Algorithms and Data Structures 2020/21
Week 10 tutorial solutions

1. Given a point $p_0 = (x_0, y_0)$, the *right horizontal ray* from p_0 is the set of points $\{p = (x, y_0) : x \geq x_0\}$, that is, it is the set of points due right of p_0. Show how to determine whether a right horizontal ray from a given p_0 intersects a line segment $\overline{p_1 p_2}$ in $O(1)$ time, by reducing the problem to that of two line segments intersecting.

   Answer: This one is not too difficult. It depends on a simple observation - the line segment $\overline{p_1 p_2}$ is bounded in space. Hence if the right horizontal line from p_0 is to intersect $\overline{p_1 p_2}$, it must achieve this intersection within the 'bounding box' for $\overline{p_1 p_2}$. Let $p_0 = (x_0, y_0)$, $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$. We only need to consider the right horizontal line from p_0 between x_1 and x_2. Hence we define $p_3 = (x_1, y_0)$ and $p_4 = (x_2, y_0)$. Then we can test for intersection of $\overline{p_1 p_2}$ with the horizontal ray by testing whether $\overline{p_1 p_2}$ intersects $\overline{p_3 p_4}$.

2. Show how to determine in $O(n^2 \lg n)$ time whether any three points in a set of n points are co-linear.

   Answer: This one depends on observing that *if* three points are co-linear, and we draw them in a line, then one of the three points must be lowest - i.e., have the smallest y-value (if there are ties, we take the leftmost such point). Suppose p, q, r are co-linear, and p is the lowest. Then the polar angle of q wrt. p is the same as the polar angle of r wrt. p.

   Our algorithm is as follows:
   Choose every point p in turn, to act as the origin.
   Then sort all of the other points in terms of their *polar angle* wrt. p.
   Finally do a linear search of the sorted points - if any two neighboring points in the sort have equal polar angles ($\bmod \pi$), they are co-linear with p, and we can terminate with **yes**.
   Repeat until all points have been considered as the origin.

   Sorting by polar angle is not too difficult. We can use a standard sorting algorithm such as MergeSort, except that we must change the comparison operator. That can be done by observing that we can compare the polar angles of q, r wrt. p by considering the vectors $\overrightarrow{pq}$ and $\overrightarrow{pr}$. By definition, the vector $\overrightarrow{pr}$ is anti-clockwise of $\overrightarrow{pq}$ iff $(r - p) \times (q - p) < 0$. This is *almost* enough to order r and q by polar angle wrt. p. First suppose the the y-coordinate of $q - p$ is non-negative:
   If $(r - p) \times (q - p) < 0$, then r has greater polar angle wrt. p than q;
   alternatively if $(r - p) \times (q - p) > 0$, then q has polar angle greater than r *if and only if* $r - p$ has a non-negative y-coordinate.
   If $(r - p) \times (q - p) = 0$, then we need to check the y coordinate of $r - p$.
   We can do similar case analysis for $q - p$ having a negative y-coordinate.

This polar-angle comparison operator can (clearly) be evaluated in constant time for any pair q, r. Hence by plugging the new comparison into MergeSort (say), we can sort polar angles wrt. p in $\Theta(n \lg n)$ time. Finally in a linear walk through the array we can search for any pair of neighbouring points q, q' such that $(q-p)\times(q'-p) = 0$.

Notice that in the last phase (where we do a linear scan looking for neighbours with 0 cross product) we are forgetting to check for q, q' which are a polar angle π apart. This does not hurt our algorithm - even if it is the case that the 3 colinear points are such that p lies *between* q and q', we will *observe* the co-linearity whenever we choose q or q' as our base point, whichever happens first.

3. In the *online convex hull problem*, we are given the set Q of n points one point at a time. After receiving each point, we are to compute the convex hull of the points seen so far. Obviously, we could run Graham's scan once for each point, with a total running time of $O(n^2 \lg n)$. Show how to improve this slightly, by showing we can solve the online convex hull problem in $O(n^2)$.

   This is Ex. 33.3-5 of [CLRS]. Ex. 35.3-5 of [CLR].

