CS41 Lab 2

September 15, 2017

In typical labs this semester, you'll be working on a number of problems. You are encouraged to work with one or two others. You will not be handing in solutions; the primary purpose of these labs is to have a low-pressure space to discuss algorithm design. However, it will be common to have some overlap between lab exercises and homework sets.

Note: if you do not feel fully confident when it comes to asymptotics, induction, or proofs, I strongly encourage you to focus on the initial two problems first.

1. **Induction.** Using induction, show that the following summations hold for all $n \geq 0$.

   - $\sum_{k=0}^{n} k = \frac{n(n+1)}{2}$.
   - $\sum_{k=0}^{n} 2^k = 2^{n+1} - 1$.

2. **Asymptotic analysis.** Assume you have functions f and g such that $f(n)$ is $O(g(n))$. For each of the following statements, decide whether you think it is true or false and give a proof or counterexample.

   (a) $\log_2(f(n))$ is $O(\log_2(g(n)))$.
   (b) $2^{f(n)}$ is $O(2^{g(n)})$.
   (c) $(f(n))^2$ is $O((g(n))^2)$.
   (d) If $g(n)$ is $O(h(n))$, then $f(n)$ is $O(h(n))$.

3. **Close to sorted.** Say that a list of numbers is "k-close-to-sorted" if each number in the list is less than k positions from its actual place in the sorted order. (So a 1-close-to-sorted list is *actually* sorted.) Give an $O(n \log k)$ algorithm for sorting a list of numbers that is k-close-to-sorted.

   In your algorithm, you may use any data structure or algorithm from CS35 by name, without describing how it works.

4. **The Wedding Planner Problem, redux.** The Wedding Planner Problem from last week seemed challenging. This week, you'll try to break it down into simpler cases. For concreteness, here is a synopsis of the problem.

   Imagine you are a wedding planner who must help couples decide who to invite to a wedding. Couples come to you with lists of people they might invite to their wedding. They also come with demands Alice and Bob need to be invited, but if Bob is invited, do not invite Carol (they have history). When Carol, Dave, and Eve get together, they only talk about Justin Bieber (their favorite musician), so don't invite all three. However, any two of them is ok. Call these conditions constraints. People at weddings are demanding. Everything needs to be exactly perfect, and they will blame you if even one constraint is not satisfied. You're not even sure if this is possible, and it when it's not, it would be nice to have a convincing argument for the wedding couple.

(a) First, formulate the problem. What are the inputs to the wedding planner problem? How would you describe these inputs in data? (e.g. should the input be an integer? a list? a list of what? etc.) What should the output be? What does an algorithm have to achieve?

(b) Suppose you hire an assistant named Vinny to help you plan the wedding. In this case, Vinny takes the list of possible people to invite and the constraints and gives you a list of people to invite (and who to not invite) such that all constraints are met. However, Vinny is new, and you want to check to make sure he is doing his job. Design and analyze and efficient VERIFIER algorithm that takes (i) a list of people, (ii) a list of constraints, and (iii) a candidate invitation list (this is what Vinny gives you), and outputs (i.e. verifies) whether or not all constraints have been satisfied.

(c) Design a brute-force algorithm for the wedding planner problem. Provide pseudocode and explain why your algorithm is correct. (Hint: it may be helpful to use your VERIFIER algorithm as a subroutine.)