Problem Solving Workshop #14 Tech Interviews and Competitive Programming Meetup
October 22, 2016 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.

Coding Problem

Given a string S, count the number of **distinct** pairs (2-tuples) of characters such that the first character occurs in S at least once before the second character.

In other words, the pair of characters the pair (a, b) should be counted if there are two indices (i, j) in the string where S[i] == a and S[j] == b and i < j. "Distinct" means you shouldn't double-count a pair if the same pair can be found in multiple places. Note that (b, a) is not the same pair as (a, b).

Example Input: S = "acbbc"
Output: 6
Explanation: We can form the character pairs (a, c), (a, b), (b, b), (b, c), (c, b), (c, c), 6 distinct ones in total. Note that:

- We cannot form, for example, (b, a) because there is no "a" that comes after any "b".
- We can form both (b, c) and (c, b) because some "c"s occur after "b"s and some before.
- The word "distinct" in the problem statement doesn't mean the chars in the pair should be distinct ((b, b) is a valid pair); it means count each pair with the same chars only once.

Complexity guidance: analyze the complexity of your algorithm in terms of N, the length of S, and D, the number of distinct chars in N (D is potentially much smaller than N). Some possible time complexities in increasing order of goodness: O(N^2) **[Easy]** -> O(ND) **[Easy-Medium]** -> O(N + D^2) **[Medium]** -> O(N + D log D) **[Medium-Hard]** -> O(N), optimal **[Medium-Hard]**.

Algorithm Problem

Given an array A of integers, consider the problem of finding the **contiguous** subarray with the maximum sum.

Example Input: A = [4, -3, 2, -10, **3, 2 , -1, 2**, -1]
Output: (4, 7)
Explanation: These are the start and end indices of [**3, 2, -1, 2**] in the input array. That subarray has the highest sum, 6, out of any contiguous subarray. Note, from this example, that you can't assume the maximum sum subarray will contain only positive numbers, since you may need to include some negative numbers to extend your "reach" and get more positive numbers.

(i) **[Easy]** Give any valid solution to this problem. Give a time and space complexity analysis.
(ii) **[Medium]** Find an optimized solution that has complexity O(N log N) or better, where N is the length of the array.
(iii) **[Medium-Hard]** Suppose you have K processors available. (Forget about the fact that probably on most machines the O(N) algorithm is I/O-bound.) Parallelize your algorithm, so that, at least when N is large and K is relatively small, you improve the running time of your solution from part (ii) by a factor of (at least roughly) K.
(iv) **[Hard]** Suppose you now have unlimited processors and threads. Your ability to use parallelism is limited only by the fact that it takes some (assume O(1)) amount of time to start a thread, and pass arguments to it. Can you improve your previous solution?