CS 270 (Fall 2025) — Assignment 1

Due: Friday, September 12, by 11:59pm

General Instructions. The following assignment is meant to be challenging, and we anticipate that it will take most of you at least 10–15 hours to complete. Please provide a formal mathematical proof for all your claims, and present runtime guarantees for your algorithms using asymptotic (big-$O/\Omega/\Theta$) notation, unless stated otherwise. You may assume that all basic arithmetic operations (multiplication, subtraction, division, comparison, etc.) take constant time.

Submission. Homework submission will be through the Gradescope system. Instructions and links have been provided through the course website and Piazza. The only accepted format is PDF. For this homework and Homework 2, we will still allow photographs/scans of handwritten solutions, but if they are illegible, you may lose points. Starting with Homework 3, only typed solutions will be accepted, and we highly recommend producing your solutions using LaTeX (the text markup language we are also using for this assignment).

(1) You are given an unsorted array $A = (a_1, \dots, a_n)$ of n numbers. We say that a pair (a_i, a_j) is an inverse pair if $i < j$ and $a_i > a_j$. For example, if $A = (3, 4, 1, 2, 5)$, then $(3, 1)$ is an inverse pair. In total, there are 4 inverse pairs in this example: $(3, 1), (4, 1), (3, 2), (4, 2)$.

Your goal is to count how many inverse pairs are in A. Please provide an $O(n \log n)$-time algorithm with pseudocode, a running-time analysis, and a correctness proof. [Hint: think about the MergeSort.]

(2) Given integers a and n, and a prime p, could you design an algorithm to compute $a^n \bmod p$? For example, if $a = 7$, $n = 10$, and $p = 13$, the output should be 4. Please provide the pseudocode, a running-time analysis, and a correctness proof.

You may assume that p is a constant; therefore, the multiplication of two integers smaller than p takes $O(1)$ time.

(3) We defined the lexicographic order in the lecture. For any strings A and B, we use $A <_L B$ to denote that A is smaller than B in lexicographic order. Please prove that, for any strings A, B, and C, if $A <_L B$ and $B <_L C$, then $A <_L C$.

(4) We have studied the suffix array in the lecture. For simplicity, we now assume that $\Sigma = \{0, 1\}$, i.e., we only consider binary strings. As we also mentioned in the lecture, for any string S of length n, it has $(n + 1)$ suffixes.

Now we assume the following nice uniform property of S:

- For any string P of length k, the number of suffixes that start with P is upper bounded by $n/1.2^k$, i.e., P can not be the prefix of more than $n/1.2^k$ suffixes of S.

Now assume that $n = 2^{64}$ and you have 64 GB to store the lookup table. How many memory accesses do you need to determine whether A is a substring of S? Here we assume that you already have the sorted array and the lookup table, and you only need one memory query to compare whether A is a prefix of any suffix.