Homework 1

CS 5114 (Fall 2018)

Assigned on Monday, August 27, 2018.
Submit a PDF file containing your solutions on Canvas by the beginning of class on Monday, September 3, 2018.

Instructions:

- The Graduate Honor Code applies this to homework. In particular, you are not allowed to consult any sources other than your textbook, the slides on the course web page, your own class notes, and the instructor. Do not use a search engine.
- Do not forget to typeset your solutions. *Every mathematical expression must be typeset as a mathematical expression, e.g., the square of n must appear as n^2 and not as "n^2".* You can use the LaTeX version of the homework problems to start entering your solutions.
- Describe your algorithms as clearly as possible. The style used in the book is fine, as long as your description is not ambiguous. Explain your algorithm in words. A step-wise description is fine. *However, if you submit detailed code or pseudo-code without an explanation, we will not grade your solutions.*
- Do not make any assumptions not stated in the problem. If you do make any assumptions, state them clearly, and explain why the assumption does not decrease the generality of your solution.
- Do not describe your algorithms only for a specific example you may have worked out.
- You must also provide a clear proof that your solution is correct (or a counter-example, where applicable). Type out all the statements you need to complete your proof. *You must convince us that you can write out the complete proof. You will lose points if you work out some details of the proof in your head but do not type them out in your solution.*
- Describe an analysis of your algorithm and state and prove the running time. You will only get partial credit if your analysis is not tight, i.e., if the bound you prove for your algorithm is not the best upper bound possible.

Problem 1 (5 points) Solve exercise 1 in Chapter 1 (page 22 of "Algorithm Design" by Kleinberg and Tardos.

Problem 2 (5 points) Solve exercise 2 in Chapter 1 (page 22 of "Algorithm Design" by Kleinberg and Tardos.

Problem 3 (25 points) Consider the following algorithm to determine if a given positive interger n is prime. For each integer i between 2 and $\lfloor\sqrt{n}\rfloor$, check if i is a factor of n, i.e., if dividing n by i leaves a remainder of 0. If the answer is "yes" for any i, return that n is composite. Otherwise, return that n is prime. Answer the following questions about this algorithm.

(a) (5 points) Let us use $f(n)$ to denote size of the input to this problem. What is $f(n)$? Is $f(n) = 1$ (because there is only number in the input)? Is $f(n) = n$ (because the value of the input is n)? Is it something else entirely? In the last case, write down what you think this alternative is.

(b) (10 points) What is the running time $g(n)$ of the algorithm in terms of the input size? You may assume that we can check whether if i is a factor of n in $O(1)$, i.e., constant, time.

(c) (10 points) Prove that $g(n) = \Omega(f(n)^d)$ for any positive value d. In other words, prove that the running time of this algorithm for testing primality is lower bounded by any arbitrary polynomial function of the input size. You can start from any statement in the textbook or the slides but be sure to explain how you derive your conclusion from the starting statement.

Note: It is likely that you will be able to prove part (c) only if you answer part (a) correctly.

Problem 4 (25 points) (20 points) Solve exercise 5 in Chapter 2 (page 68) of "Algorithm Design" by Kleinberg and Tardos. If you decide that a statement is true, provide a short proof. Otherwise, provide a counterexample. *Note:* If you think one of the statements is true, you should not prove it for your own choices of $f(n)$ and $g(n)$. Your proof must hold for *any* pair of functions $f(n)$ and $g(n)$ where $f(n)$ is $O(g(n))$.

Problem 5 (40 points). Describe an algorithm that uses a priority queue (heap) to merge k sorted lists into one sorted list. Each sorted list has n integers. Neither k nor n is a constant.

To get you started, consider the following algorithm (let us call it `Algo1`):

1. Insert all the kn numbers into a priority queue, with each number being its own key.
2. Repeatedly report the smallest key in the queue and delete this value from the queue, until the queue becomes empty.

What is the running time of this algorithm?

Now develop a heap-based algorithm (let us call it `Algo2`) that has a better running time. Describe your algorithm, prove its correctness, and provide an analysis of the running time. Show that the running time of `Algo1` is lower-bounded by the running time of `Algo2`, i.e., if $f_1(n, k)$ is the running time of `Algo1` and $f_2(n, k)$ is the running time of `Algo2`, then $f_1(n, k) = \Omega(f_2(n, k))$. Note that the running time of both algorithms will depend on both n and k since there are nk numbers in total.