Introduction to Algorithms: 6.006
Massachusetts Institute of Technology September 27, 2016
Professors Erik Demaine, Debayan Gupta, and Ronitt Rubinfeld Problem Set 2

Problem Set 2

All parts are due on October 13, 2016 at 11:59PM. Please download the .zip archive for this problem set. Remember, your goal is to communicate. Full credit will be given only to a correct solution which is described clearly. Convoluted and obtuse descriptions might receive low marks, even when they are correct. Also, aim for concise solutions, as it will save you time spent on write-ups, and also help you conceptualize the key idea of the problem.

Part A

Problem 2-1. [10 points] **Back-solving recurrences**

(a) [2 points] If $T(n) = \Theta(n^2 \log n)$ is a solution to $T(n) = aT(n/2) + \Theta(n^2)$, where a is a *positive integer*, find all possible values of a.

(b) [2 points] If $T(n) = \Theta(n^2)$ is a solution to $T(n) = aT(n/3) + \Theta(n)$, where a is a *positive integer*, find all possible values of a.

(c) [2 points] If $T(n) = \Theta(n^2)$ is a solution to $T(n) = 4T(n/b) + \Theta(n^2)$, where b is a *positive real number*, find all possible values of b.

(d) [2 points] If $T(n) = \Theta(n^{6.006})$ is a solution to $T(n) = 5T(n/b) + \Theta(n^5)$, where b is a *positive real number*, find all possible values of b.

(e) [2 points] If $T(n) = \Theta(n^2)$ is a solution to $T(n) = 6T(n/6) + f(n)$, find one possible function f.

Problem 2-2. [20 points] **Sorting a Rectangle**

In this problem, you are given a 2D array of integers A that has n rows and m columns. (Assume m is *much* (e.g., exponentially) smaller than n.) Array A has one special property: the integers in every row of A and every column of A are non-decreasing. More formally, $A[i,j] \leq A[i,j+1]$ for every $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m-1$, and $A[i,j] \leq A[i+1,j]$ for every $i = 1, 2, \ldots, n-1$ and $j = 1, 2, \ldots, m$.

For example, one possible such 2D array is the following:

$$\begin{bmatrix} 1 & 2 & 4 & 8 \\ 3 & 5 & 5 & 10 \\ 4 & 6 & 19 & 30 \end{bmatrix}.$$

Describe and analyze an algorithm with running time $O(nm \lg m)$ that produces a sorted 1-D array of length nm that has exactly the same elements as A. For example, for the 2D array given above, the algorithm should return $[1, 2, 3, 4, 4, 5, 5, 6, 8, 10, 19, 30]$.

Problem 2-3. [25 points] **Sorting Venture Capital Offers for 6006LE**

The code you wrote for the hip new 6006LE search engine in the last problem set is attracting the interest of venture capitalists, who are lining up to bring you truckloads of venture capital dollars—literally!

You have a parking lot with parking spots numbered $1, 2, \ldots, n$, to which n venture capitalists drive money-filled trucks of sizes $a_1, a_2, \ldots, a_n$ (leaving no extra parking spaces). You'd like to sort the trucks by size (so you can work your way down the list of trucks later talking with the venture capitalists about their terms), but you can only see along a limited distance of the parking lot at any one time, so (1) you can only compare the sizes of two trucks at most k parking spots away from each other[1], and (2) you can only tell trucks at distance at most k from each other[1] to swap places.

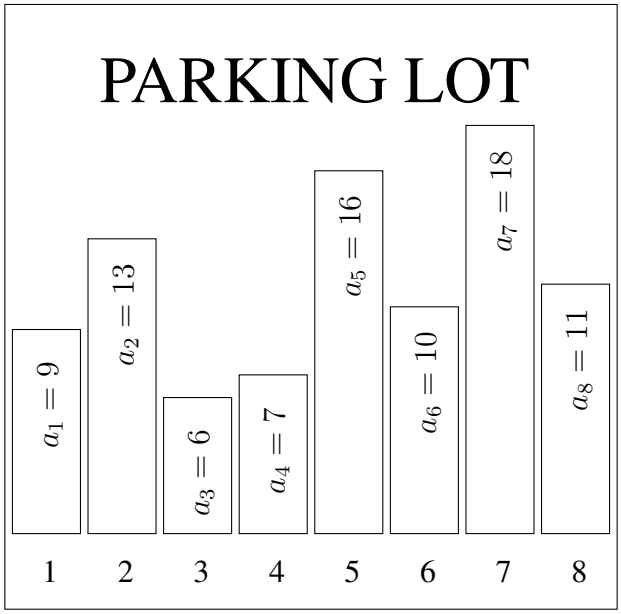

Figure 1: Bird's-eye view of a parking lot.

