CS41 Homework 5

This homework is due at 11:59PM on Sunday, October 27. Write your solution using LaTeX. Submit this homework in a file named `hw4.tex` using **github**. This is a **10 point assignment**. This is a partnered homework. You should primarily be discussing this homework with your partner.

It's ok to discuss approaches at a high level. In fact, I encourage you to discuss general strategies. 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 partner/group *while in lab*. In this case, note (in your post-homework survey) who you've worked with and what parts were solved during lab.

The main **learning goals** of this homework are to get more practice with algorithm design, especially as it pertains to graph algorithms and greedy algorithms.

1. **Recurrence Relations.** Solve the following recurrence relations:

   (a) $A(n) = 5A(n/3) + 2n$,
   $A(1 = 6$

   (b) $M(n) = 4M(n/2) + 3n$,
   $M(1) = 4$

   Note: You must use the Substitution Method for one of the problems and Recursion Trees for the other.

2. **Summer camp triathlon** (Kleinberg and Tardos, 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. **Evaluating investment strategies.**

   An investment company has designed several new trading strategies and wants to compare them. Unfortunately, the strategies were tested on different commodities over different time

periods, so the total profit earned is not a fair comparison. Instead, the company wants to compare each strategy's profit to the maximum possible profit that could have been made (with perfect hindsight) during the same time span. Each strategy was allowed to buy a fixed amount of the commodity once and then sell what it bought at some later date.

During each testing period, the investment company recorded the minimum and maximum prices that were reached each day. Your algorithm receives a list of n pairs, where the i^{th} pair has the minimum price of the commodity and the maximum price of the commodity on day i. Your algorithm should output the largest price difference that could have been achieved by buying at the minimum price on day i and selling at the maximum price on day $j > i$.

For example, suppose $n = 3$ and the (min, max) prices were $[(8, 14), (1, 2), (4, 6)]$. Then you should return a per-unit profit of 5, corresponding to buying for 1 on day 2 and selling for 6 on day three (recall that the trading strategies must buy first, and can't sell on the same day).

Clearly, there is a simple algorithm that takes time $O(n^2)$: try all possible pairs of buy/sell days and see which makes the most money. Design a divide and conquer algorithm to determine the best profit in hindsight more efficiently. Set up a recurrence that describes the running time of your algorithm, and solve it to show that your algorithm is faster than $O(n^2)$.