Problem Solving Workshop #43 Tech Interviews and Competitive Programming Meetup
February 23, 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, hackerrank.com
Books: Cracking the Coding Interview, Elements of Programming Interviews
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, "Chain Lightning" (Easy-Medium)

In a video game, you have a cluster of N enemy minions, each of whose hit points are given in an array. You have a spell called "chain lightning". When cast, chain lightning strikes a minion, dealing D damage, and then bounces to a different minion.

Each bounce does K fewer hit points of damage than the previous, until the next bounce would do 0 or less damage, at which point the spell does not bounce to any more minions. In other words, the first enemy hit takes D damage, the next one D - K, the one after that D - 2K, and so on for as long as the damage is greater than zero. The spell cannot bounce to any enemies it has already hit, so each enemy can only be hit by it once, or not at all.

You're implementing the code for this spell, and you want it to automatically target so as to kill the maximum number of minions. A minion is killed by the spell if it bounces to the minion, and the minion's hit points are <= the damage the spell deals to the minion. Determine the order in which the minions should be hit to maximize the number of them that are killed. (If multiple sequences are equally good, pick any one.)

Example Input: minion_health = [35, 100, 60, 90], D = 100, K = 35
Output: indexes_of_hit_minions = [1, 2, 0]
Explanation: The 3 hits will be 100 damage, then 100-35 = 65 damage, then 65-35 = 30 damage. We use the 100 damage hit on the 100 health minion (at index 1, 0-indexed), then the 65 damage hit on the 60 health minion (at index 2). The third hit can't kill any additional minions, so it doesn't matter what you do with it (we arbitrarily picked to hit the minion at index 0). There are many other equally good solutions here, such as [3, 0, 1].

Minor follow-up: If we know the number of hits will be a small constant because K is large enough, but the number of minions can be large, does that change your solution to get a better time complexity?

Problem #2, "Median of Many" (Medium)

You're given M ***sorted*** arrays of 32-bit signed integers, each of size N. Your goal is to find the median of all the combined integers. In other words, if all the values were placed into a single list of size MN (keeping all duplicates), what would be the median? If there are an even number of elements, you can assume the median will be defined as the smaller of the two median elements.

Example Input: [[2, 5, 6, 7], [2, 4, 5, 10], [3, 12, 14, 17]]
Output: 5
Explanation: if all of these numbers were placed together into a sequence and then sorted, the resulting sequence would be [2, 2, 3, 4, 5, 5, 6, 7, 10, 12, 14, 17]. The two middle elements are 5 and 6, so the median is 5. (If the list had been of odd size, there would have been just one middle element, and that would be the median.)

Of course there is the obvious approach of merging all the elements into a single array, sorting them, and then trivially finding the median, but this has time complexity O(MN log(MN)). The goal is to find an approach much better than O(MN). You should be leveraging the fact that the arrays are sorted.

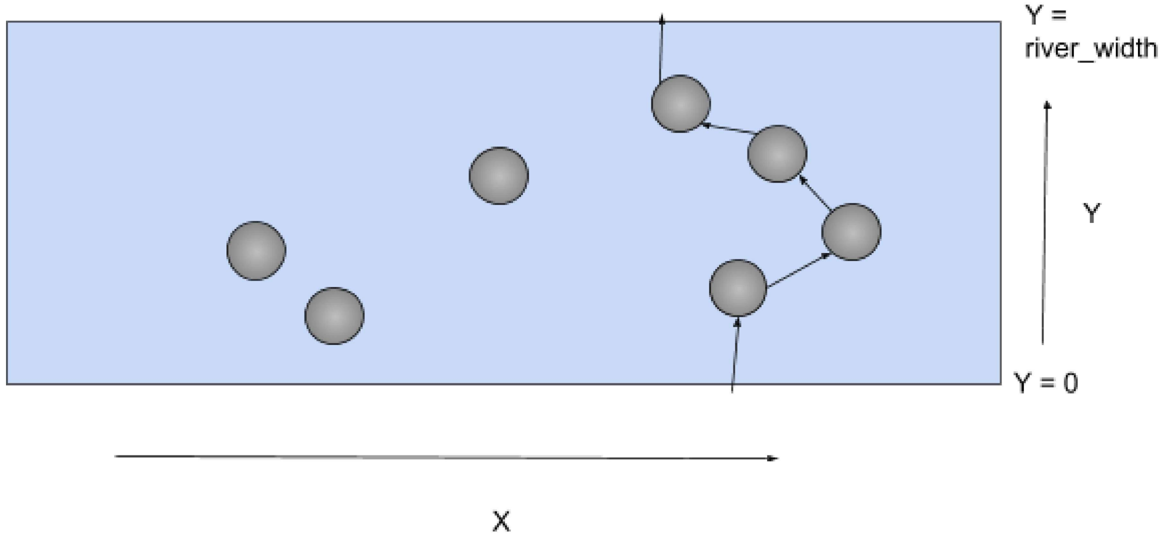
Problem #3, "Skipping Stones"

In a river, there are some stepping stones you can use to cross the river by jumping from stone to stone. You start at any point of your choice on the river's lower bank (the line Y = 0), and you need to get to any point on the upper bank (Y = river_width) by jumping from stone to stone until you make it across.

Assume the radius of each stone is negligible, and therefore we can treat each stone as a point having a precise (x, y) coordinate. You can jump at most 1 unit of distance per jump, and after each jump, you must land safely on a stone or river bank -- you can't land in the water! The jump distance limit applies both when jumping between stones, and between a stone and a river bank or vice versa. Distance is measured as Euclidean distance: distance = sqrt(x_delta^2 + y_delta^2).

(a) [Medium-Hard] Given a list of the stone positions, and the river_width, determine whether it's possible for you to cross the river.

Example Input: stones = [(1.5, 1), (2, 0.5), (2.5, 1.6), (4, 0.7), (4.6, 1.2), (4.2, 1.8), (3.5, 2)], river_width = 2.5
Output: Yes, possible.
Explanation: The situation is illustrated in the diagram below. The arrows represent the path by which you can cross the river. The first three stones in the input are the ones on the left, then the next four are the ones used to cross, in order from bottom to top (that they appear in that order in this example is a coincidence).



In time complexity analysis for this problem, you can assume that the stones are distributed in a way that looks fairly random over the rectangular region of space where the stones are present, X = Xmin...Xmax, Y = 0...river_width, where Xmin and Xmax are the stones with the lowest and highest x-coordinates. That is, the placement of the stones is not contrived in some special way. You can also assume that the number of stones is approximately C * (Xmax - Xmin) * river_width, for some reasonably-sized constant C. Note that this means that on average, a 1x1 square patch of the river will contain some constant number of stones.

(b) [Hard] Additionally, for each stone, you're given a probability between 0 and 1 that the stone will sink the moment you step on it. If you step on a stone and it happens to sink, you fail to cross the river. You successfully cross the river only if none of the stones you step on sink (the riverbanks never sink). Select a path through the river that minimizes your overall chance of failure, assuming each stone's probability of sinking is independent of all the others.

Complexity Goals

1. O(N log N) where N is number of minions. In the follow-up, we can solve in O(N + H log H) if there are H hits.
2. O(M log N log (MN)) is a great complexity. Something close to that is good too.
3. (a) can be solved in O(N) if there are N stones and the stated assumptions about the distribution of stones are met. (b) can be done in O(N log N) time.