**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(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.

1. Creating a for loop that loops through all elements. If this number is less than the least number recorded, then make this the new least number, then return at the end whatever the number was. This runs at worst case O(n) if it runs through the entire loop and best case at Ω(n) since the loop still needs to run through the entire loop to identify a minimum value.
2. You can use merge sort to order the list, then if the list is odd then choose the middle number, and if it is even choose the middle two numbers divided by 2. Merge sort runs at O(nlgn), and the if statement that determines the size of the list is O(1) to select the element, so at the end we get O(nlgn).

c) Loop through each element. If the list already has 10 elements, then find the smallest one using the minimum, and if it is smaller than the next element, then remove that element, and add the next element. This runs at worst case O(n) because you have to loop through the entire element, and the if statements do not scale up to make the algorithm any less efficient than finding the minimum.

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?

This is not really that useful despite the fact that a binary search is slightly more efficient, an insertion sort still requires shifting the elements after the insertion point to make space the new element, which means the time spent on shifting elements still remains O(n), so the insertion sort algorithm would still be O(n^2)

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?

Insertion sort because it has a best case Omega(N) when the input is almost sorted since it only needs to make minor adjustments to correct the position of the elements. This is compared to an unoptimized merge sort, which would run at O(nlgn) due to the need to recursively split the elements. Therefore, the insertion sort would run faster.

1. An unoptimized merge sort or a heap sort?

Netiher is faster than the other since they both have a worst and best case run time of nlgn. However, a heap sort requires less space complexity since it is an in-place sorting algorithm and does not break apart the array, so in that sense it may be faster. Again however, merge sort is better computationally since it does not require the constant computations involved in for instance calculating the left child like a heap sort would, so in that sense it is faster than a heap sort, and fewer computations may be needed if the list is already sorted for a merge sort since fewer comparisons will be needed, whereas with a heap sort, how close the array is to being ordered is always irrelevant since it always turns the input into a binary heap.