   Answer: It is good to first think about why the naïve algorithm (re-running Graham's scan each time a point is added) will only lead to the bound $O(n^2 \lg(n))$. This is because, if Graham's scan is $O(n \lg(n))$, this means there is some $c \geq 0$ such that the running time of Graham's scan is $\leq cn \lg(n)$ for sufficiently large n. This gives us the following upper bound

$$\sum_{k=3}^{n} ck \lg(n)$$

   of the running time of the naïve online algorithm. Taking only the terms from $k = n/2$ to $k = n$, this would give us an expression as large as $\frac{n^2}{4}(\lg(n) - 1)$ which is of the form $\Theta(n^2 \lg(n))$. So we need to work harder...

   In coming up with our better online algorithm, note that the points $p_1, \ldots, p_n$ may arrive in any order, the ordering of them does not imply any geometric relationship (it is important to remember this, to distinguish from the scenario of the standard Convex Hull algorithm, where we sort all points together first, and then access points in order of polar angle)

   For our improved algorithm, we assume that the convex hull of a set of points is *represented* by its points presented in *anti-clockwise* order, starting with the *bottom-most* point (if there is more than one point with this y-coordinate, then take the left-most of these). In the online setting, we will consider a number of such "convex hulls', as points are added.

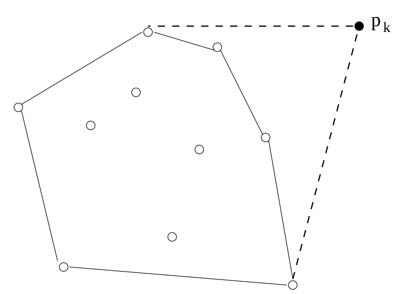
   The naïve algorithm was defined by running the convex hull algorithm again every time a new point was added - this having the time-bound $O(k \lg(k))$ for the update for the kth point.

Our better algorithm will just use $O(k)$ work to update the convex hull for the k-th point. We will not *need* to do the sorting of points (which is what drives the $O(n \lg(n))$ running time of Graham's scan) when we are just adding a new extra point to a current convex hull. This is because if we have the convex hull $C(k-1)$ of the points $\{p_1, \ldots, p_{k-1}\}$, then this lists the points of that hull in counterclockwise order from the bottom-most (and left-most if breaking ties) point in that set.

Now consider the work we need to do in updating the convex hull $C(k-1)$ to obtain $C(k)$, i.e., to consider the change on adding p_k.

- First suppose p_k sits in the interior of $C(k-1)$. Then $C(k)$ is equal to $C(k-1)$. We can detect this case by the fact that if $p_{1,k-1}, \ldots, p_{\ell(k-1),k-1}$ are the points of $C(k-1)$ in anti-clockwise order from the lowest one $p_{1,k-1}$, then for every $1 \leq i \leq \ell(k-1)$, $p_{i,k-1} \to p_{i+1,k-1} \to p_k$ involves a *left-turn* (note we perform $i+1$ in a wrap around fashion, with $p_{\ell(k-1)+1,k-1} =_{\mathrm{def}} p_{1,k-1}$)
- Alternatively suppose that p_k is on the exterior of $C(k-1)$. Then the convex hull consists of p_k plus one continuous segment from $C(k-1)$ - usually this segment will consist of most of the existing points on the convex hull $C(k-1)$. In this scenario, the left/right turns of the convex hull $C(k-1)$ wrt. p_k can be characterised as follows:
  - In considering the points of $C(k-1)$ from the bottom-most point $p_{1,k-1}$ onwards, let i be the first point such that $p_{i,k-1} \to p_{i+1,k-1} \to p_k$ involves a *right turn*.
  - Subsequent to finding the i defined above, let j be the first index *after* i such that $p_{j,k-1} \to p_{j+1,k-1} \to p_k$ involves a *left turn*.
  - It is easy to argue that $p_{h,k-1} \to p_{h+1,k-1} \to p_k$ is always a left turn from $h = j$ all the way round to $h = i-1$.
  - The convex hull $C(k)$ is equal to $p_{1,k-1} \ldots, p_{i,k-1}, p_k, p_{j,k-1} \ldots, p_{\ell(k-1),k-1}$.

The following picture is a good reference point for the discussion.



Now we observe that the updating of $C(k-1)$ to become $C(k)$ just involves a linear scan of the existing points (of which there are at most $k-1$) on $C(k-1)$, with the "work done" in considering each point, being just constant-time (doing the left-turn/right-turn test). Overall the update for the new point p_k takes $O(n)$ time.