Amy Bui
HW 1

1 Left Turn and Convexity

(a) Derive and verify that, for the given points $A = (x_a, y_a)$, $B = (x_b, y_b)$, and $C = (x_c, y_c)$ (in this order), the sign of the determinant is positive if A, B, and C appear in counterclockwise (ccw). The determinant can be calculated by:

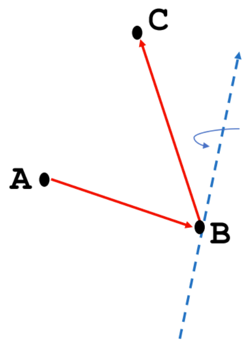
$$\begin{aligned} D = \begin{vmatrix} x_a & y_a & 1 \\ x_b & y_b & 1 \\ x_c & y_c & 1 \end{vmatrix} &= x_a \begin{vmatrix} y_b & 1 \\ y_c & 1 \end{vmatrix} - y_a \begin{vmatrix} x_b & 1 \\ x_c & 1 \end{vmatrix} + 1 \begin{vmatrix} x_b & y_b \\ x_c & y_c \end{vmatrix} \\ &= (x_a y_b - x_a y_c) + (x_c y_a - x_b y_a) + (x_b y_c - x_c y_b) \end{aligned}$$

Notice the result is the cross product of the two vectors $\overrightarrow{AB} = \langle x_b - x_a, y_b - y_a, 0\rangle$ and $\overrightarrow{BC} = \langle x_c - x_b, y_c - y_b, 0\rangle$, i.e. $\overrightarrow{AB} \times \overrightarrow{BC}$:

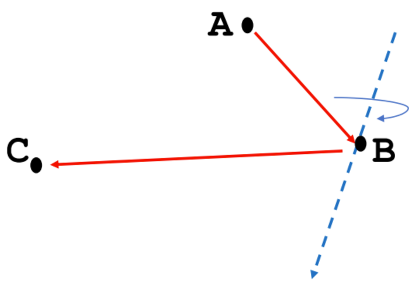
$$\begin{aligned} \overrightarrow{AB} \times \overrightarrow{BC} = \begin{vmatrix} i & j & k \\ x_b - x_a & y_b - y_a & 0 \\ x_c - x_b & y_c - y_b & 0 \end{vmatrix} \\ = \big[(x_a y_b - x_a y_c) + (x_c y_a - x_b y_a) + (x_b y_c - x_c y_b)\big] k \end{aligned}$$

By the right-hand rule convention, when the cross product is positive, the direction along the path from $A \to B \to C \to A$ of the boundary of ΔABC is ccw, where our C is to the "left" of directed line $\overrightarrow{AB}$, because the direction of the vector perpendicular to the two vectors is in the positive direction (Fig. 1a); when the cross product is negative, the direction along the path from $A \to B \to C \to A$ of the boundary of ΔABC is clockwise, because the direction of the vector perpendicular to the two vectors is in the negative direction (Fig. 1b). Therefore, the sign of the determinant can tell us when three points are given in ccw or cw order.

Reference: [1] [2]



(a) Positive cross product



(b) negative cross product

Figure 1: Visualization of $\overrightarrow{AB} \times \overrightarrow{BC}$

(b) Describe an algorithm to test if polygon P (given by a circularly linked list of its n vertices in order) is convex.

From class, a convex polygon is such that, walking along the boundary of the polygon, all angles "turn" in the same "direction", resulting in all internal angles being less than 180° and only consecutive segments intersecting.

*Let there exist a method that outputs the sign of the determinant of three given coordinate points, and call it DeterminantSign. Assume it runs $O(1)$ time by plugging in the coordinate values in the formula(s) like those mentioned in part (a).

Input: A circularly linked list of the n ordered vertices of polygon P. We will call these vertices in their sequence $p_1, p_2, ..., p_n$, where p_1 is arbitrarily selected as the "first" vertex we examine in the algorithm below, p_2 is the "next" node/vertex of p_1, etc.

Output: **true** or **false** if P is convex.

```
isConvex($P$):

1. $A \leftarrow p_1$; $B \leftarrow p_2$; $C \leftarrow p_3$          $\Theta(1)$
2. direction $\leftarrow$ DeterminantSign($A, B, C$)          $\Theta(1)$
3. for $i \leftarrow 2$ to $n$          $\Theta(n)$
4.         $A \leftarrow B$; $B \leftarrow C$; $C \leftarrow C$.next          $\Theta(1)$
5.         turn $\leftarrow$ DeterminantSign($A, B, C$)          $\Theta(1)$
6.         if turn $\neq$ direction then return false
7. return true
```

Correctness : The algorithm examines each adjacent angle of the polygon formed by three adjacent points. The direction (in terms of left/right or positive/negative) of the first arbitrary (interior) angle chosen is given by direction (line 2) and formed from 3 vertices. I show by induction that this algorithm works for all polygons of $k \leq n$ vertices, where $3 \leq k \leq n$.

For $k = 3$ vertices (triangle), it is trivially true that P is a convex polygon, and the direction of the k angles match and all are acute. For $k = \{3, ..., i\}$, let all the directions of the angles up to the angle about p_{i-1} (i.e. $\angle p_{i-2}, p_{i-1}, p_i$) match the first, thus far, and therefore $\{p_1 ... p_i\}$ is convex. The next angle formed with the $i \rightarrow \mathsf{next}^{th}$ vertex, given by $\angle p_{i-1}, p_i, p_{i \rightarrow \mathsf{next}}$, will have a turn that either does or doesn't match direction. If it matches, P is convex because all turns are the same. If it doesn't, P is not convex because the angle makes it either a concave polygon or complex polygon.

