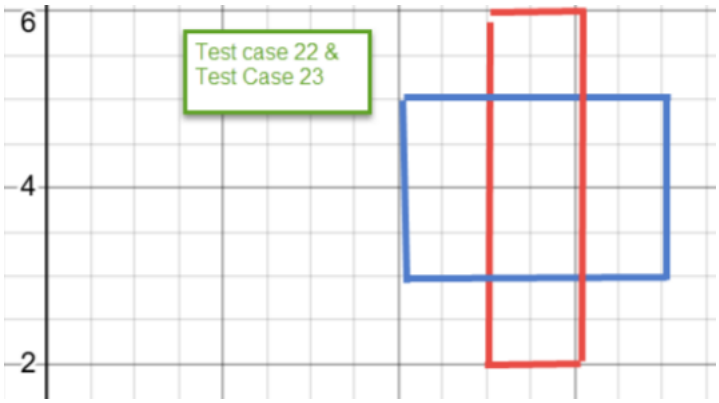
Due to the nature of the problem and the overlap of the loop to satisfy several conditions, my test case will officially commence in the following configuration
Since we can see one shape has cut through both opposite sides of the rectangle.....
It would have been close to contemplate logical expressions to this level until the issue actually arose as part of more difficult test cases....

TEST CASE 22 & 23



We can see the test case fails.....

```
rect1Topright: 6
rect2TopRight: 5
rect1BottomLeft: 2
rect2BottomLeft: 3
HERE
-2
-3
The overlapping area is: 6
```

It is interpreting the dimensions of the blue rectangle to be the overlap...
I have performed some analysis below:

```
rect1Topright: 6
rect2TopRight: 5
rect1BottomLeft: 2
rect2BottomLeft: 3
HERE
-2
-3
The overlapping area is: 6
```

This is getting the value derived from the blue rectangle and it is incorrect

```java
if (rect1TopRight[1]<rect2TopRight[1])
{
    System.out.println("HERE1");
    width = Math.abs(0-rect2BottomLeft[0]) - Math.abs(0-rect1TopRight[0]);
    System.out.println(width);
    height = Math.abs(0-rect2BottomLeft[1]) - Math.abs(0-rect1TopRight[1]);
    System.out.println(height);

    return (Math.abs(width) * Math.abs(height));
}
}

//Same code as above, but flipped  (test case 9)
if(rect2BottomLeft[0]<rect1BottomLeft[0])
{
    if (rect2TopRight[1]<rect1TopRight[1])
    {
        System.out.println("HERE2");
        width = Math.abs(0-rect1BottomLeft[0]) - Math.abs(0-rect2TopRight[0]);
        System.out.println(width);
        height = Math.abs(0-rect1BottomLeft[1]) - Math.abs(0-rect2TopRight[1]);
        System.out.println(height);

        return (Math.abs(width) * Math.abs(height));
    }
```

We know that the x axis is less

We need something to treat different if X axis is smaller rect1BottomLeft[0] (which blue is smaller than red rect1BottomLeft[0])...
And also Y axis is smaller rect1TopRight[1](which it is, blue smaller than red rect1TopRight[2])..
This observation would be suitable for Test Case 8.
But for Test Case 22 and 23, we can also see that blue rectangle has gone through both sides of the red.....
To translate this into coding, we need to check if the blue rectangle has a top right (x axis) which is greater than red rectangle Y axis...

IF this is the case, then we are interested in using the width = width of the red rectangle.
And we will need height to be the same as blue rectangle...

Normally we are used to seeing this code function properly for test cases such as Test Case 8. We can see performing operation for width:

```java
width = Math.abs(0-rect1BottomLeft[0]) - Math.abs(0-rect2TopRight[0]);
```

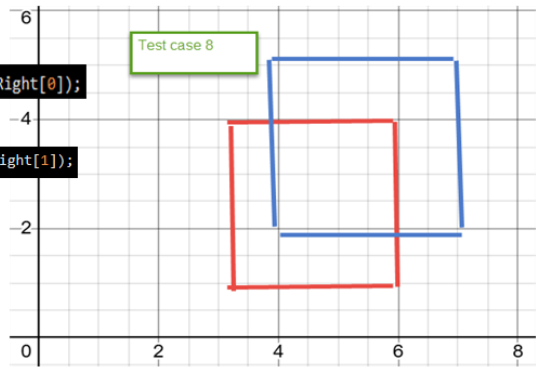
= Math.abs(4-6) = 2

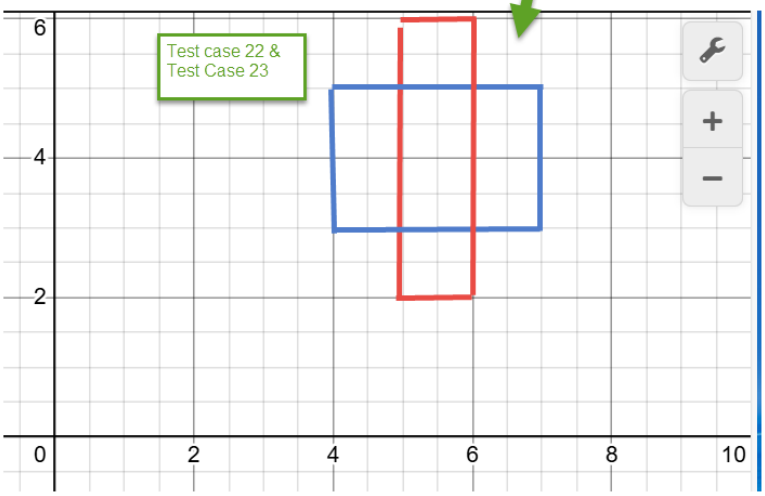
```java
height = Math.abs(0-rect1BottomLeft[1]) - Math.abs(0-rect2TopRight[1]);
```

=Math.abs(2-4) = 2

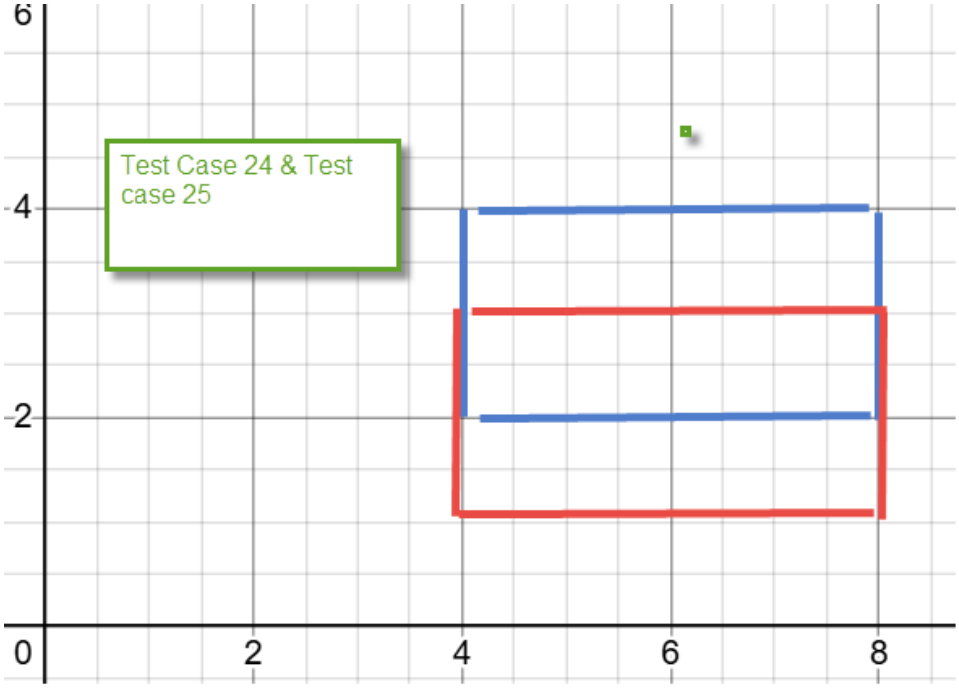
Overlap = 2 x 2

OR
=width = Math.abs(3-7) = 2
height = Math.abs(4-7) = 2





I also ran into difficulties when performing the following:



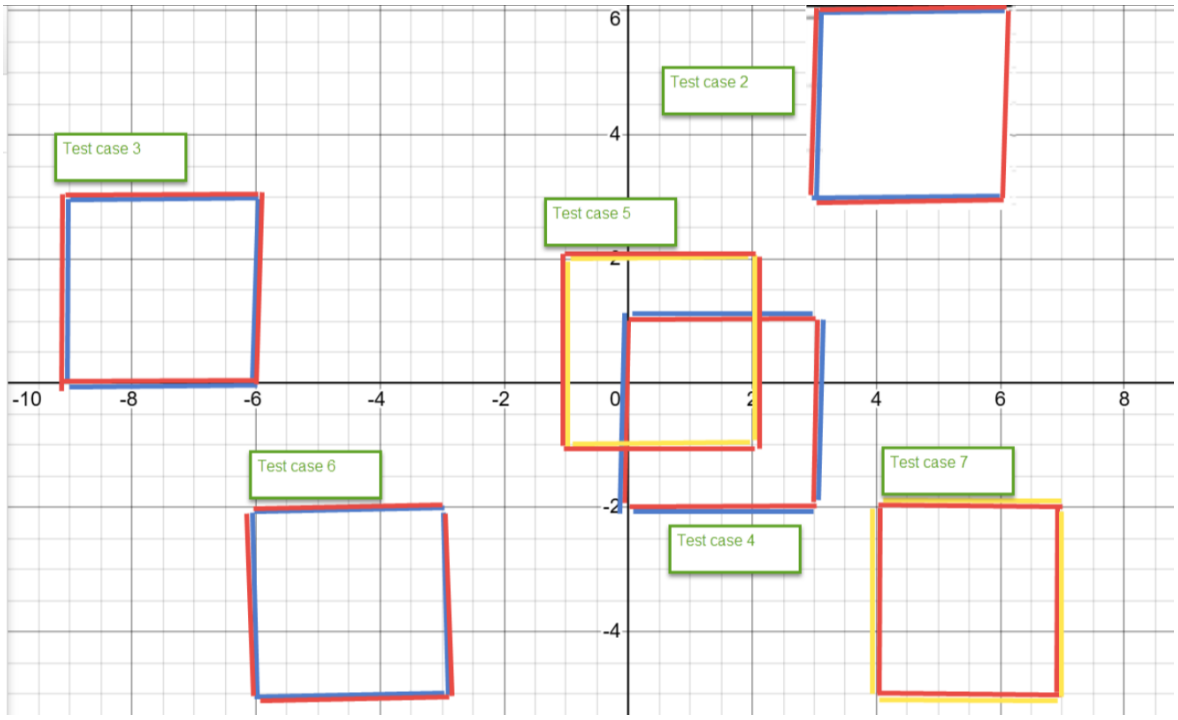
with test case 24 and test case 25 it is performing assumption that shape is inside shape

we can understand from the width and height perspective since the calculation focused on relationship between bottom left of both shapes and top right of both shapes. (from the X axis perspective).

