**COMP232 – Data Structures & Problem Solving**

**Fall 2023**

**Nam Nguyen**

**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(n2) 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.

**Answer:**

Before performing each of the computations above on the unordered list, I will merge-sort the list in ascending order for easier computations.

1. To find the minimum value of the sorted list, we will basically grab the first index of the list, which results in the runtime of O(1).
2. To find the median value of the sorted list, we can first determine whether the size of the list is even or odd. If odd, we will grab the (size/2)th index. If even, we will retrieve two integers at the following indices: (size/2) and (size/2 – 1). Then, we can take the average of the integers as the median. The runtimes of both cases O(1).
3. To find the 10 largest values, we can simply return the last 10 indices of the list. The runtime will also be O(1) as with the other computations as we are only grabbing indices.

The runtime of my algorithm will be the runtime of the merge sort algorithm plus the runtime of the corresponding computation that we want to perform, which will eventually turn out to be the O(nlgn)

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?

**Answer:**

Personally, this is a good and useful idea to implement. As we are assuming the array to the left of the ith value is already sorted, we know we can conduct a binary search on it. In the worst case, instead of going through the whole left array to find the appropriate spot for the ith value, we now only have to perform half the number of comparisons that we must initially do.

The runtime of a binary search is O(logn), which makes the ultimate runtime of the algorithm O(nlogn). Thus, the idea of implementing binary search will help the improve the overall runtime of the Insertion Sort.

4. Complete the heapSort method in the HeapSort class so that given an array of integers they are sorted into descending 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:

1. An unoptimized merge sort or insertion sort?

* I expect the insertion sort to run faster. With the nearly sorted list, the merge sort will ultimately break down every element of the list and merge them back together like it always does. Thus, the runtime will still be O(nlogn). However, with insertion sort, the number of comparisons will be cut significantly and can potentially be as fast as O(n) due to having to compare only the last few unsorted elements with the sorted section of the array.

1. An unoptimized merge sort or a heap sort?

* In this case, like the merge sort algorithm, the heap sort algorithm will run normally, meaning it will heapify the list, swap elements, and trickle nodes down as appropriate. Thus, the two algorithms will basically have the same runtime of O(nlogn). However, due to heap sorting being in-place sorting, it is reasonable to prefer heap sorting.