Complexity : Let accessing the next adjacent node in the linked list be a $O(1)$ time operation. Each line in the algorithm runs $O(1)$ time except for the for loop in lines 3 - 6. The for loop examines the "turn" of only each adjacent triple on the list of vertices. Therefore, the for loop runs $O(n)$ time. We are not concerned with any sorting of the vertices here. Therefore, the total runtime of the algorithm is $O(n)$.

2 Unordered Divide-and-Conquer Convex Hull

(a) For P_1 and P_2 to share a support line, the line must have at least one vertex from each polygon on it, and all the vertices of both polygons must be to one side of the line. If P_1 and P_2 share no vertices, then they are either 1) completely disjointed, 2) one is completely interior to the other, or 3) their edges intersect.

If they are **disjointed**, then P_1 and P_2 can only have 2 support lines in common, both of which would create the new edges of the convex hull created from $P_1 \cup P_2$, regardless of the values of n_1 and n_2. In other words, each vertex of P_1 has 1 or 0 shared support line with P_2. (Fig. 2)

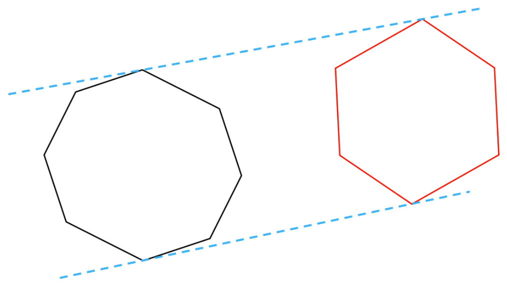


Figure 2: Shared support lines (blue) of two disjointed polygons.

If one is **interior** to the other, regardless of the values of n_1 and n_2, then there are 0 support lines, because any support lines the interior polygon has would intersect the exterior polygon, and any support lines the exterior polygon has would never share a vertex from the interior polygon. (Fig. 3)

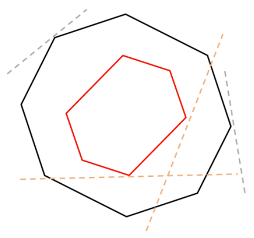
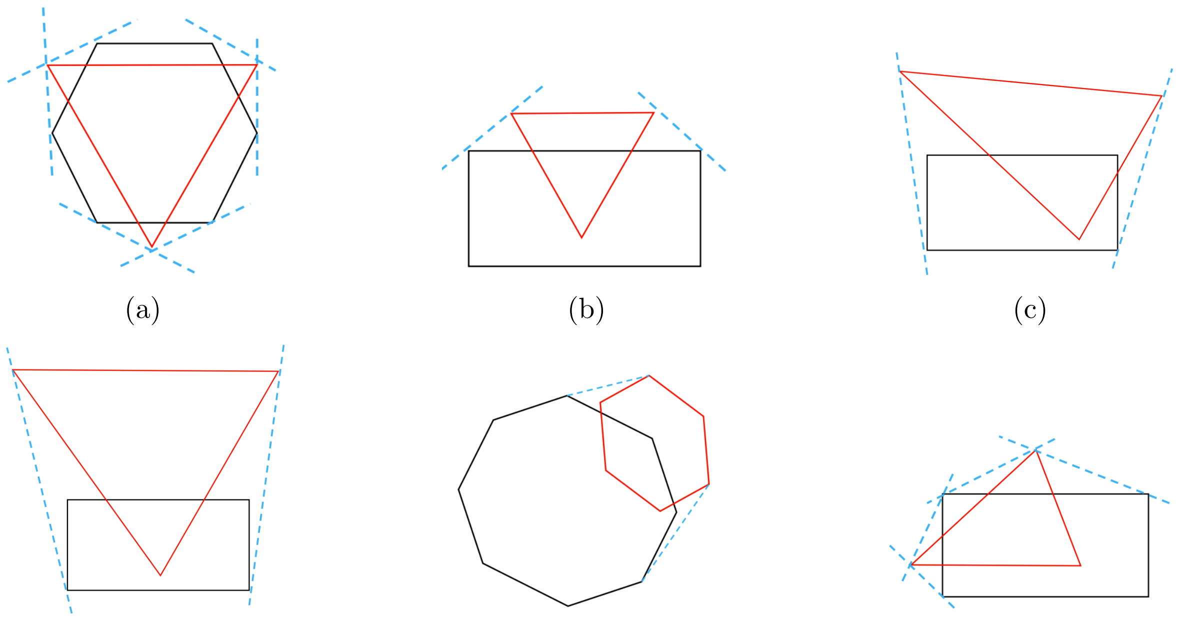


Figure 3: Two polygons with no shared support lines. The exterior polygon's support lines (grey) never touch the interior polygon, and the interior polygon's support lines (orange) bisects the exterior polygon.

