null 3 null null
3 1 null 2 null null null
3 2 1 null null null null

Process returned 0 (0x0)   execution time : 0.506 s
Press any key to continue.
```

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.**

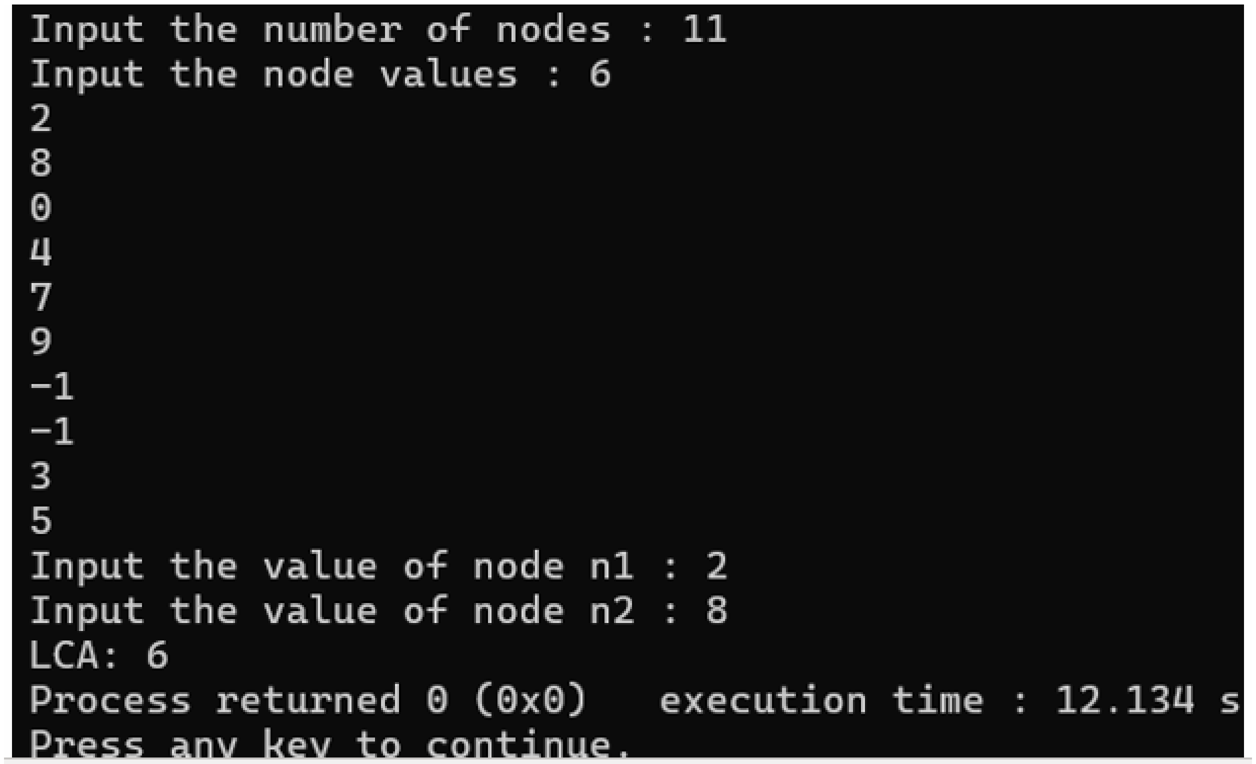
Assumption: Node may be a descendant of itself.

