1. Suppose that in Graham scan, pairs of points (but not three or more) may have the same x-coordinates. How can you modify the algorithm to handle this degeneracy.

For the upper hull:
Sort first with respect to increasing x-coordinates, and then (in case of ties) with respect to decreasing y-coordinates.

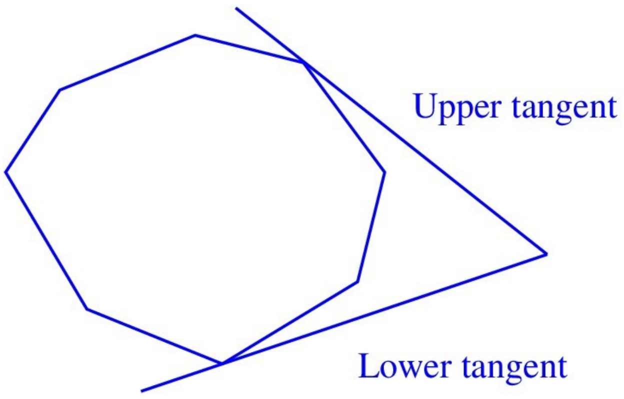
2. [Incremental Convex Hull construction]

(a) Let C be a convex polygon, and P a point. Propose an algorithm to determine whether P is inside or outside C. Running time?

Consider the horizontal ray emanating from P. Find out how many edges of C intersect with the ray. If the count is 1, then P is inside C. If the count is 0 or 2, then P is outside C. If the ray meets two consecutive edges of C, count this as a single intersection.

(b) Let $P_1, P_2, \ldots, P_n$ be a set of n points in the general position. We want to compute CH($P_1, P_2, \ldots, P_n$). Use Part (a) to convert CH($P_1, P_2, \ldots, P_i$) to CH($P_1, P_2, \ldots, P_{i+1}$). Total running time?

First check whether P_{i+1} is inside or outside CH($P_1, P_2, \ldots, P_i$).
Inside: CH($P_1, P_2, \ldots, P_{i+1}$) = CH($P_1, P_2, \ldots, P_i$).
Outside: Draw two tangents and discard points between them.



Running time is O(n^2).

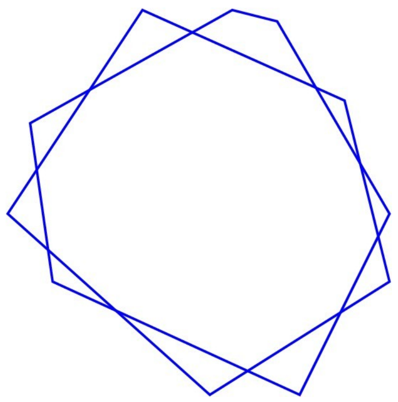
(c) Propose an O(n log n)-time algorithm using this idea.

Sort the points first. Then the check of Part (a) is not necessary. Tangent computations may start from the rightmost point of CH($P_1, P_2, \ldots, P_i$), which is P_i. As many as Θ(n) vertices may be traversed (and removed) in each iteration. But once a vertex is removed, it will never come back again. Amortized analysis shows that total running time for computing all the tangents is O(n). You may take the number of corners on the current hull as the potential.

3. Let S and T be two disjoint sets of points in the Euclidean plane. S and T need not be horizontally separated. You have computed the two convex hulls CH(S) and CH(T).

(a) Propose an O(n)-time algorithm to merge these two hulls to CH($S \cup T$), where $n = |\, S \cup T \,|$.

There may be too many tangents to construct.



Better idea: Get four sorted lists from UH(S), LH(S), UH(T) and LH(T). Merge them and then use Graham scan.

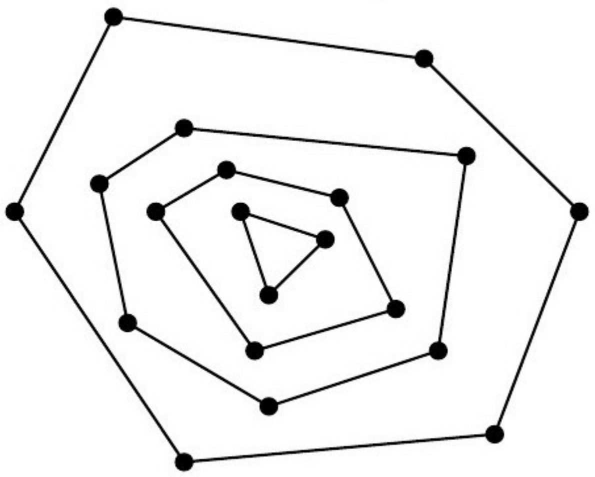
(b) Prove that the problem of Part (a) cannot be solved in o(n) time in the worst case.

