COMP232 – Data Structures & Problem Solving
Fall 2023

Homework #6 – Sorting
[50 points]

1. Implement the `insertionSort` method in the `CS232SortableLinkedList` class in the `hw06` code. This method should perform an insertion sort on the linked list and must run in $O(n^2)$ time. Note: You do not need to change any of the links, swap the elements in the nodes instead. The `No1Tests` class contains tests that you can use to check your implementation of this functionality.

2. Consider each of the following computations on an initially unordered list of integer values:

   a. Find the minimum value.
   b. Find the median (i.e., the middle value).
   c. Find the 10 largest values.

For each computation above:

   i. Briefly describe, in a few sentences or pseudo code, an efficient algorithm to perform the computation.
   ii. Give and briefly justify an asymptotic upper bound on your algorithm's worst case running time.

3. In our implementation of Insertion Sort we worked backward from the i-1st position while swapping to find the location at which to insert the ith value. This is essentially a linear search. However, because we know that the first i-1 values are already sorted we could have used a binary search to find the proper location at which to insert the ith value. Is this a useful idea? Why or why not?

I don't think that a binary search would be necessarily more helpful in this case. I actually think the runtime outcome would be pretty much the same because we would still have to traverse the list for our comparisons and then swap. So it would not improve any run time and would only really help us after the array is sorted because that's how binary search is effective. It is meant to cut search time in half based on how the array is sorted. So if we try this technique we would still need to sort the array and by the time we would use binary search to find the location, the elements should already be sorted into the correct location.

4. Complete the `heapSort` method in the `HeapSort` class so that given an array of integers they are sorted into <u>descending</u> order in O(n lg n) time using `HeapSort`. **You can run the `main` method to see if your sort works.**

5. In an application where we need to sort lists that we know will already be nearly sorted indicate which sort would you expect to run faster and briefly justify your answer:

a. An unoptimized merge sort or insertion sort?

I would say that insertion sort would be faster because if the list is mostly or completely sorted the runtime would have a O(n). An unoptimized merge sort may not account for a pre-sort list and will still divide and conquer, making the run time O(nlgn), which would be less efficient than a linear growth rate. Even if the runtime of insertion sort is closer to O(n^2) because of a couple swaps, it would still require less memory because it's sorting a pre-existing array instead of creating more.

b. An unoptimized merge sort or a heap sort?

Both of these functions still run all the loops, regardless of whether the list is pre-sorted or not so both run times would be O(nlgn). However, merge sort seems to take more memory because it breaks the initial array into two separate arrays. Also, the recursive nature of merge sort also could also cause it to be less efficient than heap sort, based on my implementation.