Assignment #2

SOFE 2715U: Data Structures

Winter 2018

Instructor: Dr. Shahryar Rahnamayan

Date: March 6, 2018

Dhanushga Lionel-100616831

Allan Santosh-100557518

Mohtasim Siddiqui-100635463

A: Pseudocode:

```
Algorithm max_value_of_reserve(vector n)

Input: N number of vectors
Output: Max value of oil that can be collected

If ( n = 1 ) then
Return vector (x2 - x1)

If ( n > 1 ) then
For I ê 1 to n-1 do
    Value ê 0
    For I ê 1 to n-1 do
Cur <-- Vector[i]
            If (currect vector y  >= previous vector  ) {
            Cur = cur + x2[j] - x1[j]
}
Else if
(currect vector y  < previous vector  ) {
Store the vector in new Array (x2[j]-x1[j],x1[i]-x2[j],y[i]-y[j]
Store the vector in new Array (x1[j]-x2[j],x1[i]-x2[j],y[i]-y[j]
}
Else{
Store the vector in new Array (x2[j]-x1[j],x1[i]-x2[j],y[j]-y[j]
Store the vector in new Array (x1[j]-x2[j],x2[i]-x1[i],y[j]-y[i]


        Execute(vector x1, vector x2, checkslope)

        For I ê0 to n-1 do
            cur ê v[j].vector
            value = maxvalue of (cur)
Display value;
```

B: Code

```
/**
*This program implements the Oil Well problem for assignment 2
*@author Dhanushga Lionel, Allan Santosh, Mohtasim Siddiqui
*@version 1.0
*@since 2018-03-06
*/
//Declaring libraries to use
import java.util.*;
import java.util.Vector;
```

```
public class Oilwell {

    ArrayList<String> oillocation = new ArrayList<String>(); //Make a new arraylist for string
    ArrayList<Integer> oillocationsize = new ArrayList<Integer>(); // Arraylist for integers
    ArrayList<Integer> newreserves = new ArrayList<Integer>(); // Arraylist for integers

    int x1, x2, y;//Declare variables

    public Oilwell (ArrayList<String> oillocation) {
        this.oillocation = oillocation;
    }
    /** This method checks the integers
    public void execute(int x1, int x2, boolean check) {

        for (int i = 0; i < oillocation.size(); i++) {
            Scanner stringtoint = new Scanner (oillocation.get(i));
            x1 = stringtoint.nextInt();
            x2 = stringtoint.nextInt();
            oillocationsize.add(checklayeramount(x1,x2));
        }
    }
    /** This method checks which is lower or which is higher
    public int checklayeramount (int lower, int higher) {
        return higher-lower;
    }
    /** This method checks pairs and returns a<b or it returns false
    public boolean checkpairs(int ax1, int ax2, int bx1, int bx2) {

        double a = ax1 * ax2;
        double b = bx1 * bx2;

        if (a != b) {
            return a < b;
        }

        else {
            return false;
        }
    }
/**
*We pick a point to start drilling a hole
*Then we sort all other points on the line by the angle of the point we picked
*By iterating on the other points in the order of the slope use a rotating line sweep
*.If we encounter the first point of an oil layer, we add the value of this layer to the current result
```

```
*f we encounter the second point, we subtract the value of this layer from the current result
*/

    public static void main(String[] args) {

        int numlayers = 0;
        Scanner inputnumlayers = new Scanner (System.in);
        Scanner inputlocations = new Scanner (System.in);
        ArrayList<String> oillocation = new ArrayList<String>();

        while (numlayers == 0) {
            numlayers = inputnumlayers.nextInt();
        }

        for (int i = 0; i<numlayers; i++) {
            oillocation.add(inputlocations.nextLine());
        }


        Oilwell reserves = new Oilwell (oillocation) ;

        int value = 0;
        for (int i = 0; i < numlayers ; i++) {
            int cur = 0;
            for (int j = 0; j< numlayers; j++) {
                if (oillocation.get(i) == oillocation.get(j)) {
                    if(oillocation.get(i)) >= oillocation.get(j) && oillocation.get(i)
<= oillocation.get(j)){
                        cur = cur + oillocation.get(j)-oillocation.get(i);
                    } }
                    else if (reserves.get(j).y <  reserves.get(i).y) {
                        newreserves.add(oillocation.get(j).x2 -
oillocation.get(j).x1 , oillocation.get(i).x1 - oillocation.get(i).x2, reserves.get(i).y-
reserves.get(j).y);
                        newreserves.add(oillocation.get(j).x1 -
oillocation.get(j).x2 , oillocation.get(i).x1 - oillocation.get(i).x1, reserves.get(i).y-
reserves.get(j).y);
                }
                else {
                    newreserves.add(oillocation.get(j).x2 - oillocation.get(j).x1 ,
oillocation.get(i).x1 - oillocation.get(i).x1, reserves.get(j).y-reserves.get(i).y);
                    newreserves.add(oillocation.get(j).x1 - oillocation.get(j).x2 ,
oillocation.get(i).x2 - oillocation.get(i).x1, reserves.get(j).y-reserves.get(i).y);
                }
            }

            reserves.execute(x1, x2, newreserves.checkpairs(oillocation.get(i).x1,oillocation.get(j).x2,oillocation.get(j).x1,oillocation.get(j).x2));
            for (int i = 0; i < newreserves.size(); j++) {
                cur = reservers.get(j).x1;
                if (cur>value) {
                    value = cur;
                    System.out.println(value);
                }
            }

        }

    }

}
```