Use a procedure like merge sort to get

$T(n) = 2\,T(n/2) + o(n) = o(n \log n)$.

4. [Onion layers]

Let S be a set of n points in the plane. Let L_1 denote the set of vertices of CH(S). Remove the points of L_1 from S, and compute the convex hull of S again. Let L_2 denote the set of vertices of this convex hull. Repeat.



What is the worst-case running time for computing all onion layers if you use

(a) Jarvis march
(b) Graham scan.

Give tight bounds.

(a)

$O(nh_1 + (n-h_1)h_2 + (n-h_1-h_2)h_3 + \cdots) = O(nh_1 + nh_2 + nh_3 + \cdots) = O(n^2)$
This is a tight bound because we may have a single layer with n vertices.

(b) There may be $\Theta(n)$ layers, so running Graham scan these many times takes $O(n^2)$ time again.

For both the algorithms, deletion of points from S can be implemented in a total of $O(n^2)$ time.

5. You are given n points in the plane in general position. Arrange the points to a list P_1, P_2, …, P_n such that for different i, j, the segments P_iP_{i+1} and P_jP_{j+1} do not intersect except perhaps at some endpoint, and each $P_iP_{i+1}P_{i+2}$ is a right turn.

Assume that our convex-hull algorithm outputs the corners of the output hull in a clockwise fashion starting from any vertex.

First algorithm: Let $L_1, L_2, \ldots, L_k$ be the onion layers of S. We need to list the points of the layers with some caution. We can start the listing from any point of L_1, and stop just before completing the polygon. Let Q be the last point listed. From Q, compute the appropriate tangent to L_2 (the tangent having all points of L_2 on or to the right of it is appropriate). Then, list the points of L_2 in clockwise fashion starting from the point of tangency, and stop just before coming back to that point. Again, draw the appropriate tangent from the last point listed to the next inner layer L_3. Start listing points of L_3 starting from the point of tangency. Repeat until all the layers are considered.

Second algorithm: Compute $L = \mathrm{CH}(S) = (Q_1, Q_2, \ldots, Q_h)$. List $Q_1, Q_2, \ldots, Q_h$. Remove $Q_1, Q_2, \ldots, Q_{h-1}$ from S. Unlike the previous algorithm, we keep Q_h in S. We then again compute $\mathrm{CH}(S)$. It is easy to see that Q_h must belong to this second hull. List the points of the second hull starting immediately after Q_h and stopping immediately before Q_h. Then, remove from S the point Q_h and all the points on the second hull except the one listed last. This process is repeated until S contains only one point.

The running time of both these algorithms is $O(n^2)$ if we use Jarvis march for each convex-hull computation.

6. (a) Prove that the number e of edges in any triangulation of CH(S) with $|S| = n$ satisfies $2n - 3 \leq e \leq 3n - 6$.

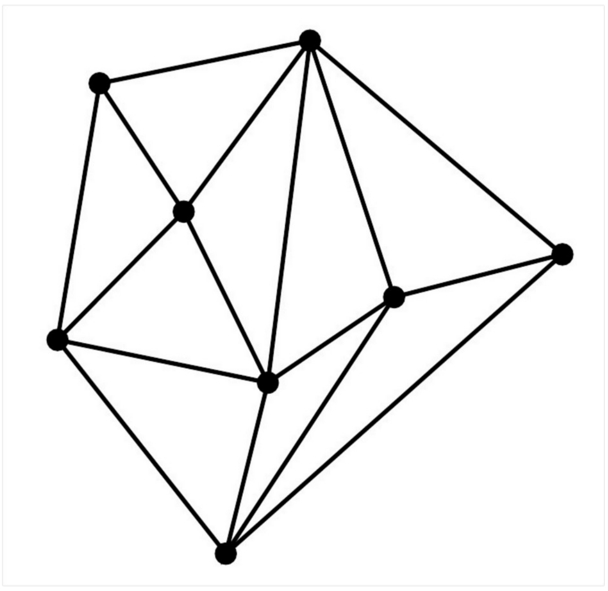
Let t be the number of triangles, and h the number of vertices on CH(S). By Euler's formula, we have

$$n - e + (t + 1) = 2.$$

Each edge belongs to two faces, so

$$2e = 3t + h.$$

Eliminate t. Then use $3 \leqslant h \leqslant n$.



