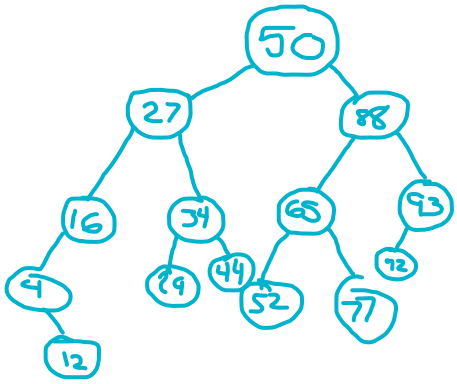
Naseem Auguste

Before starting the exercise, go through the lecture notes, examples, and binary search tree practice problems pdf file.

Computer Science I - Exercise
Binary Search Trees

1. Draw the binary search tree that results from inserting the following values into an initially empty binary search tree in the following order: 50, 27, 16, 88, 34, 65, 52, 77, 93, 4, 12, 29, 44, 92



2. What are the outputs of a pre-order, in-order and post-order traversal of the final binary search tree drawn in question 1?

In-order: 4 , 12 , 16 , 27 , 29 , 34 , 44 , 50 , 52 , 65 , 77 , 88 , 92 , 93

Pre-order: 50 , 27 , 16 , 4 , 12 , 34 , 29 , 44 , 88 , 65 , 52 , 77 , 93 , 92

Post-order: 12 , 4 , 16, 29 , 44 , 34 , 27 , 52 , 77 , 65 , 92 , 93 , 88 , 50

3. If a search was conducted for the value 37 in the final binary search tree from question #1, which nodes would get visited? (List them in the order they get visited.)

50 → 27 → 34 → 44 → 37

4. Write a function which returns the smallest value stored in a *non-empty* binary search tree. The prototype is below:

```
int minVal(struct treenode* root) {
```

int min = root → data;

if(root → left == NULL)

return root;

else

min = root → left → data;

minVal(root → left)

```
}
```

5. Write a function which returns the number of leaf nodes in a binary search tree. The prototype is below:

```
int numLeafNodes(struct treenode* root) {
```

int leaves.

if(root → left && root → right == NULL)
    return leaves++,

if(Root → left != NULL)
  numLeafNodes(root → left)
if(root → right != NULL)
  numLeafNodes(root → right)

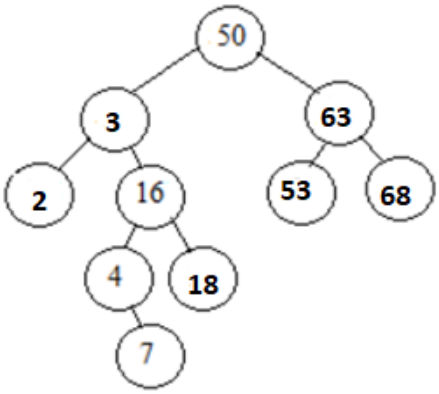
```
}
```

6. Pass the following tree's root and 16 to the following function and explain what does the following function do and returns?

```
struct treenode* q6(struct treenode* root, int x) {
  if (root == NULL)
    return NULL;

  if (root->data > x) {

    struct treenode* tmp = q6(root->left, x);
    if (tmp == NULL)
      return root;
    else
      return tmp;
  }
  else
    return q6(root->right, x);
}
```

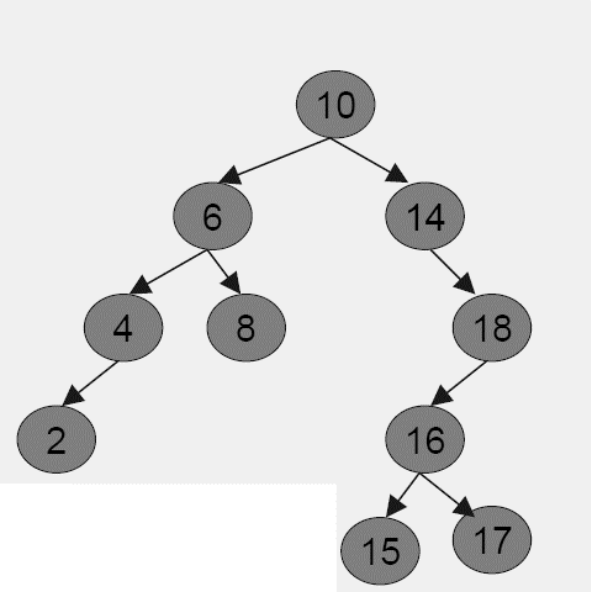


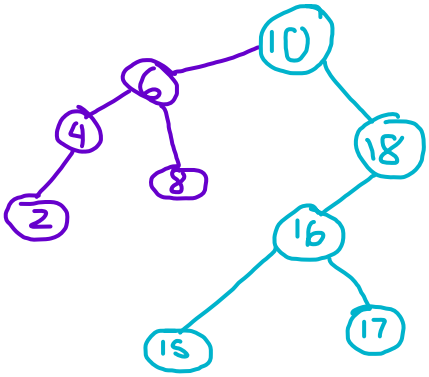
50 → 3 → 16 → 18

returns 18

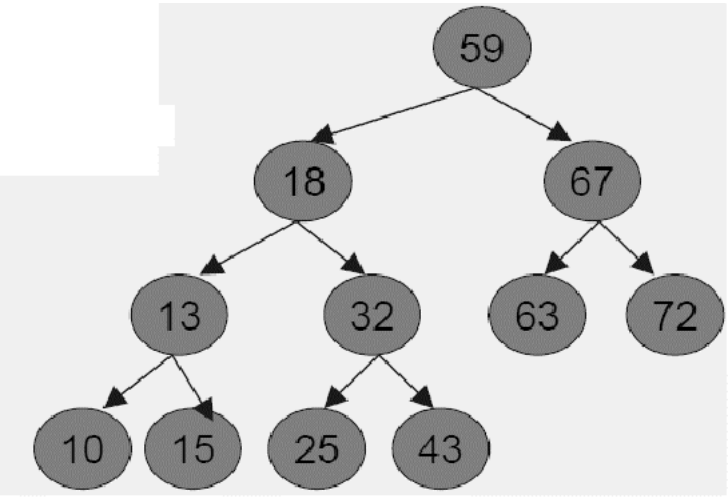
returns the next value that is greater than x