C: Big-oh Analysis: O(n^2 log n) is the answer

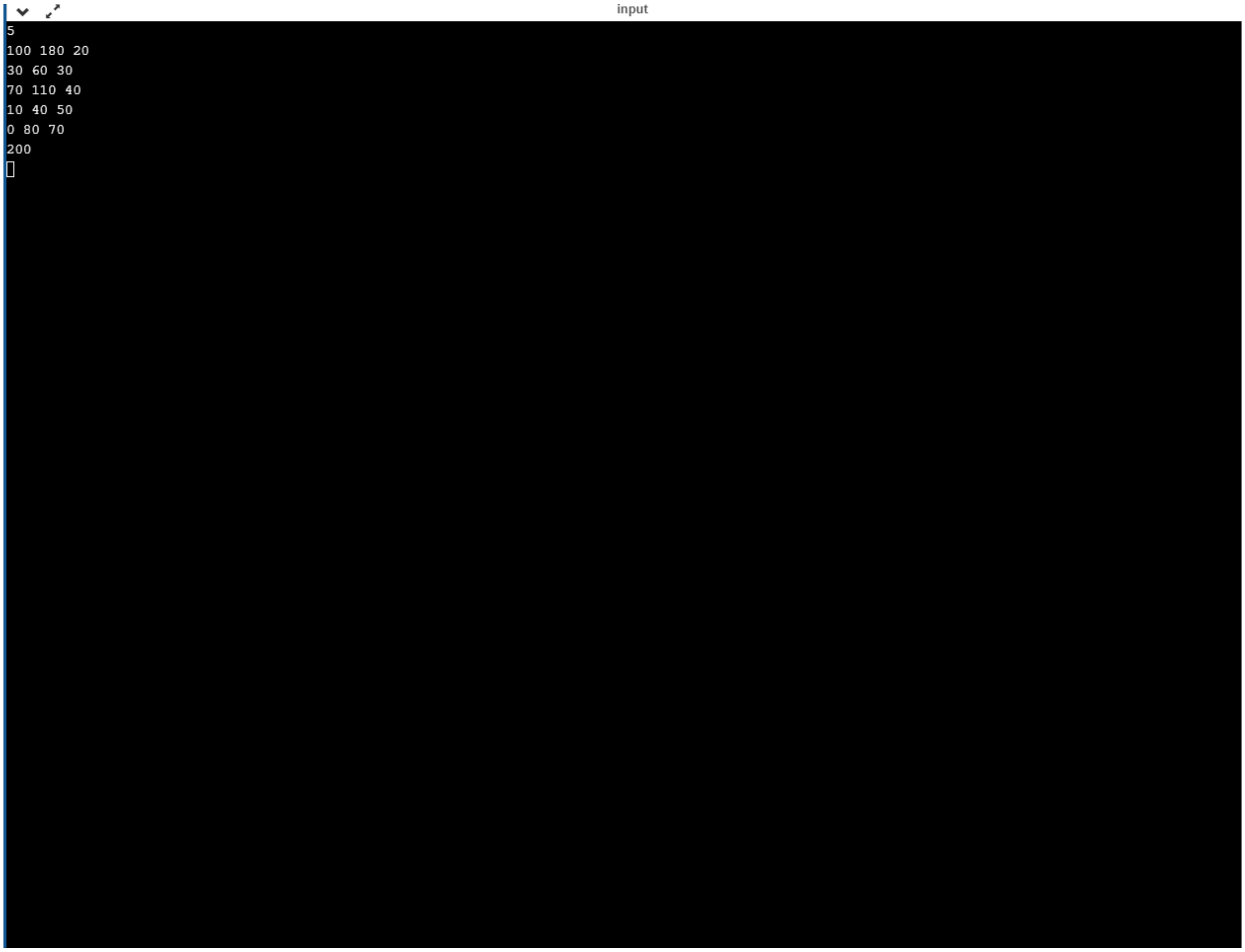
If the wells were on a horizontal line, the well will only hit on one, so the best thing is to hit the largest one. Other than that, the it is possible to touch two of the oil layers endpoints. If we did O(n^3), each pair of points of the input are on the horizontal line and check to see if oil layers are hit, will be too slow. We can instead use a rotating sweep line. We can pick a point x, where we will drill a well, then sort all other points not on that horizontal line by the angle of x and the other point. Then we can rotate the well going through x by iterating on the other points in the order of the slope.If we encounter the first point of an oil layer, we add the value of this layer to the current result, if we encounter the second point, we subtract the value of this layer from the current result. The algorithm therefore runs in O(n^2 log n) time.