But we had not focussed on the relationship between Y axis..
This is a key component in deciding if there is a shape fully inside another one.....
We can clearly see this is not the case

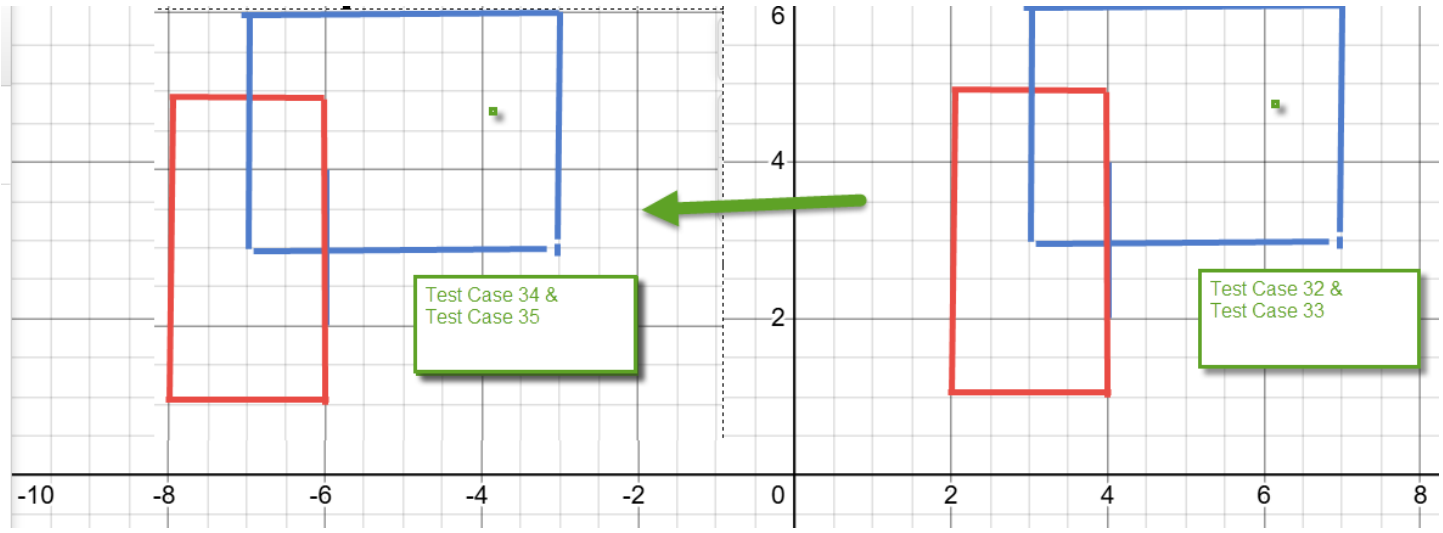
I have now completed all the test cases performing in X and Y axis >=0

During my earlier part of the testing, I had focussed on performing the following...
The idea was to give a good idea of the relationship between two shapes and how it impacts the calculation with respect to the absolute values and as to whether it entails adding widths/heights or performing subtraction... I am fairly convinced I now need to transpose the shapes I completed in the positive area similar to below.



I am trying a few as below:

TEST CASE : 34 & 35



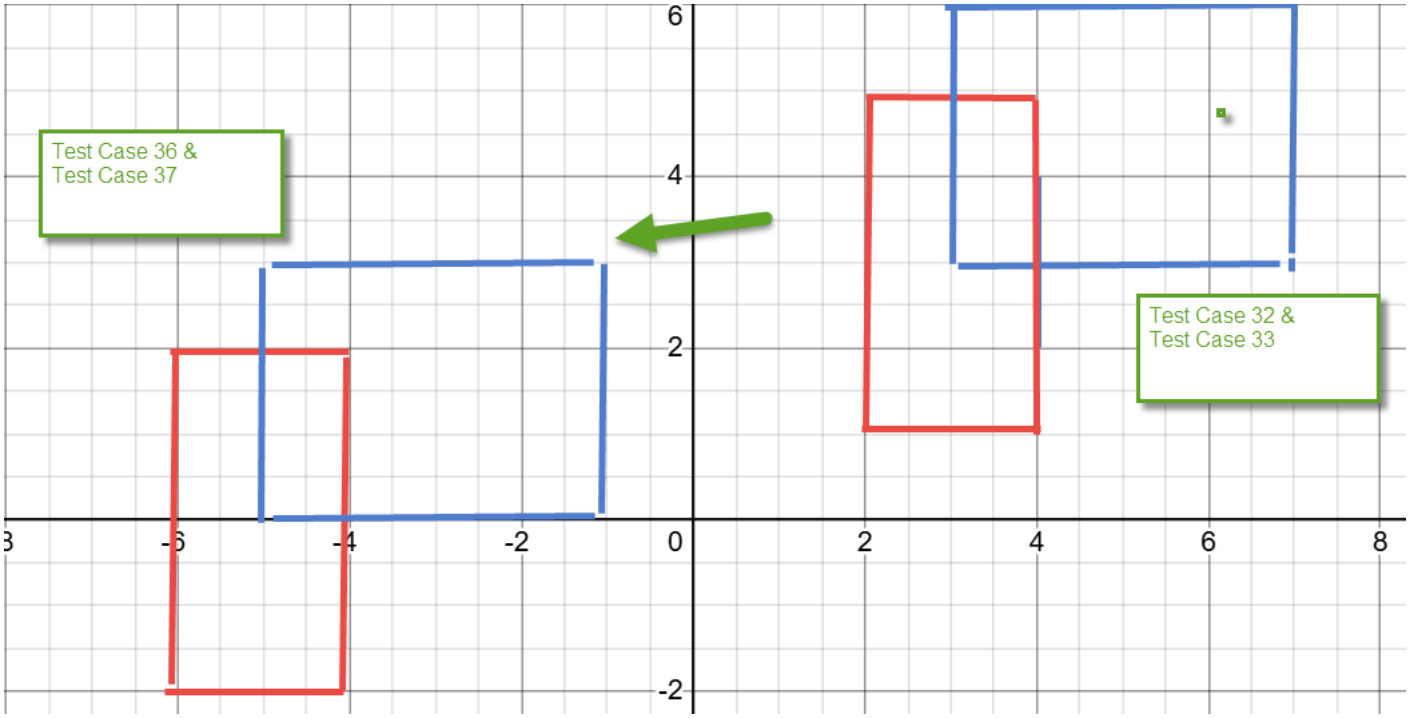

```
//TEST CASE 34
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-8, 1},  new int[]{-6, 5},new int[]{-7, 3}, new int[]{-3, 6}));
//
```

```
The overlapping area is: 2
```

```
//TEST CASE 35 - flip of the above
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-7, 3}, new int[]{-3, 6},new int[]{-8, 1},  new int[]{-6, 5}));
//
```

```
The overlapping area is: 2
```

I expected above to be fine since we are not utilizing shapes which are stretching across the x or y axis. For the following case, I am very sure the outcome will be incorrect...




```
 //TEST CASE 36
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-6, -2}, new int[]{-4, 2},new int[]{-5, 0},  new int[]{-1, 3}));
//
```

```
The overlapping area is: 0
```

We can see in fact that it has not triggered any of the logical expressions.. Which in fact is actually correct since I programmed for exact overlay and also all coordinates in positive axis.

I think below is the best logical starting point....

I have basically considered that if a shape's y coordinate (bottom left) is negative and the top right y coordinate is positive... We would simply add the difference in width (in relation to 0).... This is perfectly fine....

```
if (rect1TopRight[1]==rect2TopRight[1]
   &&rect1BottomLeft[1]==rect2BottomLeft[1])
   {
       System.out.println("EXACT overlap");

       if ((rect2BottomLeft[1]<0 && rect2TopRight[1]>0))

       {
           width = Math.abs(0-rect2BottomLeft[0]) + Math.abs(0-rect2TopRight[0]);
       height = Math.abs(0-rect2BottomLeft[1]) + Math.abs(0-rect2TopRight[1]);

       }

       else
       {

       width = Math.abs(0-rect2BottomLeft[0]) - Math.abs(0-rect2TopRight[0]);
       System.out.println(width);
       height = Math.abs(0-rect2BottomLeft[1]) - Math.abs(0-rect2TopRight[1]);
       System.out.println(height);

       }
       return (Math.abs(width) * Math.abs(height));
```

NOTE: We do not need replica of the above code (ie if the rectangles were swapped in the parameter), since it would make absolutely no difference since are testing for equality.....

We know this is not the case when dealing with partial overlaps....

I think my best option is to consider my code... We know I have coded for positive axis only.. But we know for the following circumstance

```
if ((rect2BottomLeft[1]<0 && rect2TopRight[1]>0))
```

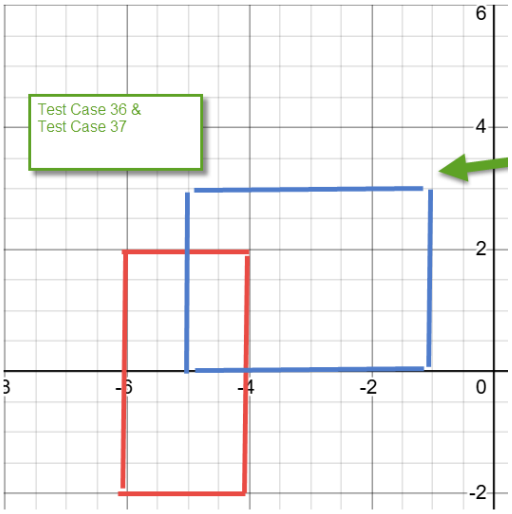
As on the left, the operand actually becomes negative as oppose to positive..

This has created an issue, since I would literally need lots repeat code to perform an addition as oppose to subtraction...

HOWEVER IN ORDER TO AT LEAST ENSURE THAT IT APPEARS WITHIN THE MAIN LOGIC FLOW OF THE CODE, I NEED TO CONSIDER CHANGING THE MAIN LOGICAL EXPRESSIONS AS FOLLOWS....

```


            //if all in the positive axis  or
            //need to check if any shapes cross an axis, since we know the calculation will
            //be impacted since we need to either add/subtract the width or height
            //effectively we need to check
            //rectangle1 width crosses the x axis
            //rectangle1 height crosses the y axis
            //rectangle2 width crosses the x axis
            //rectangle2 height crosses the y axis

            if ((rect1BottomLeft[1]>=0 && rect2BottomLeft[1]>0
            && rect1TopRight[1]>=0 && rect2TopRight[1]>=0)
            || (rect1BottomLeft[0]<0 && rect1TopRight[0]>0)  //rectangle1 width crosses the x axis
            || (rect1BottomLeft[1]<0 && rect1TopRight[1]>0)  //rectangle1 height crosses the y axis
             || (rect2BottomLeft[0]<0 && rect2TopRight[0]>0)  //rectangle2 width crosses the x axis
            || (rect2BottomLeft[1]<0 && rect2TopRight[1]>0))  //rectangle2 height crosses the y axis
            {

                //whether the shapes are physically overlapped
                //looks at x axis
```

It makes logical sense to run the above code again...




```
//TEST CASE 36
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-6, -2}, new int[]{-4, 2},new int[]{-5, 0},  new int[]{-1, 3}));
//
```

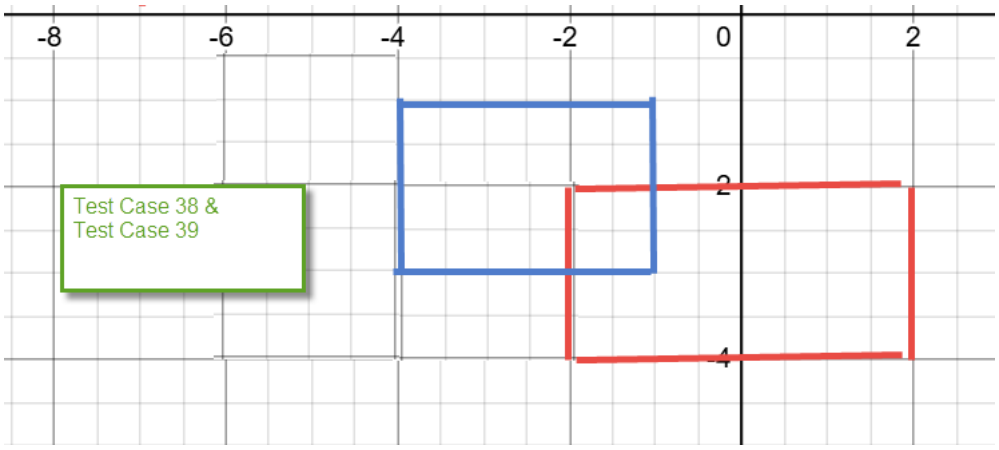
```
//TEST CASE 37 - flip of the above
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-5, 0},  new int[]{-1, 3},new int[]{-6, -2}, new int[]{-4, 2}));
//
```

```
The overlapping area is: 2
```

This is exactly positive news, so it suggests that if one rectangle crosses the y axis, there is no impact.

