Problem Solving Workshop #19 Tech Interviews and Competitive Programming Meetup

January 29, 2017 https://www.meetup.com/tech-interviews-and-competitive-programming/

Instructor: Eugene Yarovoi (can be contacted through the group Meetup page above under Organizers)

More practice questions: leetcode.com, glassdoor.com, geeksforgeeks.org
Books: Elements of Programming Interviews, Cracking the Coding Interview
Have questions you want answered? Contact the instructor, or ask on Quora. You can post questions and follow the instructor and other people who write about algorithms.

Try to find optimized solutions, and provide a time and space complexity analysis with every solution.

Problem #1, "Stock Analysis"

You have stock price data over a long period of time. Assume fixed sampling intervals (say 1 data point per second), so the data looks like an array of prices, e.g. [22.83, 22.84, 22.81, ...]. Sometimes, when there's no data for a particular point in time, the information will appear as 0, so data could look like [22.83, 0, 22.84, 22.81]. Zero is therefore interpreted as a special value (it is assumed it cannot be a stock price).

A financial software developer wants to preprocess the data in such a way that when they later ask repeated queries of the form "what fraction of datapoints were missing between two timestamps T_1 and T_2?" (T_1 and T_2 different in each query), they can find the answer without having to scan the whole dataset.

(i) **[Easy]** Give any solution that makes searching for the fraction of missing datapoints between two timestamps possibly happen faster than if the data were not preprocessed.
(ii) **[Easy-Medium]:** Make queries happen in O(1) time.
(iii) **[Medium] Follow-up:** Now suppose there's data for more than 1 stock over the same time period. The stocks are numbered 0 through S-1 if there are S stocks. Preprocess the data so you can answer queries of the form "what total fraction of datapoints is missing between timestamps T_1 and T_2 over all stocks numbered between S_1 and S_2?" (This problem could be tough if you don't see the trick, so move on to another problem if you get stuck for a long time.)

Problem #2, "Amazing Numbers"

In an array, define an amazing number as an integer whose value is less than or equal to its index. For example, in the array [2, 1, 0, 5], there are 2 amazing numbers because 2 > 0 (not an amazing number), 1 <= 1 (is an amazing number), 1 <= 2 (is an amazing number), and 5 > 3 (is not an amazing number).

You're given an array and you're allowed to rotate it (without loss of lgenerality, to the right) by any amount. You want to rotate the array in a way that maximizes the count of amazing numbers. For the

example of [2, 1, 0, 5], if you rotated it by two, you would get [0, 5, 2, 1], which results in 3 amazing numbers (the highest possible amount because 5 can never be amazing in an array of size 4). Output the amount of rotation needed to maximize the amazing number count, as well as the count itsef.

Example Input 1: 2, 1, 0, 5
Ouptut: rotation 2, count 3

Example Input 2: 1, 0, 0
Output: rotation 1, count 3
Explanation: rotating the array by 1 to [0, 1, 0] makes all the numbers be amazing. The solution with a rotation of 2 would have also worked -- in this case, you can output either.

Complexity Guidance: [very easy] $O(n^2)$, **[medium]** O(n log n) , **[medium-hard]** O(n)

Problem #3, "Parallel Palindromes"

Consider the problem of checking whether a given string is a palindrome, but the check should be performed ignoring non-letters and letter case. For example, "A man, a plan, a canal, Panama!" should still be considered a palindrome.

(i) **[Easy]** Give an **in-place** (O(1) extra space), O(n) algorithm for checking whether a given string is a palindrome under this deifnition of palindrome.

(ii) **[Medium]** Suppose now we have a very large string to check, and we have K threads available for this task. How would you make use of the multiple threads available to solve the problem in O(N / K) time? (It may be O(N / K + K), but you may assume K is a relatively small number and is small relative to N / K). **Your algorithm does not need to be in-place (extra space is allowed).**

(iii) **[Hard]** How would you solve this algorithm "roughly in-place" with K threads in the same O(N / K) time? "Roughly in-place" means that ***each thread*** should only use O(1) (or some minimal amount like O(log N) maybe) extra space. Essentially, the total amount of extra space used by all threads should be of a much smaller order of magnitude than the size of the input.