CHAPTER 11
GEOMETRY ALGORITHM – CONVEX HULL

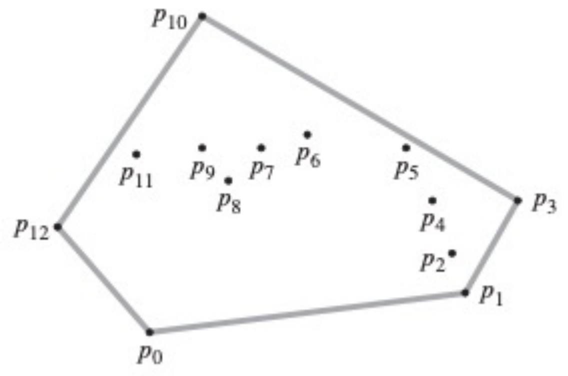
11.1 Learning Objectives

1. Students are able to understand the concept of convex hull.
2. Students are able to understand the monotone chain algorithm for convex hull.
3. Students are able to solve problems related to convex hull.

11.2 Material

11.2.1 Convex Hull

Given a set of points Q in 2-D. The **convex hull** of Q, denoted by CH(Q), is the **smallest convex polygon** P for which each point in Q is either on the boundary of P or in its interior. We implicitly assume that all points in the set Q are unique and that Q contains at least three points which are not colinear. Intuitively, we can think of each point in Q as being a nail sticking out from a board. The convex hull is then the shape formed by a tight rubber band that surrounds all the nails. The figure below shows a set of points and its convex hull, where Q = {p_0, p_1, ..., p_{12}} and its convex hull CH(Q) in gray.

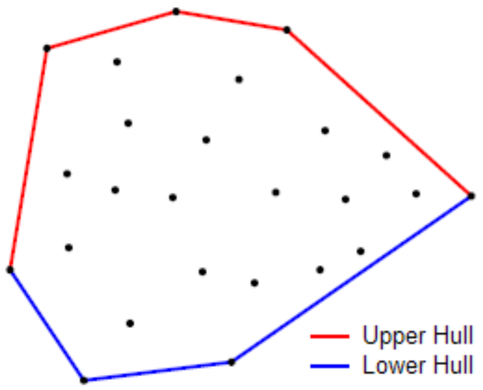


There are several algorithms in finding the convex hull, such as Jarvis's march algorithm, Graham's scan algorithm, monotone chain algorithm, etc. In this chapter, we will only discuss about the monotone chain algorithm.

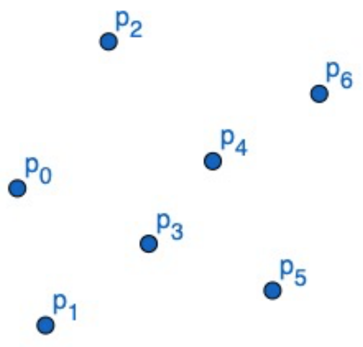
11.2.2 The Monotone Chain Algorithm

The monotone chain algorithm constructs the convex hull of a set of 2-dimensional points. It does so by first sorting the points lexicographically (first by x-coordinate, and in case of a tie, by y-coordinate), and then constructing lower and upper hulls of the points.

A lower hull is the lower part of the convex hull. It runs from its leftmost point to the rightmost point in counter-clockwise order. An upper hull is the remaining part of the convex hull. In the following picture, the upper hull is the red part of the convex hull, while the lower hull is blue.

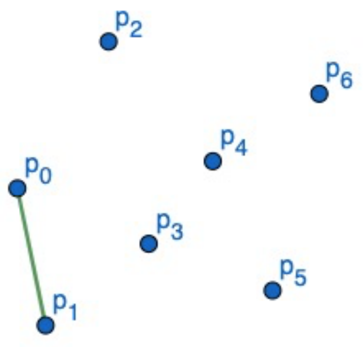


For example, given 7 points in 2-dimensional space as in the following picture. Note that the points are already sorted by lowest X, and lowest Y if tie.