Perhaps it would make sense now examining outcome if one rectangle crossed the y axis

<u>TEST CASE: FAIL</u> (We can see that generic logical is not compliant when a rectangle crosses the y axis)

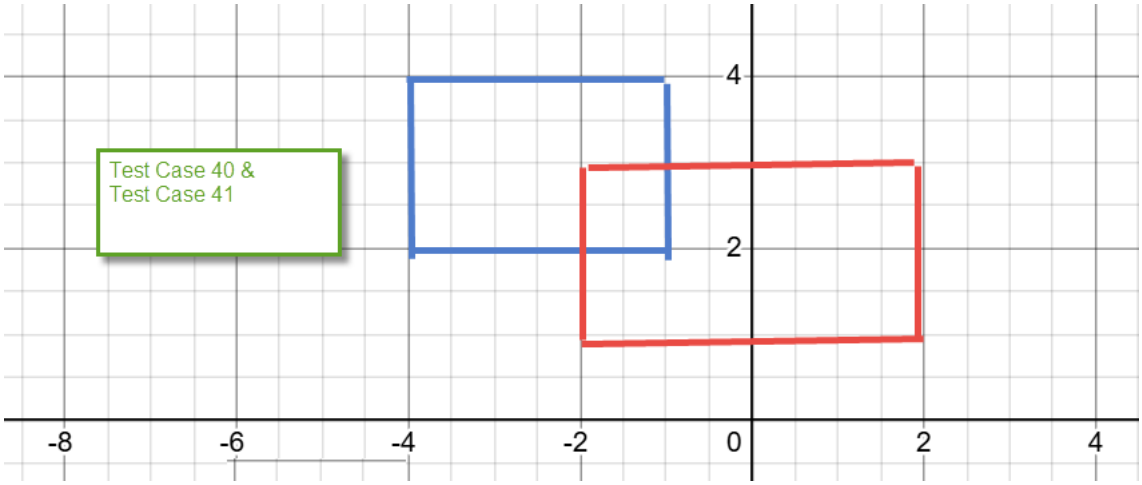



```
//TEST CASE 38
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-4, -3},  new int[]{-1, -1},new int[]{-2, -4}, new int[]{2, 2}));
//
```

```
rect1Topright: -1
rect2TopRight: 2
rect1BottomLeft: -3
rect2BottomLeft: -4
HERE1
1
3
The overlapping area is: 3
```

It might be worth seeing if the fail is still the same if we perform above x axis

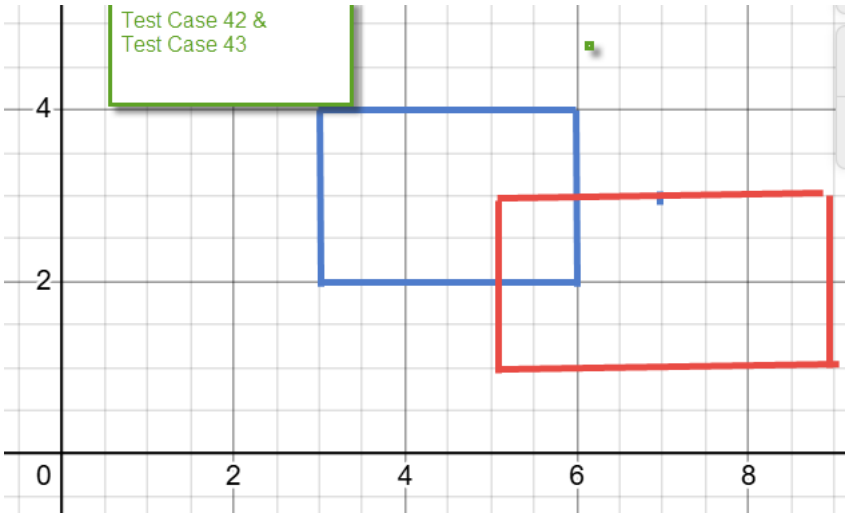
TEST CASE:




```
//TEST CASE 40 - flip of the above
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-4, 2}, new int[]{-1, 4},new int[]{-2, 1},  new int[]{2, 3}));
//
```

```
rect2BottomLeft: 2
1
-1
MUST7
The overlapping area is: 1
```

I will examine this in the positive axis



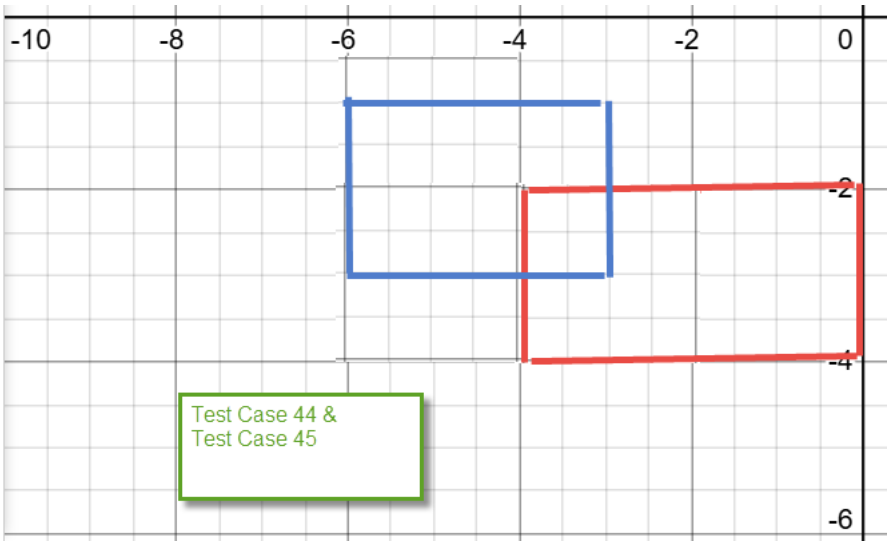

```
//TEST CASE 42 - flip of the above
System.out.println("The overlapping area is: " + overlappingArea(new int[]{3, 2},  new int[]{6, 4},new int[]{5, 1}, new int[]{9, 3}));
```

```
rect2BottomLeft: 1
-1
-1
MUST8
The overlapping area is: 1
```

SO, it appears that all the failed test cases are below the X axis and in which a shape is crossing the Y axis.

So just to confirm this is the case, I will run the same test above but ensure that the most right rectangle is resting exactly on the Y axis...

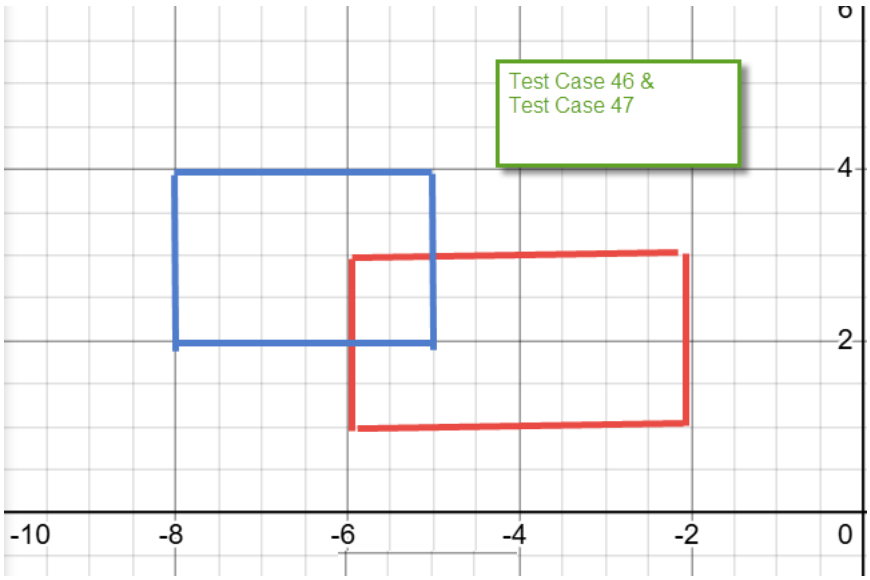
<u>TEST CASE:</u>



```
//TEST CASE 44 - flip of the above
//System.out.println("The overlapping area is: " + overlappingArea(new int[]{-4, -4}, new int[]{0, -2},new int[]{-6, -3},  new int[]{-3, 1}));
//

 //TEST CASE 45 - flip of the above
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-6, -3},  new int[]{-3, 1},new int[]{-4, -4}, new int[]{0, -2}));
//
```

```
rect2BottomLeft: -4
1
1
MUST8
The overlapping area is: 1
```

I will also run the test above the x axis but in negative quadrant:



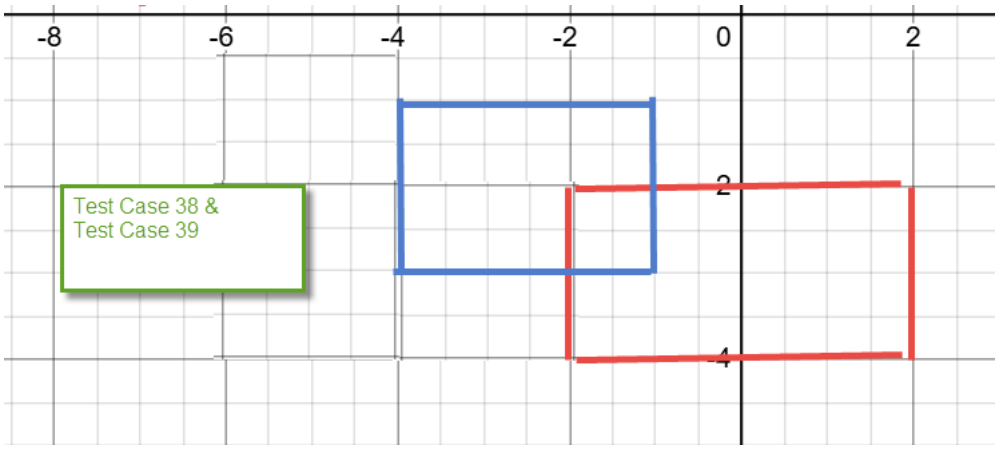

```
//TEST CASE 46 - flip of the above
//System.out.println("The overlapping area is: " + overlappingArea(new int[]{-6, 1},  new int[]{-2, 3},new int[]{-8, 2}, new int[]{-5, 4}));
//

//TEST CASE 47 - flip of the above
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-8, 2}, new int[]{-5, 4},new int[]{-6, 1},  new int[]{-2, 3}));
//
```

```
rect2BottomLeft: 2
1
-1
MUST7
The overlapping area is: 1
```

So it appears I have finally gained an understanding of the overlaps and also the implications of crossing the y axis.

So now, I will closely analyse the code flow for the following case. I am guessing at the area it fails, I need to include further logic to check if the rectangle crosses the y axis...
Once I am satisfied with this, perhaps I need to also look at the outcome if two rectangles cross the Y axis..

