/\*The algorithm works by first going through the data, calculating the width and height of each set of four corners where y1 == y2 and y3 == y4 are the same. Then, whatever the height is for that set of four corners, the height is added to an list x(width) times. For example, if the height = 2 and the width = 4, 2 is added to the list 4 times. The list, h, contains the height for every x value within the total area of the data set. Since the algorithm only goes through the initial data once, the time complexity of this part is O(n).

With this list of heights, the algorithm uses a stack to pop or push indexes of the list, depending on the height at that index in the list. For each item in the list, if the stack is empty or the height of the current index is greater than the height of the index at the top of the stack, the current index is pushed to the stack. If the algorithm comes across a height that is less than the top of the stack, then it pops the stack until there is 1 item left, calculating the area of each height, and adding it to the previous sum, then comparing the area to the maxArea. It repeats this process for the rest of the heights left in the array. These two unnested while loops both have a time complexity of O(n).

Combining the first and second parts gives you a time complexity of 3\*O(n) = O(n)

\*/

**import** java.io.\*;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.Collections;

**import** java.util.Scanner;

**import** java.util.Stack;

**import** java.util.List;

**public** **class** area {

**public** **static** **void** main(String[] args) **throws** IOException {

//read in the file

Scanner in = **new** Scanner(System.***in***);

System.***out***.println("Paste the file path with no quotations: ");

String input = in.nextLine();

FileReader file = **new** FileReader(input);

//FileReader file = new FileReader();

BufferedReader buffer = **new** BufferedReader(file);

//read in array size

String line = buffer.readLine();

//store size

**int** size = Integer.*parseInt*(line);

//store x and y coordinates

List<Integer> x = **new** ArrayList<Integer>();

List<Integer> y = **new** ArrayList<Integer>();

**while** ((line = buffer.readLine()) != **null**) {

String[] data = line.split("\\s+");

x.add(Integer.*parseInt*(data[0]));

y.add(Integer.*parseInt*(data[1]));

//System.out.println(Arrays.toString(data));

}

//array of every height at every x within data set

List<Integer> h = **new** ArrayList<Integer>();

**for** (**int** i = 0; i < size - 2; i+=2) {//O(n)

**int** Ax = x.get(i+1);

**int** Ay = y.get(i+1);

System.***out***.print("A:("+Ax+","+Ay+") ");

**int** Bx = x.get(i+3);

**int** By = y.get(i+3);

System.***out***.print("B:("+Bx+","+By+") ");

**int** Cx = x.get(i);

**int** Cy = 0;

System.***out***.println("C:("+Cx+","+Cy+") ");

**int** width = Bx - Ax;

**int** height = Ay - Cy;

//add heights to list

h.addAll(Collections.*nCopies*(width, height));

}

Stack<Integer> s = **new** Stack<>();//lifo

**int** maxArea = 0; //max area

**int** top; // top of stack

**int** areaOfTop; //area of column at top of stack

**int** i = 0;

**int** n = h.size();

**while** (i < n)//O(n)

{

System.***out***.println("i: "+i);

//print stack

System.***out***.println(Arrays.*toString*(s.toArray()));

// If h[i] is > than h[top], push it to stack

**if** (s.empty() || h.get(s.peek()) <= h.get(i)) {

s.push(i++);//increase index

}

// If ht[i] is < than h[top] of stack, calculate the area

**else**

{

top = s.peek(); // store the top

s.pop(); // pop the top

// Calculate the area

**if**(!s.isEmpty()) {

areaOfTop = h.get(top) \* (i - s.peek() - 1);

System.***out***.println("areaOfTop: "+areaOfTop);

// check max area

**if** (maxArea < areaOfTop) {

maxArea = areaOfTop;

}

System.***out***.println("maxArea: "+maxArea);

}

}

}

//last index

System.***out***.println("i: "+i);

System.***out***.println(Arrays.*toString*(s.toArray()));

// pop rest of stack and calculate areas and compare to max

**while** (s.empty() == **false**)//O(n)

{

top = s.peek();

s.pop();

**if** (s.empty()) {//add area of last item

areaOfTop = h.get(top) \* i;

}

**else** {

//calculate area and add it to

areaOfTop = h.get(top) \* (i - s.peek() - 1);

}

//compare to maxArea

System.***out***.println("areaOfTop: "+areaOfTop);

**if** (maxArea < areaOfTop) {

maxArea = areaOfTop;

System.***out***.println("maxArea: "+maxArea);

}

}

System.***out***.println("Final Area: "+maxArea);

}

}//Total Time Complexity: O(n) + O(n) + O(n) = O(n)

Input 3

Text

Description automatically generated

Text

Description automatically generated

Input 4

Lots of lines…

