Assignment 1

Computational Geometry (CS60064), Group 6

Suhas Jain	19CS30048
Sarthak Johnson Prasad	18CS10049

Question 1

Given n points on 2D-plane, propose an algorithm to construct a simple polygon P with all the given points as vertices, and only those. Provide its proof of correctness and deduce its time complexity (A simple polygon is one in which no two edges intersect each other excepting possibly at their endpoints).

Answer:

Algorithm:

Let the set of points on 2D-Plane be $\mathcal{S}$ and the simple polygon to be constructed is denoted as $\mathcal{P}$. Further we denote the n vertices of polygon $\mathcal{P}$ as v_1,v_2, ..., v_n. Here $\mathcal{P}_i$ stands for polygon obtained after the execution of phases 1 through i of the algorithm. $\mathcal{CH}(\mathcal{Q})$ denotes the convex hull of set $\mathcal{Q}$. Let's come to the algorithm now. For initialisation, we select three points s_1, s_2, $s_3 \in \mathcal{S}$ such that no other point in $\mathcal{S}$ lies within $\mathcal{CH}(\{s_1, s_2, s_3\})$. Let $\mathcal{S}_1 := \mathcal{S} \setminus \{s_1, s_2, s_3\}$. On i-th iteration ($1 \leq i \leq n$ - 3) :

- **Step 1:** First we choose one point, lets name it $s_i \in \mathcal{S}_i$ at random such that no remaining point of $\mathcal{S}_{i+1} := \mathcal{S}_i \setminus \{s_1\}$ lies within the convex hull formed by the polygon formed till now and the chosen point, i.e, $\mathcal{CH}(\mathcal{P}_{i-1} \cup \{s_i\})$.
- **Step 2:** Next we find an edge on the polygon $\mathcal{P}_{i-1}$ formed till now, (v_k,v_{k+1}) that is completely visible from s_i, and we replace it with two edges (v_k,s_i) and (s_i,v_{k+1}).

The steady growth of the size of the polygon formed results in us getting the required polygon from the set of points as its vertices.

Correctness:

We proceed to prove the correctness of both Step 1 and Step 2 of our proposed algorithm. For Step 1, we claim that a point $s_i \in \mathcal{S}_i$ always exists. For example, take the point that lies closest to $\mathcal{CH}(\mathcal{P}_{i-1})$.
For Step 2, we prove that a point p which lies outside $\mathcal{P}$, lies outside $\mathcal{CH}(\mathcal{P})$ then there exists at least one edge which is completely visible from p. We can show this by induction: First, we compute the vertices of $\mathcal{CH}(\mathcal{P})$, and consider the chain from the leftmost supporting vertex and the right one.

- **Case 1:** If the chain consists of only one edge - The chain must be visible since both the vertices are visible from p.
- **Case 2:** If chain consists of multiple edges (lets say k) - We consider the leftmost edge $\mathcal{E}$. If this edge is visible, we are done. Else, consider the leftmost edge $\mathcal{E}_1$ which is in front of $\mathcal{E}$ (and faces p). The leftmost point of $\mathcal{E}_1$ must be visible from p. Thus we find at most k-1 edges whose left and right endpoints are visible from p.

Time Complexity:

When we start the algorithm we need to take a random point and find two more points which are closest to it, this step takes $O(n)$ time. After this we need to do Step 1 and Step 2 $O(n)$ number of times. In Step 1, one way to proceed is to find the point which lies closest to $\mathcal{CH}(\mathcal{P}_{i-1})$. For finding such a point we need to find distance of all the points from the $\mathcal{CH}(\mathcal{P}_{i-1})$, this takes $O(n^2)$ time. The essence of the whole algorithm is to compute all edges which are completely visible (Step 2). This also requires at most $O(n^2)$ time. Step 1 and Step 2 combined take $O(n^2)$ time. As this needs to be repeated $O(n)$ number of times, the whole algorithm requires at most $O(n^3)$ time.

Question 2

2a

A convex polygon P is given as counter-clockwise ordered sequence of n vertices, in general positions, whose locations are supplied as (x, y) co-ordinates on the x-y plane. Given a query point q, propose an algorithm to determine in $O(\log n)$ time and $O(n)$ space, including pre-processing, if any, whether or not P includes q.

