Homework 4

Out: 10.12.23
Due: 10.25.23

1. [Bubble Sort, 10 points]
Describe an implementation of a modified bubble-sort algorithm that uses two stacks and up to three additional variables to sort a sequence of n elements. The elements are initially stored in one of the stacks, and after your algorithm is run, should be stored in order in a stack. Be sure to specify when your algorithm should stop running.

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. [Sum Elements, 10 points]
Describe a Θ(n log 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.

4. [Sorting, 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?

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. [Skip List, 50 points]
This task builds on the Linked List problem from the previous homework. Your starting point should be your implemented *LinkedList.h*, which you should incorporate into this new design.

You are provided with a partial implementation of a templated doubly-linked list in file *SkipList.h*. It contains the templated *Node* class and a templated *LinkedList* class

from the previous assignment, and a *SkipList* class.

Design a skip list class, which consists of a doubly-linked list with only sentinel -∞, +∞ nodes at its top-most level, and contains all elements at its bottom-most level list. A node added to any given level has a probability of ½ to be added to the next level up, and node that is not added to a given level cannot be added to any of the higher levels. (Thus, the number of levels should be about lg(n), where n is the number of elements.) Your design should incorporate your implementation of a doubly-linked list from the previous assignment, and use it to implement the skip list.
Do not modify the class definitions.

The skip list should contain the following methods and members:

- SkipList – Constructor of an empty skip list consisting of the top-level list only, which accepts the linked lists sentinel values.
- ~SkipList() – Destructor of a skip list.
- search – Tests if a value x is contained in the skip list, returns a pointer to its location in the bottom-most list if it exists, or the location of its predecessor in the bottom-most list, if does not exist.
- insert – Accepts 'data' to be inserted. Inserts node with data, returns pointer to the inserted node in the bottom-most list if node is inserted, NULL if node already exists and thus is not inserted.
- printData – Prints the content of the skip list (data values only, separated by spaces and new lines), (data values only, separated by spaces), utilizing the LinkList class' printData method.
- print – Print the entire content of the skip list (including nodes' addresses and pointers), utilizing the LinkList class' printData method, may be used for debugging purposes.
- topList – pointer to the beginning of the top-most list of the skip list.
- randSeed – seed to be used with the provided getRand() random number generator, for grading standardization (see description below).

Note that the sentinel values are dependent on the type T, and therefore may be declared as variables.

The printData function in the provided *main.cpp* could give the following output:

```
SkipList data:
-2147483648 2147483647
-2147483648 2 2147483647
-2147483648 2 5 6 9 2147483647
-2147483648 1 2 3 5 6 7 9 2147483647
```

Implement the *SkipList* class such that it could be compiled and ran with the provided *main.cpp* file. To do so, you are allowed to add methods and members to the *LinkedList* class and to the *SkipList* class.

A note on random number generation:
When generating random numbers via the *srand* library function, it is best practice to initialize it using a different random seed every time (e.g the time of day). However, a truly random skip list makes grading difficult. As a workaround, we provide you with a fixed randSeed (initially 330), and a getRand() function, for grading standardization.

In your skip list implementation, call srand(this->randSeed) once in the SkipList constructor, and then call the provided *getRand()*, which returns either 0 or 1 with a 50% chance, when inserting nodes to the skip list. If it returns a 1, then insert node to the next level of the skip list.

Submit your modified *SkipList.h* file only. Your code must compile and run on the lab computers with a variation on the provided *main.cpp* file.

Your code will be graded based on:

★ Correct functionality

★ How easy it is to read and understand

★ Whether it demonstrate good design and style