Finally, the most number of support lines that can occurr at one vertex is 2, because that is the number of edges a vertex has that could intersect the other polygon. It may have 0 shared support lines in the case that one vertex of P_1 is interior to P_2; it may have 1 shared support line in the case that only one of the edges formed intersects the other polygon, then one of the points of that edge will share a support line with one of the points in the other polygon (Fig. 4b - 4f). Or it can have 2 shared support lines when both edges of the vertex intersects edges of the other polygon (Fig. 4a and 4f). This can happen at most $2 \times \texttt{min}(n_1, n_2)$, limited by the number of vertices of the lesser polygon and depending on how the two polygons overlap.

Therefore, the number of common support lines of P_1 and P_2 is at most $2 \times \texttt{min}(n_1, n_2)$.



(a) (b) (c)

(d) (e) (f)

Figure 4: Possible shared support lines (blue) formed in various ways two polygons overlap. The red polygon indicates it has fewer or equal number of vertices than the black polygon.

Reference: [3]

(b) I specify an algorithm that computes the convex hull of convex polygons $P \cup Q$. Since they are already known to be convex, their vertices are already sorted.

Input: The sorted vertices of convex polygons P and Q, ordered respectively by $\{p_1, ..., p_n\}$ and $\{q_1, ..., q_m\}$. Note that $|P| = n$ and $|Q| = m$, and let $N = n + m$.

Output: $\mathcal{CH}(P \cup Q)$

```
MergeConvexHulls($P$, $Q$):

  1. $X \leftarrow$ Merge $P$ and $Q$.          $\Theta(N)$
  2. Find the convex hull of sorted $X$ using any convex hull finding algorithm on a
     presorted list. I suggest GrahamScan.
     So $Y \leftarrow$ GrahamScan($X$)          $\Theta(N)$
  3. return $Y$
```

The runtime is $O(N)$ because it takes $O(N)$ time to sort the already sorted elements of P and Q into one list X. And then we are running Graham Scan on a presorted list, which itself is then $O(N)$ time.

(c) I specify a divide-and-conquer algorithm that uses MergeConvexHulls without presorting to find the convex hull of an arbitrary set of n points in set S.

```
DC-ConvexHull($S$):

  1. $m \leftarrow \frac{n}{2}$
  2. $L \leftarrow S[0 : m - 1]$ and $R \leftarrow S[m : n - 1]$
  3. $L' \leftarrow$ DC-ConvexHull($L$)
  4. $R' \leftarrow$ DC-ConvexHull($R$)
  5. $S' \leftarrow$ MergeConvexHulls($L'$, $R'$)          $\Theta(n)$
  6. return $S'$
```

The runtime is $O(n \log n)$ because it divides S in half to find the convex hulls of the separated set by making recursive call to the divide-amd-conquer algorithm, and takes linear time to "merge" the two convex hulls/polygons that were found, which results in the convex hull formed by the two convex polygons.

3 Three

Given a set S of n planar points in general position and in arbitrary order, I construct these polygons with the following algorithms. Any figure is meant to be on xy-plane.

(a) A simple monotone polygon R whose vertices are exactly the vertices of S, which contains the line segment given by the min and max x-coord vertices of S.

MonotoneHull(S):

```
1. $S' \leftarrow$ sort $S$ by x-coordinate. $\Theta(n \log n)$
2. Create the line segment ($y = mx + b$) from the first and last vertices in $S'$ (min
   and max x-coord vertics of $S$). Let $b$ be this line's y-intersection, and $m$ the slope.
3. for $i \leftarrow 0$ to $n$        $\Theta(n)$
4.       $B[i] \leftarrow y_{S'[i]} - mx_{S'[i]}$
         (the y-insersection of the line given by
         the point $S'[i]$ and the slope $m$)
5. $j \leftarrow 1$, $R[0] \leftarrow S'[0]$
6. for $i \leftarrow 1$ to $n$        $\Theta(n)$
7.       if $B[i] \geq b$
8.          then $R[j] \leftarrow S'[i]$, $j \leftarrow j + 1$
9. for $i \leftarrow n - 1$ downto 0        $\Theta(n)$
10.      if $B[i] < b$
11.         then $R[j] \leftarrow S'[i]$, $j \leftarrow j + 1$
12. return $R$
```

This algorithm runs $\Theta(n \log n)$ time due to soring the points by the x-coordinates (line 1). We can find, in constant time, the slope and y-intersection of the line segment **L** given by the min and max x-coord points in S' (line 2, Fig. 5). Using that m and b, we can create a list of the y-intercepts of each point in S' on a line of slope m, which is linear time. This tells us whether or not a point in S is above or below the main line segment (Fig. 6). Finally, we compute R by finding the list of points above the line segment in order of the sorted S' (loop 6), and adding to it the points below the line segment (found the same way in reverse order and discarding any duplicate end points) (loop 9, Fig. 6), this runs in linear time. (Alternatively, if for each point we found above **L** were removed from S', we can simply reverse and append the remaining S' to R). The resulting polygon is monotone with respects to our line segment **L**, because all lines perpendicular to **L** intersects R at most twice (Fig. 7).

