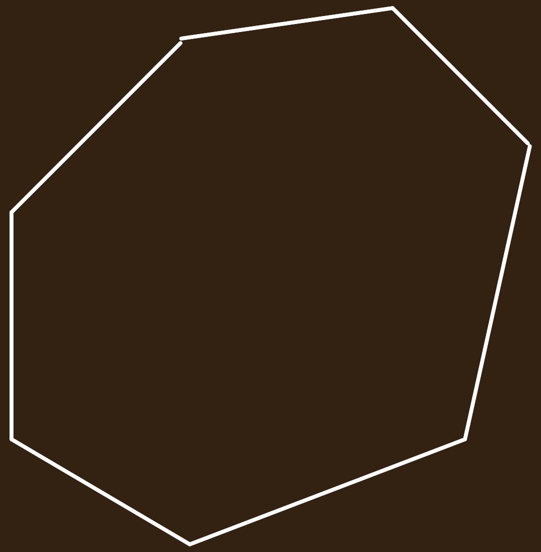
1. You are given a convex polygon P with n corners. Assume that P is specified by the coordinates of the n corners in clockwise direction. Propose algorithms for the following parts.

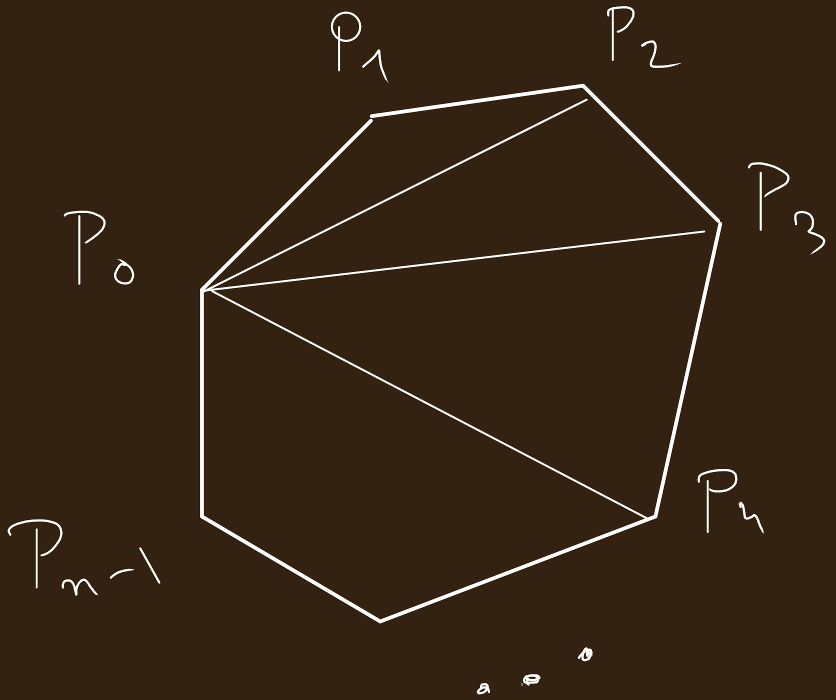
   (a) Compute the perimeter of P.



Add the lengths of the n sides.

Length of a side is the Euclidean distance between the two endpoints.

(b) Compute the area of P.



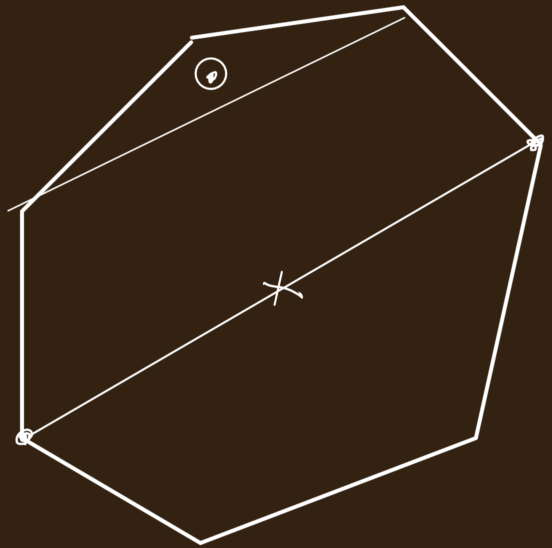
Triangulate the polygon, and add the areas of the triangles.

