Raed Abuzaid

CSCI 335

Project 2

Time Results

	Input 1	Input 2	Input 3
Vector	0.00252194 s	0.00887956 s	0.0754643
List	0.0640515 s	1.15027 s	23.272 s
Heap	0.00382101 s	0.00984581	0.0438931
AVL tree	0.0375182	0.152114	0.665504

Time Complexities

Vector Median:

- **Lower_bound** to perform a binary search to locate the correct position to insert to the correct position: **O(log n)**
- We end up with **O(n* log n)**, this is because for every instruction we had to either insert or erase an element, which overshadows the binary search

List Median:

- Finding the **position** to insert, In the worst case: **O(n)**

- To **Insert** and **delete** in the worst case we have to traverse the whole list: **O(n)**
- Given that insertion and deletions take **O(n)** and we have **n** instructions, we get **O(n^2)**

Heap Median:

- Both **insertion** and **removal** operations in a heap have a complexity of **O(log n)**
- Since for every instruction we either insert or remove we end up with **O(n* log n)**

Tree Median:

- Insertion and deletion operations in an AVL tree have a time complexity of **O(log n)**
- SInce we perform either insertion or deletion for every iteration in the loop we get a complexity of: **O(n* log n)**

For the most part the complexity for each of the iterations of the ADT lined up with the expected complexity. The vector has a time complexity of **O(n* log n)** just like the heap and the tree. We see that it grows slowly much like the heap and the tree. For the list we see that the time grows rapidly as we grow at **O(n^2).** If you multiply the number of instructions by themselves and then by the time the numbers match up. (0.0640515) * (4 * 4) = 1.0246 which is close to the 1.15 that we received. And 1.15 * (4 * 4) is about 18 seconds. LIkewise for heap and the tree they grow quite slowly compared to the list, although since there were not as many costly operations using the heap it is faster despite growing at a similar pace to the tree as they are both **O(n* log n).**