TEST CASE: Examining code flow



I performed following code modifications:

```
else
{
    //This will be new code here to ascertain if a rectangle appears on both sides of the Y axis.
    //Based on the above statement, we know that rectangle1 top right coordinate is less than rect2 top right
    //we will know check if rect2 runs on both sides of the Y axis
    //At moment, we do not know implications if both rectangles cross the Y axis.. But it is ignored at this moment in time
    //This is in reference to test case 38 and 39
    if (rect2BottomLeft[0]<0 && rect2TopRight[0]>0)
    {
        width = Math.abs(0-rect2BottomLeft[0]) - Math.abs(0-rect1TopRight[0]);
        System.out.println(width);
        height = Math.abs(0-rect2TopRight[1]) - Math.abs(0-rect1BottomLeft[1]);
        System.out.println(width);
         System.out.println("MUST15");
    }

    else
    {

        System.out.println("HERE1");
        width = Math.abs(0-rect2BottomLeft[0]) - Math.abs(0-rect1TopRight[0]);
        System.out.println(width);
        height = Math.abs(0-rect2BottomLeft[1]) - Math.abs(0-rect1TopRight[1]);
        System.out.println(height);
    }
```

To check if a rectangle spans across Y axis

I also performed same again, but this referring to rectangle1

```java
                                //This will be new code here to ascertain if a rectangle appears on both sides of the Y axis.
        //Based on the above statement, we know that rectangle1 top right coordinate is less than rect2 top right
        //we will know check if rect2 runs on both sides of the Y axis
        //At moment, we do not know implications if both rectangles cross the Y axis.. But it is ignored at this moment in time
        //This is in reference to test case 38 and 39
        if (rect1BottomLeft[0]<0 && rect1TopRight[0]>0)
        {
            width = Math.abs(0-rect1BottomLeft[0]) - Math.abs(0-rect2TopRight[0]);
            System.out.println(width);
            height = Math.abs(0-rect1TopRight[1]) - Math.abs(0-rect2BottomLeft[1]);
            System.out.println(width);
             System.out.println("MUST16");
        }
                                else
                                {
                            System.out.println("HERE2");
                            width = Math.abs(0-rect1BottomLeft[0]) - Math.abs(0-rect2TopRight[0]);
                            System.out.println(width);
                            height = Math.abs(0-rect1BottomLeft[1]) - Math.abs(0-rect2TopRight[1]);
                            System.out.println(height);
                                }

                            return (Math.abs(width) * Math.abs(height));
```

To check if a rectangle spans across Y axis

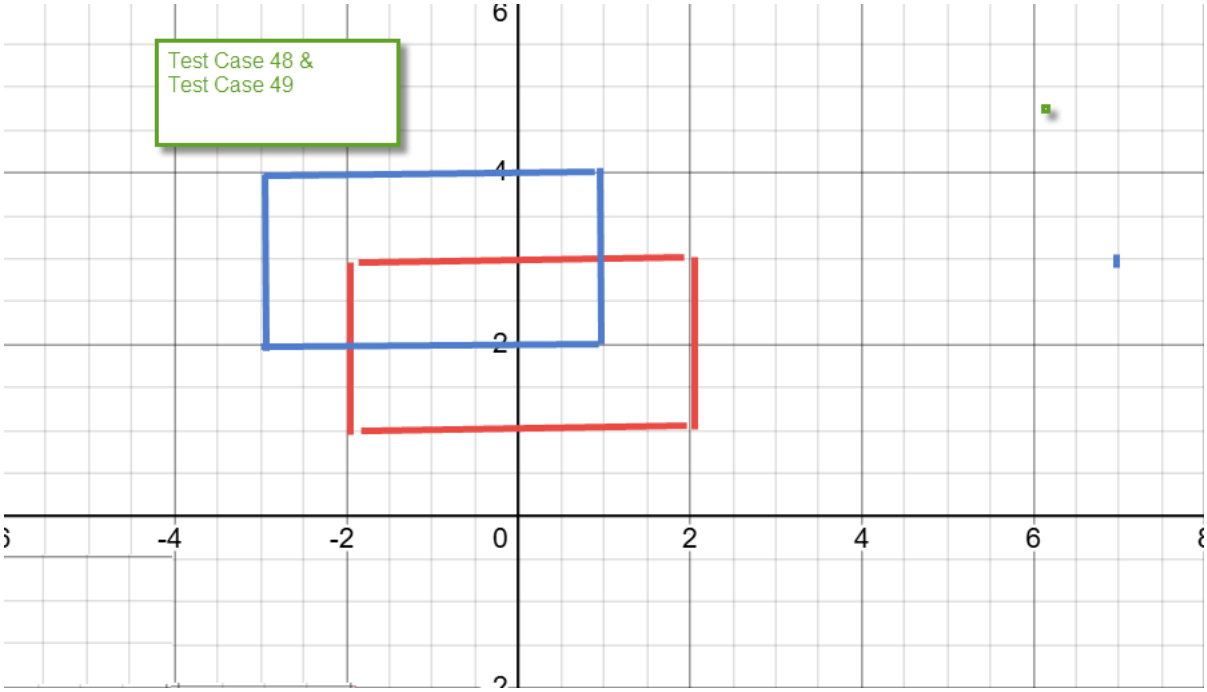
```java
//TEST CASE 38
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-4, -3},  new int[]{-1, -1},new int[]{-2, -4}, new int[]{2, 2}));
//

   //TEST CASE 39 - flip of the above
//System.out.println("The overlapping area is: " + overlappingArea(new int[]{-2, -4}, new int[]{2, 2},new int[]{-4, -3},  new int[]{-1, -1}));
//
```

```
rect2TopRight: 2
rect1BottomLeft: -3
rect2BottomLeft: -4
1
1
MUST15
The overlapping area is: 1
```

So, I am now going to explore both rectangles crossing the Y axis.

I will first try above X axis, since this was an area where there were there were no known issues..

TEST CASE: FAIL




```
//TEST CASE 48
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-2, 1}, new int[]{2, 3},new int[]{-3, 2},  new int[]{1, 4}));
//
```

```
rect2TopRight: 4
rect1BottomLeft: 1
rect2BottomLeft: 2
1
-1
MUST7
The overlapping area is: 1
```

```
//TEST CASE 49
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-3, 2},  new int[]{1, 4},new int[]{-2, 1}, new int[]{2, 3}));
//
```

```
rect2TopRight: 3
rect1BottomLeft: 2
rect2BottomLeft: 1
1
-1
MUST8
The overlapping area is: 1
```

I will now quickly investigate the area in this section of code and branch it with if else (similar to 1 rectangle spanning the y axis).
I have modified this area of code

```java
else
{
    //this now tackles test cases 48 and 49 where both rectangles span across the Y axis
    if ((rect1BottomLeft[0]<0) && (rect1TopRight[0]>0)
        && (rect2BottomLeft[0]<0) && (rect2TopRight[0]>0))
    {
       width = Math.abs(0-rect2BottomLeft[0]) - Math.abs(0-rect1TopRight[0]);
       System.out.println(width);
       height = Math.abs(0-rect1BottomLeft[1]) - Math.abs(0-rect2TopRight[1]);
       System.out.println(height);
       System.out.println("MUST20");
    }

    else
    {

    //width = Math.abs(0-rect2BottomLeft[0]) - Math.abs(0-rect1TopRight[0]);

    width = Math.abs(0-rect1BottomLeft[0]) - Math.abs(0-rect2TopRight[0]);
    System.out.println(width);
    height = Math.abs(0-rect2BottomLeft[1]) - Math.abs(0-rect1TopRight[1]);
    System.out.println(height);
    System.out.println("MUST7");
    }
```

New code to tackle both rectangles stretching across Y

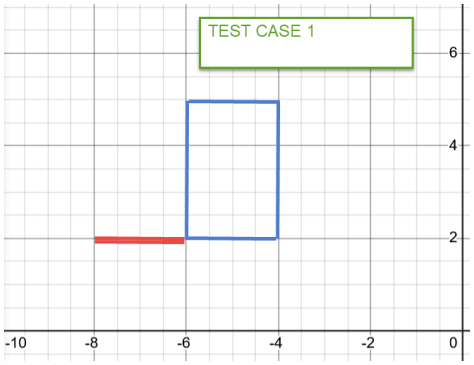
Once again, I have to make identical change in which the rectangles are swapped when parameter is passed to the method argument.

```java
else
{

    //this now tackles test cases 48 and 49 where both rectangles span across the Y axis
    if ((rect2BottomLeft[0]<0) && (rect2TopRight[0]>0)
        && (rect1BottomLeft[0]<0) && (rect1TopRight[0]>0))
    {
       width = Math.abs(0-rect1BottomLeft[0]) - Math.abs(0-rect2TopRight[0]);
       System.out.println(width);
       height = Math.abs(0-rect2BottomLeft[1]) - Math.abs(0-rect1TopRight[1]);
       System.out.println(height);
       System.out.println("MUST21");
    }

    else
    {
    width = Math.abs(0-rect2BottomLeft[0]) - Math.abs(0-rect1TopRight[0]);
    System.out.println(width);
    height = Math.abs(0-rect1BottomLeft[1]) - Math.abs(0-rect2TopRight[1]);
    System.out.println(height);
    System.out.println("MUST8");

    }
    return (Math.abs(width) * Math.abs(height));
}
```

New code to tackle both rectangles stretching across Y

I believe I have covered all the test scenarios possible
I am going to formally present all test cases. If I identify an arrangement in which I feel I should explore with a new scenario, I will present it and it will be identifiable in my test cases in the code.

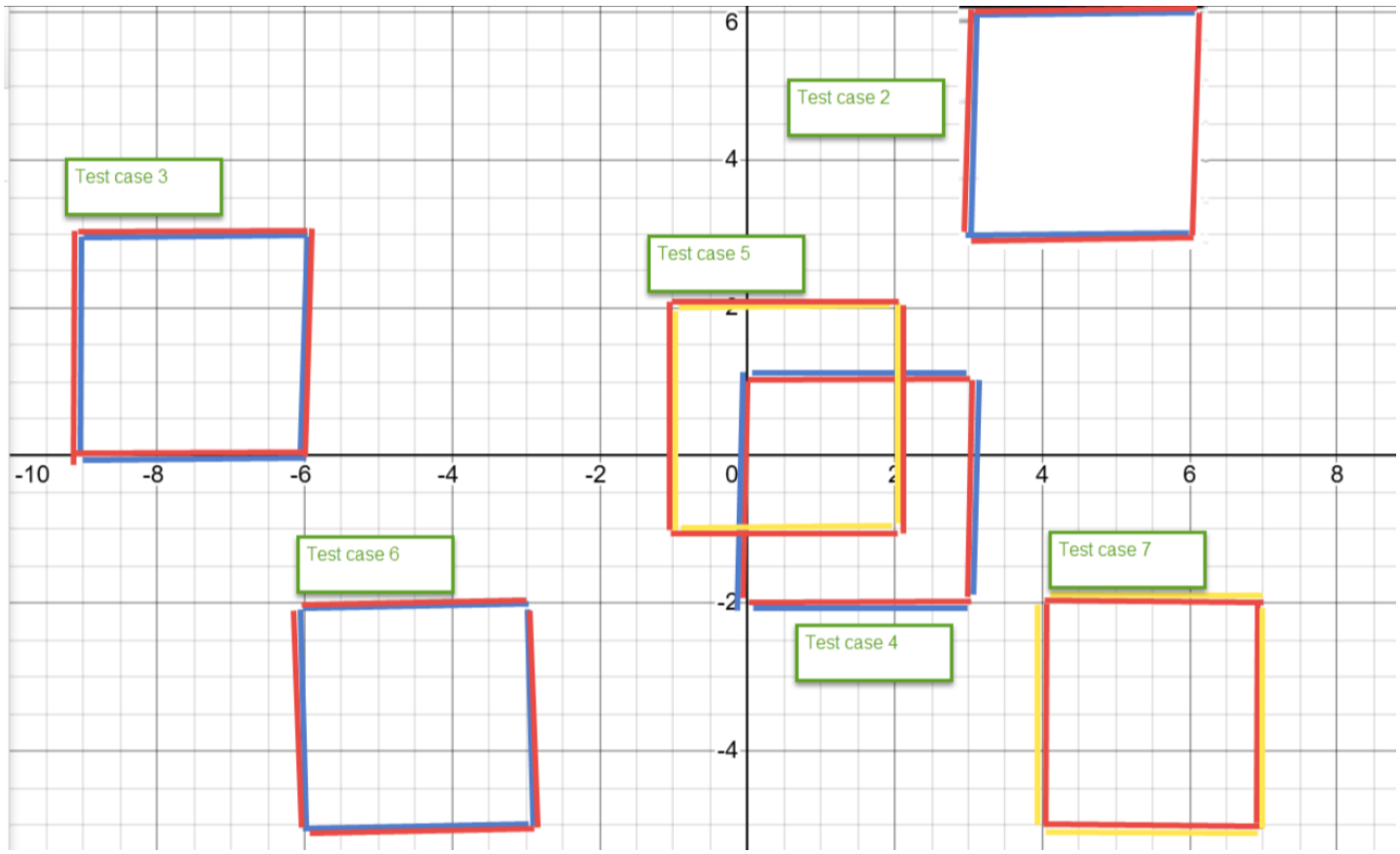
TEST CASE 1:




```
  //TEST CASE 1 - NO OVERLAP
 //System.out.println("The overlapping area is: " + overlappingArea(new int[]{-6, 5},  new int[]{-5, 5},new int[]{-8, 2}, new int[]{-4, 3}));
//                                                        //rect1bottomLeft     //rect1TopRight   //rect2BottomLeft    //rect2TopRight
```

```
NO OVERLAP FOUND
```