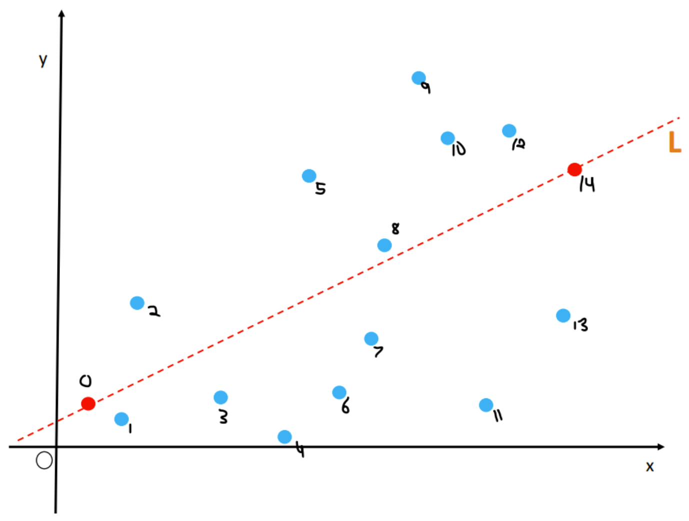


Figure 5: Set S of n points, labeled in order sorted by their x-coordinates. Line **L** is given by the min and max x-coordinate points in S.

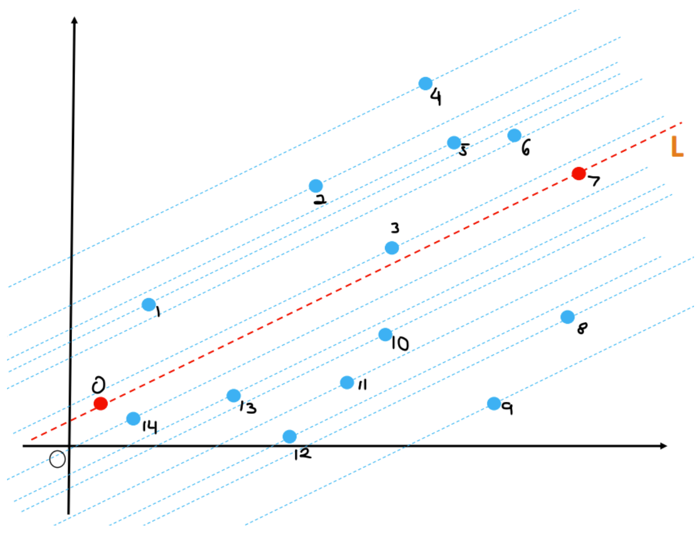


Figure 6: Every point in S on a line with slope m is sorted by their y-intercepts above **L** (ascending x-coord order) and below **L** (descending x-coord order).

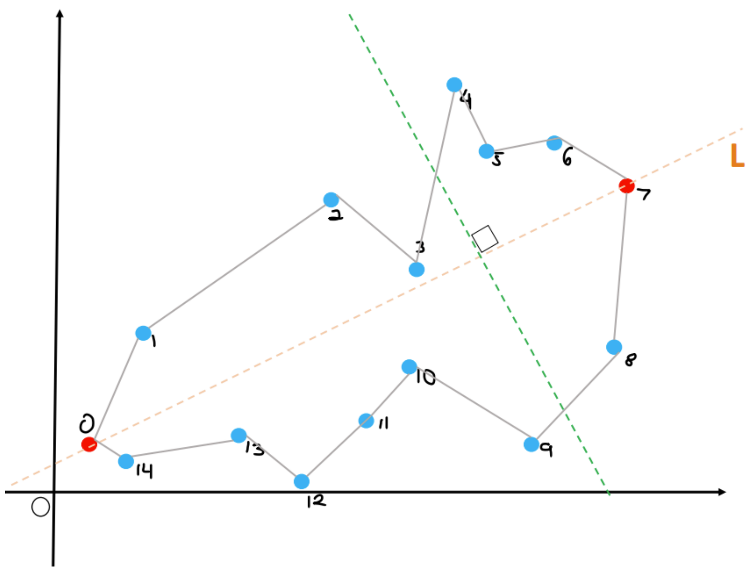


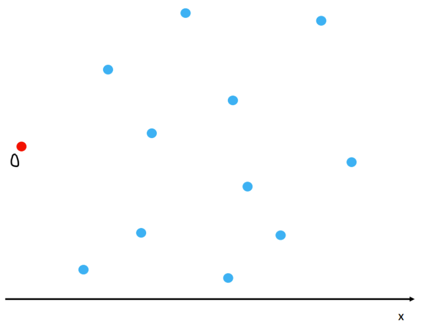
Figure 7: Resulting monotone polygon R from set S. Every line orthongonal to **L**, such as the green line, intersects the polygon at most twice.

(b) A simple starshaped polygon P whose vertices are exactly the points in S for which the min x-coord point in S "sees" every point in P.

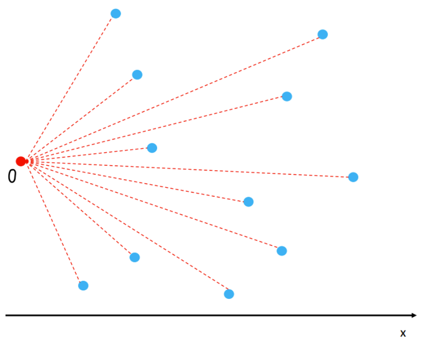
StarHull(S):

```
1. $p_0 \leftarrow$ smallest x-coordinate point in $S$.          $\Theta(n)$
2. for $i \leftarrow 0$ to $n-1$          $\Theta(n)$
3.         $p_i \leftarrow S[i]$
4.         if $p_i = p_0$
5.             then $M[i] \leftarrow \infty$
6.             else $M[i] \leftarrow$ slope $m$ of line $\overline{p_0 p_i}$,          $m = \frac{y_i - y_0}{x_i - x_0}$
7. $P \leftarrow$ sort $S$ by their slope values in $M$ (decending).          $\Theta(n \log n)$
8. return $P$
```

