CS140 - Assignment 5

Due: Sunday, Oct. 16 at 11:59pm

For this assignment, you may (and are encouraged to) work with a partner. It can be anyone in the class.

1. **3 points**] *Extra credit*

   The assignment is due at the usual time on Sunday. However, to encourage you to finish it early (and have a proper break), if you submit the assignment by 11:59pm on Friday, you will receive 3 points of extra credit on this assignment.

2. [**8 points**] Stacks and queues revisited

   Last week you showed that you could implement a queue using two stacks. In fact, you showed using the aggregate amortized analysis technique that `enqueue` and `dequeue` have amortized constant time. Now prove the same amortized constant time using the accounting method for amortized analysis.

3. [**30 points**] Ski Optimization

   You work at a ski rental place and you would like to design an algorithm that matches skiers to skis. Ideally every skier should get a pair of skis whose length matches his or her height. Unfortunately, in general this is not possible. So, to measure how good a particular match of ski to skier is, we'll use the *disparity* between them, which we'll define as the absolute value of the difference between the length of the skis and the height of the skier. Now we want a way to assign skis to skiers that minimizes the sum of the disparities.

   The input to this problem is an array of n skiers (each skier is a pair comprising their name and their height) and an array of $m \geq n$ pairs of skis (each ski pair is just the height of the skis). These arrays are given in sorted order from shortest to tallest.

   (a) (2 points) Since the first goal is to have *an* algorithm, you start simple. Describe a brute-force algorithm that naively explores all possible options and explain how long it would take to execute this algorithm on a computer that performs 1 billion operations per second if there were 20 skiers and 20 pairs of skis.

   (b) (2 points) After deciding that you can't wait that long, you consider the following greedy "algorithm": since the skiers and skis are already sorted, consider the skiers one-by-one from shortest to tallest. For each skier under consideration, assign that skier the pair of skis that most closely match that skier's height. Then remove that skier and pair of skis from consideration and repeat the process. Find a small example where this gives a solution that is worse than optimal. You'll need to provide a particular small set of skier heights, a small set of ski heights, show the solution produced by the greedy "algorithm", and then show a solution that is better.

(c) (4 points) In search of something better, you realize that if there is a short person and a tall person, it is never better to give the shorter person a taller pair of skis than were given to the tall person. In other words, there exists an optimal solution to the ski problem in which there are no "inversions" in which persons x and y, with x shorter than y, are assigned skis such that x has longer skis than y. It turns out that this conjecture is not hard to prove, particularly if you perform a detailed case analysis. Enumerate the cases that you would consider in proving this conjecture and then give a short proof of any one of the cases. (A few sentences should suffice for the proof.)

(d) (4 points) Assume there are n people and n pairs of skis. Describe (in English sentences) a fast algorithm for assigning skis to skiers, briefly explain how the proof of correctness would work, and give the running time of your algorithm.

(e) (18 points) Finally, consider the general case that $m \geq n$.

i. In simple and clear pseudo-code or English, describe a recursive algorithm for solving this problem. For now, assume that "solving" means just finding the number which is the sum of the disparities in an optimal solution (that is, the sum of the differences between the skiers and their skis in an optimal solution). Make sure to describe the base cases and the recursive call(s).

ii. Next, describe how you would implement this algorithm using dynamic programming. In particular, describe what the DP table looks like and the order in which the cells would be filled in.

iii. What is the running time and space of your algorithm?