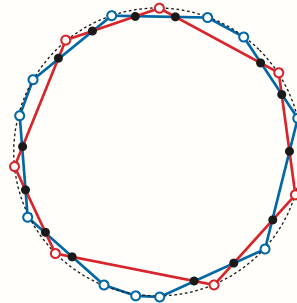


1. Suppose we are given two convex polygons P and Q , with m and n vertices, respectively. Each polygon is represented as a circular doubly-linked list of vertices in counterclockwise order.
 - (a) What is the *exact* maximum number of times that P and Q can intersect, as a function of m and n ? (A tight $O(\cdot)$ bound is worth significant partial credit.)

Solution: This intersection of any two convex sets is convex, and every convex subset of a line segment is either empty, a single point, or a line segment. Thus, the intersection of any edge of P and the area enclosed by Q is either empty or a line segment. (The single-point case is ruled out by general position.) It follows that the intersection of any edge of P and Q (just the boundary) is either empty or exactly two points. So there are at most $2m$ intersection points.

A symmetric argument implies that there are at most $2n$ intersection points. It follows that $|P \cap Q| \leq 2 \min\{m, n\}$.

Moreover, this bound is tight in the worst case, for all m and n . Without loss of generality, suppose $m \leq n$. Arbitrarily choose m red points and n blue points on the unit circle, such that none of the $n + m$ resulting circular arcs have two red endpoints. Let P be the convex hull of the red points, and let Q be the convex hull of the blue points. Each edge of P intersects Q exactly twice, so $|P \cap Q| = 2m$.



- (b) Describe and analyze an algorithm to compute all intersection points of P and Q . Analyze your algorithm as a function of m and n .

Solution: We use a variant of the Bentley-Ottmann sweep-line algorithm.

The sweep-line intersects at most two edges of each polygon. Thus, we can implement the sweep dictionary using a simple array of length 4, and every dictionary operation takes $O(1)$ time.

Similarly, at each event, we calculate the next event by brute force in $O(1)$ time. The next event is either the right endpoint of an edge that crosses the sweep line, or the intersection of two edges that cross the sweep line. Crudely, this is at most $4 + \binom{4}{2} = 10$ possible events. At each right endpoint, we can find the next edge of P or Q in $O(1)$ time by traversing the polygon data structure.

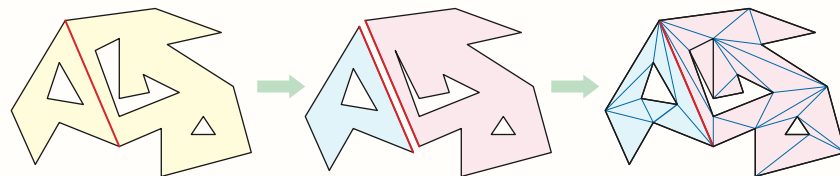
The algorithm processes one event for each vertex of P and Q and each intersection point. By part (a), the total number of events is at most $m + n + 2 \min\{m, n\}$, and we process each event in $O(1)$ time. Thus, the overall algorithm runs in $O(n + m)$ time. ■

2. A polygon with holes P is a connected area with a polygonal outer boundary P_0 and one or more disjoint polygonal inner boundaries P_1, P_2, \dots, P_h .
- (a) Suppose you are given a polygon P with h holes and a total of n vertices. Exactly how many diagonals and triangles are in any frugal triangulation of P , as functions of n and h ? Justify your answer.

Solution (induction): Let T be an arbitrary frugal triangulation of P . I will prove that T has $n + 3h - 3$ diagonals and $n + 2h - 2$ triangles, by induction on the quantity $n + 3h$.

Assume that every triangulation of a polygon with n' vertices and h' holes, where $n' + 3h' < n + 3h$, has $n' + 3h' - 3$ diagonals and contains $n' + 2h' - 2$ triangles. There are two cases to consider:

- If P is a triangle with no holes ($n = 3$ and $h = 0$), the claim is trivial. We needed 0 diagonals to partition P into 1 triangle.
- Otherwise, T must have at least one diagonal pq . There are two cases to consider.
 - First, suppose p and q lie on the same boundary polygon of P . Then pq partitions P into two polygons P_1 and P_2 with holes, and partitions T into triangulations of those polygons. Suppose P_1 has n_1 vertices and h_1 holes, and P_2 has n_2 vertices and h_2 holes. Because p and q appear in both smaller polygons, we have $n_1 + n_2 = n + 2$. P and Q each have at least three vertices, so $3 \leq n_1 \leq n - 1$ and $3 \leq n_2 \leq n - 1$. Finally, each hole of P is a hole in either P_1 or P_2 , so $h_1 + h_2 = h$.