Area of a triangle:

$$\sqrt{s(s-a)(s-b)(s-c)}$$

$$\text{abs}\left(\frac{1}{2}\begin{vmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{vmatrix}\right)$$

(c) Find a point inside P in O(1) time.

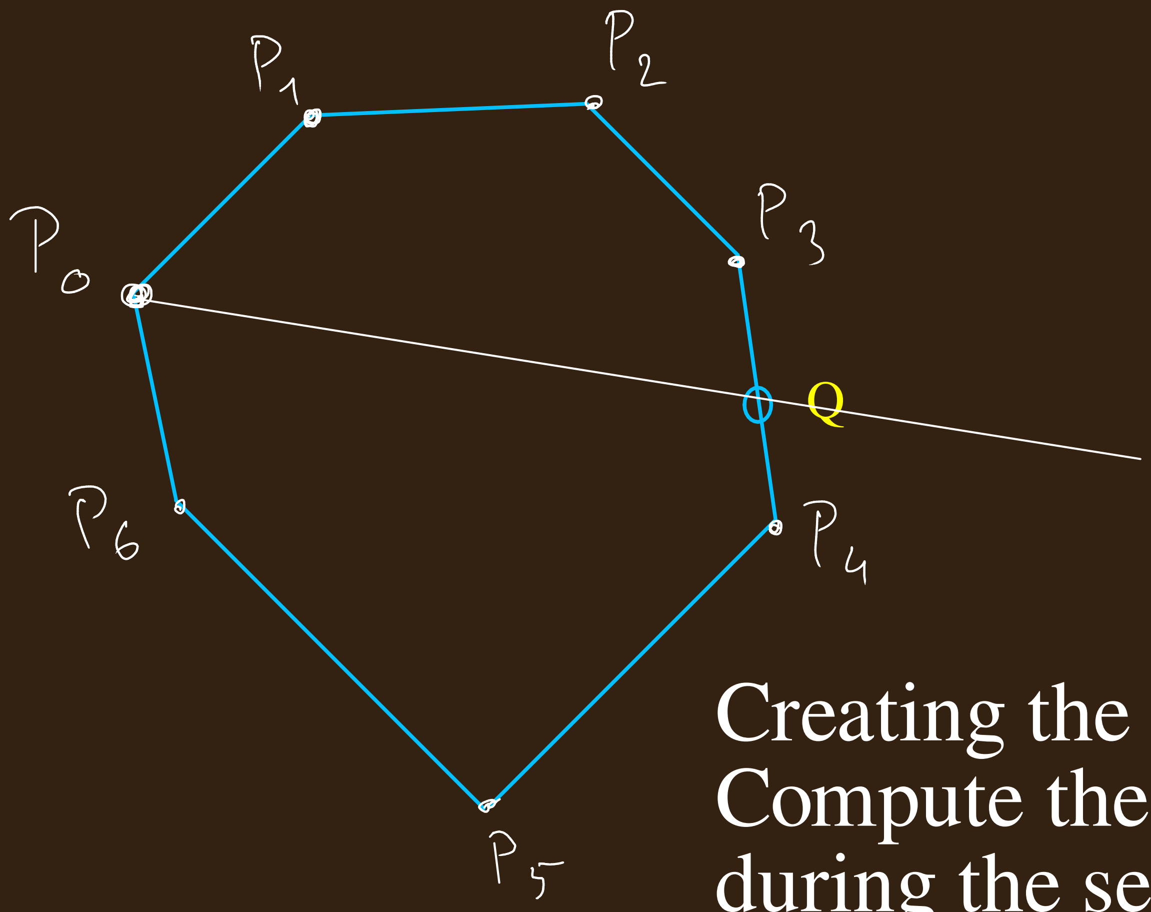


Computing the centroid of the corners takes O(n) time.

You can take the mid point of any two non-adjacent corners.

You can take the centroid of the triangle with any three corners of P.

(d) A point Q is given on an edge of P (Q is not a corner). Propose an O(log n)-time algorithm to identify the edge.