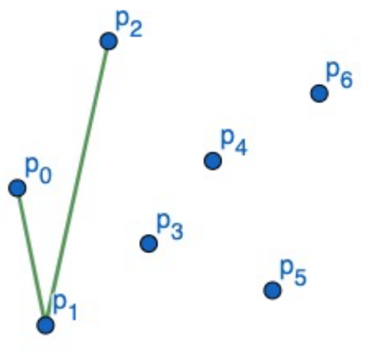


Here is the illustration of the monotone chain algorithm to build the lower hull first:

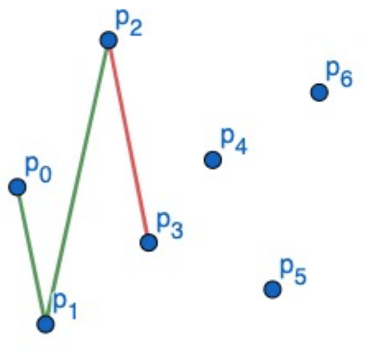
1. Insert the first two leftmost points into the lower hull.

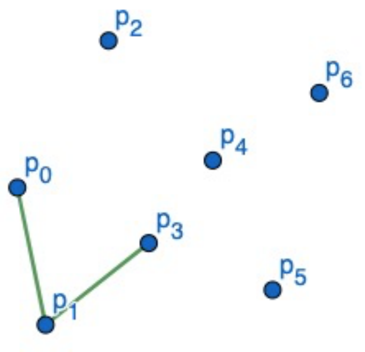


2. Check whether P_0, P_1 (the top two points in lower hull), and P_2 creates a counter-clockwise turn. It is yes, so put P_2 into the lower hull.

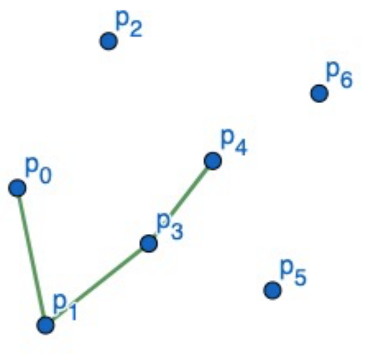


3. Check whether P_1, P_2 (the current top two points in the lower hull), and P_3 creates a counter-clockwise turn. Unfortunately, it is no, so remove P_2 from the lower hull. Then, check whether P_0, P_1 (the current top two points in the lower hull), and P_3 creates a counter-clockwise turn. It is yes, so put P_3 into the lower hull.

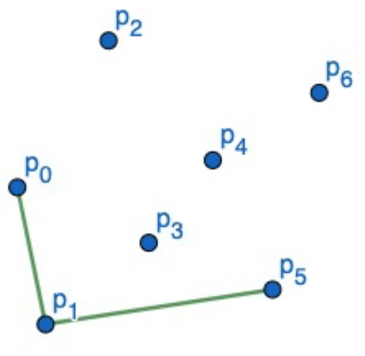




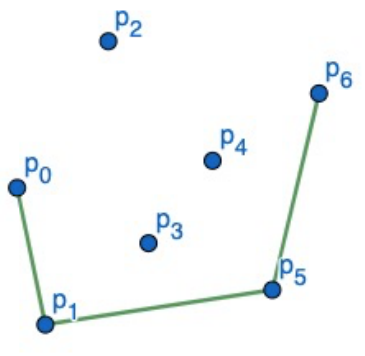
4. Check whether P_1, P_3 (the current top two points in the lower hull), and P_4 creates a counter-clockwise turn. It is yes, so put P_4 into the lower hull.



5. In a similar way, the points P_4 and P_3 are removed from the lower hull, and P_5 is put into the lower hull.



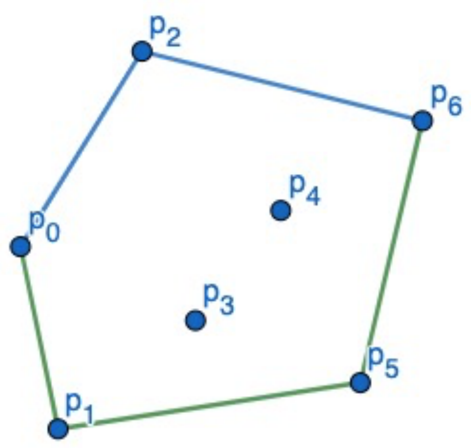
6. The point P_6 is then put into the lower hull.



Finally, building the lower hull is done. The lower hull contains points $\{P_0, P_1, P_5, P_6\}$.