D: Results

Sample 1:

Input:	Output:

5
100 180 20
30 60 30
70 110 40
10 40 50
0 80 70
	200

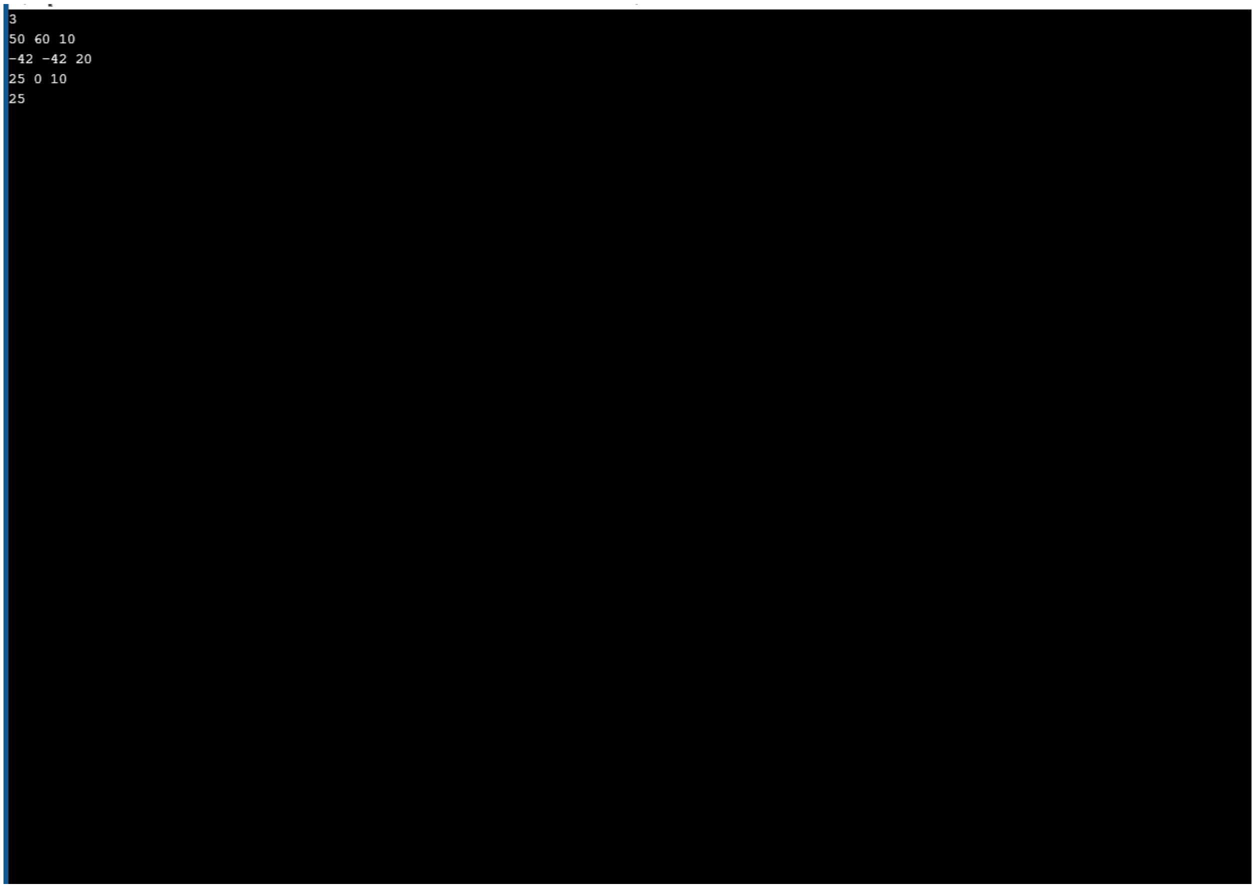


```
5
100 180 20
30 60 30
70 110 40
10 40 50
0 80 70
200
```

