Homework 4: Divide and Conquer

Be sure to read the grading guidelines and style guidelines. Especially to see the suggested format for describing algorithms.

We sometimes describe how long are justifications or proofs are. These lengths are intended to help you estimate how much detail we're expecting, you should not take those estimates as hard length-limitations.

Our solutions for any individual problem will fit in approximately one page or less. **Beginning with this homework, if you submit an answer substantially longer than 2 pages, the TAs are allowed to stop reading at the end of page 2.**

You are allowed (and encouraged!!) to collaborate with each other. Brainstorming is much easier to do in a group than alone! But you must follow the collaboration policy (which includes needing to write your submission on your own).

You will submit to gradescope; we will have a different box for each problem, so please give yourself extra time to submit.

1. Setting the baseline [10 points]

Later on this homework, you will use the following library function

`ShortestCycleThrough`
Input: A directed graph G, with non-negative edge weights. A vertex v of G.
Output: The length of the shortest cycle that goes through v
Running Time: $\mathcal{O}(E \log V + V \log V)$

In this problem, you'll design the function. This question will **not** use divide and counquer.

(a) Give pseudocode that will return the length of the shortest cycle through v (just return the length, not the cycle itself). Our pseudocode is about 6 lines.

(b) Give a brief proof of correctness (our proof is about 4-5 sentences).

2. Planar D+C

In this problem, you'll design a divide and conquer algorithm on planar graphs. A graph is **planar** if it can be drawn on a piece of paper without any edges crossing. For example, trees and grids are planar graphs, but a clique on 5 vertices is not.

You will use the following library to write a divide-and conquer algorithm

`PlanarSeparator`
Input: A planar graph $G = (V, E)$
Output: A set S of vertices such that:

- $|S| \leq \mathcal{O}\left(\sqrt{|V|}\right)$
- $G - S$ is disconnected, and every weakly connected component[a] of $G - S$ has at most $\frac{2|V|}{3}$ vertices.

Running Time: $\mathcal{O}(V)$

[a] A *weakly connected component* of a directed graph is exactly a connected component of the undirected graph you get by ignoring the directions on the edges.

where $G - S$ refers to the graph created by deleting S and every edge with at least one endpoint in S. The intuition for the seperator algorithm is that it gives you a relatively small set of vertices, which (when removed) separate the graph into relatively balanced pieces.

For some examples:

- In a balanced tree, the root node (just by itself) is a separator
- In a grid, a column or row "near the middle" is a separator

You may use the following facts without proving them:

- If G is planar, then $G - S$ is planar for every set S.
- If G is planar, then it has $\mathcal{O}(V)$ edges. In analyzing running time in planar graphs, E's are always replaced with V's (this convention sometimes simplifies the running time).

Now, our problem: given a directed planar graph with non-negative edge weights, we're looking for the shortest cycle in the graph (measuring by sum of the edge weights).

(a) Give a 1-2 sentence summary of the main idea of your algorithm (this helps us when grading).

(b) Describe your algorithm. You may use the `PlanarSeparator` algorithmm as well as the `ShortestCycleThrough` algorithm from problem 1. You may return just the **length** of the cycle rather than the cycle itself.

(c) Give a 1-3 sentence summary of your proof (this helps us when grading and should help you formulate your thoughts before jumping into the proof).

(d) Prove your algorithm correct.

(e) The running time analysis is trickier than usual here. Give a recurrence that describes the worst-case behavior of `PlanarShortCycles` and state the final big-$\mathcal{O}$. You should not show significant algebra (if you need it), but you should include a few sentences to justify why your analysis is correct.
For this problem, we will give full credit to any analysis that is (1) correct, (2) shows the running time is better than $\mathcal{O}\left(V^2\right)$ (even if the analysis doesn't give the tight big-$\mathcal{O}$)

3. Programming

In this problem you'll write real java code! You will submit to gradescope where it will be autograded.

You have two arrays: $A[], B[]$ both are sorted in increasing order, but they can have very different lengths. Let m be the length of $A[]$ and n be the length of $B[]$. Each element of $A[]$ and $B[]$ is unique across both arrays. Design an algorithm that, given input k, finds the k^{th} smallest element combined between the two lists (i.e., the one which would be at index k, if you combined and sorted the arrays, and if the combined array is one-indexed).

Your algorithm should run in time "poly-logarithmic" in n, m: that means that log factors (like $\log(m)$ or $\log^3(m) \cdot \log^4(n)$) are fine, but expressions with variables raised to constants (like m or $\sqrt{n}$) are not. There is a divide and conquer algorithm that runs in $\Theta(\log(m) + \log(n))$.

A few details:

- The starter file with a method stub is on the course webpage. Don't change the name of the file (gradescope will look to compile an uploaded file of the same name) and don't change the method name or parameters
- Writing additional functions is allowed (and probably a good idea).
- You should not need to import any additional java packages.
- There are stress-tests provided on gradescope. To meet the efficiency requirements you must:
  - Pass all the stress tests
  - Have a program which theoretically meets the poly-logarithmic condition (we will check for this by hand and adjust scores if you passed the stress-tests but did not meet the theoretical requirement).

4. MinSubarrayProduct

Given an array A containing n non-zero integers, find the minimum **product** of a subarray (you only need to return the value of the minimum product).

We define the empty product (i.e., the product of an empty subarray) to be 1.

(a) Give a 1-2 sentence summary of the main idea of your algorithm (this helps us when grading).

(b) Give pseudocode (and/or English) to describe a divide and conquer algorithm for this problem. You **must** use divide and conquer (see the hints for that that means).

(c) Give a 1-3 sentence summary of your proof (this helps us when grading and should help you formulate your thoughts before jumping into the proof).

(d) Prove your algorithm is correct (you almost certainly want induction). In your inductive step, you may have points where there are two similar cases. You may write "the other case is similar" and "the proof of the other statement is similar." to avoid repeating yourself. You must still be clear on what statements you are showing and which cases are being skipped.

(e) Give the big-$\mathcal{O}$ running time for your algorithm. Briefly (3-4 sentences) justify your response. Your response must include a recurrence, but the arithmetic to solve the recurrence need not be included.

Hint: You **must** use a divide & conquer algorithm for this problem. There may be other (faster!) algorithms – you must use divide & conquer. If you have any question about whether your solution qualifies as D&C, ask in office hours. In general

- If you're working on a portion of the array, and you make a recursive call where the parameters change by 1 or 2, you're probably doing dynamic programming, not divide and conquer (D&C would probably involve breaking the array in half).
- If you're not using recursion, you're not doing divide and conquer.
- If you make recursive calls, but don't use their output, you're probably not doing divide and conquer.

Hint: Negative numbers make this a bit more complicated than the sum algorithm from section! If you never say "a negative times a positive is a negative" in your explanation to yourself of your algorithm, you probably don't have the correct algorithm!!