

- Suppose you are given a set B of n disjoint axis-aligned rectangles in the plane. Describe an algorithm to find a line that intersects the largest number of boxes in B .

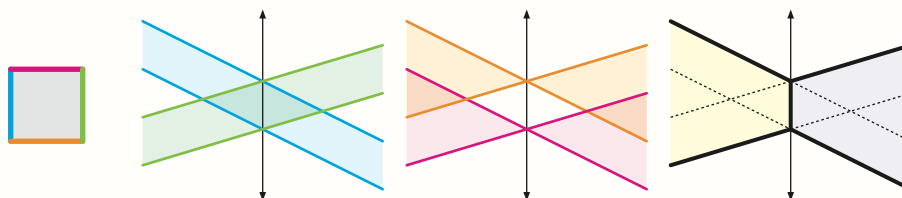
[Hint: What is the dual of a rectangle?]

Solution (bow ties): We consider the problem through the lens of the duality transformation $(a, b) \Leftrightarrow y = ax - b$. General position implies that is a subset of boxes is stabbed by a vertical line, then that same subset is stabbed by a non-vertical line, so we only need to consider lines with well-defined dual points.

Consider a rectangle $r = [a^-, a^+] \times [b^-, b^+]$. A line ℓ intersects r if and only if at least one vertex of r lies on or above ℓ and at least one vertex of r lies on or below ℓ . Thus, the dual of r consists of all points ℓ^* that are neither above all four lines dual to vertices of r nor below those four lines. This region in the dual plane is an infinite “bow-tie” bounded by two pairs of rays whose basepoints lie on the y -axis.

$$\left\{ (x, y) \mid \begin{array}{l} x < 0 \\ y \leq a^-x - b^- \\ y \geq a^+x - b^+ \end{array} \right\} \cup \left\{ (x, y) \mid \begin{array}{l} x > 0 \\ y \leq a^+x - b^- \\ y \geq a^-x - b^+ \end{array} \right\}$$

The vertices of the dual bow-tie are the points $(0, -b^-)$ and $(0, -b^+)$ dual to the lines containing the horizontal edges of r .



The internal structure of the dual of an axis-aligned rectangle.

From left to right: Slabs dual of the vertical edges, double-wedges dual to the horizontal edges, and regions dual to negative- and positive-slope stabbing lines

Let P be the set of vertices of the input rectangles B . We can build the arrangement of dual lines P^* in $O(n^2)$ time. Define the *box-depth* of a cell in this arrangement as the number of dual bow-ties that contain the cell.

The upper envelope of P^* has box-depth zero; this cell contains point dual to lines that pass entirely below R . Assuming general position, the box-depth of two adjacent cells differ by at most 1. For any edge e of the line arrangement, we can determine in $O(1)$ time the dual line p^* that contains e , the rectangle r that has p as a vertex, and whether e lies on the boundary or the interior of the dual bow-tie r^* . Thus, we can label each cell with its box-depth in $O(n^2)$ time, by traversing the dual graph of the arrangement, starting at the upper envelope.

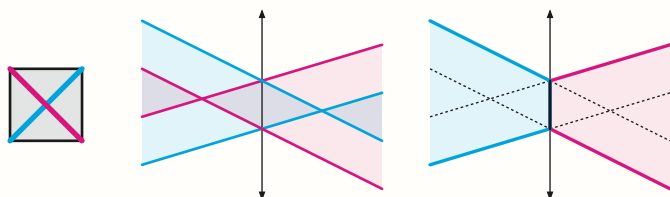
Finally, we return the maximum box-depth among all cells in the dual arrangement. The entire algorithm runs in $O(n^2)$ time. ■

Solution (diagonals): General position implies that is a subset of boxes is stabbed by a vertical line, then that same subset is stabbed by a non-vertical line, so we only need to consider lines with finite slope. Similarly, we do not need to consider horizontal lines. (Even if we can't assume general position, those cases can be handled separately in $O(n \log n)$ time.)

We separately look for the best line with negative slope and the best line with positive slope. A line with negative slope intersects a rectangle $r = [a^-, a^+] \times [b^-, b^+]$ if and only if it intersects its negative-slope diagonal $(a^-, b^+)(a^+, b^-)$. Similarly, a line with positive slope intersects r if and only if it intersects its positive-slope diagonal $(a^-, b^-)(a^+, b^+)$. I will describe an algorithm to find the best line with non-positive slope in $O(n^2)$ time; a symmetric algorithm finds the best line with non-negative slope.

