CS41 Homework 7

This homework is due at 11:59PM on Sunday, November 2. This is a 13-point homework. Write your solution using LaTeX. Submit this homework using **github** as a **.tex** file. This is a **partnered homework**. You should primarily be discussing problems with your homework partner.

It's ok to discuss approaches at a high level with others. However, you should not reveal specific details of a solution, nor should you show your written solution to anyone else. The only exception to this rule is work you've done with a lab teammate *while in lab*. In this case, note (in your **homework submission poll**) who you've worked with and what parts were solved during lab.

> Any time you are asked for an algorithm, you should *of course* include a description of your algorithm, a proof of correctness, and a runtime (including halting!) analysis. This instruction holds for homeworks for the entire semester (unless the problem specifically says you do not need to include those proofs).

1. **Can this graph exist?** (Kleinberg and Tardos, 4.29)
   The *degree* of a node in an undirected graph is the number of edges it has. (For example, a node with only one neighbor has degree 1, and the root of a binary tree has degree 2.)

   Given a list of n natural numbers $d_1, d_2, \ldots, d_n$, give an efficient algorithm which can decide whether there exists an undirected graph $G = (V, E)$ whose node degrees are precisely the numbers $d_1, d_2, \ldots, d_n$. (That is, if $V = \{v_1, v_2, \ldots, v_n\}$ then the degree of v_i should be exactly d_i.) G should not contain multiple edges between the same pair of nodes (e.g. $e = \{u, v\} \neq e' = \{u, v\}$), or "loop" edges $\{u, u\}$ where both endpoints are the same node. Prove that your algorithm is correct.

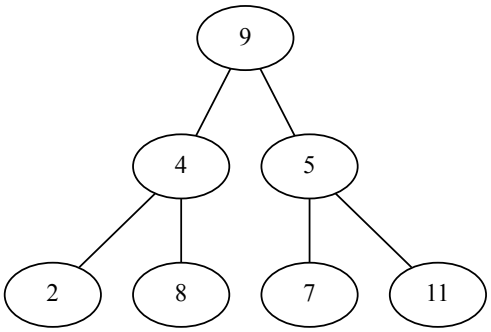
2. **Summer camp triathlon** (K& T 4.6)
   Your friend is working as a camp counselor, and is in charge of organizing activities for a set of junior-high-school-age campers. One of the plans is the following mini-triathlon exercise: each contestant must swim 20 laps of a pool, then bike 10 miles, then run 3 miles. The plan is to send the contestants out in a staggered fashion, via the following rule: the contestants must use the pool one at a time. (In other words, first one contestant swims the 20 laps, gets out, and starts biking. As soon as this person is out of the pool, a second contestant begins swimming the 20 laps; as soon as the second person is out and starts biking, a third contestant begins swimming, and so on.)

   Each contestant has a projected *swimming time* (the expected time it will take them to complete the 20 laps), a projected *biking time* (the expected time it will take them to complete the 10 miles of bicycling), and a projected running time (the expected time it will take them to complete the 3 miles of running). Your friend wants to decide on a *schedule* for the triathalon: an order in which to sequence the starts of the contestants. Let's say that the *completion time* of a schedule is the earliest time at which all contestants will be finished with all three legs of the triathalon, assuming they each spend exactly their projected swimming, biking, and running times on the three parts. (Again, note that participants can bike and run simultaneously, but at most one person can be in the pool at any time.) What's the best order for sending people out, if one wants the whole competition to be over as early as possible?

More precisely, give an efficient algorithm that produces a schedule whose completion time is as small as possible.

3. **Finding the minimum in a tree**

Consider an n-node complete binary tree T, where $n = 2^h - 1$ for some h. Each node v in T is labeled with a unique real number x_v. A node v in T is a **local minimum** if for all neighbors w of v, we have $x_v < x_w$. For example, in the tree below, 5 is a local minimum, because its neighbors all have higher values $(7, 9, 11)$.



Access to vertex labels is restricted; you can only see the node labels by *probing* the node. Your goal is to find a local minimum using the fewest number of probes possible.

Given a complete binary tree T, design an algorithm FindMin(T) to find a local minimum using $O(\log(n))$ probes to the nodes of T.

4. **Integer multiplication: further improvements?**

Recall the Karatsuba algorithm for integer multiplication for class, which multiplies two n-digit base-N numbers a, b by:

$$\begin{aligned} a \cdot b &= (a_L N^{n/2} + a_r)(b_L N^{n/2} + b_R) \\ &= a_L b_L N^n + (a_L b_R + a_R b_L) N^{n/2} + a_R b_R \\ &= A N^n + (C - A - B) N^{n/2} + B, \end{aligned}$$

where the three $(n/2)$-digit multiplications are:

- $A = a_L \cdot b_L$
- $B = a_R \cdot b_R$
- $C = (a_L + a_R) \cdot (b_L + b_R)$

Consider an algorithm for integer multiplication of two n-digit base-N numbers where each number is split into three parts, each with $n/3$ digits.

(a) Design such an algorithm, similar to the integer multiplication we did in class. Your algorithm should describe how to multiply two integers using only six multiplications (instead of the straightforward nine).

(b) Determine an asymptotic upper bound for the running time of your algorithm. (Write it as a recurrence, and then solve the recurrence.) You should use partial substitution.

(c) Is this algorithm asymptotically faster than Karatsuba multiplication? That is, is it better to use the algorithm that breaks an integer into three parts, or two parts?

(d) **(extra challenge)** Suppose there were an algorithm that used only five multiplications of $n/3$-digit numbers instead of six. Determine an asymptotic upper bound for the running time of such an algorithm. Would this algorithm be asymptotically faster than Karatsuba multiplication?

5. **(extra challenge) Making change**

Consider the problem of making change for n cents out of the fewest number of coins. Assume that n and the coin values are positive integers (cents).

(a) Describe a greedy algorithm to solve the problem using the EU coin denominations of 50 cents, 20 cents, 10 cents, 5 cents, 2 cents, and 1 cent. Prove your algorithm is optimal.

(b) Describe a greedy algorithm to solve the problem using the US coin denominations of quarters (25), dimes (10), nickels (5), and pennies (1). Prove your algorithm is optimal.

(c) Suppose the country of Algorithmland uses denominations that are powers of c for some integer c. This country uses $k+1$ denominations of $c^0, c^1, \ldots, c^k$. Show that your greedy algorithm works in Algorithmland as well.

(d) Design a currency system of your choosing with at least three coin denominations such that a greedy algorithm does *not* yield a minimum number of coins for some amount of m cents. Assume that one of your denominations has value one, so a solution exists for all values of m.