TEST CASE 2=>7




```
 //TEST CASE 2 - EXACT OVERLAP
  System.out.println("The overlapping area is: " + overlappingArea(new int[]{3, 3},  new int[]{6, 6},new int[]{3, 3}, new int[]{6, 6}));
//                                                        //rect1bottomLeft     //rect1TopRight   //rect2BottomLeft    //rect2TopRight
```

```
rect1Topright: 6
rect2TopRight: 6
rect1BottomLeft: 3
rect2BottomLeft: 3
EXACT overlap
-3
-3
The overlapping area is: 9
```

```
//TEST CASE 3 - EXACT OVERLAP
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-9, 0},  new int[]{-6, 3},new int[]{-9, 0}, new int[]{-6, 3}));
//                                                    //rect1bottomLeft     //rect1TopRight   //rect2BottomLeft    //rect2TopRight
```

```
rect1Topright: 3
rect2TopRight: 3
rect1BottomLeft: 0
rect2BottomLeft: 0
EXACT overlap
3
-3
The overlapping area is: 9
```

```
//TEST CASE 4 - EXACT OVERLAP
System.out.println("The overlapping area is: " + overlappingArea(new int[]{0, -2},  new int[]{3, 1},new int[]{0, -2}, new int[]{3, 1}));
//                                                    //rect1bottomLeft     //rect1TopRight   //rect2BottomLeft    //rect2TopRight
```

```
rect1Topright: 1
rect2TopRight: 1
rect1BottomLeft: -2
rect2BottomLeft: -2
EXACT overlap
The overlapping area is: 9
```

```
//TEST CASE 5 - EXACT OVERLAP
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-1, -1},  new int[]{2, 2},new int[]{-1, -1}, new int[]{2, 2}));
//                                                    //rect1bottomLeft     //rect1TopRight   //rect2BottomLeft    //rect2TopRight
```

```
rect1Topright: 2
rect2TopRight: 2
rect1BottomLeft: -1
rect2BottomLeft: -1
EXACT overlap
The overlapping area is: 9
```

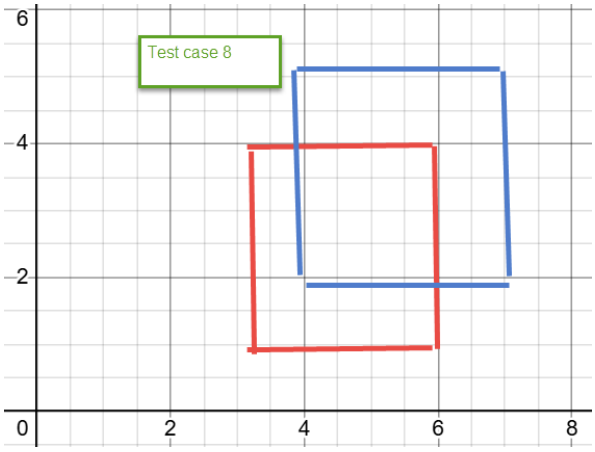
```
//TEST CASE 6 - EXACT OVERLAP
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-6, -5},  new int[]{-3, -2},new int[]{-6, -5}, new int[]{-3, -2}));
//                                                    //rect1bottomLeft     //rect1TopRight   //rect2BottomLeft    //rect2TopRight
```

```
rect1Topright: -2
rect2TopRight: -2
rect1BottomLeft: -5
rect2BottomLeft: -5
EXACT overlap
3
3
The overlapping area is: 9
```

```
//TEST CASE 7 - EXACT OVERLAP
    System.out.println("The overlapping area is: " + overlappingArea(new int[]{4, -5},  new int[]{7, -2},new int[]{4, -5}, new int[]{7, -2}));
  //                                                    //rect1bottomLeft     //rect1TopRight   //rect2BottomLeft    //rect2TopRight
```

```
rect1Topright: -2
rect2TopRight: -2
rect1BottomLeft: -5
rect2BottomLeft: -5
EXACT overlap
-3
3
The overlapping area is: 9
```

TEST CASE 8:




```
//TEST CASE 8
    System.out.println("The overlapping area is: " + overlappingArea(new int[]{3, 1},  new int[]{6, 4},new int[]{4, 2}, new int[]{7, 5}));
  //                                                    //rect1bottomLeft     //rect1TopRight   //rect2BottomLeft    //rect2TopRight
```

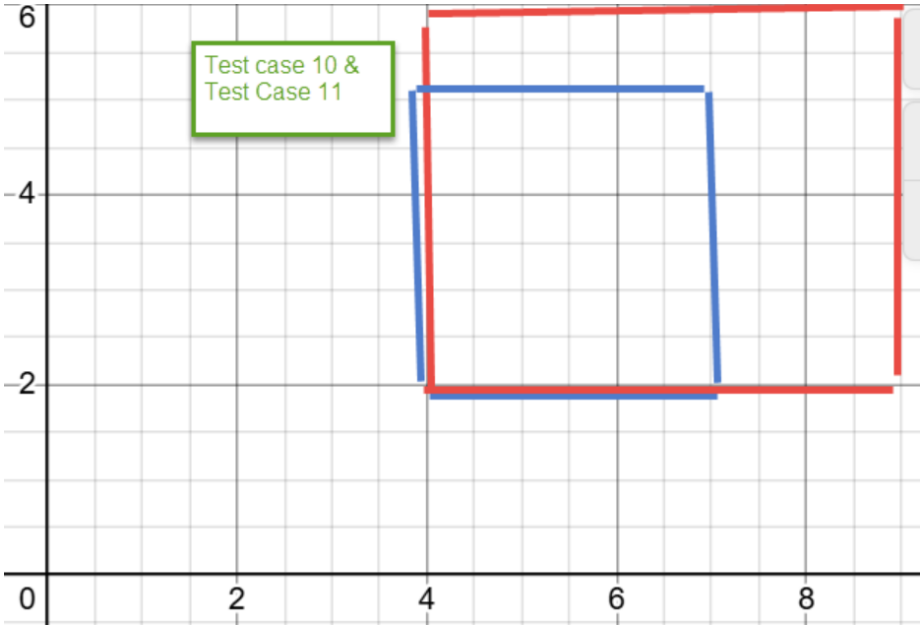
```
rect1Topright: 4
rect2TopRight: 5
rect1BottomLeft: 1
rect2BottomLeft: 2
HERE1
-2
-2
The overlapping area is: 4
```

TEST CASE 9: PASSING RECTANGLE2 FIRST INTO THE METHOD.....

```
//TEST CASE 9 - (flip of test case 8)
    System.out.println("The overlapping area is: " + overlappingArea(new int[]{4, 2}, new int[]{7, 5},new int[]{3, 1},  new int[]{6, 4}));
  //                                                    //rect1bottomLeft     //rect1TopRight   //rect2BottomLeft    //rect2TopRight
```

```
rect1Topright: 4
rect2TopRight: 5
rect1BottomLeft: 1
rect2BottomLeft: 2
HERE1
-2
-2
The overlapping area is: 4
```

TEST CASE 10 and TEST CASE 11:




```
//TEST CASE 10
    System.out.println("The overlapping area is: " + overlappingArea(new int[]{4, 2}, new int[]{7, 5},new int[]{4, 2},  new int[]{9, 6}));
//                                                          //rect1bottomLeft     //rect1TopRight   //rect2BottomLeft    //rect2TopRight
```

```
rect1Topright: 5
rect2TopRight: 6
rect1BottomLeft: 2
rect2BottomLeft: 2
MUST1
-3
-3
The overlapping area is: 9
```

```
//TEST CASE 11   same as above flipped
    System.out.println("The overlapping area is: " + overlappingArea(new int[]{4, 2},  new int[]{9, 6},new int[]{4, 2}, new int[]{7, 5}));
  //
```

```
rect1Topright: 6
rect2TopRight: 5
rect1BottomLeft: 2
rect2BottomLeft: 2
MUST3
-3
-3
The overlapping area is: 9
```

TEST CASE 12 and TEST CASE 13:


```
//TEST CASE 12   this will follow same principle as test case 10
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{3, 1},  new int[]{9, 6},new int[]{4, 2}, new int[]{7, 5}));
//
```

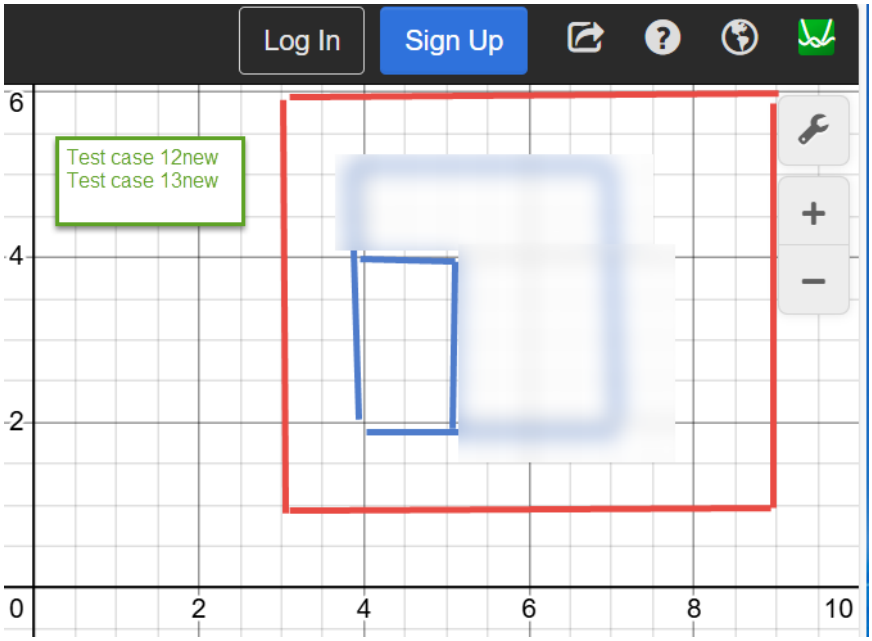
```
rect1Topright: 6
rect2TopRight: 5
rect1BottomLeft: 1
rect2BottomLeft: 2
MUST3
-3
-3
The overlapping area is: 9
```

```
//TEST CASE 13   same as above, but flipped
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{4, 2}, new int[]{7, 5},new int[]{3, 1},  new int[]{9, 6}));
//
```

```
rect1Topright: 5
rect2TopRight: 6
rect1BottomLeft: 2
rect2BottomLeft: 1
MUST1
-3
-3
The overlapping area is: 9
```

TEST CASE 12new and TEST CASE 13new:

