Instructions: As in previous homeworks.

Problem 5.1: A point $p$ in 2D is specified by its $x$-coordinate $p.x$ and $y$-coordinate $p.y$.

An array of points $P[1, \ldots, n]$ in 2D is ascending iff $P[1].x < P[2].x < \cdots < P[n].x$ and $P[1].y < P[2].y < \cdots < P[n].y$. An array of points $Q[1, \ldots, m]$ is descending iff $Q[1].x < Q[2].x < \cdots < Q[m].x$ and $Q[1].y > Q[2].y > \cdots > Q[m].y$.

Describe an efficient algorithm for the following problem: given an ascending array $P[1, \ldots, n]$ and a descending array $Q[1, \ldots, m]$, find a common point, i.e., some $i, j$ with $P[i] = Q[j]$, if one exists. [Note that if one exists, the answer must be unique (why?).] Aim for $O(\log n + \log m)$ running time. [A slower algorithm with $O(\log n \log m)$ running time will still get partial credit.]

(For example: for the ascending sequence $\langle (1, 0), (4, 1), (5, 4), (8, 5), (13, 7) \rangle$ and the descending sequence $\langle (0, 11), (2, 10), (8, 5), (9, 2), (11, 1) \rangle$, there is a common point $(8, 5)$.)

Problem 5.2: Consider the following problem: We are given a set $R$ of $n_1$ rectangles in 2D, where the sides of the rectangles are parallel to the $x$- and $y$-axes (i.e., the rectangles are not rotated) and the rectangles do not overlap. We are also given a set $P$ of $n_2$ points in 2D. For every point $p \in P$, we want to find the rectangle $r_p \in R$ that contains $p$. (Note that if $p$ is not covered by any of the rectangles in $R$, then $r_p$ is undefined, but otherwise $r_p$ is unique because of the nonoverlapping assumption.) Let $n = n_1 + n_2$.

---

1 As usual, this means giving a pseudocode description (not actual code!), along with explanation or justification of correctness (especially if it is not obvious), and analysis of the running time.
(a) (3.0 points) First give an \(O(n \log n)\)-time algorithm for the special case when all rectangles of \(R\) are assumed to intersect a given horizontal line \(\ell\).

[Hint: sort...]

(b) (7.0 points) Now describe an algorithm to solve the general problem in \(O(n \log^2 n)\) time or better.

[Hint: use divide-and-conquer and part (a) as a subroutine.]

Problem 5.3: Consider the following problem: given a number \(n\), compute \(n!\) (the factorial). Here, we measure running time in terms of the number of bit operations. Notice that \(n!\) has \(\Theta(\log(n!)) = \Theta(n \log n)\) bits.

(a) (2.0 points) Recall that Karatsuba’s algorithm can multiply two \(k\)-bit integers in \(O(k^{1.59})\) time. Using Karatsuba’s algorithm as a subroutine, show that we can multiply a \(k\)-bit integer with an \(\ell\)-bit integer (\(\ell \geq k\)) in \(O(\ell k^{0.59})\) time.

(b) (2.0 points) Analyze the naive iterative algorithm to compute \(n!\) (i.e., for \(i = 1\) to \(n\), multiply the current answer with \(i\)). Show that using (a), this algorithm runs in \(O(n^2 \log^{1.59} n)\) time.

(c) (6.0 points) Next, design and analyze a faster divide-and-conquer algorithm to compute \(n!\), running in \(O(n^{1.59} \log^{1.59} n)\) time.

[Hint: solve a more general problem, of computing \((M + 1)(M + 2) \cdots (M + n)\) for any given \(n\) and \(M\)...]

\(^2\) As usual, in a multi-part question, if you are unable to solve (a), you can still do (b) under the assumption that (a) has been solved.