Homework #2

Due Date: 11:59pm Saturday 2 November 2013

*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

Write two C++ functions that sort an input array of integers (in place) using Heapsort and Quicksort (described in the textbook).

1. Run your algorithm on a random array of sizes 5, 10, 25, 50, 100, 500, 1000, and 10000. Measure the CPU time for each such run for each algorithm, 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).
2. Repeat part (1) above but on sorted arrays i.e. generate a random array of integers, and sort it

first (both in increasing and decreasing order) and plot the running time of both Heapsort and Quicksort. Which one is better?

3. Modify your Quicksort to choose a *random* pivot instead of the last element of the array, and
4. Repeat parts (1) and (2). Does it improve?

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 $r = \text{ceil}(\lg n)$ 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. Which one looks better?

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 **CMP302.HW##.First.Last.zip** where HW## is the homework number e.g. HW01, First and Last are your first and last name. So, if your name is Mohamed Aly, and this is homework #1, the file should be named **CMP302.HW01.Mohamed.Aly.zip.** Failing to follow these instructions will cost you points.

Acknowledgment: Some problems are adapted from Erik Demaine.