

15-451/651 Algorithm Design & Analysis, Fall 2026

Homework #4

Homework #4, Due: Wednesday, September 30th at 11:59 pm

This homework a short-answer problem followed by two written problems. The short-answer problems are graded on correctness only (we reserve the right to give a point penalty for providing a long answer to a short answer question). You **should not** provide long-winded justification or proofs unless explicitly asked for. Long-answer problems require you to justify your answers and provide proofs unless stated otherwise. On this homework, there is a SOLO problem and some GROUP problems. You may work in a group of up to 3 people on the GROUP problems. **In all cases the write-up you submit must be written by you alone. You may not share your written solutions with each other. You can discuss the problem as a group, but then must write up your final solutions independently in your own words.**

Submission: Solutions to the problems should be submitted as a PDF file using gradescope, with the answer to each problem starting on a new page. Each problem has a specified page limit that you should stick to. Your solutions **must** be typeset, **not handwritten**. You should use US letter-size paper, a font size no smaller than 10pt, margins no smaller than 1in. 10 points on this homework will be allocated to correctly following the style guidelines (see the Written HWs section of the [course policies](#)).

0 **Collaboration statement** Fill out the [collaboration form](#) to disclose your group members for this homework. If you worked solo, you must still fill out the form. Note that you will have to fill out this form separately for every written homework.

1. (SOLO, 15 pts) **Short-answer problems** (Limit: 1 page)

- (a) **Amortized Assortment** Suppose that the state of some data structure can be represented by a tuple of non-negative integers. Suppose the initial state of the data structure is such that the values are all zero. On that data structure, there exist various operations, which change the state of the data structure and incur some cost while doing so. For each set of operations below, give a potential function such that the amortized cost of every operation is $O(1)$.

	Operation	Before	After	Cost	
i.	(i)	$\langle x, y \rangle$	$\langle x + 1, y \rangle$	1	$\Phi(\langle x, y \rangle) = ?$
	(ii)	$\langle x, y \rangle$	$\langle 0, x \rangle$	$3x$	
	Operation	Before	After	Cost	
ii.	(i)	$\langle x, y \rangle$	$\langle x + 1, y \rangle$	1	$\Phi(\langle x, y \rangle) = ?$
	(ii)	$\langle x, y \rangle$	$\langle x, y + 1 \rangle$	1	
	(iii)	$\langle x, 2x \rangle$	$\langle 2x, 2x \rangle$	$2x$	
	(vi)	$\langle 2y, y \rangle$	$\langle 2y, 2y \rangle$	$2y$	

	Operation	Before	After	Cost	
iii.	(i)	$\langle x, y, z \rangle$	$\langle x + 1, y, z \rangle$	1	$\Phi(\langle x, y, z \rangle) = ?$
	(ii)	$\langle x, y, z \rangle$	$\langle 0, x + y, z \rangle$	x	
	(iii)	$\langle x, y, z \rangle$	$\langle x, 0, y + z \rangle$	y	

2. (GROUP, 25 pts) **Increasing Summed Arrays** (Limit: 1 page)

You are given an input array of n integers $a_1, \dots, a_n$, each of which can be positive, negative, or zero. You would like to form a new array $b_1, \dots, b_r$ by summing contiguous disjoint subsets of a . More precisely, $b_1, \dots, b_r$ is a **summed array** if

$$b_1 = \sum_{i=1}^{k_1} a_i, \quad b_2 = \sum_{i=k_1+1}^{k_2} a_i, \quad \dots \quad b_r = \sum_{i=k_{r-1}+1}^n a_i$$

where k_1 is the index of the most rightmost item of a that forms b_1 . and k_2 is the index of the rightmost item of a that forms b_2 , and so on. For example, given the array $a = [7, 3, 10, 9, 3, -5, 21, 2, -1, 7]$, one possible summed array is $b = [10, 19, 27]$.

$$\underbrace{7, 3}_{10}, \underbrace{10, 9, 3}_{19}, \underbrace{-5, 21, 2, -1, 7}_{27}$$

Another valid summed array is $[20, 9, -2, 29]$.

$$\underbrace{7, 3, 10}_{20}, \underbrace{9}_9, \underbrace{3, -5}_{-2}, \underbrace{21, 2, -1, 7}_{29}$$

Given $a_1, \dots, a_n$, give an efficient dynamic programming algorithm to output the length of a longest-possible summed array that is strictly increasing, i.e., that $b_1 < b_2 < b_3 < \dots < b_r$, and for which r is as large as possible. Your algorithm should run in $O(n^3)$ time.

