Homework 4

Out: 10.8.20
Due: 10.21.20

1. [Insertion Sort, 10 points]
   Insertion sort utilizes a linear search to scan backward through the sorted sub-array A[1..j-1]. Can we use binary search instead in order to improve the overall worst-case runtime of insertion sort? Explain, and provide the improved runtime if your answer is yes.

2. [Cool Sort, 10 points]
   A new sorting algorithm, 'cool sort' is provided below. Does the algorithm correctly sort all arrays of positive n integers? Explain or provide a counter example.

```
coolSort(A) // input: array A of size n
    for i=1 to n-1
            x = A[i]
            k = 0
            for j = 0 to i-1
                    if A[j] <= x
                            k++
                            swap A[k] and A[j]
            swap A[k+1] and A[i]
```

3. [Range, 10 points]
   Describe an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range [a .. b] in O(1) time. What is the preprocessing runtime? Explain.

4. [Sum Elements, 10 points]
   Describe a $\Theta(n \lg n)$ algorithm that determines if for a given integer x, there are two elements in a given array of integers S, whose sum is x. Explain why the runtime applies.

5. [Heaps, 10 points]
   a. Show the content of an initially empty max heap after inserting each of the following numbers one at a time: 6, 9, 4, 12, 15. Show your work.
   b. Show the content of the heap above after an extract-max operation.

6. [In-Place Sorting, 50 points]
   We saw in class different sorting algorithms, and we generally assumed the input to be sorted wasj an array. Here, the provided input is an unordered linked list of n elements with integer values. We are interested in sorting this list in increasing order (smallest first), in O(n log n) worst case time, while using constant space (also called in-place sorting). Note that recursion requires additional space on the stack, so should

be avoided in this case.

You are provided with the following files: *LNode.h*, *LNode.cpp*, *LSorter.h*, *LSorter.cpp*, and *main.cpp*.

A linked list node, *LNode*, is implemented in *LNode.h* and *LNode.cpp*.
The *LSorter* class with the *sortList* method are declared in *LSorter.h*. The *sortList* method, which you need to implement, is declared as follows:

```
class LSorter {
public:
    LNode* sortList(LNode* head);
};
```

This method accepts as input the head node of the list to be sorted, and returns the head node of the sorted linked list. Your implementation should be included in the *LSorter.cpp* file. You may add additional classes and/or methods as you see fit, but everything should be included in this file, and none of the other files may be modified.

Finally, the provided *main.cpp* file may be used to test your implementation. You may assume that your input consists of non-negative integers with a maximal value of 1,000,000.

Modify and submit *LSorter.cpp* only. The submitted file should compile and run with the rest of the files. We will run it against large linked list and measure the run time.

Partial credit will be given for an in-place solution with a runtime that is worse than O(n log n).