CPSC 335 Homework 1 (85 points)

**Deadline:** Sunday, March 5, 11:59 PM

This homework can be done individually or in a group of maximum **3** people. For a group of 2 or 3 members, each of the group members will have to submit, even though the answers will be the same for all the members of the group. All members of the group will get the same grade. If one group member does not submit, that person gets 0. Indicate in an additional .txt file, the names and email ids of members in the group. If working individually, indicate in the .txt, your name and email id.

## #1 [20 points]

In class we learnt how to apply the Heap sort algorithm to a set of numbers. Please refer to this algorithm (“Heap Sort”) on Canvas. For a set of numbers, the steps in the first part involving Build Max heap are posted on Canvas: “Heap sort Part 1 Build Max heap”.

Working with the same set of numbers, now that the heap is built, sketch the next steps of the algorithm, which will eventually sort the array. Draw the tree and corresponding array at each step, just like the steps in the partial example solution.

## #2 [10 points]

The analysis of recursive algorithms gives rise to recurrence relations. For Merge Sort, when n is a power of 2, the recurrence relation is:

T(n) = 1 if n = 1

T(n) = 2T(n/2) + n if n > 1

Using the recursion tree for merge sort, we were able to solve this recurrence relation into its closed form solution: n log n + n

Prove by induction that T(n) = n log n + n and hence is O(n log n)

## #3 [15 points]

Is there a heap T storing seven distinct elements such that a preorder traversal of T yields the elements of T in sorted order? How about an inorder traversal? How about a postorder traversal?

Let these distinct elements be: 1, 2, 3, 4, 5, 6, 7

## #4

## [15 points]

Briefly describe an algorithm for merging *k* sorted lists, each of length *n/k,* whose worst-case running time is O(nk). Clearly describe the algorithm in words, don’t write the pseudo code. Explain mathematically that runtime is in fact O(nk).

An algorithm for merging k sorted lists, each with a length of n/k where n is the total number of elements and k is the total amount of list would involve creating an empty “result” list, and using a function for merging 2 sorted lists and appending that list into the empty “result” list and then merging the current sorted “result” list with the next list available, and appending each value from the next “k” list being merged into the “result” list and continue until there are no more k lists remaining and return the result list that is one sorted list of all k lists.

1. First is creating an empty list, result= [].

2. Create a for loop that would start at i = 0, and goes for as long as i < klists.size() and incrementing 1, i++ (klists is a list of k lists).

3. Use mergeSort function comparing empty list “result” and list[i], and append sorted list into “result” and increment I counter.

4. The algorithm should keep calling mergeSort(result, list[i]) until i not less than klists.size().

5. After the last klists list is merged into the “result” list, return the “result” list as the answer.

## [25 points]

Briefly describe an algorithm for merging *k* sorted lists, each of length *n/k,* whose worst-case running time is O(n log k). Clearly describe the algorithm in words, how you will implement it; don’t write the pseudo code. Justify the running time. Explanation should be clear: doesn’t have to be a mathematical derivation.

To bring the runtime from