Raitah Jinnat
Prof. Tojeira
CSCI 335
Project 2 Report

What time (μs) did it take on each input (number of instructions) for each implementation?

Implementation	Vector	Linked List	Heap	AVL Tree
Input				
4000	13,474.5	58,955.5	4,854.58	4,947.42
16000	96,431.4	526,826	7,074.62	8,653.92
64000	$1.18694 * 10^6$	$8.77131 * 10^6$	31,281.6	38,506.8

Complexity analysis of each method (O-notation)

vectorMedian with an overall complexity of $O(n^2)$ performs a binary search for each positive integer-represented instruction to insert it into the vector of results in O(log(n)) time. For an instruction corresponding to removing the median from the vector of results, the sequential operations of finding and removing the median from the vector of results have a worst case time complexity of O(n).

listMedian has an overall complexity of $O(n^2)$ since it goes through n instructions and inserts them in the list of results, iterating through that list to maintain its sorting and possibly remove a median with worst case time O(n), in a similar fashion as the vector implementation.

For heapMedian, we perform n times for the amount of instructions passed through the for loop in O(n) time. For each of these instructions, operations such as push and pop on the small and large heaps of n items are performed in O(log(n)) time, while the method rebalances the heaps as needed. heapMedian therefore has an overall complexity of O(nlog(n)).

treeMedian also has an overall time complexity of O(nlog(n)). For each instruction passed into the for loop taking O(n) time, the AVL operations performed upon it to insert the instruction and then rebalance the binary trees accordingly if needed take O(log(n)) time.

How do my results line up with O-complexity?

My results of heapMedian and treeMedian line up with their overall time complexity as their times of implementation do not significantly differ by magnitude and were the fastest times out of the implementations conducted during this project. However, the list implementation resulted in the slowest time by several orders of magnitude, despite its algorithmic time complexity being the same as the vector implementation. The vector implementation stores elements in a contiguous memory block while linked lists only use nodes connected to each other by pointers. Non-contiguous memory allocation for a linked list therefore enables poorer caching slowing that specific implementation down for the same method.