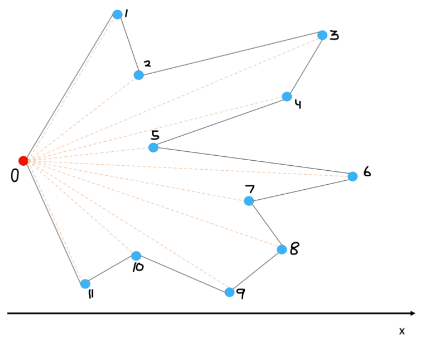
This algorithm runs $\Theta(n \log n)$ time due to sorting the points in S by their assigned slopes with regards to p_0 in M (line 7). Sorting the points by their slopes ensures that the smallest x-coordinate point in S "sees" every point in the resulting starshaped polygon P, because every line segment created by p_0 and a point in P would not intersect another edge (Fig. 8). I prove this by induction on the size of S.



(a) The smallest x-coord point in set S is the leftmost. This point is called p_0.



(b) Compute the slope of all line segments from p_0 to all other points in S.



(c) Sorting the vertices by the slope values. p_0 is a kernal of star P.

Figure 8

For a set S where $n = 3$ (a triangle), it is trivially a starshaped polygon and the edges of the leftmost vertex (p_0) has edges to the other two points. p_1 is given by the point that forms the greater slope with p_0. p_2 is given by the remaining point and is located "below" p_1 because the slope of $\overline{p_0p_2}$ is less than the slope of $\overline{p_0p_1}$, and they do not intersect inside the triangle (Fig. 9).

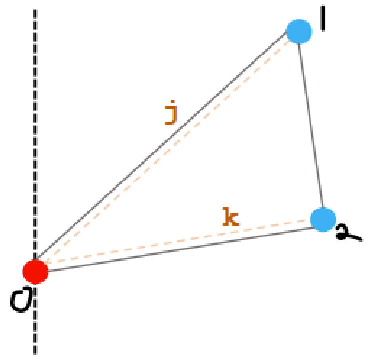
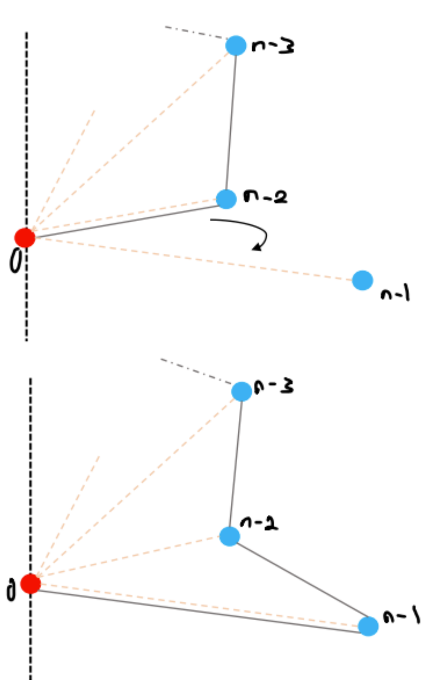
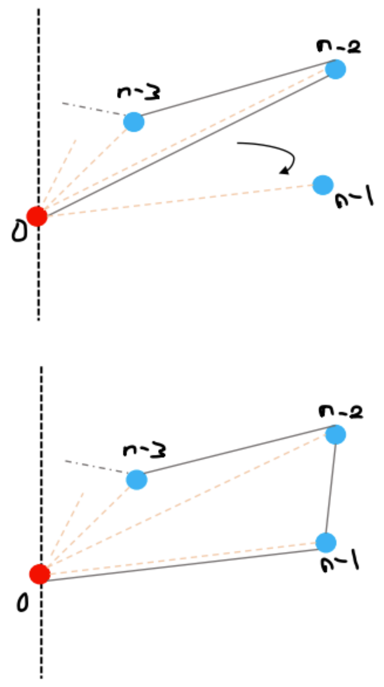


Figure 9: Slopes are labeled in orange dashed lines. The dashed black line is the vertical line given by the x-coord of p_0. Slope j is greater than slope k. Vertices then ordered $\{0, 1, 2\}$, and creates a star shape, and all points, not just p_0, are kernals.

Let the algorithm hold for set S of size n for the first the $n-1$ vertices ($\{p_0, ..., p_{n-2}\} \in P$), such that all the line segments created from p_0 to every other point in P is also in P. Since the first $n - 1$ vertices of P have already been found, that means they are ordered by their slopes with p_0. Then, the n^{th} point of the set S must appear below the last point in P because the slope of $\overline{p_0p_{n-1}}$ is less than the slope of $\overline{p_0p_{n-2}}$. The new edges of polygon P formed by $\overline{p_{n-2}p_{n-1}}$ and $\overline{p_0p_{n-1}}$ replace the previous edge of $\overline{p_0p_{n-2}}$. Since $\overline{p_0p_{n-1}}$ does not intersect $\overline{p_0p_{n-2}}$ execpt at p_0, the line segment given by $\overline{p_0p_{n-2}}$ still remains inside of polygon P (Fig 10).