Building the upper hull can be done in a similar way, except that it goes from the rightmost point to the leftmost point. In this case, we go from P_6 until P_0. After it is done, the upper hull contains points $\{P_6, P_2, P_0\}$.

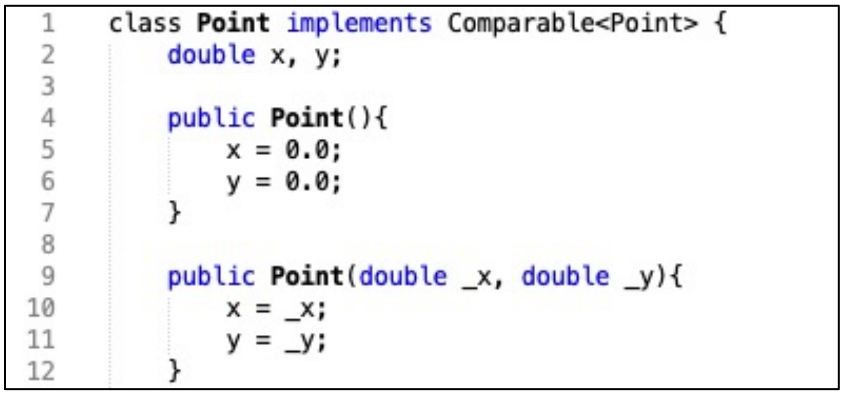
In the following picture, the lower hull is illustrated by the green segments, and the upper hull is illustrated by the blue segments.



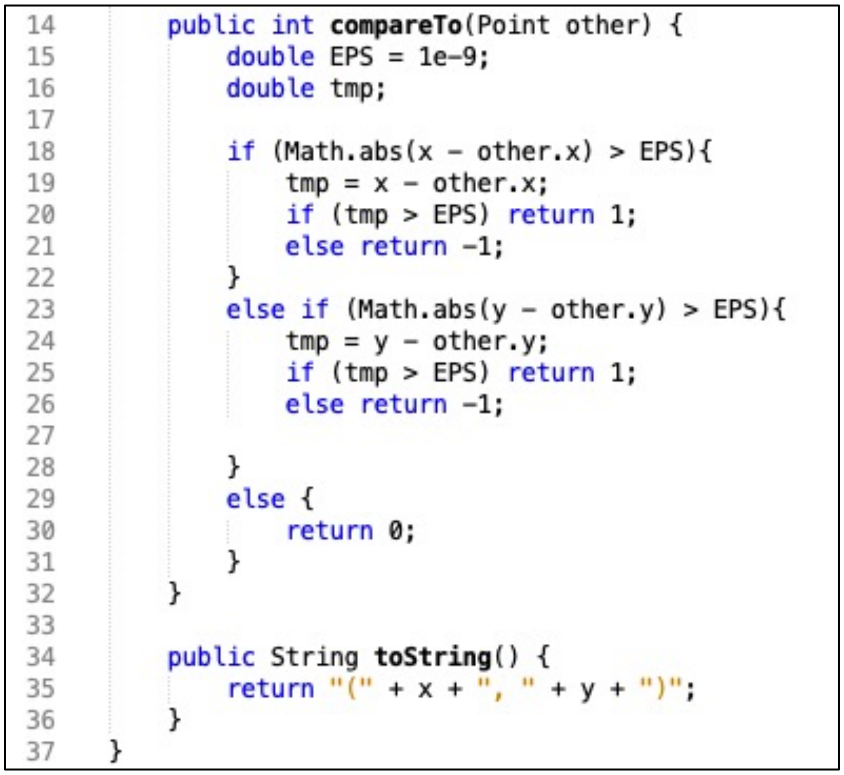
After completing both the lower hull and upper hull, then both are combined into a single convex hull but remove the last point of each lower hull and upper hull first, because it's the same as the first point of the other list.

11.2.3 Java Implementation

We will create the **Point** class that implements Comparable class, and it contains two attributes, x and y, both are in double type. Also, we override the compareTo method so that we can order the points based on the lower X, and the lower Y if tie.

