Colby Wirth

COS 285

Assignment 5

Dr. Behrooz Mansouri

25 October 2024

Analysis for Program 5 sub-tasks

File Input and Insert Functions

The input function, MyDataReader.readFileToBST, has two subroutines: Parsing the tsv file with a while loop, and building the tree. The first subroutine's Time Complexity is $O(6n)$ as each line has 6 attributes which are parsed once by the MyDataReader.lineToReport function. This simplifies to $O(n)$. After parsing a line, a Node is instantiated and added to the Binary Search Tree with the BinarySeachTree.insert function. The insert function has an average/best case time complexity of $O(log(n))$. This is because a best case and average case BST is perfectly balanced, therefore the number of elements to be searched is halved with each layer of the tree. The worst case for this insert function is $O(n)$. This occurs if the BST resembles a chain, and every element has to be parsed for insertion.

Search Function

The Search function, BinarySeachTree.search, has an average/best case time complexity of $O(log(n))$. This function operates with a modified in-order traversal algorithm, where the left node is always checked before traversing its sub-tree. The result is that the number of Nodes to parsed is halved with each iteration, resulting in $O(log(n))$ time complexity. The worst case time complexity is $O(n)$ if a tree resembles a chain, and every element has to be parsed when searching.

Validate BST Function

The validate function, BinarySearchTree.isValidBST, will always execute at $O(n)$ time complexity as it uses a pre-order traversal to check every node in the tree.

Sorted ArrayList

The sorted arraylist function, BinarySearchTree.toSortedArrayList, uses an in-order traversal algorithm to visit every element of the tree for all cases with a valid BST. This executes in $O(n)$ as each element is parsed one time.

Clone Function

The cloning function, BinarySearchTree.clone, uses a pre-order traversal algorithm to visit every element of the tree for all cases with a valid BST. This executes in $O(n)$ as each element is parsed one time, and added to the cloned tree.

Finding Popular Artists Function

The popular artist search function, BinarySearchTree.popularArtist checks all right child Nodes from the root, and compares them for equality with the Song.compareTo method. This is done with only the rightmost nodes in the Tree. For a balanced BST, the execution will occur in $O(log(n))$ time complexity, as the algorithm will search h Nodes, where h is the height of the tree and $h = \lfloor log(n) \rfloor$. The worst case is occurs when the BST is a chain of increasing nodes, as the program will execute in $O(n)$.

Filtering

The filtering function, BinarySearchTree.filterByView, will execute in $O(log(n)$ time complexity with best/average cases as it uses a modified post-order traversal to remove half of the tree with each method call. In worse cases, the function will execute in $O(n)$ if the tree resembles a chain.