Wali Khan

Professor Sun

CS 253

14 August 2020

HW2

Question 1:

I believe my algorithm runs in O(n) time. My algorithm uses two while loops. I keep three ArrayLists; one of the elements in pre-order in tree1 ("**cp1**"), the pre-order of Positions in tree1 ("**pos1**"), and the pre-order of elements of tree2 ("**cp2**"). The outer while loop runs until the two pre-order ArrayLists of elements of the two trees the same. In the inner while loop, the loop terminates when the element at a specific int **i** is the same for the two pre-order lists of the two trees. Inside the inner while loop we use an int variable to find the index of the Position in **pos1** responsible for the two elements at int **i** for not being the same. Once this Position is located, we rotate it in Tree **copy1**. Then both **pos1** and **cp1** are updated. Once the two elements of **cp1** and **cp2** are the same at a given index int **i** is incremented. Until the two pre-order lists of the two trees are not the same, the loop keeps going, and the lists have at most n elements so the algorithm runs in O(n) time.

Question 2:

Usually, inserting elements in to the RandomizedTreeSet, is slightly faster than inserting elements in to the SelfBalanacingTreeSet, in my case an AVLTree. However, in some instances inserting elements in to the SelfBalancingTreeSet is faster, but in the majority of cases the RandomizedTreeSet was faster. This is because the RandomizedTreeSet does not perform any extra height balancing update methods. This is why sometimes the SelfBalancingTree is faster

***the variables which are used in the TreeRotator.java file are bolded**

because sometimes the heights can become too long in the RandomizedTreeSet causing it to take longer to find a viable node to insert an element.

Between extracting elements using the removeMin() method from the PQ interface and extracting elements using an inOrder traversal, the inOrder traversal was faster. This is because every time removeMin() is called it searches for the smallest element in the tree usually the left most Position. In contrast to the inOrder method, which uses recursion, adds a Position to the list once a Position is visited and declared not null. It traverses both left and right subtrees recursively. Due to the recursive nature of the inOrder traversal method, a base case is reached faster thus terminating the recursion extracting elements faster than the PQSort method which terminates once the tree is empty and all Positions have been visited and removed from the tree using the removeMin() method.

I wrote a method test which takes the time it takes to insert elements from an ArrayList to a given type of TreeSet. It records the long of the start time and returns the long of the time elapsed in milliseconds. The main method tests both how long (in milliseconds) it takes to insert elements in to a SelfBalancingTree, as well as RandomizedTreeSet. The main method also tests how long the priorityQueueSort() and inOrderTraversal() take to execute.

***the variables which are used in the TreeRotator.java file are bolded**