Algorithms & Data Structures 2024/25

Practical Week 9

I realise that model answers tend to be easily available. Please however try to come up with your own solutions. If you do get stuck then ask the demonstrators for help.

0. Finish anything that may be left over from last week. (This is particularly important for any and all exercises on asymptotic notation.)
1. For each of the following functions $f(n)$ and $g(n)$, determine which of the following statements applies (and justify your answer): $f(n) = o(g(n))$, $f(n) = \Theta(g(n))$, or $f(n) = \omega(g(n))$.
   (a) $f(n) = 1000n^2$, $g(n) = n^3$.
   (b) $f(n) = 2.1^n$, $g(n) = 2^n$
   (c) $f(n) = 5n$, $g(n) = \log_2(3^n)$
   (d) $f(n) = n^2$, $g(n) = \frac{n^2}{\log_2 n}$
2. For each of the following statements (where f, g, h are non-negative functions), determine whether it is true or false, and give a proof or a counterexample.
   (a) If $f(n) = \omega(g(n))$ and $g(n) = O(h(n))$, then $f(n) = \omega(h(n))$.
   (b) If $f(n) = o(g(n))$, then $f(n) = O(g(n))$.
   (c) If $f(n) = \omega(g(n))$ and $g(n) = \Theta(h(n))$, then $f(n) = \omega(h(n))$.
   (d) If $f(n) = O(g(n))$ but not $g(n) = O(f(n))$, then $f(n) = o(g(n))$.
3. Explain how one might modify the Merge function (used within MergeSort) to achieve a constant $O(1)$ number of comparisons in certain circumstances. What are those circumstances?
4. Suppose you are given an array $A[]$ of length n containing numbers. Suppose further that you are given access to a random number generator `random(k)` that returns a random integer from $\{0, \ldots, k-1\}$, for any parameter k. How do you generate a random permutation of the elements in $A[]$ in linear time?
5. In terms of real-world performance (as opposed to asymptotic analysis), it may be beneficial for recursive sorting algorithms to not recurse all the way down to problems of size 1 but stop recursion earlier, replacing those recursive calls with (ideally) fast algorithms to sort those small subproblems. This is to avoid excessive amounts of expensive stack operations. For example, an algorithm that can sort any sequence of five numbers using as few comparisons as possible may be a candidate for replacing recursive calls on five elements.

   Design and implement (in a language of your choosing) an algorithm that sorts any sequence of five elements using at most seven comparisons. The algorithm should be "in place", that is, you shouldn't be using auxiliary arrays. Test on all $5! = 120$ permutations of the input $[1, 2, 3, 4, 5]$ (you may find it useful to write a function that automatically generates, and feeds into your sorting algorithm, those permutations).
6. **Solving this question requires familiarity with basic probability theory. Please feel free to skip the calculation of the expected number of comparisons if you feel that you are not sufficiently familiar with probability theory.**

   Consider the following sorting algorithm.

   MonkeySort $(a_1, \ldots, a_n \in \mathbb{R})$
   1: **while** $(a_1, a_2, \ldots, a_n)$ is not sorted **do**
   2: randomly permute $a_1, a_2, \ldots, a_n$
   3: **end while**

In the literature this algorithm is also variously known as “StupidSort”, “BogoSort”, or “Slow-Sort”.

Provide more detailed code, in particular for testing for sortedness.

This being a randomised algorithm, we need to express the number of comparisons it makes as a random variable. If you know your way around very basic probability theory: what’s that random variable’s expectation, asymptotically?

You may assume that the input elements are pairwise distinct.