Homework #2

*Please present a **printed** report with all your answers, explanations, and sample plots. Submit also a soft copy of the source code and binaries used to generate these results. Please note that copying of any results or source code will result in ZERO credit for the whole homework.*

Problem 1

Suppose that you are given n red and n blue water jugs, all of different shapes and sizes. All red jugs hold different amounts of water, as do the blue ones. Moreover, for every red jug, there is a blue jug that holds the same amount of water, and vice versa.

Your task is to find a grouping of the jugs into pairs of red and blue jugs that hold the same amount of water. To do so, you may perform the following operation: pick a pair of jugs in which one is red and one is blue, fill the red jug with water, and then pour the water into the blue jug. This operation will tell you whether the red or the blue jug can hold more water, or that they have the same volume. Assume that such a comparison takes one time unit.
Remember that you may not directly compare two red jugs or two blue jugs.

1. Describe a deterministic algorithm that uses $O(n^2)$ comparisons to group the jugs into pairs.
2. Prove a lower bound of $\Omega(n \lg n)$ for the number of comparisons that an algorithm solving this problem must make. (*Hint*: check the proof of lower bounds of comparison sorts).

Problem 2

Given a set of n numbers, we wish to find the i largest in sorted order using a *comparison-based* algorithm. Describe an algorithm that solves this problem with the required criteria below and justify why that is the case:

1. Space $\Theta(1)$ and worst case running time $O(n \lg n + i)$
2. Space $\Theta(1)$ and *expected* worst case running time $O(n + i \lg i)$ [Hint: randomized select]
3. Space $\Theta(1)$ and worst case running time $O(n + i \lg n)$ [Hint: priority queues]

Problem 3

Describe a $\Theta(n)$-time algorithm that, given a set S of n integers and another integer x, determines whether or not there exists two elements in S whose elements sum to x. Analyze its running time to verify it takes $\Theta(n)$.

Problem 4

Write a C++ function that sorts an input array of integers using **Radix sort** with **Counting sort** as the auxiliary stable sorting algorithm.

1. Assume the inputs are unsigned 64-bit integers and that $r = 8$ bits (the number of digits for each

number) i.e. each 64-bit number is represented as 8 digits in base-$2^r = 256$. Run your algorithm on random arrays of sizes 100, 10^3, 10^4, 10^5, 10^6, 10^7. Measure the CPU time for each such run and plot the results. To get more accurate timings, you should take the average of 25 runs for each input array size (with a random array each time). Does the time look linear in the input size n? Why or why not?

2. Repeat part (1) above but setting r every time to be r = ceil(lg n) for every n. Does the run time look better?
3. Plot the results form part (2) above with the results of your implementation of Quicksort from homework #1. Which one looks better?

Problem 5

You are given a set of n **static** keys that need to be inserted in a hash table for fast search. You are required to implement a hash table, in C++, that supports search operations in constant time in the worst case. Use the universal hash functions that are defined by $h_{ab}(k)=((a\,k+b)\,mod\ p)\,mod\ m$ for $a\in(1,2,..,p-1)$ and $b\in(0,1,...,p-1)$ for some prime number p. Set $m = n$ for all experiments below.

1. Implement the perfect hashing scheme described in the lectures (and textbook). Note that the size of the secondary hash table for each slot $m_i=n_i^2$.
2. Try inserting n random keys into the hash table for n = 1000, 10000, 100000. For each n, search 1000 keys (500 that are already inserted and 500 that are not). Plot the average search time for each n with n on the x-axis.
3. Repeat (1) and (2) for a normal hash table, with collisions resolved by chaining i.e. linked lists instead of secondary hash tables, with new keys inserted at the head of the lists. Compare the average search time as a function the number of keys in the hash table to that using perfect hashing. [Note: you can use a ready made implementation of a linked list, e.g. from the C++ STL].

Instructions

Please submit a soft copy of the solutions together with source code and binaries in one zip file. The file should be named as **CMP448.HW##.BN##.First.Last.zip** where HW## is the homework number e.g. HW01, BN## is your bench number, First and Last are your first and last name. So, if your bench number is 26, your name is Mohamed Aly, the file should be named **CMP448.HW01.BN26.Mohamed.Aly.zip.** Failing to follow these instructions will cost you points.