(a) A left turn to the next smaller slope point. The kernal still sees all other previous points.



(b) a right turn to the next smaller slope point. The kernal still sees all other previous points.

Figure 10: The turn to the next point doesn't matter if the points are sorted by the slopes from p_0. The kernal still sees all previous points if the next point in the set has the next smaller slope.

The n^{th} point cannot lie "above" the $n-1^{th}$ point, because that means the slope of $\overline{p_0 p_{n-1}}$ was greater than the slope of $\overline{p_0 p_{n-2}}$. By contradiction, p_{n-1} and p_{n-2} were not in correct sorted order.

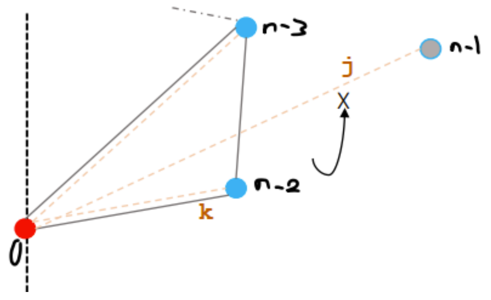
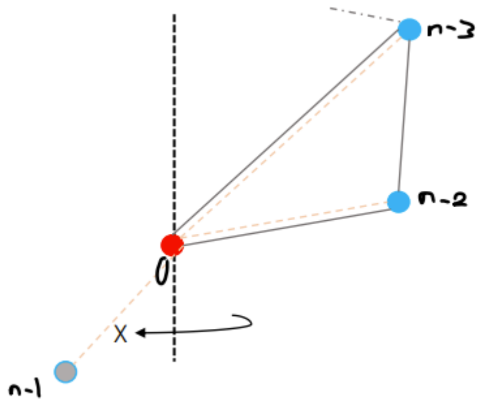
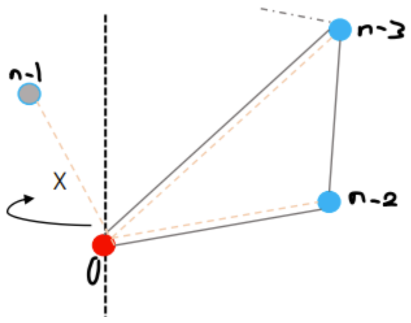


Figure 11: Slope j is greater than slope k. Therefore the above points are out of order.

No points can be "behind" p_0, i.e. have a smaller x-coord value than p_0, because p_0 was determined to be the smallest x-coord point in S (Fig. 12).



(a) The smallest x-coord point in set S is the left-most. This point is called p_0.



(b) Compute the slope of all line segments from p_0 to all other points in S.

Figure 12

(c) For a set of points Q such that $\forall q \in Q$, $q \notin S$, describe algorithms to determine if q is interior or exterior to P and R.

I assume that R is ordered as described above in part (a), where the first point stored is the min x-coord point, and the rest of the points are ordered going up and clockwise for the polygon. Let $|R| = n$. I give a $\Theta(\log n)$ time algorithm:

IntOrExtToMono(q, R):

```
1.  $r_{\text{min-x}} \leftarrow R[0]$
2.  $r_{\text{max-x}} \leftarrow$ max x-coord of $R$;    max $\leftarrow$ index of $r_{\text{max-x}}$ in $R$
3.  **if** $x_q < x_{\min}$ **then return** exterior
4.  **if** $x_q > x_{\max}$ **then return** exterior
5.  $m \leftarrow$ slope of $\overline{r_{\text{min-x}} r_{\text{max-x}}}$ (line segment $L$);    $b \leftarrow$ y-intercept of $\overline{r_{\text{min-x}} r_{\text{max-x}}}$
6.  $b_q \leftarrow$ y-intercept of the line given by point $q$ and slope $m$.
7.  **if** $b_q > b$ **then**
8.      left $\leftarrow 0$; right $\leftarrow$ max; $i \leftarrow \frac{\texttt{left}+\texttt{right}}{2}$
9.      **while not** $(x_{R[i]} < x_q$ **and** $x_q < x_{R[i+1]})$
10.         **if** $x_q < x_{R[i]}$ **then** right $\leftarrow i - 1$
11.         **else** left $\leftarrow i + 1$;
12.         $i \leftarrow \frac{\texttt{left}+\texttt{right}}{2}$
13.     $j \leftarrow R[i]$; $k \leftarrow R[i+1]$
14.     dir $\leftarrow$ Determinant$(k, q, j)$
15.     **if** dir $> 0$ **then return** exterior    (*above L, and ccw with neighbors*)
16.     **else return** interior    (*above L, and cw with neighbors*)
17. **else** left $\leftarrow$ max; right $\leftarrow n - 1$; $i \leftarrow \frac{\texttt{left}+\texttt{right}}{2}$
18.     **while not** $(x_{R[i]} > x_q$ **and** $x_q > x_{R[i+1]})$
19.         **if** $x_q > x_{R[i]}$ **then** right $\leftarrow i - 1$
20.         **else** left $\leftarrow i + 1$;
21.         $i \leftarrow \frac{\texttt{left}+\texttt{right}}{2}$
22.     $j \leftarrow R[i+1]$; $k \leftarrow R[i]$
23.     dir $\leftarrow$ Determinant$(k, q, j)$
24.     **if** dir $> 0$ **then return** interior    (*below L, and ccw with neighbors*)
25.     **else return** exterior    (*below L, and cw with neighbors*)
```