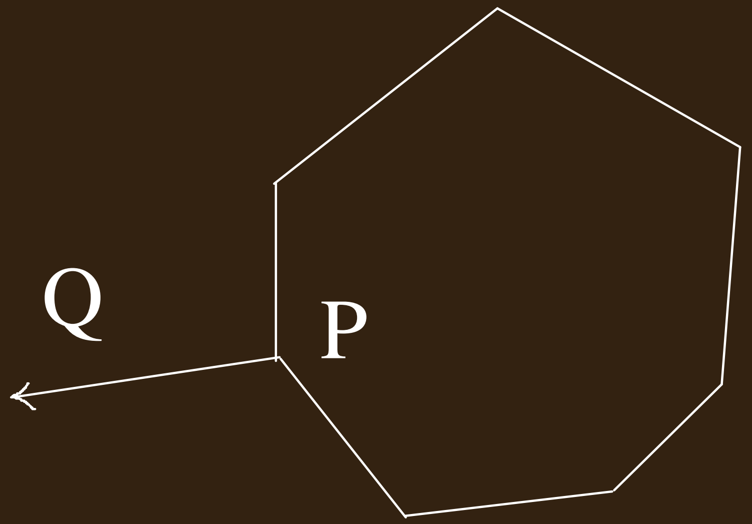
The side array with respect to P_0Q can be used for binary search. It consists of (a possibly empty) left block followed by (a possibly empty) right block. Binary search recovers the left-to-right crossover point.

Creating the entire array takes O(n) time. Compute the sides only at the indices accessed during the search.

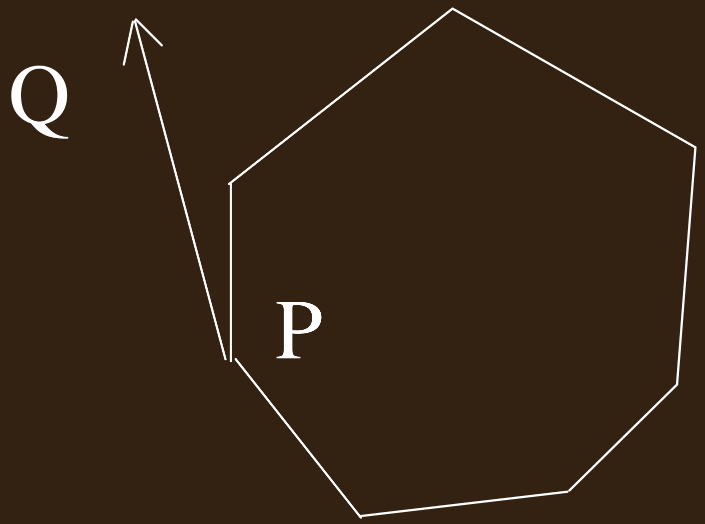
(e) You are given a point Q not on the boundary of P. Propose an algorithm to determine whether Q is inside or outside P. Running time?

We may use the previous algorithm. But several cases should be considered.