(b) How can you use the incremental convex-hull construction to triangulate CH(S)? Running time?

Join the new point with the points of tangency, and also with all the discarded points.

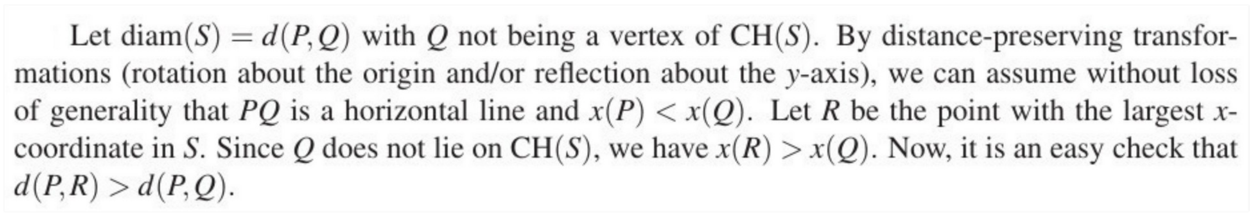
Initial sorting: O(n log n)
Triangulation: O(n) amortized time

7. [Farthest Pair]

Let S be a set of n points in general position in the plane. We want to find P, $Q \in S$ such that $d(P, Q)$ is the maximum. This distance is called the diameter of S.

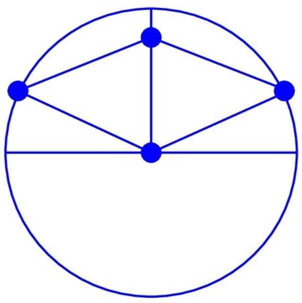
(a) Prove that P and Q are vertices of CH(S).

Solution



Let $\mathrm{diam}(S) = d(P,Q)$ with Q not being a vertex of $\mathrm{CH}(S)$. By distance-preserving transformations (rotation about the origin and/or reflection about the y-axis), we can assume without loss of generality that PQ is a horizontal line and $x(P) < x(Q)$. Let R be the point with the largest x-coordinate in S. Since Q does not lie on $\mathrm{CH}(S)$, we have $x(R) > x(Q)$. Now, it is an easy check that $d(P,R) > d(P,Q)$.

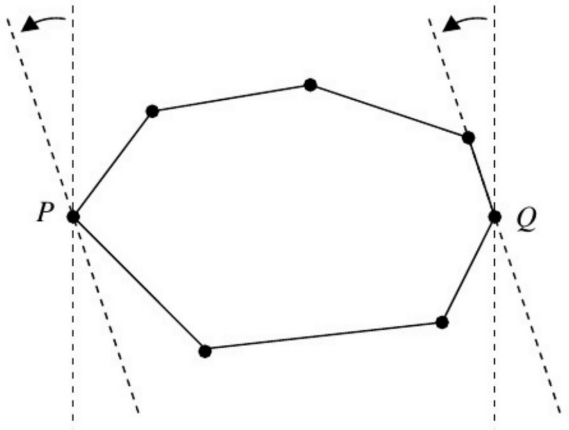
(b) Let P be a point on CH(S). Demonstrate that the distances of the points on CH(S) from P need not be unimodal.



(c) Let Q, Q' be consecutive vertices on CH(S). Let L be the line QQ'. The perpendicular distances of the vertices of CH(S) from L are unimodal. Let P be the farthest point from L. Build a collection C of pairs (P, Q) and (P, Q'). Prove that the farthest pair (P, Q) can be found in C.

Solution

Let $\text{diam}(S) = d(P,Q)$. We can assume, without loss of generality, that PQ is horizontal, and $x(P) < x(Q)$. But then, P (resp. Q) is the point in S with the minimum (resp. maximum) x-coordinate. Therefore, $\text{CH}(S)$ lies completely between two vertical lines passing through P and Q. Rotate these two lines simulltaneously (clockwise or counter-clockwise) until one of these lines coincide with an (the first) edge of $\text{CH}(S)$. This procedure is demonstrated in the following figure. It is now an easy check that $(P,Q) \in C$ (in the figure, for example, we have $Q = Q_{i+1}$ and P is a farthest Q_k from the line L_i passing through Q_i and Q_{i+1}).