```
#include <iostream>
using namespace std;
struct TreeNode
{
   int val;
   TreeNode* left;
```

```
    TreeNode* right;
    TreeNode(int x) : val(x),left(NULL),right(NULL) {}
};
TreeNode* lowestCommonAncestor(TreeNode* root,TreeNode* p,TreeNode* q)
{
    if(!root||root==p||root==q)
        return root;
    TreeNode* left=lowestCommonAncestor(root->left,p,q);
    TreeNode* right=lowestCommonAncestor(root->right,p,q);
    if(left && right)
        return root;
    return left?left:right;
}
TreeNode* insertLevelOrder(int arr[],TreeNode* root,int i,int n)
{
    if(i<n)
    {
        TreeNode* temp=new TreeNode(arr[i]);
        root=temp;
        root->left=insertLevelOrder(arr,root->left,2*i+1,n);
        root->right=insertLevelOrder(arr,root->right,2*i+2,n);
    }
    return root;
}
TreeNode* findNode(TreeNode* root,int val)
{
    if(!root||root->val==val)
        return root;
    TreeNode* leftSearch=findNode(root->left,val);
    if(leftSearch)
        return leftSearch;
    return findNode(root->right,val);
}
int main()
{
    int n,n1,n2;
    cout<<"Input the number of nodes : ";
    cin>>n;
    int arr[n];
    cout<<"Input the node values : ";
    for(int i=0;i<n;i++)
        cin>>arr[i];
    cout<<"Input the value of node n1 : ";
    cin>>n1;
    cout<<"Input the value of node n2 : ";
    cin>>n2;
    TreeNode* root=insertLevelOrder(arr,root,0,n);
    TreeNode* p=findNode(root,n1);
```

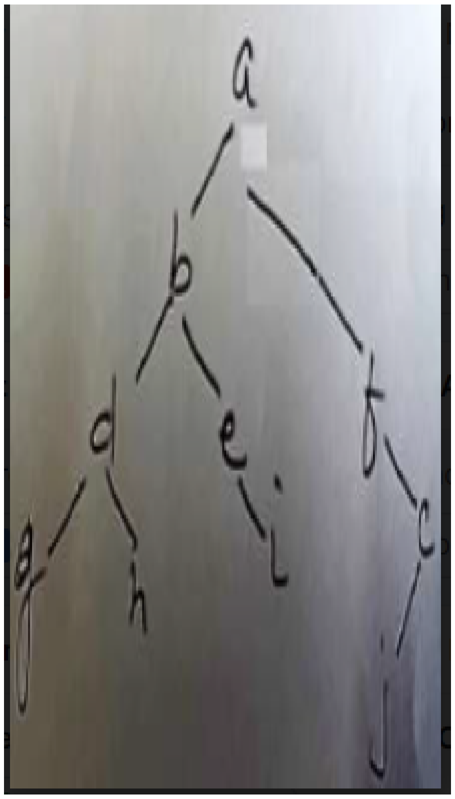
```
   TreeNode* q=findNode(root,n2);
   TreeNode* lca=lowestCommonAncestor(root,p,q);
   if(lca)
      cout<<"LCA: "<<lca->val;