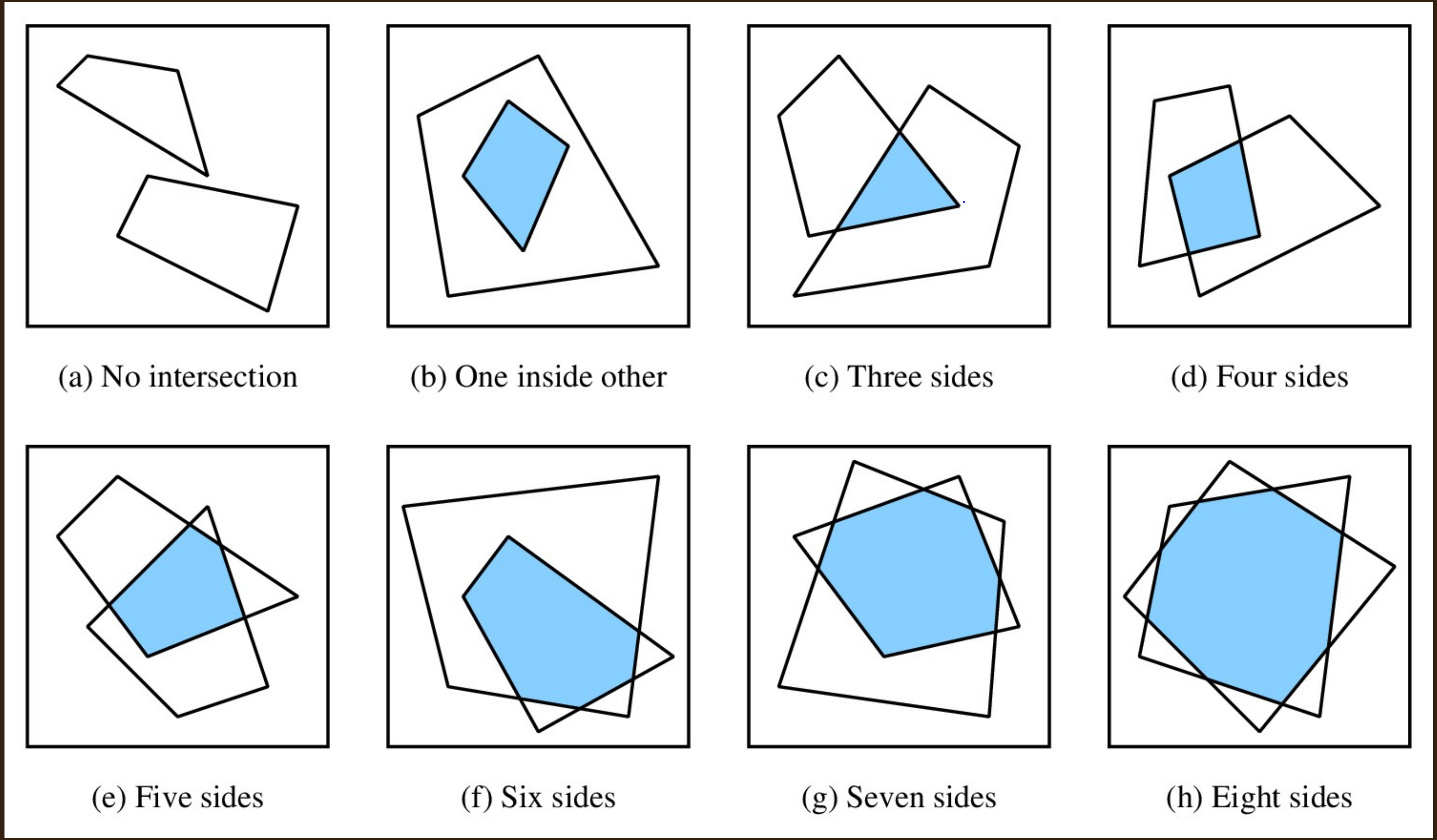
No intersection with P



Tangency



2. You are given two convex quadrilaterals. Propose an efficient algorithm to compute the intersection of the given quadrilaterals.



