Laboratory work # 3

Student: HU Riqian
Student ID: 20321114
Timus Name: hduads2022_20321114
Mail: jhlxhrq@163.com

Problem # 1444. Elephpotamus

Screenshot from Timus:

9822956	11:55:24 11 Apr 2022	hduads2022_20321114	1444. Elephpotamus	Java 1.8	Accepted		0.296	5 180 KB

Explanation of algorithm:

1. Build the polygon along the first point depending on the atan2 function;

2. If the polygon turn around, then let the point start after the turning.

Computational complexity of algorithm:

$$O(n \log n)$$

Source code:

```
import java.io.*;
import java.util.Arrays;

public class Elephpotamus {
    public static void main(String[] args) throws IOException {
        new Elephpotamus().run();
    }

    private Point[] POINTS;
    private int SIZE;
    private int STEP = 0;
    StreamTokenizer in;
    PrintWriter out;

    int nextInt() throws IOException {
        in.nextToken();
        return (int) in.nval;
    }

    void run() throws IOException {
        in = new StreamTokenizer(new BufferedReader(new
InputStreamReader(System.in)));
        out = new PrintWriter(System.out);
        initPoints();
```

```
        sortPoints();
        paveBoard();
        outputStep();
        out.flush();
    }

    void initPoints() throws IOException {
        SIZE = nextInt();
        POINTS = new Point[SIZE + 1];
        for (int i = 0; i < SIZE; i++) {
            POINTS[i] = new Point(nextInt(), nextInt());
            POINTS[i].setIndex(i + 1);
        }
    }

    void sortPoints() {
        for (int i = SIZE - 1; i >= 0; i--) {
            POINTS[i].reset(POINTS[i].getX() - POINTS[0].getX(),
POINTS[i].getY() - POINTS[0].getY());
        }
        Arrays.sort(POINTS, 1, SIZE, Point::SOLVE);
    }

    void paveBoard() {
        for (int i = 0; i < SIZE - 1; i++) {
            int d1_x = POINTS[i].getX() - POINTS[0].getX();
            int d1_y = POINTS[i].getY() - POINTS[0].getY();
            int d2_x = POINTS[i + 1].getX() - POINTS[0].getX();
            int d2_y = POINTS[i + 1].getY() - POINTS[0].getY();
            if (d1_x * d2_y - d1_y * d2_x <= 0 && d1_x * d2_x + d1_y * d2_y <
0) {
                STEP = i;
                break;
            }
        }
    }

    void outputStep() {
        out.println(SIZE);
        out.println(POINTS[0].getIndex());
        for (int i = 0; i < SIZE - 1; i++) {
            out.println(POINTS[(STEP + i) % (SIZE - 1) + 1].getIndex());
        }
    }

}

class Point {
    private int x;
    private int y;
    private int index;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public void reset(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX() {
        return x;
```

```
    }

    public int getY() {
        return y;
    }

    public void setIndex(int index) {
        this.index = index;
    }

    public int getIndex() {
        return index;
    }

    public static int SOLVE(Point P1, Point P2) {
        if (P1.getX() * P2.getY() == P1.getY() * P2.getX() && P1.getX() *
P2.getX() + P1.getY() * P2.getY() >= 0) {
            if (P1.getX() * P1.getX() + P1.getY() * P1.getY() < P2.getX() *
P2.getX() + P2.getY() * P2.getY())
                return -1;
            else
                return 1;
        }
        else if (Math.atan2(P1.getY(), P1.getX()) < Math.atan2(P2.getY(),
P2.getX()))
            return -1;
        else
            return 1;
    }
}
```