CS 473 (Spring 2025)
Homework 1 (due Feb 6 Thu 10am)

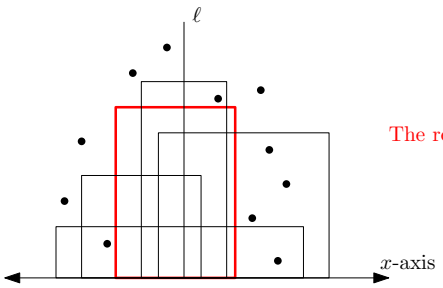
Instructions: Carefully read `http://courses.grainger.illinois.edu/cs473/sp2025/policies.html#hw` and `http://courses.grainger.illinois.edu/cs473/sp2025/integrity.html`. In particular, note the following:

- **Groups of up to three people may submit joint solutions.** Each problem should be submitted by exactly one person, and the beginning of the homework should clearly state the Gradescope names and email addresses of each group member. In addition, whoever submits the homework must tell Gradescope who their other group members are.
- **Submit your solutions electronically on the course Gradescope site as PDF files.** Submit a separate PDF file for each numbered problem. If you plan to typeset your solutions (which we highly recommend), you may use the LaTeX solution template on the course web site. If you must submit scanned handwritten solutions, we recommend that you use a black pen on blank white paper and a high-quality scanner app (or an actual scanner, not just a phone camera).
- You may use any source at your disposal—paper, electronic, or human—but you ***must* cite *every* source that you use, and you *must* write everything yourself in your own words** (see the course web pages).
- Your solution will be judged not only for correctness but also for clarity and style (again, see the course web pages).

Problem 1.1: A rectangle r is called *grounded* if the bottom edge of r lies on the x-axis (thus, a grounded rectangle can be specified by 2 x-coordinates and 1 y-coordinate).

Given a set R of grounded rectangles and a set P of points in 2D with $n = |R| + |P|$, we want to find an *empty* rectangle $r \in R$, i.e., a rectangle that does not have any point of P inside, if report that none exists.

(a) (50 pts) Consider the special case when all rectangles of R intersect a given vertical line ℓ. Design and analyze an $O(n \log n)$-time algorithm to solve this special case. [Hint: don't use divide-and-conquer here; solve the problem directly by examining the points and rectangles in increasing y-order. You may assume no two points have the same x- or y-coordinates.]



(b) (50 pts) Design and analyze an efficient algorithm to solve the original problem (without the restriction about ℓ), by using divide-and-conquer (and using part (a) as a subroutine).

Problem 1.2: Given three sets of integers A, B, and C with a total size $|A| + |B| + |C| = n$, we want to decide whether there exist elements $a \in A$, $b \in B$, and $c \in C$ such that $c = a + b$. It is not difficult to obtain a near $O(n^2)$ time algorithm for this problem. Here, we will explore subquadratic algorithms for some interesting special cases.

(a) (20 pts) Give an algorithm that runs in $O(|A||B| \log n)$ time, assuming that C is stored in a sorted array.

(b) (40 pts) For the case when $A, B, C \subset \{1, \ldots, n^{1.5}\}$, give an algorithm that runs in $O(n^{1.5} \log n)$ time, by using polynomial multiplication or convolution (on vectors of length $n^{1.5}$).

(c) (40 pts) For the case when $C \subset \{1, \ldots, n^{1.5}\}$ and A and B are arbitrary sets of integers (with no restrictions on the range), give an algorithm that runs in $O(n^t)$ time for some constant t strictly smaller than 2.
[Hint: for A and B, divide into intervals of length $n^{1.5}$. For intervals with fewer than r elements, use the algorithm in (b). For intervals with more than r elements (how many such intervals can there be?), use the algorithm in (a). How should we set the parameter r?]

Problem 1.3:

(a) (20 pts) Show that the product of an $n \times \ell$ with an $\ell \times n$ matrix can be computed in $O(\ell^{0.81} n^2)$ time by using Strassen's algorithm as a subroutine.

(b) (80 pts) We are given n strings $s^{(1)}, \ldots, s^{(n)} \in (\Sigma \cup \{\texttt{?}\})^*$ where `?` is the "don't care" symbol. Each string $s^{(i)}$ has length exactly ℓ.
We want to determine whether there exist two strings $s^{(i)}$ and $s^{(j)}$ $(i \neq j)$ that *match*. Here, two strings $s^{(i)} = s^{(i)}_1 s^{(i)}_2 \cdots s^{(i)}_\ell$ and $s^{(j)} = s^{(j)}_1 s^{(j)}_2 \cdots s^{(j)}_\ell$ in $(\Sigma \cup \{\texttt{?}\})^*$ are said to match iff for all $k \in \{1, \ldots, \ell\}$, we have $s^{(i)}_k = s^{(j)}_k$ or $s^{(i)}_k = \texttt{?}$ or $s^{(j)}_k = \texttt{?}$.
The trivial algorithm runs in $O(\ell n^2)$ time. Describe a faster algorithm.
[Hint: use the idea from Clifford and Clifford's algorithm, but replace convolution with matrix multiplication...]