COS 330: Great Ideas in Theoretical Computer Science

Fall 2026

Problem Set 0

Module: Introduction

PSet 0 is a warmup, and works a little differently from the rest of the PSets:

- Unlike later PSets, this one **doesn't follow the usual Problem 1 / Problem 2 / Problem 3 structure** – it's a loose collection of smaller problems meant to help you dust off background material (proofs, probability, etc.) before the course ramps up.
- **It is not graded, and there's nothing to submit** – the only goal is to help you self-assess where you're at and get back into the rhythm of problem solving.
- **Fully open**: use any resources you like (collaboration, AI, office hours, etc.) to help you complete it. But our recommendation is to try to solve the problems with no help first, and ask for clarification only if you get stuck.
- We still suggest taking a serious stab at each problem and then writing up your solution carefully, as if you were submitting it for credit. This is the best way to get the benefit of the exercise.

Problem 1 : Proofs Practice

Let $a = [a_1, a_2, \ldots, a_n]$ be a list of n distinct numbers. An inversion of a is a pair of indices $i < j$ such that $a_i > a_j$, and we write $\text{inv}(a)$ for the total number of inversions of a. An inversion is adjacent if $j = i + 1$. For example, the list $[3, 1, 2]$ has exactly two inversions – the pairs $(1, 2)$, since $3 = a_1 > a_2 = 1$, and $(1, 3)$, since $3 = a_1 > a_3 = 2$ – of which only $(1, 2)$ is adjacent.

Note that, directly from the definition, a is sorted in increasing order (that is, $a_1 < a_2 < \cdots < a_n$) if and only if $\text{inv}(a) = 0$.

(a) Prove that if a has no adjacent inversions, then a has no inversions at all.

(b) Suppose $a_i > a_{i+1}$ for some i, and let b be the list obtained from a by swapping the entries at positions i and $i + 1$. Prove that $\text{inv}(b) = \text{inv}(a) - 1$.

Proving the "Obvious"

Parts (a) and (b) are statements you would probably believe on sight, and that's the point: the entire exercise is producing a write-up with no gaps in it.

In part (b), for example, most of the work is in the pairs of positions you expect to be unaffected by the swap. "All the other pairs are clearly unchanged" is exactly the kind of sentence that could hide an error (and that would cost you presentation points on a real PSet).

A good habit is to make sure that every statement you write is precise. Writing generic ambiguous statements can make it easier to overlook a case (and it's actually a common failure mode for AI assistants, too). Even if the statement is correct, it can be unjustified. Remember that correct $\neq$ justified.

(c) Consider the following process, starting from any list a: while the current list has at least one adjacent inversion, pick one (any one you like) and swap those two entries. Prove that the process always halts, that the list is sorted when it does, and that the number of swaps performed is exactly $\text{inv}(a)$ – no matter which adjacent inversion is picked at each step. Conclude that the process never performs more than $\binom{n}{2}$ swaps, and give a list for which it performs exactly $\binom{n}{2}$ swaps.

Problem Solving Tips

This problem actually asks you to prove four related things, building on the previous parts. It's a good idea to break it down into pieces and tackle them one at a time, rather than trying to prove everything at once.

Problem 2 : Algorithms Practice

Let $a = [a_1, a_2, \ldots, a_n]$ be a list of n integers. A peak is an index $i \in \{2, \ldots, n-1\}$ such that $a_i \geq a_{i-1}$ and $a_i \geq a_{i+1}$.

(a) Prove that if $n > 2$, $a_1 < a_2$, and $a_n < a_{n-1}$, then a has at least one peak.

(b) You are given a list a of n ($n > 2$) integers such that $a_1 < a_2$ and $a_n < a_{n-1}$. Describe an $O(\log n)$-time algorithm to find a peak of a.

Problem 3 : Probability Practice

(a) Let X be a random variable such that there exists a single number c such that $\Pr[X = i] = \frac{c}{i^2}$ for all integers $i \geq 1$. Find the unique c such that X is indeed a well-defined random variable, and then compute $\mathbb{E}[X]$.

Aligning Expectations

You might need to find the value of some infinite series to solve this problem. You don't have to derive the value of the series yourself. Asking an AI assistant (or another online tool) to compute an infinite series for you is entirely appropriate.

(b) Let $X \sim \text{Unif}[a, b]$ be a uniform random variable on the interval $[a, b]$. That is, X is equally likely to be any real number between a and b. Find $\mathbb{E}[X]$.

You may use without proof the fact that the PDF of a uniform random variable on $[a, b]$ is $\frac{1}{b-a}$ on the interval $[a, b]$ and 0 otherwise. You may also use without proof the fact that the CDF of a uniform random variable on $[a, b]$ is 0 for $x < a$, 1 for $x > b$, and $\frac{x-a}{b-a}$ on $[a, b]$.

