Jared Yu

Homework 6

Algorithms for Data Science

1. References: https://en.wikipedia.org/wiki/Tree_traversal#Post-order_(LRN)
https://www.youtube.com/watch?v=4zVdfkpcT6U
https://www.youtube.com/watch?v=Vzqaz4MDGvc
https://stackoverflow.com/questions/4547012/complexities-of-binary-tree-traversals
https://www.youtube.com/watch?v=T68vN1FNY4o
In a binary tree each vertex has a maximum of two child nodes and a minimum of zero child nodes. Using post-order traversal, the algorithm will search along the left subtree recursively and then the right subtree recursively.

   Searching along the left subtree, the algorithm will first check if a left child exists in each node. Once it sees that there is no left child, it will check for a right child of the node. If no right child exists, the node is visited one time total and the algorithm moves on to another node. If a right child exists, then it will continue to search along that branch. If the algorithm is searching back up after having scanned the left and right children, the algorithm will be in the process of exiting a recursion for a node and finally visit it. Therefore, whether a vertex has zero children, one child, or two children, the algorithm will visit the node exactly once. In a binary tree with n vertices, then each node will be visited exactly one time, where the time spent at each node is constant amount of time.

   The time for this function would be $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 1$. The reason is that each node must call the recursion for a left and right subtree, hence $2 \cdot T\left(\frac{n}{2}\right)$, and then 1 because the algorithm must visit the "root" which is the node itself taking constant time.

   This formula matches the format to utilize the Master theorem of $T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^d)$, where $a = 2$, $b = 2$, and $d = 0$ (since the constant C is $\Theta(1)$). Then, it can be shown that $\log_b a = \log_2 2 = 1 > d = 0$, and therefore $T(n) = \Theta\left(n^{\log_b a}\right) = \Theta\left(n^{\log_2 2}\right) = \Theta(n^1) = \Theta(n)$.

2. Reference: https://www.geeksforgeeks.org/print-bst-keys-in-the-given-range/
https://www.geeksforgeeks.org/find-the-minimum-element-in-a-binary-search-tree/
https://stackoverflow.com/questions/29157030/time-complexity-for-finding-the-minimum-value-of-a-binary-tree

CONTAINS(a, b, root):

```
// Initialize the root
current_node = root.get()

while (current_node) {
  // Check if node in range
  if (current_node->key <= b and current_node->key >= a)
    return true
  // Otherwise move down AVL tree
  if (current_node->key < a)
    current_node = current_node->left.get()
  else
    current_node = current_node->right.get()
}

// Return not found
return false
```

The CONTAINS() function will check if the AVL tree contains a value between a and b. If the AVL tree contains such a value within the range, it will return true and false otherwise. The function works by first initializing the variable `current_node` to the root node of the AVL tree with `root.get()`. Then, it will use a while-loop to search through the AVL tree. Since the AVL tree is a balanced binary search tree with a height that is balanced through rotations, this is the most efficient way to check if either a value is contained in the AVL tree or if a value within a range is contained within the tree.

In the while-loop, the loop first check if the current node's key is less than or equal to b or greater than or equal to a. If this is the case, then it returns true, indicating the biologists have found structures in the tree with specific weight between a and b inclusive. If the current node's key does not fall in the range, it will either search left if the key is less than a, or search right otherwise. The process repeats until a node's key is within the range, or the AVL tree reaches a leaf node at which point the while-loop will exit and return false.