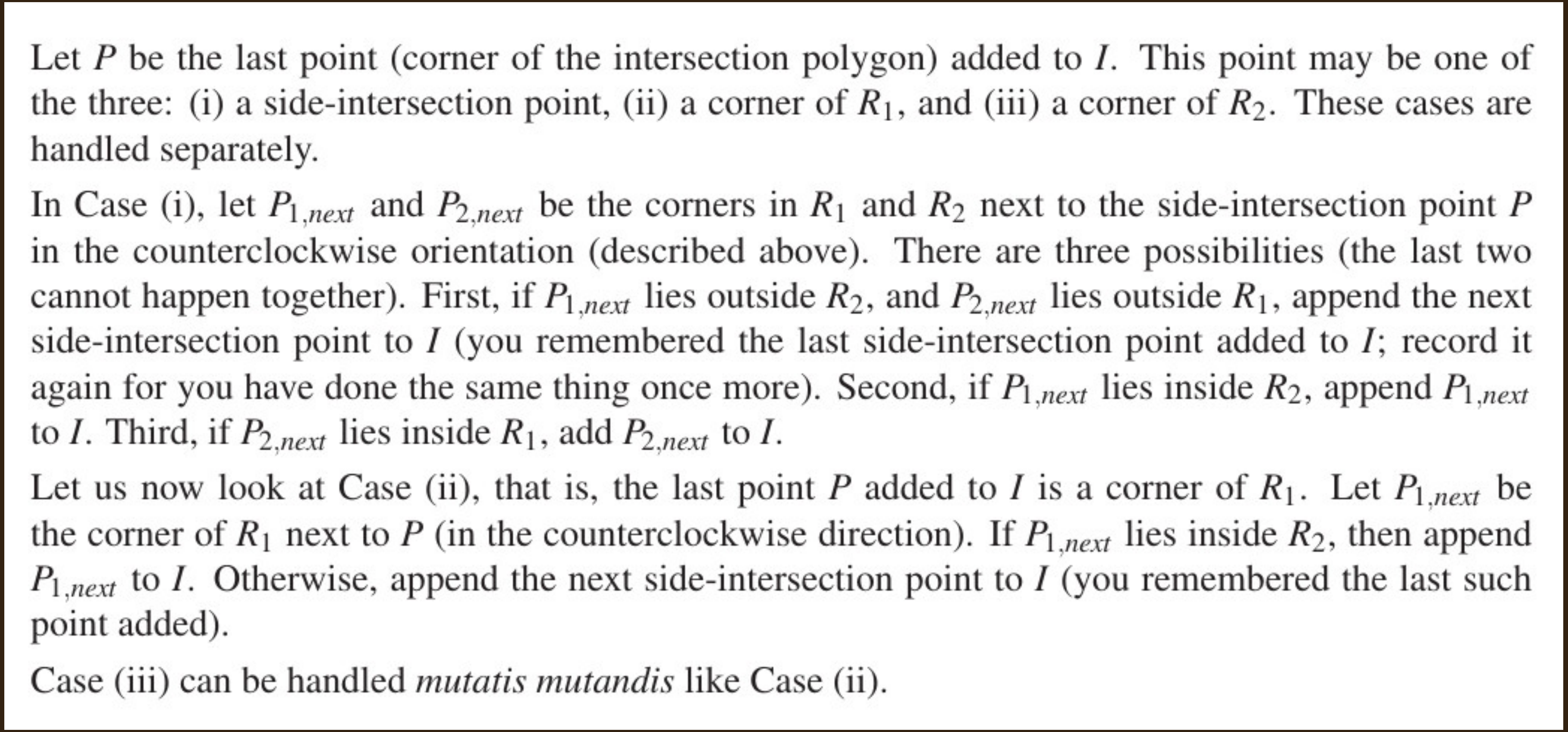
First, compute all the points of intersections of the sides of R_1 with the sides of R_2. Here, sides are to be treated as segments (not lines). Let there be l such points of side intersection. We have $0 \leqslant l \leqslant 8$. All these l points will be corners of the desired intersection polygon I. But the intersection polygon may contain one or more corners from one or both of the input quadrilaterals. You need to insert these quadrilateral corners in between the l side-intersection points computed above.

Arrange the l side-intersection points in the counterclockwise order (starting from any one of them). In order to do that, obtain the average (center of mass) of these l points, compute the angles of the l points with respect to the average (use `atan2`), and sort the l points with respect to these angles.

You now have three lists of points, all arranged in the counterclockwise order. The l side-intersection points, the four corners of R_1, and the four corners of R_2. By the next point in each list, we mean the next point in the counterclockwise direction in that list. Moreover, let P be a side-intersection point resulting from the intersection of the side s_1 of R_1 with the side s_2 of R_2. The corner of R_1 next to P is that endpoint of s_1, that follows P in the counterclockwise orientation. Likewise, the corner of R_2 next to P is defined.

Let us first handle the special case $l = 0$. This happens when either the quadrilaterals are disjoint or one of these is completely contained in the other. In the first case, all corners of each quadrilateral must lie outside the other quadrilateral, whereas in the second, all four of the corners of one quadrilateral lie inside the other (checking for one suffices). For disjoint quadrilaterals, the intersection polygon is empty, whereas in the other case with $l = 0$, it is the the contained quadrilateral.