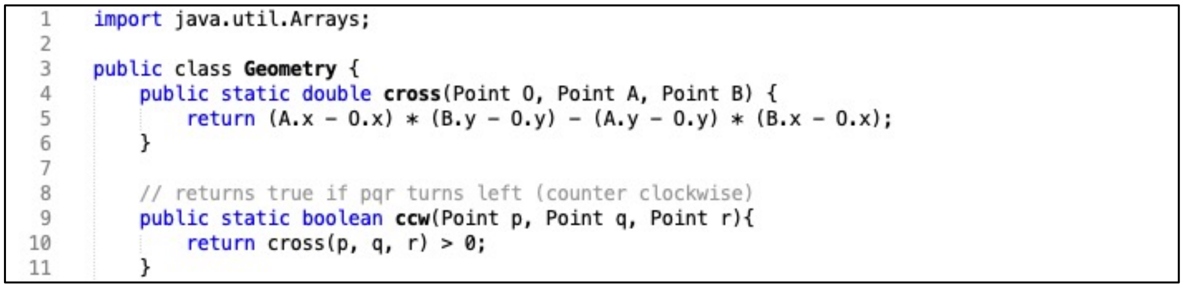


```
class Point implements Comparable<Point> {
    double x, y;

    public Point(){
        x = 0.0;
        y = 0.0;
    }

    public Point(double _x, double _y){
        x = _x;
        y = _y;
    }
```

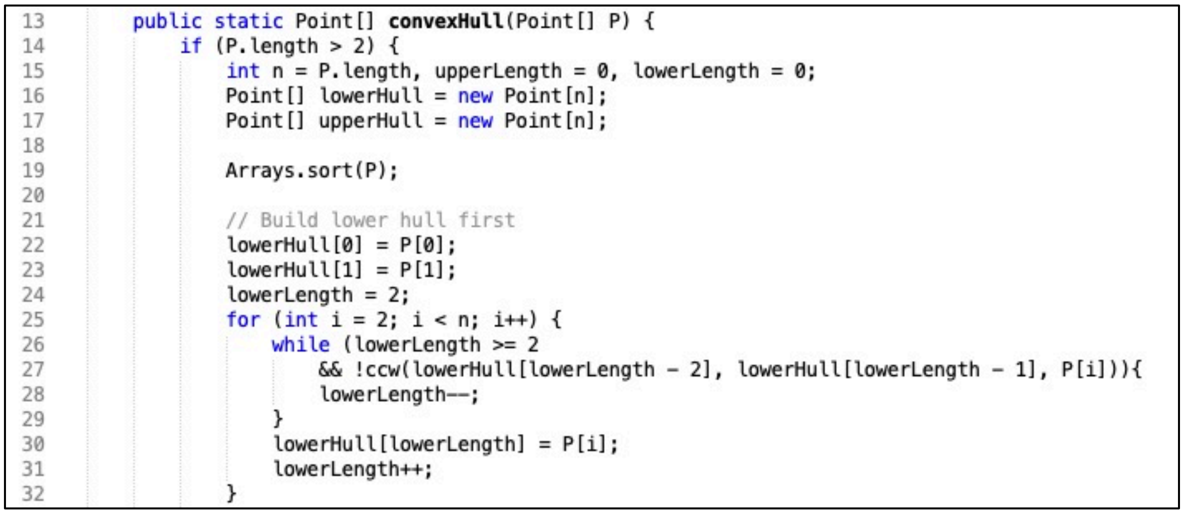


```java
    public int compareTo(Point other) {
        double EPS = 1e-9;
        double tmp;

        if (Math.abs(x - other.x) > EPS){
            tmp = x - other.x;
            if (tmp > EPS) return 1;
            else return -1;
        }
        else if (Math.abs(y - other.y) > EPS){
            tmp = y - other.y;
            if (tmp > EPS) return 1;
            else return -1;

        }
        else {
            return 0;
        }
    }

    public String toString() {
        return "(" + x + ", " + y + ")";
    }
}
```