I will now explore with a smaller inner rectangle



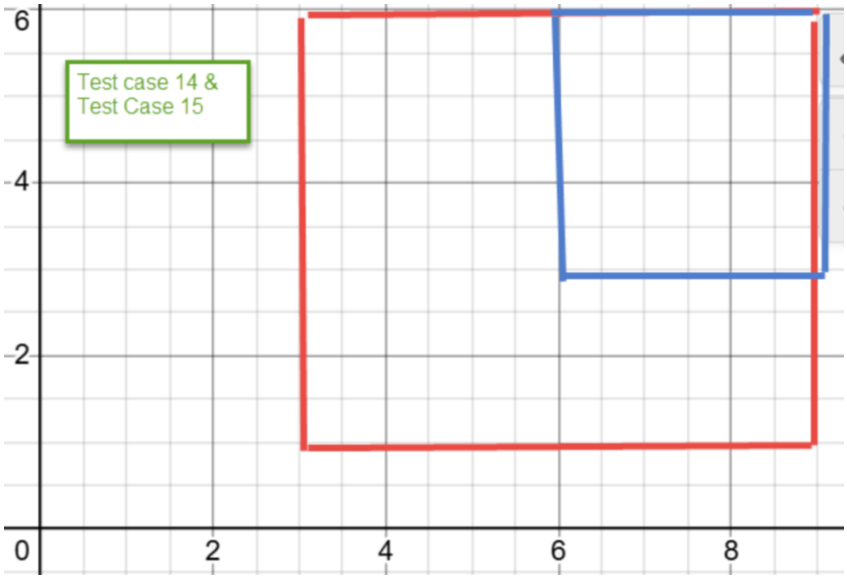

```
//TEST CASE 12NEW   this will follow same principle as test case 10
  //System.out.println("The overlapping area is: " + overlappingArea(new int[]{3, 1},  new int[]{9, 6},new int[]{4, 2}, new int[]{5, 4}));
//
```

```
rect1Topright: 6
rect2TopRight: 4
rect1BottomLeft: 1
rect2BottomLeft: 2
MUST3
-1
-2
The overlapping area is: 2
```

```
//TEST CASE 13NEW   this will follow same principle as test case 10
  System.out.println("The overlapping area is: " + overlappingArea(new int[]{4, 2}, new int[]{5, 4},new int[]{3, 1},  new int[]{9, 6}));
//
```

```
rect1Topright: 4
rect2TopRight: 6
rect1BottomLeft: 2
rect2BottomLeft: 1
MUST1
-1
-2
The overlapping area is: 2
```

TEST CASE 14 and TEST CASE 15: FAIL




```
  //TEST CASE 14
  System.out.println("The overlapping area is: " + overlappingArea(new int[]{3, 1}, new int[]{9, 6},new int[]{6, 3},  new int[]{9, 6}));
//
```

```
rect1Topright: 6
rect2TopRight: 6
rect1BottomLeft: 1
rect2BottomLeft: 3
-6
-3
MUST7
test
The overlapping area is: 18
```

We can see that it has obtained the height (3) to be correct....
But for some reason, it has obtained width (8) which is totally incorrect.
It has appeared in this section of code in the following test case scenarios:
48,49 40,41

But first, I will try test case 15

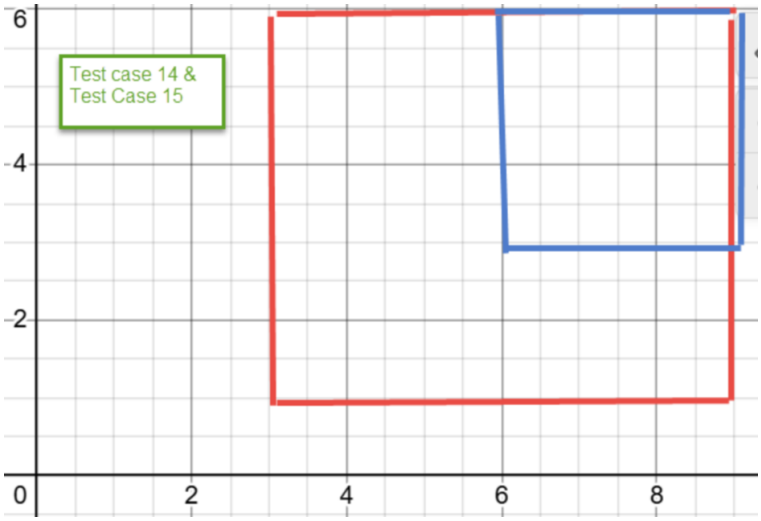
TEST CASE 15: FAIL

```
    //TEST CASE 15   same as above, but flipped
  System.out.println("The overlapping area is: " + overlappingArea(new int[]{6, 3},  new int[]{9, 6},new int[]{3, 1}, new int[]{9, 6}));
//
```

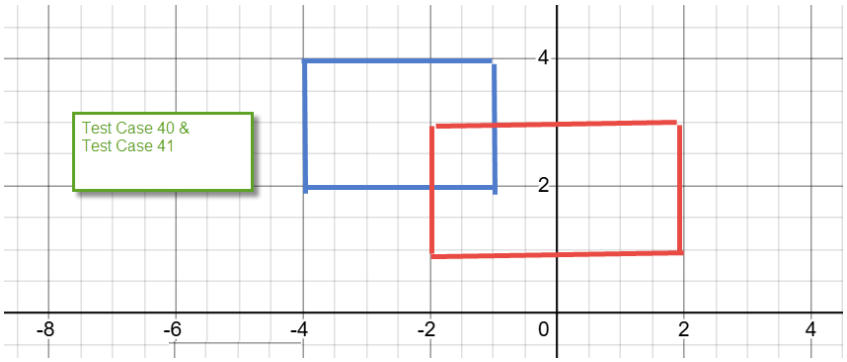
```
rect1Topright: 6
rect2TopRight: 6
rect1BottomLeft: 3
rect2BottomLeft: 1
MUST1
-3
-3
-6
-3
MUST8
The overlapping area is: 18


** Process exited - Return Code: 0 **
```

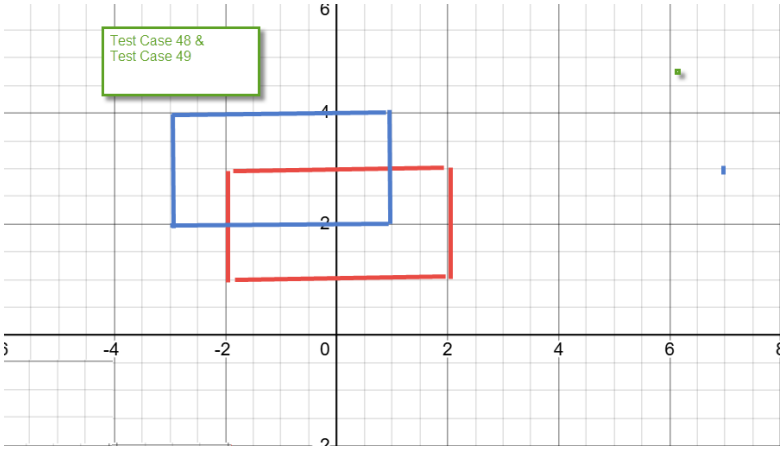
I am going to really need to understand 48,49 40,41 with 14 and 15

ISSUES HERE AND SAME AREA OF CODE.. I can only speculate that I need to reference if the rectXBottom (X,Y) coincides with both rectangles or the rectXTopRight (X,Y) coincides with both rectangles.. We would then to calculate the overlap area to be the nested rectangle...
I would need to perform this for both scenarios
I think all this is in respect to having widened the scope of the main if loop... I allowed certain scenarios enter based on coincinding coordinates.. And most likely it requires re-instating in this area of code... It has to be remembered that several areas of code are effectively clashing in this main if block...



<u>NO ISSUES HERE AND SAME AREA OF CODE</u>



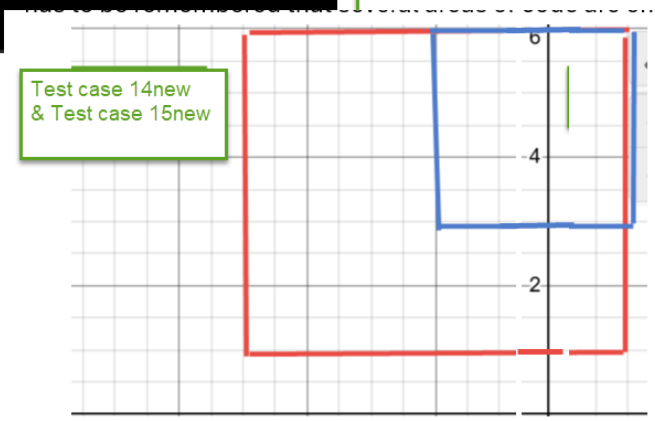



```
263
264                 else
265                 {
266                     //this now tackles test cases 48 and 49 where both rectangles span across the Y axis
267                     if ((rect1BottomLeft[0]<0) && (rect1TopRight[0]>0)
268                         && (rect2BottomLeft[0]<0) && (rect2TopRight[0]>0))
269                     {
270                        width = Math.abs(0-rect2BottomLeft[0]) - Math.abs(0-rect1TopRight[0]);
271                        System.out.println(width);
272                        height = Math.abs(0-rect1BottomLeft[1]) - Math.abs(0-rect2TopRight[1]);
273                        System.out.println(height);
274                        System.out.println("MUST20");
275                     }
276
277                     else
278                     {
279
280                     //width = Math.abs(0-rect2BottomLeft[0]) - Math.abs(0-rect1TopRight[0]);
281
282                     width = Math.abs(0-rect1BottomLeft[0]) - Math.abs(0-rect2TopRight[0]);
283                     System.out.println(width);
284                     height = Math.abs(0-rect2BottomLeft[1]) - Math.abs(0-rect1TopRight[1]);
285                     System.out.println(height);
286                     System.out.println("MUST7");
287                     }
288
289                     System.out.println("test");
290                     return (Math.abs(width) * Math.abs(height));
291                 }
```

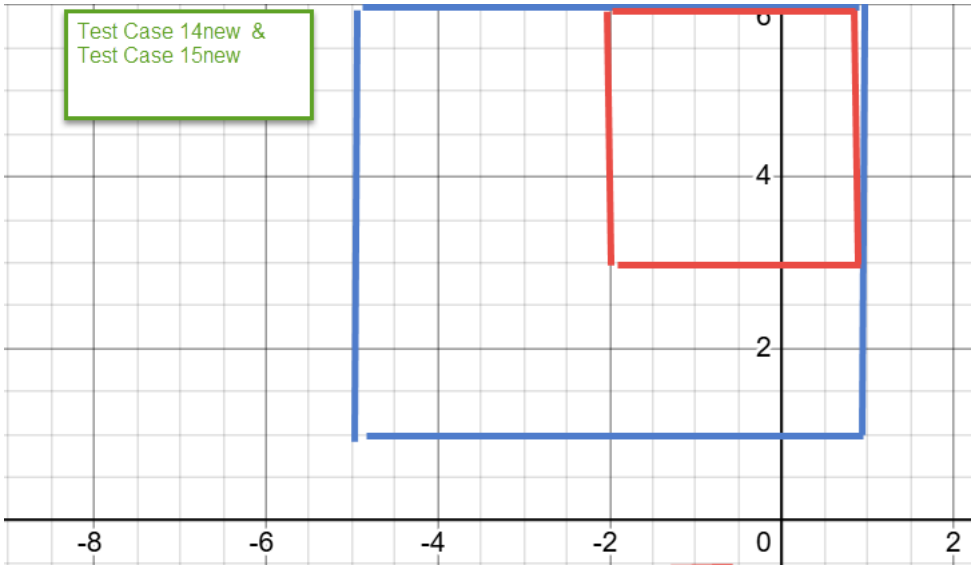
I am looking to make the change in this section area of code...
But if the associated if is examined, we are dealing with both rectangles spanning the X axis..

