Problem Solving Workshop #41 Tech Interviews and Competitive Programming Meetup

January 21, 2019 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, geeksforgeeks.org
Books: Elements of Programming Interviews, Cracking the Coding Interview
Have questions you want answered? Contact the instructor on Meetup, 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, "Interviewing Candidates"

A company has 2N candidates they wish to interview. They have a NY office and an LA office, each of which can interview half of them, N candidates. For each candidate, the cost of flying them out to each of the two locations may be different. Given each candidate's cost of flying out to each office, determine where each candidate should be interviewed to minimize the total cost (if there's more than one optimal arrangement, return any one).

Example Input: [(Name = John, NY cost = 8, LA cost = 5), (Name = Larry, NY cost = 12, LA cost = 1), (Name = Susan, NY cost = 15, LA cost = 15), (Name = Rachel, NY cost = 7, LA cost = 6)]

Output: NY : [Susan, Rachel], LA: [John, Larry]

Explanation: with this arrangement, it costs 5 for John, 1 for Larry, 15 for Susan, and 7 for Rachel. That's a total cost of 28, which is the smalest possible.

(a) Solve
Follow-up (Medium-Hard, might want to skip and come back to): what if there are C candidates and N slots per office, but C > 2N? For each candidate, we have the option of not interviewing that candidate, which has 0 cost. We still want to interview N candidates at each of the two offices.

Problem #2, "Corrupted Sequence"

You had a neat list that contained the numbers [1, 2, 3, 4, ...]. Then, some of the entries were deleted, making the list look, for example, something like [2, 5, 7, 8, 9, 12, ...]. Given the list of remaining entries, and a parameter K, what is the K-th smallest (0-indexed) value that was deleted (K >= 0, and K < number of entries that were deleted).

Example Input: [2, 5, 7, 8, 9, 12], K = 6
Output: 13

Explanation: The entries missing from the original, in order are: [1, 3, 4, 6, 10, 11, 13, 14, ...]. The K-th index (0-indexed) has value 13.

(Easy) Solve it in O(N) time, where N is the size of the array.
(Medium) Solve it better than O(N).

Problem #3, "Task Parallelization" (Medium-Hard)

We have a queue of compute jobs. Each task has an associated amount of time that it would take to complete it. To maintain fairness, ***we want to run the jobs in the order they appear in the queue***, but since there's no dependency between them, we may run several jobs in parallel.

The jobs are to be run in a batch format: we take some number of jobs from the front of the queue (our choice how many), and dispatch them as a batch to an executor service, to be run in parallel. Since they run in parallel, the completion time of the entire batch is equal to the completion time of the longest-running job in the batch. The executor service has some limitations, so the entire batch has to complete before the executor service can accept another batch. The executor service has some integer value P, which represents the maximum number of jobs that it can accept per batch (we can send less than or equal to P jobs in any given batch).

(a) Given the list of running times of jobs and a ***parallelization capacity P***, how long will it take to complete all the jobs, with the optimal partitioning of the job queue into batches?

Example Input: [5, 15, 20, 3], P = 2
Output: 28
Explanation: We can dispatch 5 by itself (5 time), then [15, 20] (20 time), then 3 by itself (3 time). This is a total of 28 units of time.

(b) Now, if instead you're given the list of running times of jobs and a ***target time T*** in which you want to complete all the tasks, what is the necessary parallelization capacity P for the executor service?

Example Input: [5, 15, 20, 3], T = 30.
Output: minimum P = 2
Explanation: If we can have 2 per batch, we can dispatch 5 by itself (5 time), then [15, 20] (20 time), then 3 by itself (3 time). This is a total of 28 units of time, which fits in the budget. This isn't possible if P = 1, since then completing the jobs would take 5 + 15 + 20 + 3 time, which exceeds the budget, so P = 2 is required.

Problem #4, "Square Subsets"

You're given an array of small positive integers. Calculate the number of distinct non-empty subsets of this array (really sub-multi-sets, as the array can have duplicates), such that the product of all the numbers in the sub-multiset is a perfect square.

Example Input: [7, 7, 6, 10, 15, 7]
Output: 3
Explanation: the distinct subsets whose products are perfect squares are {7, 7, 6, 10, 15}, {6, 10, 15}, and {7, 7}. 7*7 = 49, a perfect square, and 6*10*15 = 900, which is a perfect square because 30*30 = 900. 7*7*6*10*15 is a perfect square too, considering it is the product of two other perfect squares ($a^2 * b^2 = (ab)^2$). Note that even though the two 7s could be sampled from different indexes, that doesn't affect the answer in this case, because we are only counting distinct subsets (and there's no way to form any new perfect-square subsets with the 3rd 7).

(a) (Medium-Hard) Solve this when the array can potentially be large, but all numbers are < 15.
(b) (Hard) Same but all numbers < 70.

Hint: consider under what conditions the product of some N numbers will be a perfect square. Namely, a number will be a perfect square if in its prime number factorization, all the distinct prime factors appear an even number of times. For example, the prime factorization $2^4 3^2$ is a perfect square because it can be grouped like $(2^2 3^1)^2$.

This is a slightly simplified version of https://codeforces.com/problemset/problem/895/C, which has an online judge. Be careful that their version is a little different (just a few minor extra steps).

Complexity Guidance

1. Main problem: O(N), follow-up: $O(CN^2)$
2. (a): given in problem, (b): O(log N)
3. $O(N^3)$ where N is number of jobs is pretty good. Then in increasing order of goodness: $O(NP^2)$, $O(N^2 \log N)$, O(N P log P)
4. Because the individual numbers have been constrained to have values < 15 or < 70, the values are not a parameter (it's possible that your algorithm may scale poorly with higher values). Your algorithm may therefore "technically" be O(N) where N is the number of values. But, your algorithm should be in practice be efficient enough that it can run in seconds for N = 1000.