Supposing that finding the x-min and x-max coordinate point in sorted polygon R is constant time, then this algorithm runs $\Theta(\log n)$ time for a point q. This works by 1) detecting in constant time if point q is above or below the line segment L given by the min and max points by comparing the y-intercepts; 2) Finding the two adjacent points q is located between by their x-coordinates. This is done in logarithmic time because detecting above/below L means half or a part of the points of R doesn't have to be checked. Then we run something similar to binary search to find q's x-coord neighbors (see while loops in line 9 (above L) and line 18 (below L)). Finally, using q and its neighbors j and k, we can use the formula in question 1 to calculate the determinant of the three points to see the whether or not they are ordered in a ccw (positive determinant) or cw (negative determinant) manner (I pick the order starting with the point with the highest x-coord of the three, k). If q is above L (line 8-16), then a ccw orientation with its neighbors means it is exterior to R (Fig. 13.a), otherwise if it is above L and cw, it is interior (Fig. 13.b). If q is below L (line 17-25), then a ccw orientation with its neighbors means it is interior to R (Fig. 13.c), otherwise it is exterior (Fig. 13.d). And finally, q is clearly exterior if it's x-coord is less than the min's, or if it is greater than the max's (Fig. 13.e). Running this algorithm for a set of points Q of size m would then be time $\Theta(m \log n)$.

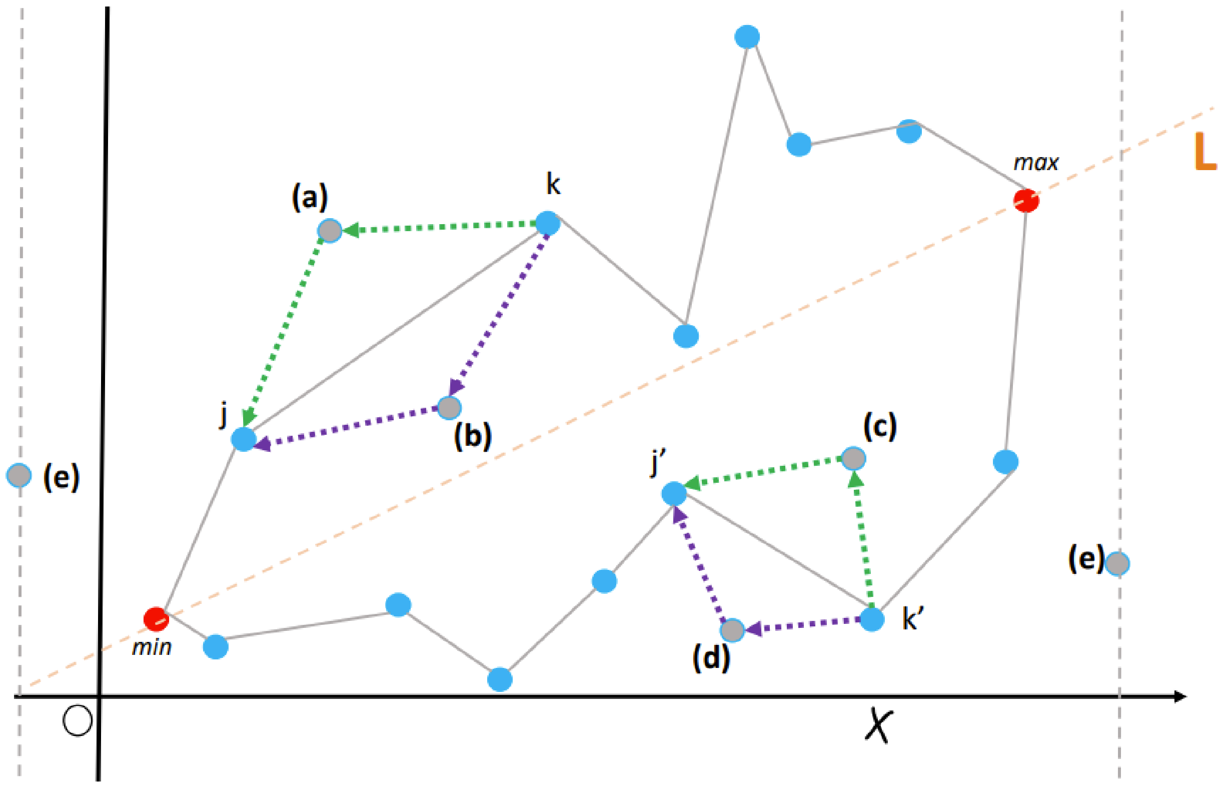


Figure 13: Modified monotone polygon R from Fig. 7, annotated to demonstrate whether a point q (here $q \in \{a, b, c, d, e, e\}$) that is not in R, is interior or exterior to R. See discussion above.

