Convex hulls in 2D

The problem: Given a set P of n points in the plane, find their convex hull.

Properties of the convex hull

- A point is on the CH if and only of it is *extreme* (a point p is extreme if there exists a line l through it such that all other points are on or on one side of l).
- An edge is on the CH if and only of it is *extreme* (a line l is extreme if all points in P are on or on one side of it).
- A point p is **not** on the CH if and only if p is contained in the interior of a triangle formed by three other points of P.
- The points with minimum/maximum x-coordinate are on the CH.
- The points with minimum/maximum y-coordinate are on the CH.
- Walking counter-clockwise (ccw) on the boundary of the CH you make only left turns.
- Consider a point p inside the CH. The points on the boundary of the CH are encountered in sorted radial order wrt p.

Algorithm: Brute force

Idea: Find all extreme edges

```
Algorithm BruteForce (input: points $P$)
    • for all distinct pairs of points $(p_i, p_j)$:
        – if edge $(p_i, p_j)$ is extreme, output it as CH edge
```

Questions:

- How do you check if an edge is extreme, and how fast?
- What is the overall running time of Algorithm BruteForce?

Algorithm: Gift wrapping

Idea: start from a point p guaranteed to be on the CH and find the edge pq of the CH starting at p; repeat from q.

```
Algorithm GiftWrapping (input: points $P$)
  • Let $p_0$ be the point with smallest x-coordinate (if more than one, pick right-most)
  • $p = p_0$
  • repeat
        for each point $q$, $q! = p$:
          ∗ compute counter-clockwise-angle (cccw) of $q$ wrt $p$
        let $p'$ be the point with smallest such angle
        //claim: edge $(p, p')$ is on the CH because...
        output $(p, p')$ as CH edge
        $p = p'$
  • until $p == p_0$
```

Questions:

1. Run Gift Wrapping on a set of points and check how it works.
2. Consider a set of points that includes *degenerate cases*, that is, cases that may need to be handled separately (usually collinear points, identical points, concurent lines, etc).
3. What is the running time of Algorithm Gift Wrapping? Express the running time as function of n (input size) and k, where k is the output size (in this case, the size of the CH).

   Note: An algorithm whose running time depends on the output size is called an *output-sensitive* algorithm.
4. How big/small can k be for a set of n points? Show examples.
5. What are the best and worst-case bounds for Gift Wrapping?
6. When is GiftWrapping is a good choice?

Algorithm: QuickHull

Idea: Similar to Quicksort. Partition the points carefully, then recurse.

```
Algorithm QuickHull (input: points $P$)

    • Find left-most point $a$ and right-most point $b$

    • Partition $P$ into $P_1$ (points left of $ab$) and $P_2$ (points right of $ab$)

    • return QuickHull($a, b, P_1$) + QuickHull($b, a, P_2$)

QuickHull($a, b, P$)
//P is a set of points all left of $ab$
//returns the upper hull of $P$

    • if $P$ is empty: return emptyset

    • for each point $p \in P$: compute its distance to $ab$

    • let $c$ be the point with max distance

    • let $P_1$ = points to the left of $ac$

    • let $P_2$ = points to the left of $cb$

    • return QuickHull($a, c, P_1$) + c + QuickHull($c, b, P_2$)
```

Questions:

1. Run QuickHull and check how it works.
2. Consider a set of points that includes *degenerate cases*, that is, cases that may need to be handled separately (usually collinear points, identical points, concurent lines, etc).
3. Write a recurrence for its running time.
4. What is the best case running time of QuickHull, and when might it happen?
5. What is the worst case running time of QuickHull, and when might it happen?
6. Consider the caase when the points are uniformly distributed. What is the average/expected complexity of Quickhull in this case?

Algorithm: Graham scan

Idea: start from a point p interior to the hull. Order all points by their ccw angle wrt p and traverse them. Maintain the CH of all points traversed so far and add the next point to it.

```
Algorithm GrahamScan (input: points $P$)

  • Find interior point $p_0$ (instead of an interior point, can pick the lowest point)
  • Sort all other points ccw around $p_0$; denote them $p_1, p_2, ...p_{n-1}$ in this order.
  • Initialize stack $S = (p_2, p_1)$
  • for i = 3 to n-1 do
      – if $p_i$ is left of (second(S), first(S)): push $p_i$ on $S$
      – else:
          * repeat: pop $S$ while $p_i$ is right of (second(S), first(S))
          * push $p_i$ on $S$
```

Questions:

1. Run Graham-scan on a small set of points and check how it works.
2. Degenerate cases: Consider a set of points that includes *degenerate cases*, that is, cases that may need to be handled separately (usually collinear points, identical points, concurent lines, etc). Consider the details you need to add to make the algorithm work in these cases.
3. Argue that once the points are sorted, the algorithm takes linear time.
4. What is the overall running time of Graham Scan? Is the algorithm output sensitive?