Let B^+ be the set of positive-slope diagonals of the rectangles in B . A line ℓ with negative slope intersects any segment $s = (a^-, b^-)(a^+, b^+)$ in B^+ , with $a^- < a^+$ and $b^- < b^+$, if and only if ℓ is above the lower endpoint (a^-, b^-) and below the upper endpoint (a^+, b^+) . The dual of all such lines is the intersection of three halfplanes:

$$D(s) = \{(x, y) \mid x \leq 0 \text{ and } y \leq a^-x - b^- \text{ and } y \geq a^+x - b^+\}$$



The internal structure of the dual of an axis-aligned rectangle.
 From left to right: Double-wedges dual to the diagonals,
 and regions dual to the negative- and positive-slope stabbing lines

Let P be the set of endpoint of segments in B^+ . We can build the arrangement of dual lines P^* and the y -axis $x = 0$ in $O(n^2)$ time. For each segment $s = (a, b)(a^+, b^+)$ in B^+ , give the upper dual line $y = a^-x - b^-$ weight 1 and the lower dual line $y = a^+x - b^+$ weight -1 . Define the *depth* of a cell in this arrangement as the number of dual regions $D(s)$ that contain the cell.

All cells in the right halfplane $x \geq 0$ have depth 0, so we can just ignore them. The depth of any cell in the left halfplane is the total weight of all lines in R^* above that cell, and (assuming general position) the depths of any two neighboring cells differ by exactly 1. We can label every cell with its depth in $O(n^2)$ time, by computing the depth of one arbitrary cell by brute force in $O(n)$ time, and then traversing the dual graph of the arrangement. Any point in a cell with maximum depth is dual to a negative-slope line that stabs the maximum number of boxes in B .

The overall algorithm runs in $O(n^2)$ time. ■

2. Suppose you are given a set B of n disjoint axis-aligned rectangles in the plane. Describe an algorithm that either finds a line that intersects *every* box in B , or correctly reports that no such line exists. Your algorithm should be faster than your solution to problem 1!

[Hint: Consider positive-slope lines and negative-slope lines separately.]

Solution: Following the hint, we separately look for a stabbing line with positive slope and a stabbing line with negative slope. General position implies that any subset of boxes that intersects a common vertical line also intersects a common non-vertical line. So we can restrict our search to lines with finite slope.

Let $[l_i, r_i] \times [b_i, t_i]$ denote the i th rectangle in B . For each index i , define

$$D_i := \{(x, y) \mid x \leq 0 \text{ and } y \leq l_i x - b_i \text{ and } y \geq r_i x - t_i\};$$

this convex region is dual to the set of *negative-slope* lines that intersect the *positive-slope* diagonal $(l_i, b_i)(r_i, t_i)$. A negative-slope line intersects *every* rectangle in B if and only if its dual point lies in *every* convex region D_i . Thus, any negative-slope stabbing line is dual to a feasible point for the following linear program:

minimize whatever subject to $y \leq l_i x - b_i$ for all i $y \geq r_i x - t_i$ for all i $x \leq 0$

We can find a feasible point for this LP, or correctly report that the LP is infeasible, in $O(n)$ expected time using Seidel's algorithm.

A nearly-identical algorithm finds a positive-slope stabbing line if one exists. The overall algorithm runs in $O(n)$ *expected time*. ■

3. Suppose you are given a set P of n points in the plane. Describe an algorithm that computes two **disjoint** axis-aligned rectangles of minimum total area, whose union contains every point in P .

[Hint: Sweep. Sweep again.]

Solution: Assume all x - and y -coordinates in P are distinct. Two axis-aligned rectangles are disjoint if and only if they are separated by a horizontal or vertical line (or both); we separately consider each of these cases.

To compute the smallest pair of boxes with disjoint x -ranges, we start by sorting the points in P by increasing x -coordinate. For any index k , let $AreaLeft(k)$ denote the area of the axis-aligned bounding box of the k leftmost points in P (stored in $P[1..k]$), and let $AreaRight(k)$ denote the area of the axis-aligned bounding box of the k rightmost points in P (stored in $P[n-k+1..n]$). We need to compute $\min\{AreaLeft(k) + AreaRight(n-k) \mid 1 \leq k \leq n-1\}$.

The following sweepline algorithm computes $AreaLeft(k)$ for all $1 \leq k \leq n$ in $O(n)$ time. Intuitively, the algorithm sweeps a vertical line from left to right, keeping track of the largest and smallest y -coordinate seen so far.

```

COMPUTEAREALEFT( $P$ ):
   $minx \leftarrow P[1].x$ 
   $miny \leftarrow \infty$ 
   $maxy \leftarrow -\infty$ 
  for  $k \leftarrow 1$  to  $n$ 
     $maxx \leftarrow P[k].x$ 
     $miny \leftarrow \min\{miny, P[k].y\}$ 
     $maxy \leftarrow \max\{maxy, P[k].y\}$ 
     $AreaLeft[i] \leftarrow (maxx - minx) \cdot (maxy - miny)$ 
  return  $AreaLeft[1..n]$ 

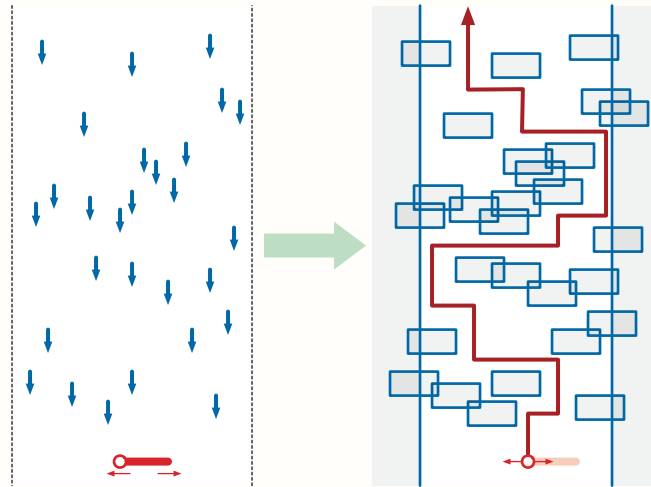
```

