Assignment 3 (due Friday, March 1, 6:00pm)

Instructions:

- Hand in your assignment using Crowdmark. Detailed instructions are on the course website.
- Give complete legible solutions to all questions.
- Your answers will be marked for clarity as well as correctness.
- For any algorithm you present, you should justify its correctness (if it is not obvious) and analyze the complexity.

   A delivery robot is trying to travel in a straight line to a destination $D$ meters away. Along this straight line, there are several robot docking stations at distances $d_1, d_2, ..., d_n$ (expressed in meters from the robot’s starting position) where the robot can stop and regenerate before continuing. The robot can travel at most $k$ meters before it must regenerate. The problem is to identify which docking stations the robot should stop at in order to minimize the number of stops. For simplicity, you may assume it is always possible to reach the destination.

   **Input:** Variables $D, k$ and $d_1 < d_2 < ... < d_n$.
   **Output:** A feasible subsequence of $d_1, d_2, ..., d_n$ with the minimum total travel time.

   (a) [4 marks] Design an greedy algorithm that solves the problem.
   (b) [1 marks] What is the asymptotic runtime complexity of your algorithm?
   (c) [5 marks] Prove that your algorithm is correct (feasible and optimal).

   As the head of the parade planning committee, you are responsible for ensuring that every decorated parade float has an appropriately sized truck to carry it.

   **Input:** Distinct parade float sizes $p_1, p_2, ..., p_n$ and distinct truck sizes $t_1, t_2, ..., t_n$.
   You can assume $t_1, ..., t_n$ are given in increasing order.
   **Output:** A permutation $\pi = \pi(1), \pi(2), \ldots, \pi(n)$ that minimizes $\sum_{k=1}^{n} (t_k - p_{\pi(k)})^2$.

   (a) [4 marks] Consider the following algorithm: for $i = 1..n$, match the truck with size $t_i$ to the unmatched parade float with the closest size $p_j$ to $t_i$. Prove that this algorithm is not optimal.
(b) [8 marks] Design a greedy algorithm for this problem and prove it is correct. 
*Hint: to prove optimality, it may be helpful to fix an arbitrary input, and consider the output of the greedy algorithm and the output of an optimal algorithm, and suppose they differ. Try to show that the “optimal” solution can be improved to obtain a contradiction.*

3. [12 marks] **Dynamic programming — Multipath Metropolis.**
Consider a city containing infinitely long horizontal streets with x-coordinates 1, 2, ..., n and infinitely long vertical streets with y-coordinates 1, 2, ..., m. Note that the city contains nm intersections forming a regular 2D grid. Currently, some of these intersections, denoted \((x_1, y_1), (x_2, y_2), ..., (x_k, y_k)\), are under construction and cannot be driven through. Your task is to determine how many different paths there are from the top left intersection (1, 1) to the bottom right intersection (n, m), if you can only drive the right (+x) and down (+y).

(a) [8 marks] Design an \(O(nm)\) dynamic programming algorithm to solve this problem. Write the recurrence for your solution, and provide pseudocode for your algorithm.

(b) [4 marks] Suppose you want to solve a new variant of this problem that asks how many different paths there are from one arbitrary intersection \((x, y)\) to another \((x', y')\) where \(x \leq x'\) and \(y \leq y'\). Describe how you could use your solution to part (a) as a black box to solve this new problem variant. (In other words, give a reduction from this new problem variant to the original problem.)

4. [16 marks] **Dynamic programming — Hungry Hungry Hippos.**
Two hippos, Alice and Bob, play a turn-based game involving a stack of pancakes. Each hippo’s goal is to eat a larger volume of pancake than the other hippo. Pancake volumes are measured in cubic inches. In each turn, a hippo can take one pancake from either the top or the bottom of the stack of pancakes, and eat it. Alice starts the game first, and the two hippos alternate turns until there are no pancakes left. Assume that Alice and Bob each play optimally. (That is, in each of Alice’s turns, she takes the action that will yield the best result, under the assumption that Bob will do the same in each of his turns.)

**Input:** A sequence of pancake volumes \(v_1, v_2, ..., v_n\) \((v_1\) is the top, \(v_n\) the bottom). 
**Output:** The total volume Alice will eat, assuming both hippos play optimally.

Design an \(O(n^2)\) dynamic programming algorithm to solve the problem. Write the recurrence(s) for your solution and provide pseudocode for your algorithm.

(Hint: Consider using two recurrences, defined in terms of each another, to simulate Alice and Bob taking alternating turns.)

5. [16 marks] **Practical programming problem**
This assignment includes an implementation question (C++ or Python), which will be submitted separately via Marmoset. See Piazza for further instructions. The deadline for this implementation question is the same as for the written assignment.