Now, suppose that $l > 0$. We build the list I of the corners of the intersection polygon of R_1 and R_2. Initialize I to empty. Add any side-intersection point to I (so now I contains only one corner at this time). Repeat the following until you have come back to the first (side-intersection) point added to I. You always need to remember the last side-intersection point added to I.


Let P be the last point (corner of the intersection polygon) added to I. This point may be one of the three: (i) a side-intersection point, (ii) a corner of R_1, and (iii) a corner of R_2. These cases are handled separately.

In Case (i), let $P_{1,next}$ and $P_{2,next}$ be the corners in R_1 and R_2 next to the side-intersection point P in the counterclockwise orientation (described above). There are three possibilities (the last two cannot happen together). First, if $P_{1,next}$ lies outside R_2, and $P_{2,next}$ lies outside R_1, append the next side-intersection point to I (you remembered the last side-intersection point added to I; record it again for you have done the same thing once more). Second, if $P_{1,next}$ lies inside R_2, append $P_{1,next}$ to I. Third, if $P_{2,next}$ lies inside R_1, add $P_{2,next}$ to I.

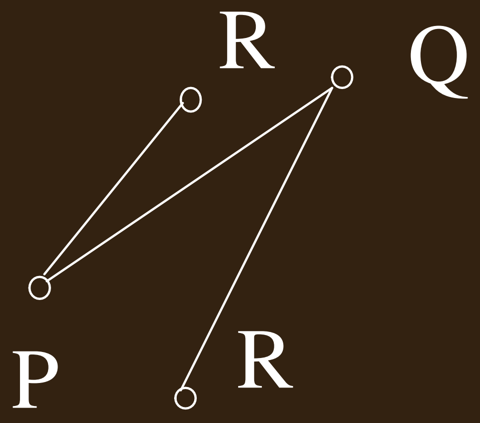
Let us now look at Case (ii), that is, the last point P added to I is a corner of R_1. Let $P_{1,next}$ be the corner of R_1 next to P (in the counterclockwise direction). If $P_{1,next}$ lies inside R_2, then append $P_{1,next}$ to I. Otherwise, append the next side-intersection point to I (you remembered the last such point added).

Case (iii) can be handled *mutatis mutandis* like Case (ii).

This solution assumes counterclockwise listing of polygon corners. It can be readily modified to work with clockwise listing.

3. You are given a set of n points in the Euclidean plane. Your task is to find out the points P,Q in the given collection such that the absolute value of the slope of the segment PQ is as large as possible. Propose an efficient algorithm for this problem. You may assume that the given points are in general position.

Sort the given points with respect to their x-coordinates. [O(n log n)]
The steepest pair appears consecutively in the sorted list. [O(n)]



Proof: Let P,Q be the steepest pair. If they are not consecutive, there exists a point R between them. R is not on the line PQ [general position].
If R is above PQ, then PR is steeper.
If R is below PQ, then RQ is steeper.

4. You are given n intervals $[a_i,b_i]$ standing for activities (like classes, seminars, and so on), all of which must be scheduled. Propose an efficient algorithm to find out the minimum number of classrooms needed. Assume that the interval endpoints are in general position.

Sort the 2n endpoints. Let the sorted list be $E_1, E_2, \cdots, E_{2n}$. Each E_i should store the info which type of endpoint (left or right) it is.

```
overlap = 0; max = 0;
for (i = 1, 2, ⋯, 2n)
    if ($E_i$ is a left endpoint)
        ++overlap
        if (overlap > max) max = overlap
    else
        --overlap
```

We sweep from $-\infty$ to $+\infty$
Interesting things (events) happen only at the endpoints.
We handle all the events.

5. Lift the general-position restriction from the previous exercise. How can you make your algorithm work (in the same running time)?

Change the sorting order slightly.

If $b_i = a_j$, put b_i before a_j.

Ties like $a_i = a_j$ or $b_i = b_j$ may be broken arbitrarily.

6. Assume that each activity has a preparation time c_i and a closing time d_i. How can you make your algorithm work in this setting?

Run the algorithm on the intervals $[a_i - c_i, b_i + d_i]$.