   else
      cout<<"LCA not found";
}
```

Outptut :



```
Input the number of nodes : 11
Input the node values : 6
2
8
0
4
7
9
-1
-1
3
5
Input the value of node n1 : 2
Input the value of node n2 : 8
LCA: 6
Process returned 0 (0x0)   execution time : 12.134 s
Press any key to continue.
```

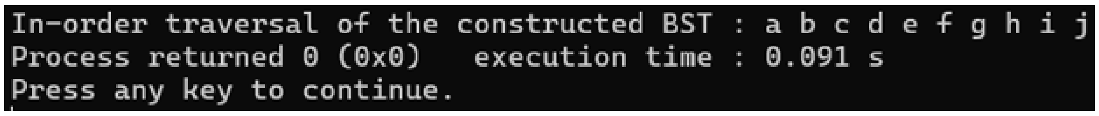
4. Given a list that depicts the in-order traversal of a binary tree,develop the binary search tree itself. Example: “gdhbeiafjc” is the in-order traversal for the BST:



```
#include <iostream>
#include <string>
using namespace std;
struct TreeNode
{
   char val;
   TreeNode* left;
   TreeNode* right;
   TreeNode(char x):val(x),left(NULL),right(NULL) {}
};
TreeNode* buildTreeFromInorder(string& inorder,int start,int end)
{
   if(start>end)
      return nullptr;
   int mid=(start+end)/2;
   TreeNode* root=new TreeNode(inorder[mid]);
   root->left=buildTreeFromInorder(inorder,start,mid-1);
   root->right=buildTreeFromInorder(inorder,mid+1,end);
   return root;
}
void printInorder(TreeNode* root)
{
   if(!root)
      return;
   printInorder(root->left);
```

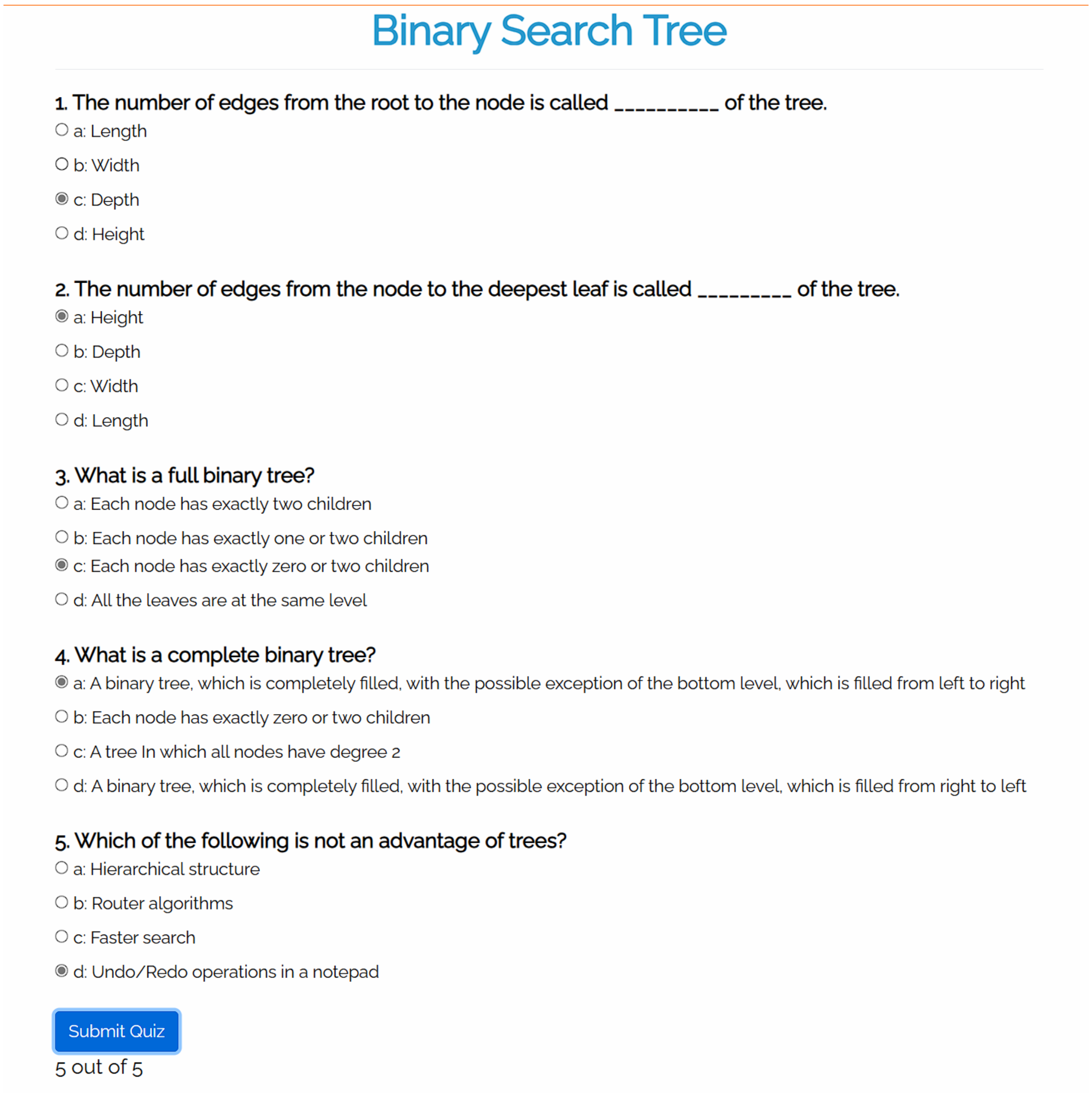
```
    cout<<root->val<<" ";
    printInorder(root->right);
}
int main()
{
    string inorder="abcdefghij";
    int n=inorder.size();
    TreeNode* root=buildTreeFromInorder(inorder,0,n-1);
    cout<<"In-order traversal of the constructed BST : ";
    printInorder(root);
}
```

Output :



```
In-order traversal of the constructed BST : a b c d e f g h i j
Process returned 0 (0x0)   execution time : 0.091 s
Press any key to continue.