(c) A p-biased coin is one that lands heads with probability p, independently on every flip. Flip a p-biased coin n times. Each time the coin is Tails, throw it away. Each time the coin is heads, keep it independently with probability q (and throw it away otherwise). Let Y be the number of heads that are kept. Find $\mathbb{E}[Y]$.

Problem 4 : Graphs Practice

Let $G = (V, E)$ be an undirected graph with n vertices and m edges. A cut is a partition of V into two sets S and $\bar{S} = V \setminus S$ (either of which may be empty). An edge crosses the cut if exactly one of its endpoints lies in S, and we write $c(S)$ for the number of edges that cross the cut.

Fix a cut S and a vertex v. Call an edge incident to v internal if its other endpoint is on the same side of the cut as v, and crossing otherwise. Write s_v for the number of internal edges at v and x_v for the number of crossing edges at v, so that $s_v + x_v = \deg(v)$.

Aligning Expectations

The only definitions you need to know for this problem are the basic definitions of a graph (vertices, edges, degree). All the other definitions are given in the problem statement. We will do this often, even with things we will cover in class. You will eventually internalize a lot of definitions that will be recurring throughout the course, but the goal of the course isn't for you to memorize any definitions. Instead, you will be learning how to read, understand, and apply definitions in context. This is a skill that will be useful for the rest of your life, not the actual definitions themselves.

(a) Let S' be the cut obtained from S by moving v to the other side (and leaving every other vertex where it is). Prove that $c(S') = c(S) + s_v - x_v$. In particular, if $s_v > x_v$, then moving v strictly increases the number of crossing edges.

(b) Consider the following procedure: start from an arbitrary cut, and while some vertex v satisfies $s_v > x_v$, move that v to the other side of the cut. Prove that the procedure halts after at most m moves.

Problem Solving Tips

The shape of this argument is worth internalizing, because it comes up all over algorithm design: repeatedly make a small local improvement, and separately argue that you cannot improve forever.

Two things to notice. First, part (b) doesn't need to know anything about what the cut

looks like – only that some quantity strictly increases at every step and can never exceed a fixed bound. Second, part (c) will extract a global guarantee (half of all the edges cross) from a purely local condition (no single vertex wants to move). Whenever a problem asks you to show that some good object exists, "start anywhere, and keep improving until you can't" is a reasonable first thing to try.

(c) Prove that the cut S produced when the procedure halts satisfies $c(S) \geq m/2$. Conclude that every graph contains a bipartite subgraph with at least half of its edges.

Problem 5 : Algorithms Challenge

Optional – Challenge Problem

Unlike the rest of PSet 0, this problem is optional. It's harder than the others, and does require a real algorithmic idea rather than just careful execution. Only attempt it if you want to push yourself beyond the core warmup material – there's no expectation that you do.

The median of a list of n numbers (assume n is odd) is the $\lceil n/2 \rceil$-th smallest entry.[1] For example, the median of the list $[3, 1, 4, 1, 5]$ is 3.

Suppose you are given a list L of n numbers and you want to find a contiguous sublist of L with exactly k elements whose median is as large as possible. A contiguous sublist of L with k elements is a sublist formed by choosing a consecutive sequence of k elements from L. Assume k is always odd.

For example, if $L = [3, 1, 4, 1, 5]$ and $k = 3$, then the contiguous sublist $[4, 1, 5]$ has median 4, which is the largest possible median of any contiguous sublist of L.

(a) Suppose you are given the following simplified problem: given a list L of n numbers and a number x, decide whether there exists a contiguous sublist of L with k elements whose median is at least x. Describe an $O(n)$-time algorithm for this problem.

(b) Using your algorithm from part (a), describe an $O(n \log n)$-time algorithm to find a contiguous sublist of L with k elements whose median is as large as possible. (Hint: You might find it helpful to design a $O(n \log(\max(L) - \min(L)))$-time algorithm first.)

Problem Solving Tips

One trick to approaching a question like this is to ask yourself "why is this problem interesting?", or in other words, "why isn't the solution obvious?". We are asked to design an $O(n \log n)$-time algorithm

A good first step is to ask "what is a brute force solution that requires minimal creativity?" In this case, that would exhaust over $n-k$ possible contiguous sublists and compute the median of a list of size k from scratch. That would take total time $O(nk)$.

[1] That is, if you sorted the list, the entry at index $\lceil n/2 \rceil$.

What do I learn from this? First, if this is an exam, I would definitely immediately write down "If $k = O(\log n)$, then the following brute force solution runs in time $O(kn) = O(n \log n)$ [and include a complete/correct analysis here]," and get my concrete partial progress. Second, I learn that the hard case is when k is large.

This may or may not ultimately lead me directly to a solution. For example, my instinct from here was actually to try and re-use information from one median computation for the next one, but I soon realized it's a reasonably advanced data structures problem to get right, but that's OK! Even though I threw all this work out and pivoted to the outline above instead, I got more confident/comfortable with the problem along the way.