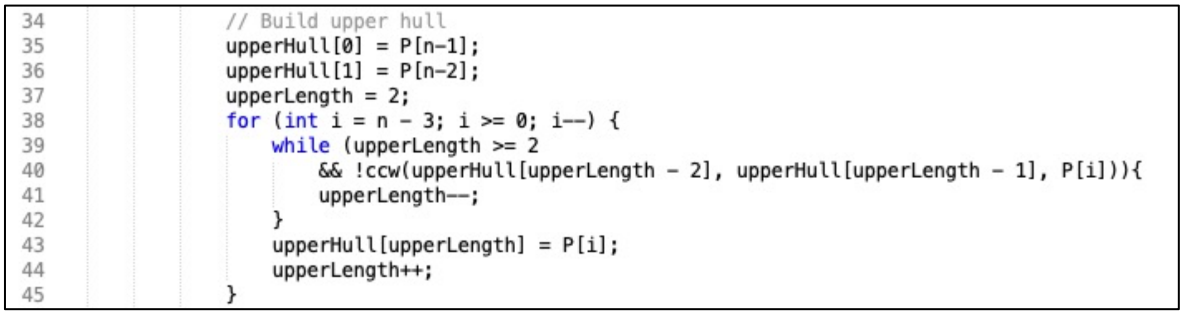
Then, we implement the **Geometry** class. It has three static methods: **cross**, **ccw**, and **convexHull**. The cross method takes three parameters, point O, point A, and point B, and it returns the cross product of vector OA and OB. The ccw method takes three parameters as well, point P, point Q, and point R, and it returns true if the points P-Q-R creates a left turn (or counter-clockwise turn).



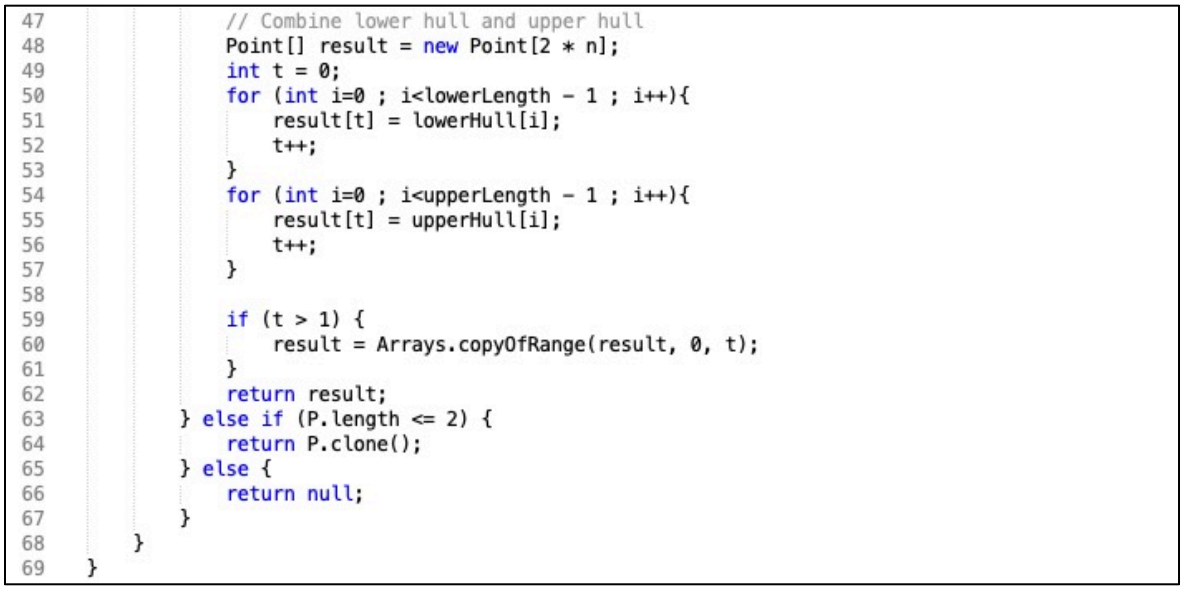
```java
import java.util.Arrays;

public class Geometry {
    public static double cross(Point O, Point A, Point B) {
        return (A.x - O.x) * (B.y - O.y) - (A.y - O.y) * (B.x - O.x);
    }

    // returns true if pqr turns left (counter clockwise)
    public static boolean ccw(Point p, Point q, Point r){
        return cross(p, q, r) > 0;
    }
```



```java
    public static Point[] convexHull(Point[] P) {
        if (P.length > 2) {
            int n = P.length, upperLength = 0, lowerLength = 0;
            Point[] lowerHull = new Point[n];
            Point[] upperHull = new Point[n];

            Arrays.sort(P);

            // Build lower hull first
            lowerHull[0] = P[0];
            lowerHull[1] = P[1];
            lowerLength = 2;
            for (int i = 2; i < n; i++) {
                while (lowerLength >= 2
                    && !ccw(lowerHull[lowerLength - 2], lowerHull[lowerLength - 1], P[i])){
                    lowerLength--;
                }
                lowerHull[lowerLength] = P[i];
                lowerLength++;
            }
```