So on paper it makes absolutely no sense in performing this implementation in the else statement without looking at the if section,,,,

I think best logical approach is to draw the two rectangles stretching across the Y axis



TEST CASE 14new and TEST CASE 15new: FAIL




```
//TEST CASE 14new   same as above, but flipped
  //System.out.println("The overlapping area is: " + overlappingArea(new int[]{-5, 6},  new int[]{1, 6},new int[]{-2, 3}, new int[]{1, 6}));
//

//TEST CASE 15new   same as above, but flipped
  System.out.println("The overlapping area is: " + overlappingArea(new int[]{-2, 3}, new int[]{1, 6},new int[]{-5, 6},  new int[]{1, 6}));
//
```

```
rect1Topright: 6
rect2TopRight: 6
rect1BottomLeft: 1
rect2BottomLeft: 3
1
-5
MUST20
test
The overlapping area is: 5
```

```
//TEST CASE 15new   same as above, but flipped
  System.out.println("The overlapping area is: " + overlappingArea(new int[]{-2, 3}, new int[]{1, 6},new int[]{-5, 1},  new int[]{1, 6}));
//
```

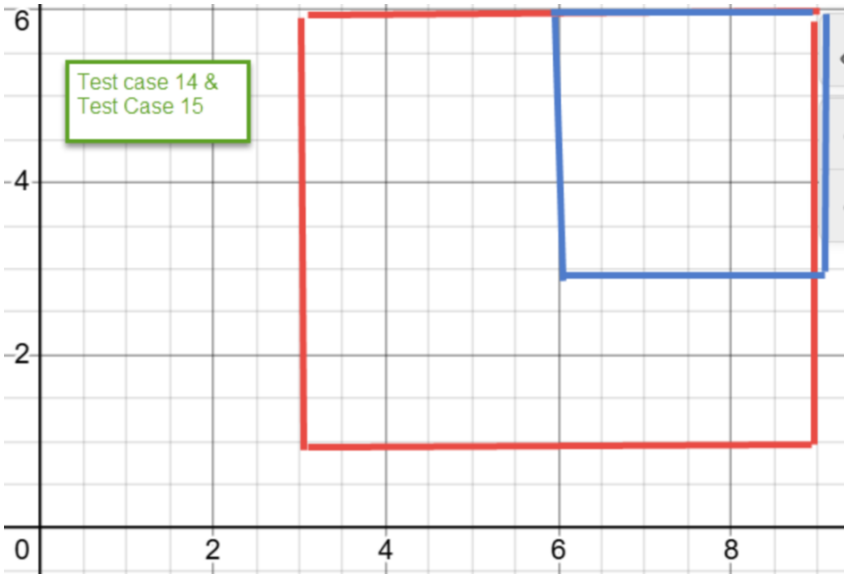
```
rect1Topright: 6
rect2TopRight: 6
rect1BottomLeft: 3
rect2BottomLeft: 1
MUST1
1
-3
1
-5
MUST21
The overlapping area is: 5
```

So, my plan of action will be to address issue with test case14 and test case 15

And then I can re-visit test case 14new and test case 15new

I am only going to implement with rogue areas the code enters.. I am hoping all test cases will sort itself out

<u>TEST CASE 14 and TEST CASE 15:</u> Re-visiting... I will just implement in the area that the code enters....




```
//TEST CASE 15   same as above, but flipped
System.out.println("The overlapping area is: " + overlappingArea(new int[]{6, -3},  new int[]{9, 6},new int[]{3, -1}, new int[]{9, 6}));
//
```

This would be the new if statement

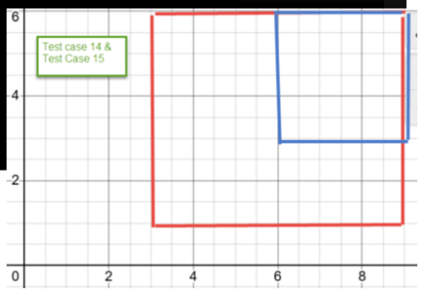
if (rect1TopRight[0]==rect2TopRight[0] && rect1TopRight[1]==rect2TopRight[1])
We now check if the x coordinate (bottom left) of rectangle1 is bigger or smaller than rectangle 2 (if rect1BottomLeft[0]>rect2BottomLeft[0])
If bigger width = Math.abs(rect1BottomLeft[0]) - Math.abs(rect1TopRight[0]) height = Math.abs(rect1BottomLeft[1]) - Math.abs(rect1TopRight[1])
else width = Math.abs(rect2BottomLeft[0]) - Math.abs(rect2TopRight[0]) height = Math.abs(rect2BottomLeft[1]) - Math.abs(rect2TopRight[1])

```
                else
                {

                //when examinin
                //instead we need the width and height of the nested rectangle
                //I can only speculate that I need to reference if the rectXBottom (X,Y) coincides with both rectangles
                //or the rectXTopRight (X,Y) coincides with both rectangles..
                //We would then to calculate the overlap area to be the nested rectangle…
                //I also believe we are in area of code where its confirmed if it nested, so just interested if either top right or bottom left pair up
                //But right now, it just does not seem so obvious if it will make right decision on with rectangle to process.
                //But if I do same for both scenarios, this should suffice
                //In practice this seems very inconsistent with associated if...

                //width = Math.abs(0-rect2BottomLeft[0]) - Math.abs(0-rect1TopRight[0]);

                width = Math.abs(0-rect1BottomLeft[0]) - Math.abs(0-rect2TopRight[0]);
                System.out.println(width);
                height = Math.abs(0-rect2BottomLeft[1]) - Math.abs(0-rect1TopRight[1]);
                System.out.println(height);
                System.out.println("MUST7");
                }

                System.out.println("test");
                return (Math.abs(width) * Math.abs(height));
            }
```

I would also then repeat this if statement but comparing if the coordinates are the same at bottom left

This will now go into else statement....

Blue rectangle is rectangle1



This is the associated code for this shape...

```
                    if (rect1TopRight[0]==rect2TopRight[0]  && rect1TopRight[1]==rect2TopRight[1])
                    {
        //We now check if the x coordinate (bottom left) of rectangle1 is bigger or smaller than rectangle 2

                    if (rect1BottomLeft[0]>rect2BottomLeft[0])
                    {
                        width =   Math.abs(rect1BottomLeft[0])  - Math.abs(rect1TopRight[0])
                        height =  Math.abs(rect1BottomLeft[1])  - Math.abs(rect1TopRight[1])
                    }
                    else
                    {
                        width=Math.abs(rect2BottomLeft[0])  - Math.abs(rect2TopRight[0]);
                        height =  Math.abs(rect2BottomLeft[1])  - Math.abs(rect2TopRight[1]);
                    }

                    }
```

OUTPUT (TEST CASE 14)

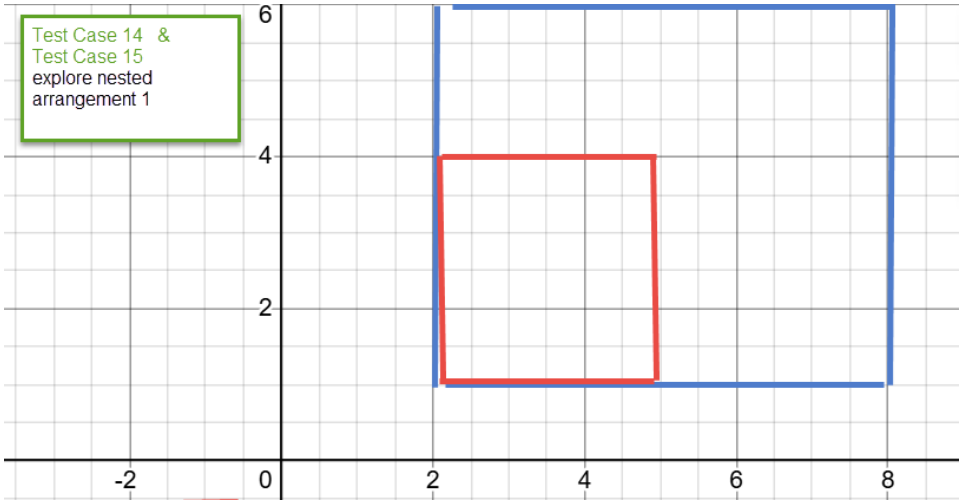
```
rect1Topright: 6
rect2TopRight: 6
rect1BottomLeft: -1
rect2BottomLeft: -3
MUST2
-3
-3
MUST23
MUST24
The overlapping area is: 9
```

OUTPUT (TEST CASE 15)

```
rect1Topright: 6
rect2TopRight: 6
rect1BottomLeft: -3
rect2BottomLeft: -1
MUST26
MUST27
test
The overlapping area is: 9
```

Now before I move onto Test case 14new and Test case 15new, I want to move the nested shape to the bottom left.. I expect there to be **issues** in this configuration...
And I will also make some new test cases where the inner shape will be nested against each corner and can also try it against the side (but not touching the corners)... It might potentially come up in my forthcoming tests already devised, but I am sure if I address now, it will save effort later...
I will try a few in each quadrant..

TEST CASE 14 and TEST CASE 15 explore nested arrangement 1:




```
//TEST CASE 14  - explore nested arrangement 1
  System.out.println("The overlapping area is: " + overlappingArea(new int[]{2, 1}, new int[]{5, 4},new int[]{2, 1},  new int[]{8, 6}));
//
```

```
rect1Topright: 4
rect2TopRight: 6
rect1BottomLeft: 1
rect2BottomLeft: 1
MUST1
-3
-3
*********************************************
AREA: 9
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 9
```

Please note it is going in the area of the code where it is calling another method. I had to use this during my implementation since it was necessary to find the smallest element. But as can be seen it has entered once...

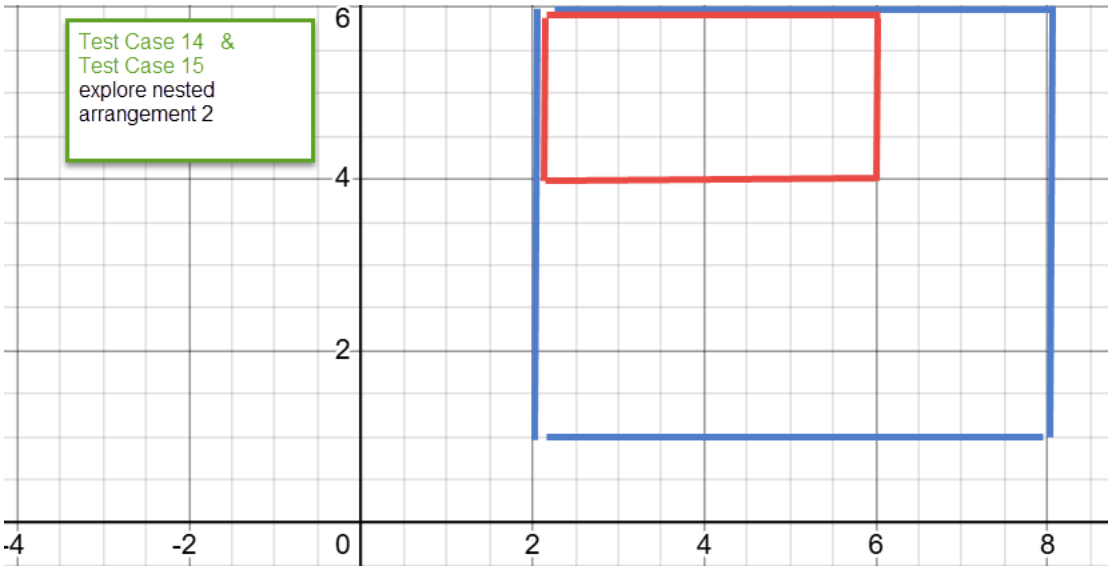
I will now try Test Case15 in the new arrangement..

```
//TEST CASE 15  - explore nested arrangement 1
  System.out.println("The overlapping area is: " + overlappingArea(new int[]{2, 1},  new int[]{8, 6},new int[]{2, 1}, new int[]{5, 4}));
//
```

```
rect1Topright: 6
rect2TopRight: 4
rect1BottomLeft: 1
rect2BottomLeft: 1
MUST3
-3
-3
*********************************************
AREA: 9
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 9
```

My next logical test case will be taking the smaller shape in the other corners..
I am also inclined to change the size also...