Sample 2:

Input:	Output:

3
50 60 10
-42 -42 20
25 0 10
	25

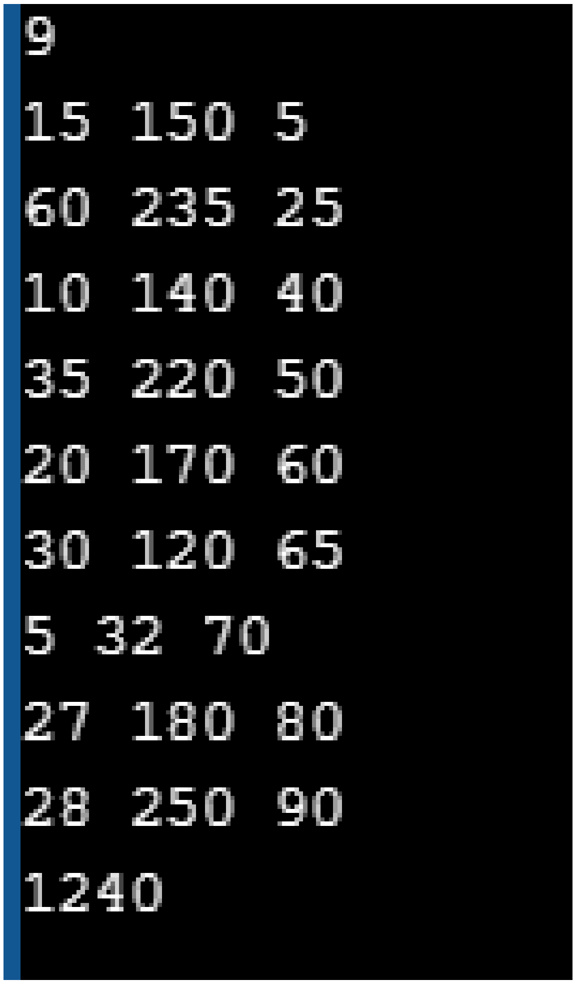


```
3
50 60 10
-42 -42 20
25 0 10
25
```

Sample 3:

Input:	Output:

9
15 150 5
60 235 25
10 140 40
35 220 50
20 170 60
30 120 65
5 32 70
27 180 80
28 250 90
	1240

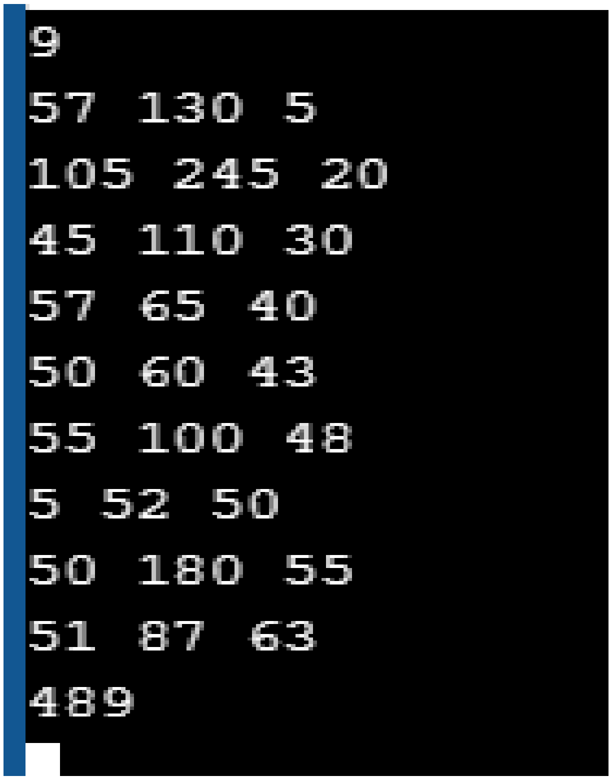


```
9
15 150 5
60 235 25
10 140 40
35 220 50
20 170 60
30 120 65
5 32 70
27 180 80
28 250 90
1240
```