```

Virtual Labs



Binary Search Tree

1. The number of edges from the root to the node is called __________ of the tree.

a: Length

b: Width

[✓ c: Depth]

d: Height

2. The number of edges from the node to the deepest leaf is called _________ of the tree.

[✓ a: Height]

b: Depth

c: Width

d: Length

3. What is a full binary tree?

a: Each node has exactly two children

b: Each node has exactly one or two children

[✓ c: Each node has exactly zero or two children]

d: All the leaves are at the same level

4. What is a complete binary tree?

[✓ a: A binary tree, which is completely filled, with the possible exception of the bottom level, which is filled from left to right]

b: Each node has exactly zero or two children

c: A tree In which all nodes have degree 2

d: A binary tree, which is completely filled, with the possible exception of the bottom level, which is filled from right to left

5. Which of the following is not an advantage of trees?

a: Hierarchical structure

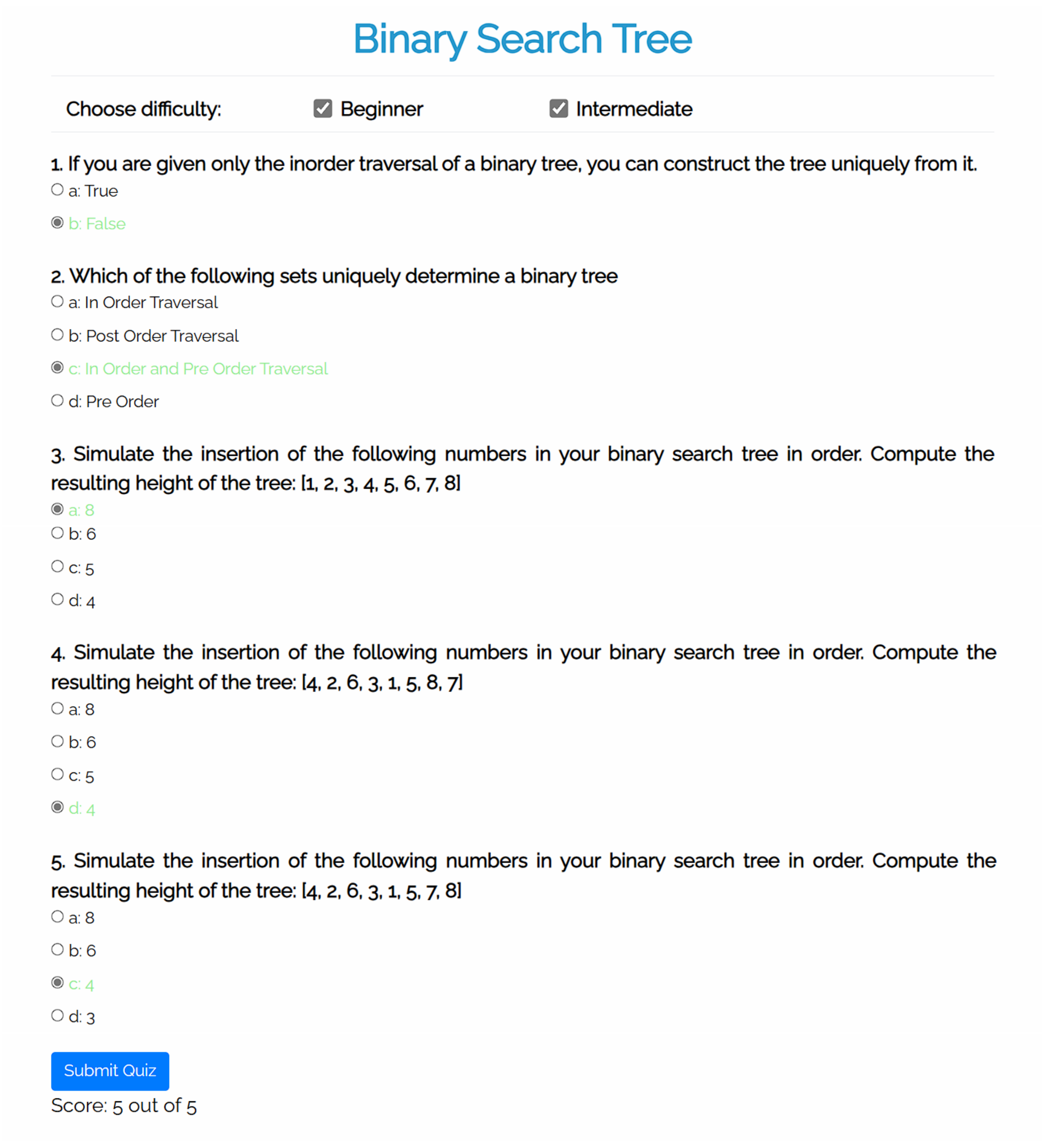
b: Router algorithms

c: Faster search

[✓ d: Undo/Redo operations in a notepad]

Submit Quiz

5 out of 5



Binary Search Tree

Choose difficulty: [✓ Beginner] [✓ Intermediate]