Once again, you do not need to provide pseudocode, it is sufficient to define your set of subproblems and give a recurrence relation in terms of those subproblems.

Example 1 if $a_i = i$ for $i = 1, \dots, n$, then the solution is $r = n$ and $b_i = a_i$ for each i . This is because the input array is already sorted and so putting $b_i = a_i$ for each i is the only solution with the maximal value of $r = n$.

Example 2 if $a_i = 0$ for all i , then the solution is $r = 1$, since any b_j is necessarily equal to 0, so there can only be one item in the output array, as the output array must be strictly increasing.

Example 3 if $a = [-100, 100, 1, 2]$, then the solution is $r = 3$ where $b = [0, 1, 2]$. Here, $b_1 = a_1 + a_2 = -100 + 100$, $b_2 = a_3$, and $b_3 = a_4$.

3. (GROUP, 25 pts) **Pittsburgh Bridge Enumeration Authority** (Limit: 2 pages)

We all know that Pittsburgh has a lot of bridges, some say the most bridges in the world, though this point draws a lot of contention because it depends how exactly you define a bridge. In this problem, we are going to settle this argument once and for all, except with a definition of bridge that might be a little bit different from the ones everyone else is talking about... the graph-theoretic definition of course!

In graph theory, a bridge is an edge in a graph that, if removed, would increase the number of connected components in the graph. In other words, an edge is a bridge if there exists a pair of vertices u and v such that every path between u and v contains that edge.

Let's say Pittsburgh is a city with n locations, connected by undirected roads. Each road connects some pair of locations u and v . To settle the age old debate, we are going to build a data structure that tells us how many bridges Pittsburgh has. Unfortunately, we do not have an up-to-date map of the city, so we only know $n - 1$ of the roads, such that, coincidentally, every pair of locations is connected by a unique path (i.e., the roads form a spanning tree). The plan is to add in the remaining roads one at a time as we find them.

Since everyone is so excited to find out the answer, they may ask us how many bridges Pittsburgh has at any point in time while we are adding the roads, and we need to be able to answer immediately with the currently correct number given the roads we have seen so far. More formally, we want to build a data structure that supports the following pair of operations. The data structure initially contains the given set of $n - 1$ roads which form a spanning tree.

- **AddRoad**(u, v): Add a road between u and v to the city.
- **NumBridges**(): Return the number of bridges in the current representation of the city (i.e., given the roads added up to this point).

Note that at the very beginning, every one of the $n - 1$ roads is a bridge, so the initial answer is $n - 1$, and the number will only decrease as more roads are added. The goal is to support **AddRoad**(u, v) in $O(\alpha(n))$ amortized time, and **NumBridges**() in constant worst-case time (your algorithm should explicitly maintain the current number of bridges, which decreases or stays the same for each **AddRoad**).

- (a) Describe the data structure(s) that your algorithm will maintain, and describe purpose of each of them, i.e., clearly explain what each of them is being used to represent. (*Hint: You should make use of the Union-Find data structure from class*).
- (b) Describe how your algorithm handles the **AddRoad** operation and maintains the current number of bridges.
- (c) Prove that your algorithm handles m **AddRoads** in $O((n + m)\alpha(n))$ time¹².

¹You may cite without proof the fact that Union-Find with path compression and union-by-size performs Union and Find in $O(\alpha(n))$ time

²You can think of the $O(n\alpha(n))$ term as preprocessing work, so the $O(m\alpha(n))$ term corresponds to an amortized cost of $O(\alpha(n))$ per **AddRoad**