(a) [5 points] Design an algorithm that *switches* two arbitrary trucks (not necessarily at distance at most k) in $O(n/k)$ truck swaps (each of distance at most k), and returns any other trucks that were moved in the process to their original positions.

(b) [5 points] Design an algorithm that *compares* two arbitrary trucks (not necessarily at distance at most k) in $O(n/k)$ truck swaps (each of distance at most k), and returns any trucks that were moved in the process to their original positions.

[1]That is, only trucks in spots i and j where $|i - j| \leq k$

(c) [15 points] Describe an algorithm that sorts the trucks in $O((n^2 \log n)/k)$ truck swaps. (*Hint:* modify an in-place $\Theta(n \log n)$ sorting algorithm, and possibly use the previous two parts.)

(d) [10 points] **(Extra credit)** Describe an algorithm that sorts the trucks in $O((n^2 \log k)/k)$ truck swaps and comparisons.

Part B

Problem 2-4. [45 points] **F1 Kart Racing**

Bowser is participating in the F1 Kart Racing Finals that take place on a circular track of integer length L. There are N competitors in total, indexed $0, 1, \ldots, N-1$; Bowser is index 0. At time $t = 0$ (the start of the race), competitor i starts at integer position p_i ($0 \leq p_i < L$) with a kart that has an integer velocity of $v_i \geq 0$. Assume that all competitors start from distinct starting positions (i.e., $p_i \neq p_j$ for all $i \neq j$), that all karts have distinct velocities (i.e., $v_i \neq v_j$ for all $i \neq j$), and that karts start at full speed (unchanging velocity v_i). Thus, at time t, kart i will be at position $(p_i + t \cdot v_i) \bmod L$ (because the track is circular so positions 0 and L are the same).

The race defines a ranking of competitors as follows. The first competitor that is passed by another competitor from behind is taken out of the competition. When only one competitor is left in the competition, they are deemed the winner, receive eternal glory, and get first position in the final ranking. A competitor s is taken out of the race by another competitor f if and only if $v_f > v_s$ and neither s nor f gets taken out of the race before f catches s; the time when this will happen is given by the following function:

$$losetime(f, s) = \begin{cases} \dfrac{p_s - p_f}{v_f - v_s}, & \text{if } p_s > p_f, \\ \dfrac{(L - p_f) + p_s}{v_f - v_s}, & \text{if } p_s < p_f. \end{cases}$$

The rank of a competitor i is defined to be 1 if they win the competition (and their finish time is $+\infty$), or $k + 1$ where k is the number of competitors whose finish time is strictly larger than i's finish time.

Note that this ranking can differ from the order of the karts' velocities, because of the differing starting positions of karts and the circular track which encourages "lapping" of other karts.

We are going to design and implement an $O(N \log N)$ algorithm that computes Bowser's final rank. (This way, we can bet all of our 6006LE venture capital and vastly multiply our resources!)

(a) [10 points] As a warmup, come up with an $O(N^2)$ algorithm that finds Bowser's final rank. (*Hint:* Can you find when competitor i leaves the race in $O(N)$ time?)

(b) [3 points] We will avoid using floating-point computations (such as noninteger division), which are inexact, and only use exact fractions via the class `Fraction`. In order to do that, we need to be able to compare two fractions. Fill in the necessary code in the `compare` method.

(c) [2 points] Write the code for the function $losetime(f, s)$ defined above.

(d) [5 points] We need to maintain a data structure of the relative order of all competitors. This is to say that, for every competitor i, this data structure maintains the competitor behind i and the one in front of i. This data structure is implemented by two Python

dictionaries `behind` and `ahead`. Write the method `remove`(i), which removes competitor i from the data structure, and correctly maintains the `behind` and `ahead` links. This should work in $O(1)$ time.

One way to find the rank of all competitors is by simulating the race, and removing the competitors that are passed by in the order those passing events happen in time. For each competitor still in the race, consider the event of their passing the competitor *immediately* ahead (we do not look at the cases when the competitor ahead is faster). If we look at all these events (of which there are at most N, one per competitor), then the one that happens earliest tells us the competitor that leaves the race first. Then, we can remove that competitor from consideration, and continue similarly.

(e) [5 points] Suppose that competitor Charlie passes competitor Bowser, competitor Alice is ahead of both of them, and competitor Deborah is behind both of them. What events might need to be added to or removed from consideration? Your answer may depend on the relative speeds v_a, v_b, v_c, and v_d of Alice, Bowser, Charlie, and Deborah.

(f) [20 points] In the previous part, you saw that at every step of the simulation, you need to find the event with the minimum time, remove $O(1)$ events, and add $O(1)$ events. For all of this, we can use the heap data structure. We will also need the data structure from part (d), but putting everything together results in an $O(N \log N)$ algorithm. Implement this algorithm in the `rank` method. (*Hint:* You may find the Python module `heapq` helpful.)