1. If you are given only the inorder traversal of a binary tree, you can construct the tree uniquely from it.

- a: True
- [✓ b: False]

2. Which of the following sets uniquely determine a binary tree

- a: In Order Traversal
- b: Post Order Traversal
- [✓ c: In Order and Pre Order Traversal]
- d: Pre Order

3. Simulate the insertion of the following numbers in your binary search tree in order. Compute the resulting height of the tree: [1, 2, 3, 4, 5, 6, 7, 8]

- [✓ a: 8]
- b: 6
- c: 5
- d: 4

4. Simulate the insertion of the following numbers in your binary search tree in order. Compute the resulting height of the tree: [4, 2, 6, 3, 1, 5, 8, 7]

- a: 8
- b: 6
- c: 5
- [✓ d: 4]

5. Simulate the insertion of the following numbers in your binary search tree in order. Compute the resulting height of the tree: [4, 2, 6, 3, 1, 5, 7, 8]

- a: 8
- b: 6
- [✓ c: 4]
- d: 3

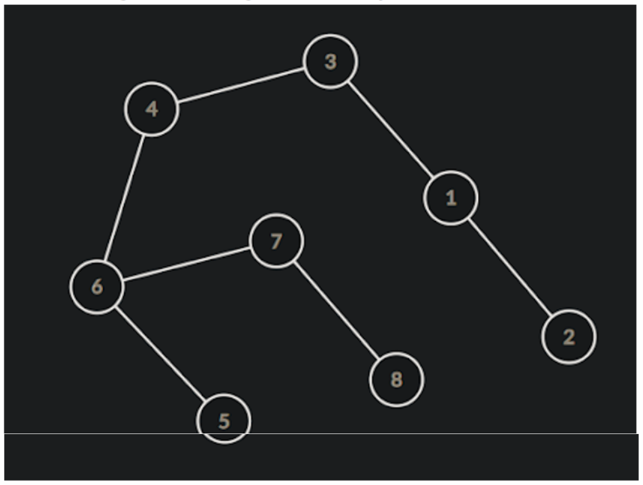
Submit Quiz

Score: 5 out of 5

Binary Search Tree

Choose difficulty: [✓ Intermediate] [✓ Advanced]

1. Following is the image of a binary search tree rooted at 4



(which child is left and which is right is arbitrary, but assigned so to maintain validity of the tree). Which of the following can be the nodes you visit when searching for the number 10.

○ a: 4 6 7 10

[✓ b: 4 6 7 8] Explanation

○ c: 4 3 1 2

○ d: 4 6 5 10

2. To find the smallest element in the subtree of a given node, we:

[✓ a: Go recursively into the left child of the current node, and stop when no left child exists.] Explanation

○ b: Go recursively into the left child of the current node, go to the right child otherwise, and stop when neither one exists.

○ c: Go recursively into the right child of the current node, and stop when no left child exists.

○ d: Go recursively into the right child of the current node, go to the right child otherwise, and stop when neither one exists.

Submit Quiz

Score: 2 out of 2