We have $n_1 + 3h_1 \leq (n - 1) + 3h < n + 3h$, so the induction hypothesis implies that T_1 has $n_1 + 3h_1 - 3$ diagonals and $n_1 + 2h_1 - 2$ triangles. Similarly, T_2 has $n_2 + 3h_2 - 3$ diagonals and $n_2 + 2h_2 - 2$ triangles. Thus, the number of diagonals in T is

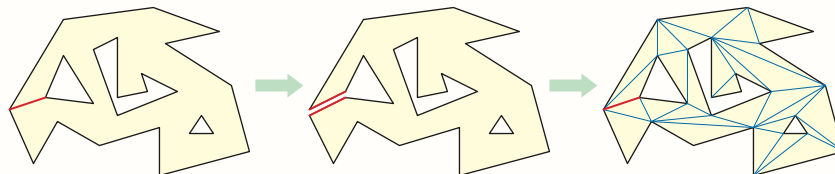
$$(n_1 + 3h_1 - 3) + (n_2 + 3h_2 - 3) + 1 = (n_1 + n_2 - 2) + 3(h_1 + h_2) - 3 = n + 3h - 3$$

and the number of triangles in T is

$$(n_1 + 2h_1 - 2) + (n_2 + 2h_2 - 2) = (n_1 + n_2 - 2) + 2(h_1 + h_2) - 2 = n + 2h - 2$$

- On the other hand, suppose p and q are on different polygon boundaries. Cutting P along pq creates a polygon P' with $n' = n + 2$ vertices and

$h' = h - 1$ holes; let T' be the triangulation of P' induced by the other diagonals in T .



We have $(n + 2) + 3(h - 1) < n + 3h$, so the inductive hypothesis implies that T' has $n' + 3h' - 3$ diagonals and contains $n' + 2h' - 2$ triangles. Thus, the number of diagonals in T is

$$\begin{aligned} n' + 3h' - 3 + 1 &= (n + 2) + 3(h - 1) - 3 + 1 \\ &= n + 3h - 3 \end{aligned}$$

and the number of triangles in T is

$$\begin{aligned} n' + 2h' - 2 &= (n + 2) + 2(h - 1) - 2 \\ &= n + 2h - 2 \end{aligned}$$

■

Solution (careful counting and Euler): Consider an arbitrary frugal triangulation of P , as a planar straight-line graph. Let V, E, D, F , and T respectively denote the number of vertices, edges, diagonals, faces, and interior triangles in this triangulation.

- Every vertex of the triangulation is a vertex of P , so $V = n$.
- Every face of the triangulation is either an interior triangle, a hole, or the outer face, so $F = T + h + 1$.
- We can count incident (edge, face) pairs in two different ways. First, every edge is incident to exactly two faces. Second, every interior triangle is incident to three edges, and every edge of P is incident to one face that is not an interior triangle. Thus, $2E = 3T + n$.
- Euler's formula now implies $V - E + F = n - \frac{1}{2}(3T + n) + (T + h + 1) = 2$, which simplifies to $T = n + 2h - 2$.
- Substituting into our earlier equation $F = T + h + 1$ implies $F = n + 3h - 1$.
- Every edge of the triangulation is either a diagonal or an edge of P , so $E = D + n$.
- Again, Euler's formula implies $V - E + F = n - (D + n) + (n + 3h - 1) = 2$, which simplifies to $D = n + 3h - 3$.

We conclude that the triangulation has $n + 3h - 3$ diagonals and $n + 2h - 2$ triangles. ■

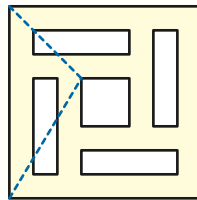
- (b) Recall the first algorithm we saw in class for triangulating a simple polygon P in $O(n \log n)$ time. Carefully describe how to modify this algorithm to compute a frugal triangulation of a polygon *with holes*. Describe *only* the necessary changes. What is the running time of your modified algorithm as a function of n and h ?

Solution: *Absolutely no changes are necessary.* In particular:

- “The edges of P ” includes both the edges of the outer polygon P_0 and the edges of the holes P_1, \dots, P_h .
- Every trapezoid inside a hole P_i is below an odd number of edges of P_i , an even number of edges of every other hole P_j (because holes are not nested), and an odd number of edges of the outer polygon P_0 (because every hole is inside P_0). Altogether, every trapezoid inside a hole is below an even number of edges of P , and thus is removed by step (2).
- Let p be the leftmost vertex of any hole P_i , and let τ be the trapezoid just to the left of p . Trapezoid τ is inside P_0 and outside every hole (because holes are disjoint), so it survives step (2). Vertex p lies in the interior of the right wall of τ . It follows that τ contains a boring diagonal (connecting P_i to another polygon boundary). It follows that the boring diagonals partition P into a set of simple polygons *without* holes.