We similarly compute $AreaLeft[1..n]$ in $O(n)$ time by sweeping from right to left. Finally, we compute $\min_k (AreaLeft[k] + AreaRight[k])$ in $O(n)$ time by brute force.

We can compute the smallest pair of boxes with disjoint y -ranges using a nearly identical algorithm (or swapping all x - and y -coordinates and using the same algorithm). Finally, we return the smaller of the results of these two algorithms. The entire algorithm runs in **$O(n \log n)$ time**; the running time is dominated by sorting. ■

4. Describe an algorithm to play Knife Storm *perfectly*, assuming you are given complete information about the falling knives in advance. (See the exam handout for more details.)

Solution (reduce to homework): We construct the free configuration space of the paddle as follows. Instead of the knives moving downward, we image the knives being fixed and paddle moving upward at unit speed. Now a configuration of the paddle is specified entirely by the left endpoint (p, t) of the paddle. The paddle intersects the i th knife at time t if and only if $x[i]-1 \leq p \leq x[i]$ and $y[i]-\frac{1}{4} \leq t \leq t[i]$, or equivalently, if (p, t) lies inside the axis-aligned rectangle $\square_i = [x[i]-1, x[i]] \times [y[i]-\frac{1}{4}, y[i]]$. Thus, the free configuration space is $\mathcal{F} = \{(p, t) \mid 0 \leq p \leq t\} \setminus \bigcup_i \square_i$.



Now our problem is to find a *y-monotone* path (meaning the y -coordinate never decreases) through \mathcal{F} from any point below all the boxes to any point above all the boxes. The boundaries of the obstacles are horizontal and vertical line segments, so aside from a 90-degree rotation, this is exactly problem 4 from Homework 1! The Homework 1 solutions describe a sweep-line algorithm that solves this problem in $O(n \log n)$ time.^a ■

^aThe statement of HW1.4 included an explicit assumption that the obstacle segments have distinct endpoints, which our boxes violate. But we can restore that condition by extending each box edge in both directions by a sufficiently small (or symbolic) positive length ε . Alternatively, we can modify the sweep-line algorithm to directly handle the horizontal sweep-line hitting bottom and top edges of boxes.

Solution (direct sweep algorithm): Following the solution to HW1.4, we simulate the game, keeping track of possible locations for the paddle. Specifically, we maintain the intersections between knives and the x -axis in a balanced binary search tree. These intersection points partition the interval $[0, 10]$ into smaller intervals; we also mark each smaller interval *live* or *dead* depending on whether it can contain the paddle.

Initially, we have only a single live interval $[0, 10]$, and our BST contains only the numbers 0 and 10. We consider two types of events, in chronological order.

- When the bottom end of a knife touches the x -axis at $(x_i, 0)$, we insert x_i into our balanced BST. We also identify the predecessor x_i^- and successor x_i^+ of x_i . The insertion splits the interval (x_i^-, x_i^+) into two smaller intervals.
 - If the interval (x_i^-, x_i^+) is live and $x_i - x_i^- > 1$, we mark the left sub-interval (x_i^-, x_i) live; otherwise, we mark it dead.
 - Similarly, if (x_i^-, x_i^+) is live and $x_i^+ - x_i > 1$, we mark the right sub-interval (x_i, x_i^+) live; otherwise, we mark it dead.
- When the top end of a knife touches the x -axis at $(x_i, 0)$, we insert x_i into our balanced BST. We also identify the predecessor x_i^- and successor x_i^+ of x_i . The insertion merges the interval (x_i^-, x_i) or (x_i, x_i^+) into a single interval (x_i^-, x_i^+) . If either (x_i^-, x_i) or (x_i, x_i^+) is live, we mark the merged interval (x_i^-, x_i^+) live; otherwise we mark it dead.

At the end of the simulation, we are again left with only the single interval $[0, 10]$. We report success if and only if that interval is live.

Our binary search tree always contains at most n numbers. Each event requires $O(1)$ tree operations and therefore takes $O(\log n)$ time. We also need $O(n \log n)$ time to sort the knife endpoints by their y -coordinates to determine the chronological order of events. Thus, the entire algorithm runs in **$O(n \log n)$ time**. ■