Sample 4:

Input:	Output:

9
57 130 5
105 245 20
45 110 30
57 65 40
50 60 43
55 100 48
5 52 50
50 180 55
51 87 63
	489

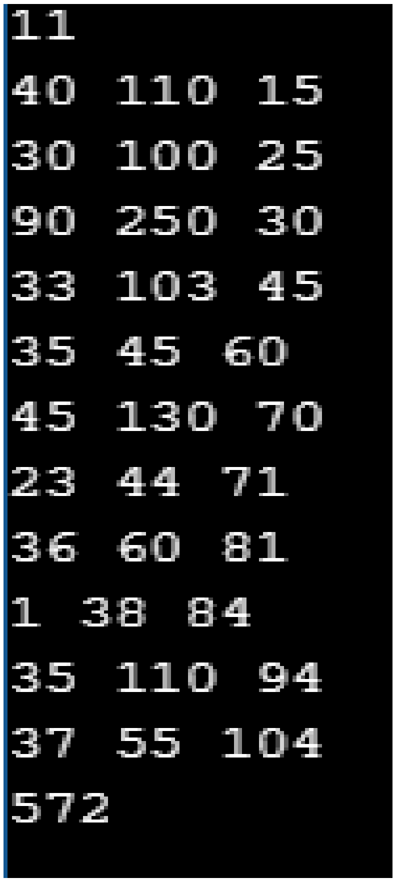


```
9
57 130 5
105 245 20
45 110 30
57 65 40
50 60 43
55 100 48
5 52 50
50 180 55
51 87 63
489
```

Sample 5:

Input:	Output:

11
40 110 15
30 100 25
90 250 30
33 103 45
35 45 60
45 130 70
23 44 71
36 60 81
1 38 84
35 110 94
37 55 104
	572



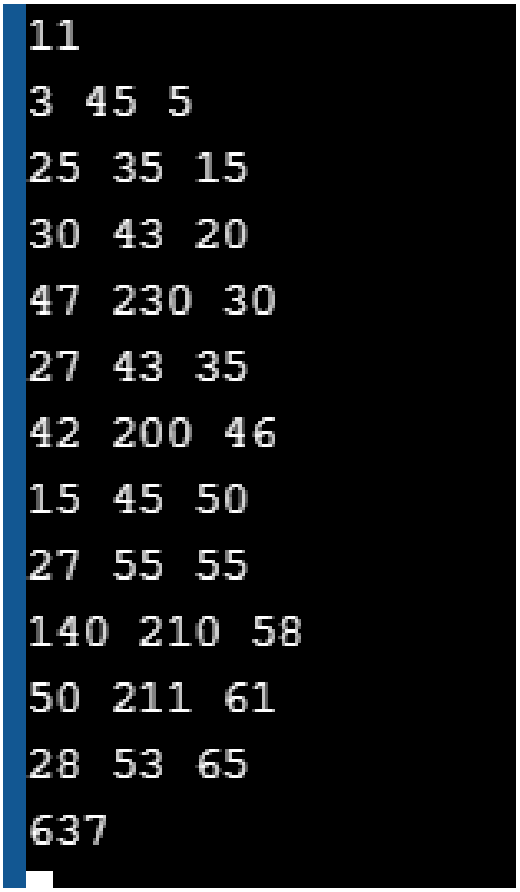
```
11
40 110 15
30 100 25
90 250 30
33 103 45
35 45 60
45 130 70
23 44 71
36 60 81
1 38 84
35 110 94
37 55 104
572
```

Sample 6:

Input:	Output:

11
3 45 5
25 35 15
30 43 20
47 230 30
27 43 35
42 200 46
15 45 50
27 55 55
140 210 58
	637

50 211 61 28 53 65	



```
11
3 45 5
25 35 15
30 43 20
47 230 30
27 43 35
42 200 46
15 45 50
27 55 55
140 210 58
50 211 61
28 53 65
637
```