Answer: The algorithm broadly involves 2 steps of computation. First is a pre-processing step which is ran only once for each polygon which is done in $O(\log n)$ time. Second is

the step where we process each query, each query is also answered in $O(log\,n)$ time. So, in total if there are Q queries they are answered in $O(Q * log\,n)$ time.

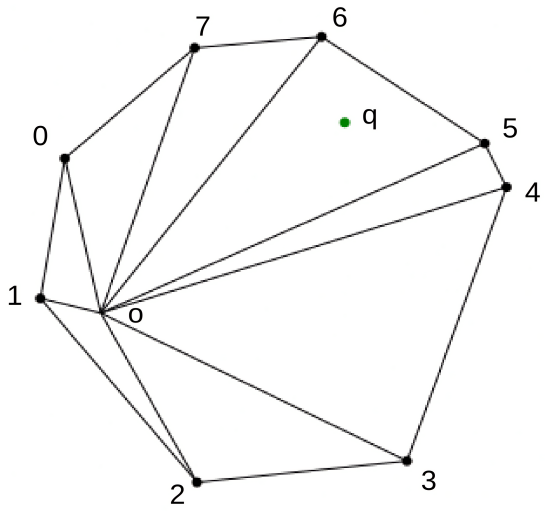

Figure 1: Sample Example Used in Explaining the Algorithm

Step 1: Pre-processing

- To run our algorithm which involves calculation of polar angles with respect to an origin, we need a point inside the polygon which we can treat as origin. To find such a point we find the centroid of triangle formed by any 3 consecutive vertices. Please note that such point will always lie inside the polygon and can be calculated in $O(1)$ time. Let the name of this point be ***o***.

- Now if we think about the array that will be obtained by calculating the polar angles of all the points of the polygon with respect to ***o*** in the sequence in which they are given, they will form a rotated sorted array. For example if print this array for the 8 sided example polygon, we get an array [1.79047, 2.89303, 5.21186, 5.81681, 0.312788, 0.431613, 0.916667, 1.24337]. In this step we need to find the index of the point which has the smallest polar angle in this array. In the given example this index will be 4 as we can clearly see in the figure also. This step can be done with the help of binary search in $O(log\,n)$ time. Let us call the index of this point as ***min_index***.
  Please Note: Before doing the binary search we do not calculate the whole array of polar angles as that would make this step $O(n)$. For binary search we need to know values of only $O(log\,n)$ points in the array, so we will calculate the polar angles only for those points while searching instead of calculating the whole array beforehand. This makes the complexity of this step $O(log\,n)$.

Step 2: Answering Each Query

- Let us call the point for which we have to answer the query as ***q***. We calculate the polar angle of ***q*** with respect to ***o***. In this step we have to find the points, between the polar angles of which polar angle of ***q*** lies. In the example the polar angle of ***q*** lies in between the points 5 and 6. We have to search for position of the point in the rotated sorted array.

- We can binary search once again with the same approach we followed earlier, where we did not calculate all the elements of the array of polar angles but just calculated the values for those angles whose values are required. As we know the ***min_index*** already this can be done in a single binary search (see code for more details on how it is implemented) which again requires polar angles of atmost $O(log\, n)$ points. Hence this step can also be executed in $O(log\, n)$ time.
  Please Note: There might be a case where the polar angle of ***q*** might be smaller than smallest polar angle in the polygon or larger than largest polar angle in the polygon. That case is handled separately in $O(1)$ time. For example, if a point lies between point 3 and point 4 in the figure.

- Now we have the segment in which ***q*** lies. Let us assume that it lies between ***i*** and ***j***. Now we can check weather ***q*** lies on left side, right side or on the line formed by ***i*** and ***j***. If it lies on left side we can say it is inside the polygon, if it is on the right side then outside the polygon otherwise it is on the polygon boundary. This can be checked in $O(1)$ time (check code for implementation details).

2b

Write a code to implement your algorithm. Construct a convex polygon with 30 vertices, and show your results for a few internal and external points.

Answer: A C++ code by the name of `assignment-1-2b.cpp` has been attached with this PDF. Comments have been added in the code to explain every step. There are also various SVG files which were generated by running the code on randomly generated polygons.
Following coloring convention has been followed for coloring each query point in the SVG:

- **GREEN** : For points inside the polygon.
- **RED** : For points outside the polygon.
- **BLUE** : For points on the boundary of the polygon.