CSC 220 Data Structures

<u>Program #8 - EASIER</u>

Advanced Sorting Algorithms

Your objective is to implement the following three algorithms:

- From the list of quadratic time algorithms: <u>Cocktail Sort</u>
- From the list of linearithmic time algorithms: <u>Quick Sort</u>
- From the list of linear time algorithms: <u>Counting Sort</u>

All of these algorithms are explained in detail in the notes and some or all of the pseudocode behind them is given. You are to implement the algorithm as described in the lecture notes. Each of these algorithms should be done in a separate function.

There is no template or .class files for this. Everything is done by you. You must have a Main class that has the entry point, but you are free to have other classes if you want. You could have a different class for each sorting method, a single class that has three methods for the three different sorts, or you could put three methods inside the Main class for sorting. You decide.

You must have a public class in its own file called Main. This public class will contain the entry point which must do the following:

- First generate an array of 20 ints, each ranging from 0 to 150 (inclusively)
  - You will need to look online to figure out how to generate random numbers in a range with Java
- Make three exact copies of that array
  - Make sure you have three copies and not three variables that point to the same copy. This is an important distinction to make.
- Send each sorting method its own copy of the array
  - This is so each sorting algorithm can start with the same array contents
- Print out the original array of ints
- Print out the sorted array of ints after applying each sorting method
  - See the Sample Output below for more clarification
- Now generate an array of 20,000 ints each within the same value range as above (i.e. 0 to 150)
  - There will be duplicate keys, but this should not pose a problem for your algorithms
- Make three exact copies of this large array
- Now use Java to time each algorithm as it runs through its copy of the array
  - You are going to have to look up online how to use Java to get the exact runtime of a block of code
  - It is VERY IMPORTANT that you do NOT include the allocation, initialization or cloning of the array of ints in your timing. The only thing I want you to time is the sorting algorithm itself (and maybe the function call to the sorting algorithm). When the algorithm is done (function returns back to main), record the time and move on to timing the next algorithm.

- Show the times each algorithm took (in milliseconds)
    - Obviously, we should see Cocktail sort taking the longest amount of time while Counting sort takes the least. Quick sort should be somewhat close to the runtime of Counting sort.

Sample Output:

Your output should look exactly like the following (albeit with different numbers for your list and different numbers for your times). Your timings might be different, but they should still be within reason. For example, I expect Cocktail to be in the triple digits, Quick to be in the single or low double digits, and Counting to be the fastest in the mid to low single digits. If your timings do not fall within this range, this is an indication that something is not working right or you aren't timing right. Please ask me for help if you can't figure it out. The spacing, values for n, text, etc must be the same.

```
$ javac Main.java && java Main
TESTING with n = 20
  Original List: 76 91 129 92 0 67 40 57 43 2 58 120 57 67 4 138 31 56 117 41
  Cocktail sorted: 0 2 4 31 40 41 43 56 57 57 58 67 67 76 91 92 117 120 129 138
  Quick    sorted: 0 2 4 31 40 41 43 56 57 57 58 67 67 76 91 92 117 120 129 138
  Counting sorted: 0 2 4 31 40 41 43 56 57 57 58 67 67 76 91 92 117 120 129 138

TIMING with n = 20,000
  Cocktail took 759.28 ms
  Quick    took 8.47 ms
  Counting took 4.63 ms
```

Note:

Use the demo_all.jar file on Moodle for help (labelled as "for all other data structures and algs" in the Visualizations section). Run it with java -jar demo_all.jar and play around with the different sorting methods (mainly cocktail, quick and counting for this assignment). Make sure you fully understand the algorithms. You **cannot** even begin to properly implement something until you fully understand it. Blindly converting the pseudocode from the notes into Java will not be enough.

The sorting algorithms I am asking you to implement are decently popular. It is possible that you can find Java implementations already online. Let me be clear about this. **<u>If you grab an implementation from an online source and try to pass it off as your own, you will receive the penalty for cheating on this assignment.</u>** You are to write each sort yourself following the algorithm from the lecture notes.

For submission:

Submit all .java source code files (in a zip folder). Do not submit any .class nor .jar files.

Bonus:

For an additional 8 bonus points, implement <u>either</u> Comb Sort, Merge Sort or Radix Sort from the extra notes (i.e. Part 3). Put this new sort method in its own function. Add the resulting sorted array to the output in the same format as the other algorithms. Also, add the runtime to the end of the timings in the same format as the other algorithms. Comb sort should be similar in time to Cocktail sort. Merge sort should be similar in time to Quicksort. Radix sort should be similar in time to Counting sort (depending on how you implemented it, it may be closer in time to Quicksort, and that's ok).

Rubric:

#	ITEM	POINTS
1	Cocktail sort works	7
2	Quick sort works	7
3	Counting sort works	7
4	Cocktail sort time is reasonable	4
5	Quick sort time is reasonable	4
6	Counting sort time is reasonable	4
7	Output matches	4
8	All directions followed	3
	TOTAL	**40**

#	PENALTIES	POINTS
1	Doesn’t compile	-50%
2	Doesn’t execute once compiled (i.e. it crashes)	-25%
3	Late up to 1 day	-25%
4	Late up to 2 days	-50%
5	Late after 2 days	-100%