**Homework #6**

**Sorting**

total points: 125

1. (5 points) Implement the insertionSort method in the CS232SortableLinkedList class in the hw6 package. This method should perform an insertion sort on the linked list and must run in 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. (40 points) 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 mode (i.e., the value that appears the most times).

 d. 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. (10 points) In our implementation of Insertion Sort we worked backward from position while swapping to find the location at which to insert the th value. This is essentially a linear search. However, because we know that the first values are already sorted we could have used a binary search to find the proper location at which to insert the th value. Is this a useful idea? Why or why not?

4. (20 points) When discussing Merge Sort we talked about two possible optimizations:

i. Switching to use Insertion Sort instead of Merge Sort once the partitions become smaller than a small fixed size (e.g. 10 items).

ii. Checking the two sub-arrays being merged and not performing the merge operation if they are already sorted (i.e. in order and do not overlap).

For each of i. and ii. discuss whether the optimization will change the best and/or worst case asymptotic bounds for Merge Sort. If the bound will change, give the new bound.

5. **(Optional ungraded question.)** Imagine that there exists an algorithm SPLITkthat can split a list of elements into sublists such that sublist contains only elements whose values are less than all elements in sublist , for . You may assume that is a power of (that is, for some ), and thus the elements will be divided equally into the  lists, and the recursive calls in the code below will continue to divide the elements in each sublist equally until the base case is reached. Imagine that there also exists an algorithm JOINk that concatenates sub-lists back into one big list. Finally, assume that SPLITk runs in linear time [ and JOINk runs in constant time .

Now consider the following sorting algorithm that uses SPLITk and JOINk:

List SORTk(List list, int k) {

 if (list.length() > 1) {

 // Split list into k sublists.

 List[] subLists = SPLITk(list, k);

 // Recursively sort all of the sublists.

 for (i=0; i<k; i++) {

 subLists[i] = SORTk(subLists[i], k);

 }

 // Rejoin the sorted sub-list into a sorted list.

 return JOINk(subLists,k);

 }

 else {

 return list;

 }

}

a. Give the part of the recurrence relation that describes the running time of the SORTk algorithm:

b. Use the technique of expansion to express from part (a) using a summation.

c. Solve the summation from (b) and give an asymptotic upper bound on the running time for the SORTk algorithm.

6. (15 points) Modify the CS232ArrayHeap(K[] keys, V[] values) constructor in the CS232ArrayHeap class in the hw6 package so that it automatically heapifies the given key-value pairs instead of requiring that they be provided in level order. This constructor method must run in worst-case time. The No6Tests class contains tests that you can use to check your implementation of this functionality. Hint: only small changes to one line of code are required.

7. (15 points) Complete the heapSort method in the HeapSort class so that given an array of integers they are sorted into descending order in time using HeapSort. You can run the main method to see if your sort works.

8. (10 points) Is Heap Sort in-place? Is it stable? Briefly justify your answers.

9. 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. (5 points) An unoptimized merge sort or insertion sort?

 b. (5 points) An unoptimized merge sort or a heap sort?

BONUS: Implement the mergeSort method in the CS232SortableLinkedList class in the hw6 package so that it performs a stable sort of the values in the list. The BonusTests class contains tests that you can use to check your implementation of this functionality.