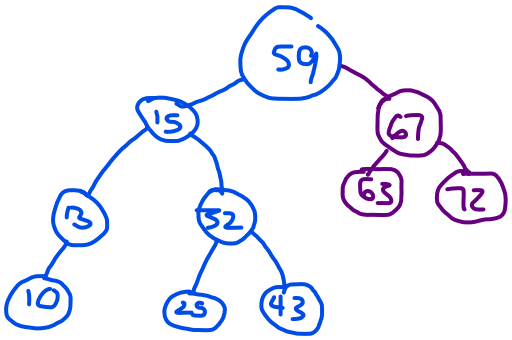
7.Consider the following BST. Draw the BST after deleting the node containing 14.





8. Consider the following BST. Draw the BST after deleting the node containing 18.





9. **a)** Indicate in few words, the purpose of the following function. The struct treenode is the same as the one defined in problem 4 on the previous page.

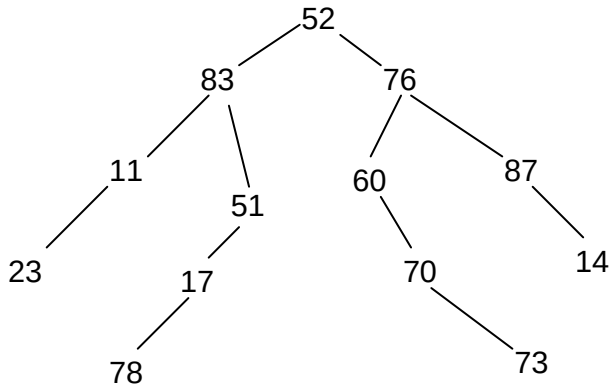
```
int f(struct treenode * p) {


    int val;
    if (p == NULL) return 0;
    val = p->data;
    if (val%2 == 0)
        return val + f(p->left) + f(p->right);
    else
        return f(p->left) + f(p->right);

}
```

sum of all values in BST

b) What does the function return, given the following tree?



Answer: 695

In each of the following questions, you will be writing functions that utilize binary trees created with the following struct:

```
struct treenode {
     int data;
     struct treenode* left;
     struct treenode* right;
};
```

10) Write a function that operates on a binary tree of integers. Your function should sum up the all of the odd numbers in the tree EXCEPT for the numbers in leaf nodes, which should instead be ignored. Make use of the function prototype shown below.

```
int sum_nonleaf_odd(struct treenode* p);
```

```
int odd = p->data;
if(p->left && p->right == NULL)
  return odd;
if (p->data % 2 == 1)
  return sumnonleafodd(p->left)+(p->right)+p->data
else
  return 0+ sumnonleafodd(p->left)+(p->right)
```

11) Write a function that returns the number of leaf nodes in the binary tree pointed to by p. Utilize the function prototype provided below.

```
int numLeafNodes(struct treenode* p);
```

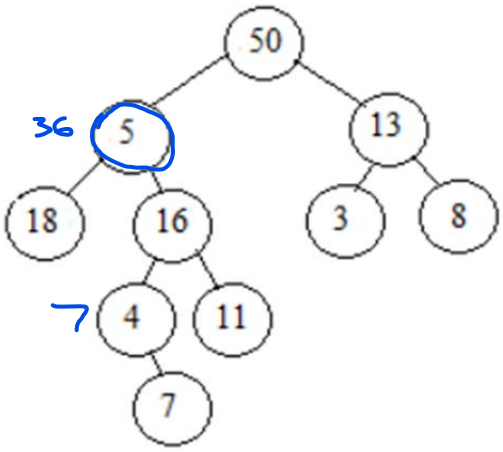
```
if(p->left && p->right = NULL)
  return p->data++;
else
  numLeafNodes(p->left)+nLN(p->right)
```

12) Write a recursive function to compute the height of a tree, defined as the length of the longest path from the root to a leaf node. For the purposes of this problem a tree with only one node has height 1 and an empty tree has height 0.

```
int height(struct treenode* root);
```

```
if(root==NULL) return 0;
else
  return height(root->left+1), height(root->right+1),
```

13. What does the function call solve(root)print if root is pointing to the node storing 50 in the tree shown below? The necessary struct and function are provided below as well. Please fill in the blanks shown below. **Draw recursion tree to find out the output.**




```
typedef struct bstNode {
     int data;
     struct bstNode *left;
     struct bstNode *right;
     } bstNode;
int solve(bstNode* root) {
     if (root == NULL) return 0;
     int res = root->data;
     int left = solve(root→left);
     int right = solve(root->right);
     if (left+right > res)
          res = left+right;
     printf("%d,", res);
     return res;
}
```

18 , 7 , 7 , 11 , 18 , 36 , 3 , 8 , 13 , 50