The algorithm still runs in $O(n \log n)$ time. ■

One fairly common mistake was to try to connect each hole P_i directly to the outer boundary P_0 with a single diagonal, and then triangulate the resulting simple polygon without holes. This is not always possible; see the example below. Fortunately, it is also not necessary!



3. In a Formula 6–7 race, cars drive along a long straight track, which we can model as the x -axis. Each car has a starting position x_i and a fixed positive speed u_i ; thus, at any time $t \geq 0$, the position of the i th car is $p_i(t) = x_i + u_i \cdot t$. The i th car wins a prize if and only if $p_i(t) = \max_j p_j(t)$ for some $t \geq 0$.

Describe and analyze an algorithm to determine which cars win a prize during a Formula 6–7 race, given the starting positions and speeds of n cars as input.

Solution (“sweep line”): The following algorithm is essentially the motorcycle-graph algorithm from Homework 2.1(b). The algorithm is based on the observation that a car cannot be in first place after it has been passed by another car; in other words, we can remove any car from the race as soon as another car overtakes it. The input is an array of records, each storing a position $car[i].x$ and a speed $car[i].u$.

«Compute time when back car coincides with front car»

OVERTAKE(back, front):

return $time \leftarrow (back.u - front.u) / (front.x - back.x)$

FORMULA67(car[1..n]):

```

sort car by x
eventQ ← empty priority queue    «future passing events»
for i ← 1 to n - 1
    time[i] ← OVERTAKE(car[i], car[i + 1])
    if time[i] > 0
        eventQ.INSERT(i, time[i])
    next[i] ← i + 1
next[n] ← ∞
winners ← {n}                    «set of winners»
now ← 0
while eventQ is non-empty
    i ← eventQ.EXTRACTMIN()
    eventQ.DELETE(next[i])        «remove car next[i]...»
    next[i] ← next[next[i]]      «...because car i passed it»
    if next[i] = ∞
        add i to winners
    else
        time[i] ← OVERTAKE(car[i], car[next[i]])
        if time[i] > 0
            eventQ.INSERT(i, time[i])
return winners

```

The entire algorithm runs in $O(n \log n)$ time. ■

Rubric: This is not the only $O(n \log n)$ -time solution.

Solution (reduction to convex hulls): We can represent each car as a ray r_i with basepoint $(0, x_i)$ and slope u_i . For any $t \geq 0$, the cross-section of these rays with the vertical line $x = t$ shows the positions of the cars at time t . Thus a car wins a prize if and only if the corresponding ray touches the *upper envelope* of the n rays, which is the intersection of the halfspaces above each ray and (as a sentinel) the halfspace $x \geq 0$. (Think of the sentinel as an additional car that starts arbitrarily far ahead of the other cars and drives backward at arbitrarily high speed.)

We can compute this halfplane intersection in $O(n \log h)$ time via duality and Chan's algorithm, where h is the number of winning cars (including the sentinel). ■

Rubric: This is not the only $O(n \log h)$ -time solution!

4. Suppose we are given a set S of n line segments in the plane in general position, with k intersecting pairs, inside a large axis-aligned bounding box. A trapezoidal decomposition of S is a planar straight-line graph constructed by building walls upward and downward from each segment endpoint *and each intersection point*, ending at the next segment *or bounding box edge* above and below.

Exactly how many vertices, edges, and trapezoids does this trapezoidal decomposition have, as a function of n and k ? (Don't count the unbounded region outside the bounding box.) Justify your answer.

Solution: There are $6n + 3k + 4$ vertices:

- 4 bounding-box corners,
- $2n$ segment endpoints,
- k intersection points, and
- $4n + 2k$ wall endpoints (one above and below each segment endpoint and each intersection point).

There are $3n + 3k + 1$ trapezoids, which we can count by considering which points defines their left walls.

- Each left segment endpoint defines the left wall of two trapezoids. So there are $2n$ “left endpoint” trapezoids.
- Each right segment endpoint defines the left wall of one trapezoid. So there are n “right endpoint” trapezoids.
- Each intersection point defines the right wall of three trapezoids. (One of these is a triangle whose right “wall” is just the intersection point.) So there are $3k$ “intersection” trapezoids.
- The only trapezoid not counted so far is the one next to the left edge of the bounding box.

If we also include the outer face, we get a total of $3n + 3k + 2$ faces.

Finally, Euler's formula implies that there are $V + F - 2 = (6n + 3k + 4) + (3n + 3k + 2) - 2 = 9n + 6k + 4$ edges. ■