Name: Mohammed Allam

UCID: 30106564

CPSC 331

Assignment 3

Part 2

Pseudocode

```
Integer BST_find_median(node root){

1.  node current_node = root;

2.  Int median_node = floor(root.size / 2) + 1;

3.  Int remaining_steps = median_node;

4.  while(remaining_steps > 0){

5.  if(current_node.left.size – remaining_steps == -1){

6.      return current_node.value;

7.  else if(current_node.left.size – remaining_steps > -1){

8.       current_node = current_node.left;
        }
        else{
9.        remaining_steps = remaining_steps – current_node.left.size – 1;

10.    current_node = current_node.right;
                }
        }
}
```

Proof

Claim: if the binary search tree invariant is satisfied, and the BST_find_median() function is executed with parameters 'node', and the BST is not null and has an odd number of nodes, and all the nodes are distinct, then the execution of the algorithm eventually halts, and when the execution halts, the median valued node is returned.

I will be using mathematical induction on the depth of the BST.

Basis: assume that the BST has a single node, this means that this single node is the root of the BST, and it has no children, which means the depth of this BST is 0.
Line 2 in the code gets the median value, in this case -> floor(1/2) + 1 which equals 1. Moving on to line 4, the while loop condition is met because the remaining_steps is greater than 0. Given that our BST has a single node, and that node has no children. The if condition on line 5 is met because the left child is equal to 0, and 0 – remaining_steps == -1.
The body of this if statements (line 6) returns the value of the current node (the root) which equals 1. And 1 is indeed the median valued node. This completes our basis step.

Inductive step:

Inductive hypothesis: let K be an arbitrary integer such that K >= -1. The inductive hypothesis concerns the execution of the algorithm when the depth of the BST is H, for any integer H such that -1 <= H <= K.

Inductive claim: To prove we must show that the conditions are also satisfied if the BST has depth of K+1.
Assuming that our BST has depth of K+1, our while loop at line 4 will always execute causing the body to also execute.

Now the binary search tree invariant #3 states that if the binary search tree is not empty then the left and right subtrees are also binary search trees, and #2 states that if this binary search tree is not empty then every value stored in the left subtree of the root is less than the value stored at the root, and every value stored in the right subtree is greater than the value stored at the root.

Note: “You can imagine the median as the middle index inside an array of odd number of elements. The median being in the middle and the count of the left and the right nodes are equal.”

Since the depth is K+1, we should have more than a single node with respect to the number of nodes is odd. And since the left and right subtrees are always smaller than the parent, and the size of each node is given:

- We first check the left subtree and compare it with remaining_steps (initially equals to median), if the difference between the left subtree size and the remaining_steps is equal to -1, that means we are pointing at the median node and we should return that node because the left nodes and the right nodes are equal and the node that we are pointing at is the middle node between the two. This can be seen at line 5 and 6.

- If the difference between the left subtree size and the remaining_steps variable is greater than -1, then we should keep going to the left subtree, and update the current node pointer. This can be seen at line 7 and 8.

- If the difference between the left subtree size and the remaining_steps is less than -1, that means we should go to the right subtree and update the remaining_steps variable. We update it by subtracting remaining_steps, the left subtree size, and 1. The 1 is for the parent node, and the left subtree is the left side of the median. And update the

current pointer node by pointing to the right subtree. This can be seen at line 9 and 10.

The computation ends once the median valued node is found and returned (line 6) as needed to complete the inductive step and complete the proof.

Time Analysis

Assuming that the size of each node is given, so I will not consider the running time of the .size() function. Which should be O(n).

Line 1 to 3 have a constant running time of 3 * O(1) -> O(1)

For line 4 and onwards, the running time of the loop depends on the number of iterations and the statements inside the body of the loop. The operations performed inside the while loop are ran at constant time O(1). So, the number of iterations of the loop are going to determine the running time. However, the upper bound for the while loop is closely related to the value of the median. The logic of the code is implemented in a way that traverses the tree and only visits the nodes that are absolutely necessary to visit which means it is taking a direct/shortest path to the median node, and it only branches to the correct side (left or right). Since this is the case, the worst case would be if the median valued node is located at the bottom of the tree (depth) which in this case the algorithm must traverse to the depth, which makes O(depth) the worst-case scenario.
Hence, the running time of the algorithm is O(d) (d = depth).