(d) If the points in S are in general position, what is the maximum size of C?

$2n$

(e) Propose an O(n log n)-time algorithm for computing the farthest pair in S.

For each edge on CH(S), the vertices most distant from the edge can be obtained by binary search in the unimodal sequence of distances. We do not need to compute the entire sequence, but compute only those distances that are needed during the search.

8. [A point sweep algorithm in one dimension]

(a) 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 $E_1, E_2, \ldots, E_{2n}$. 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
```

(b) 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_j before a_j.

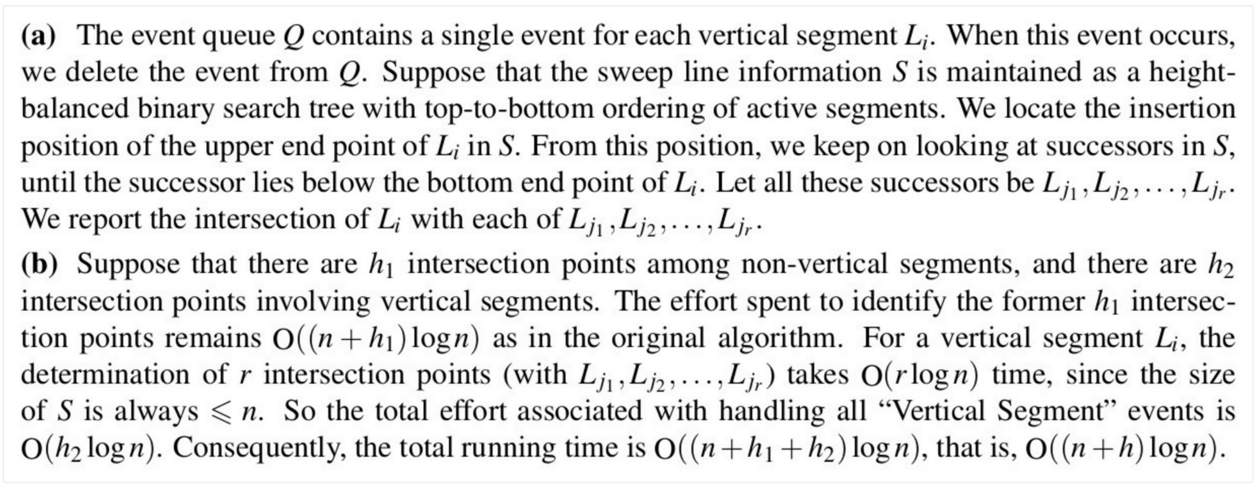
Ties like $a_i = a_j$ or $b_i = b_j$ may be broken arbitrarily.

(c) 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 $[a_i - c_i, b_i + d_i]$.

9. Suppose that in the line-sweep algorithm for the line-segment intersection problem, some lines are allowed to be vertical, but different vertical lines have different *x*-coordinates. Explain how you can handle a "vertical segment" event. Do not use rotation. Instead modify the line-sweep algorithm covered in the class.

Solution



(a) The event queue Q contains a single event for each vertical segment L_i. When this event occurs, we delete the event from Q. Suppose that the sweep line information S is maintained as a height-balanced binary search tree with top-to-bottom ordering of active segments. We locate the insertion position of the upper end point of L_i in S. From this position, we keep on looking at successors in S, until the successor lies below the bottom end point of L_i. Let all these successors be $L_{j_1}, L_{j_2}, \ldots, L_{j_r}$. We report the intersection of L_i with each of $L_{j_1}, L_{j_2}, \ldots, L_{j_r}$.

(b) Suppose that there are h_1 intersection points among non-vertical segments, and there are h_2 intersection points involving vertical segments. The effort spent to identify the former h_1 intersection points remains $O((n+h_1)\log n)$ as in the original algorithm. For a vertical segment L_i, the determination of r intersection points (with $L_{j_1}, L_{j_2}, \ldots, L_{j_r}$) takes $O(r \log n)$ time, since the size of S is always $\leqslant n$. So the total effort associated with handling all "Vertical Segment" events is $O(h_2 \log n)$. Consequently, the total running time is $O((n+h_1+h_2)\log n)$, that is, $O((n+h)\log n)$.