I assume that P is ordered as described in part (b), where the first point stored is the min x-coord point, and the rest of the points are ordered in descreasing ordered of theirs slopes created with the line segment with the min point (i.e. clockwise order). Let $|P| = n$. I give a $\Theta(\log n)$ time algorithm:

```
IntOrExtToStar($q$, $P$):

 1. $p_0 \leftarrow P[0]$
 2. if $x_q < x_{p_0}$ then return exterior
 3. $m_q \leftarrow$ slope of line segment $\overline{p_0 q}$
    $m_1 \leftarrow$ slope of line segment $\overline{p_0 p_1}$
    $m_{n-1} \leftarrow$ slope of line segment $\overline{p_0 p_{n-1}}$
 4. if $m_q > m_1$ or $m_q < m_{n-1}$ then return exterior
 5. left $\leftarrow 0$ ; right $\leftarrow n-1$; $i \leftarrow \frac{\texttt{left}+\texttt{right}}{2}$;
    $m_i \leftarrow$ slope of line segment $\overline{p_0 p_i}$;
    $m_{i+1} \leftarrow$ slope of line segment $\overline{p_0 p_{i+1}}$
 6. while not $(m_i > m_q$ and $m_q > m_{i+1})$
 7.         if $m_q > m_i$ then right $\leftarrow i - 1$
 8.         else left $\leftarrow i + 1$
 9.         $i \leftarrow \frac{\texttt{left}+\texttt{right}}{2}$
10. $j \leftarrow P[i]$
    $k \leftarrow P[i+1]$
11. dir $\leftarrow$ Determinant$(k, q, j)$
12. if dir $> 0$ then return exterior          (ccw, exterior to $P$)
13. else return interior          (cw, interior to $P$)
```

This algorithms runs in $\Theta(\log n)$ time. Since all points in P are sorted by their slopes relative to the x-min point of P, we can tell whether or not q is exterior or intertior by its slope with the x-min point. If q's x-coord is less than that of the min point, then it is trivially exterior to P (Fig. 14.c). If instead q's slope is greater than P's greatest slope (of line $\overline{p_0 p_1}$), or if q's slope is less than P's least slope value (of line $\overline{p_0 p_{n-1}}$), then it is also trivially exterior (Fig. 14.c). Finally, using an algorithm similar to binary search (line 5-9), we find the adjacent points in P where which q's slope falls betweek them. Since the points are already sorted by their slopes, finding the neighbors j and k takes logarithmic time. And finally, like before, we can calculate in constant time the determinant given the three points (I start with the point that has the lesser slope, i.e. k in the algorithm and figures) to determine the ordering orientation of the points. If the q creates a ccw turn with j and k (positive determinant), then it is exterior to P (Fig. 14.a). If q creates a cw turn with its neighbors (negative determinant), then it is interior to P (Fig. 14.b). Running this algorithm for a set of points Q of size m would then be time $\Theta(m \log n)$.

Reference: [3]

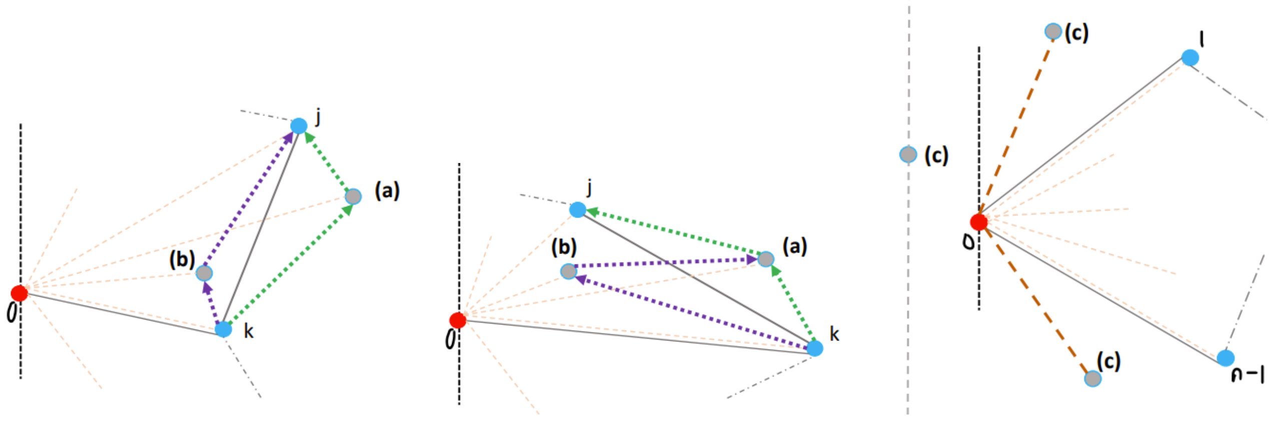


Figure 14: Modified zoomed-in view of different parts of polygon P from problem 3b, annotated to demonstrate whether a point q (here $q \in \{a, b, c\}$) that is not in P, is interior or exterior to P. Points 0 and $1, n-1, j, k$ are in P. It can be seen that for some q whose slope is between those of adjacent-slope points j, k, that q is exterior (a, green) when Δkaj is given in ccw order; conversely, q is interior (b, purple) when Δkbj is given in cw order. All other times, q is exterior, either because it is to the left of the min point 0, or its slope is greater than that of P's highest slope edge, or its slope is less than that of P's least slope edge ($q = c$).

References

[1] Cross Product of Two Vectors: https://www.cuemath.com/geometry/cross-product/

[2] Math World - Curve Orientation: https://mathworld.wolfram.com/CurveOrientation.html

[3] Jake and Diane's office hours, classmates: Stephanie, Sydney, Anju, and MacKenzie with homework problem discussions.