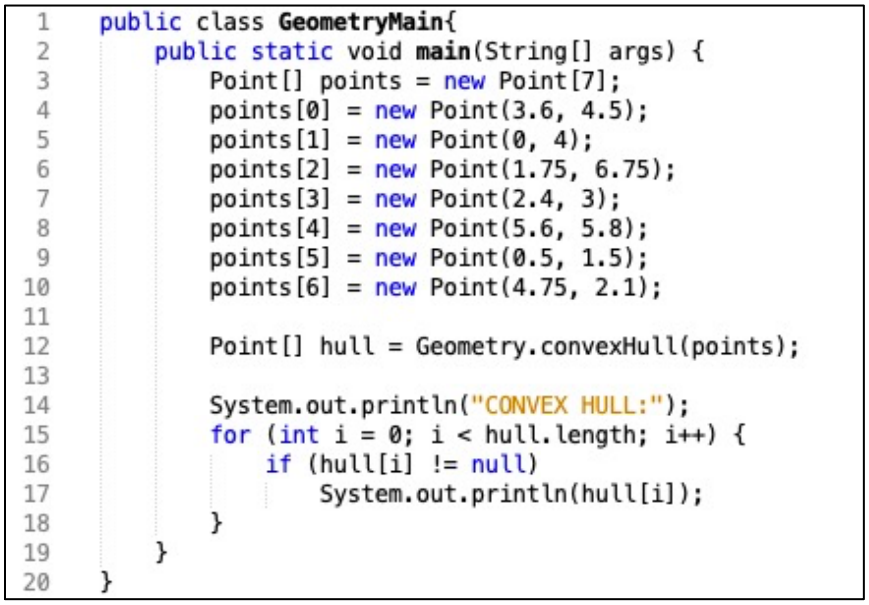


```java
            // Build upper hull
            upperHull[0] = P[n-1];
            upperHull[1] = P[n-2];
            upperLength = 2;
            for (int i = n - 3; i >= 0; i--) {
                while (upperLength >= 2
                    && !ccw(upperHull[upperLength - 2], upperHull[upperLength - 1], P[i])){
                    upperLength--;
                }
                upperHull[upperLength] = P[i];
                upperLength++;
            }
```



```java
            // Combine lower hull and upper hull
            Point[] result = new Point[2 * n];
            int t = 0;
            for (int i=0 ; i<lowerLength - 1 ; i++){
                result[t] = lowerHull[i];
                t++;
            }
            for (int i=0 ; i<upperLength - 1 ; i++){
                result[t] = upperHull[i];
                t++;
            }

            if (t > 1) {
                result = Arrays.copyOfRange(result, 0, t);
            }
            return result;
        } else if (P.length <= 2) {
            return P.clone();
        } else {
            return null;
        }
    }
}
```

Finally, the **GeometryMain** class holds the main function.



```java
public class GeometryMain{
    public static void main(String[] args) {
        Point[] points = new Point[7];
        points[0] = new Point(3.6, 4.5);
        points[1] = new Point(0, 4);
        points[2] = new Point(1.75, 6.75);
        points[3] = new Point(2.4, 3);
        points[4] = new Point(5.6, 5.8);
        points[5] = new Point(0.5, 1.5);
        points[6] = new Point(4.75, 2.1);

        Point[] hull = Geometry.convexHull(points);

        System.out.println("CONVEX HULL:");
        for (int i = 0; i < hull.length; i++) {
            if (hull[i] != null)
                System.out.println(hull[i]);
        }
    }
}
```

11.3 Assignments

1. Add **getConvexHullArea** method in the Geometry class that takes input of the convex hull of a set of points S, CH(S), and return the area of the area enclosed by the convex hull.
2. Add **getConvexHullLength** method in the Geometry class that takes input of the convex hull of a set of points S, CH(S), and return the perimeter of the convex hull.