TEST CASE 14 and TEST CASE 15 explore nested arrangement 2:



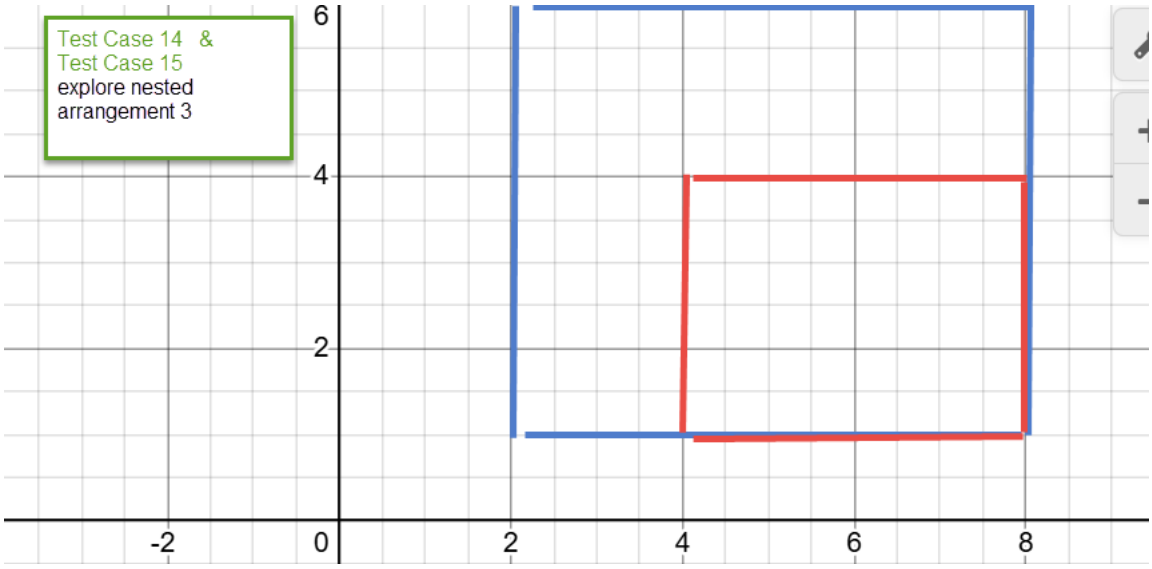

```
//TEST CASE 14  - explore nested arrangement 2
  System.out.println("The overlapping area is: " + overlappingArea(new int[]{2, 4}, new int[]{6, 6},new int[]{2, 1},  new int[]{8, 6}));
//
```

```
rect1Topright: 6
rect2TopRight: 6
rect1BottomLeft: 4
rect2BottomLeft: 1
MUST1
-4
-2
-4
-2
MUST8
The overlapping area is: 8
```

```
//TEST CASE 15  - explore nested arrangement 2
  System.out.println("The overlapping area is: " + overlappingArea(new int[]{2, 1},  new int[]{8, 6},new int[]{2, 4}, new int[]{6, 6}));
//
```

```
rect1Topright: 6
rect2TopRight: 6
rect1BottomLeft: 1
rect2BottomLeft: 4
-4
-2
MUST7
test
The overlapping area is: 8
```

TEST CASE 14 and TEST CASE 15 explore nested arrangement 3:




```
//TEST CASE 14  - explore nested arrangement 3
  System.out.println("The overlapping area is: " + overlappingArea(new int[]{2, 1},  new int[]{8, 6},new int[]{4, 1}, new int[]{8, 4}));
//
```

```
rect1Topright: 6
rect2TopRight: 4
rect1BottomLeft: 1
rect2BottomLeft: 1
MUST3
-4
-3
*********************************************
AREA: 12
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 12
```

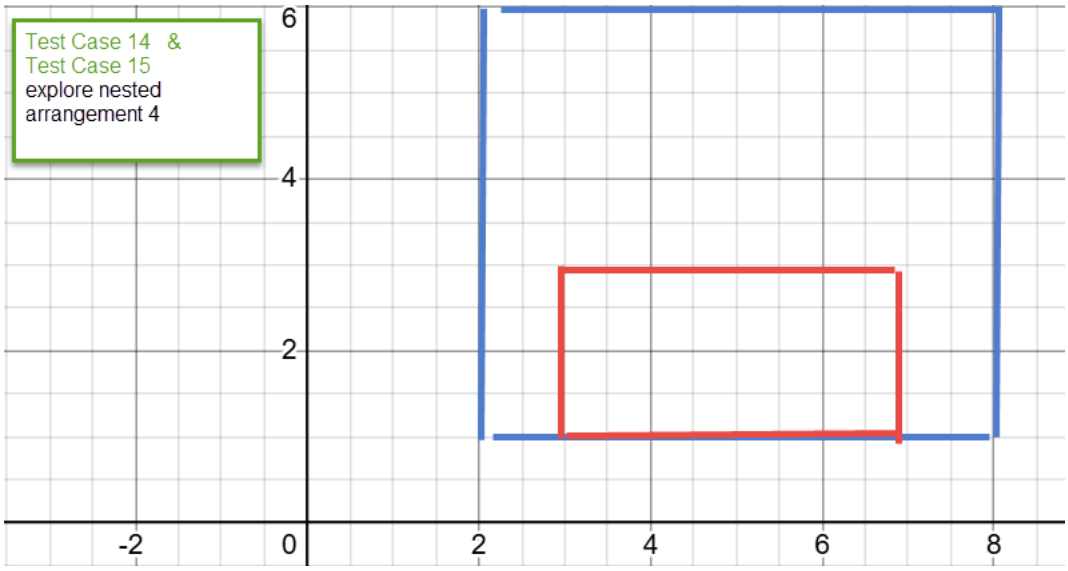
```
//TEST CASE 15  - explore nested arrangement 3
  System.out.println("The overlapping area is: " + overlappingArea(new int[]{4, 1}, new int[]{8, 4},new int[]{2, 1},  new int[]{8, 6}));
//
```

```
rect1Topright: 4
rect2TopRight: 6
rect1BottomLeft: 1
rect2BottomLeft: 1
MUST1
-4
-3
*********************************************
AREA: 12
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 12
```

So perhaps, it appears that only area of the code that required adjusting was when both shapes shared same top right hand corner as proven...

I will now move the inner shape along the edge..

TEST CASE 14 and TEST CASE 15 explore nested arrangement 4:

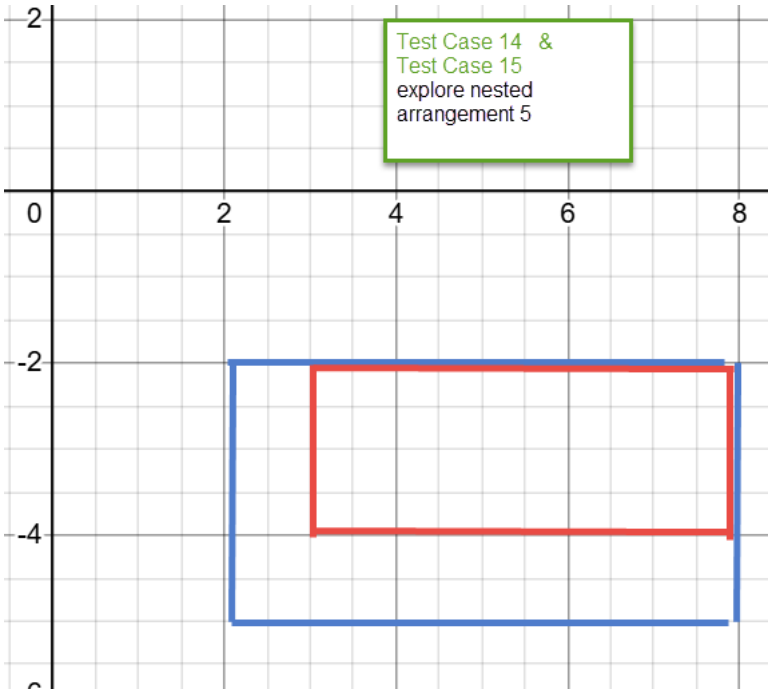



```
//TEST CASE 14  - explore nested arrangement 4
  System.out.println("The overlapping area is: " + overlappingArea(new int[]{2, 1},  new int[]{8, 6},new int[]{3, 1}, new int[]{7, 3}));
//

      //TEST CASE 15  - explore nested arrangement 4
  System.out.println("The overlapping area is: " + overlappingArea(new int[]{3, 1}, new int[]{7, 3},new int[]{2, 1},  new int[]{8, 6}));
//
```

```
rect1Topright: 6
rect2TopRight: 3
rect1BottomLeft: 1
rect2BottomLeft: 1
MUST3
-4
-2
**********************************************
AREA: 8
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 8
```

I am now going to try the configuration in different quadrants

TEST CASE 14 and TEST CASE 15 explore nested arrangement 5:



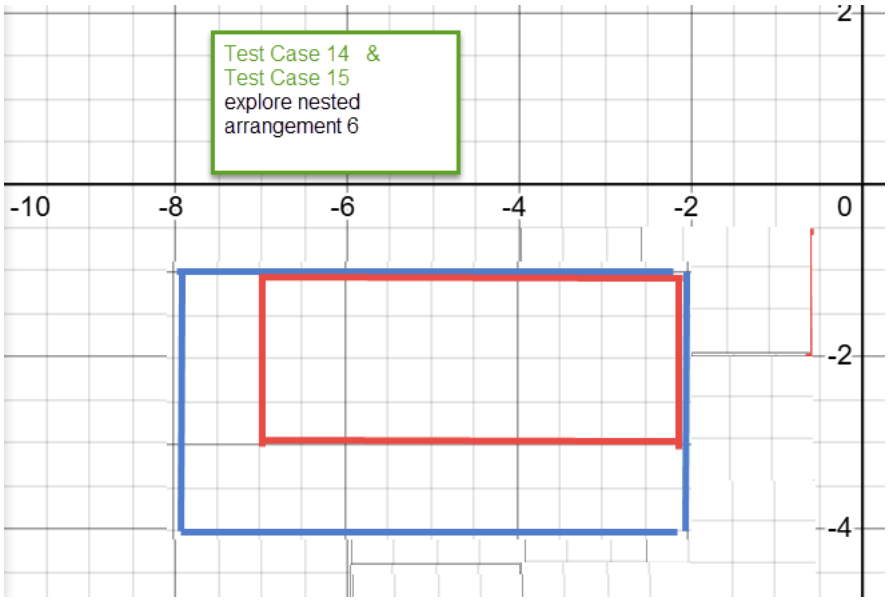

```
//TEST CASE 14  - explore nested arrangement 5
  System.out.println("The overlapping area is: " + overlappingArea(new int[]{2, -5},  new int[]{8, -2},new int[]{3, -4}, new int[]{8, -2}));
//
```

It can be seen it has hit no area of interest in the code

```
rect1Topright: -2
rect2TopRight: -2
rect1BottomLeft: -5
rect2BottomLeft: -4
**********************************************
AREA: 0
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 0
```

Before I attempt to resolve this issue, it would be good to know if the same issue persists in other quadrants...

TEST CASE 14 and TEST CASE 15 explore nested arrangement 6:



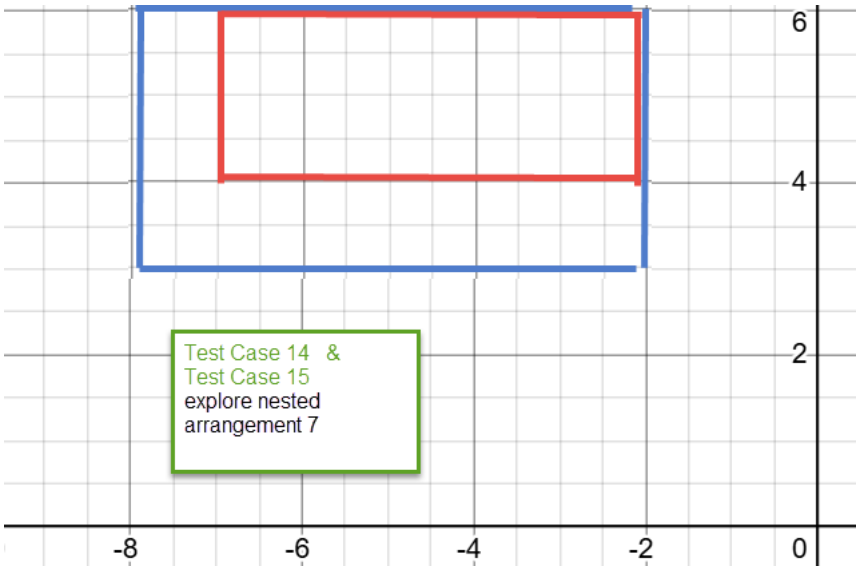

```
//TEST CASE 14  - explore nested arrangement 6
  System.out.println("The overlapping area is: " + overlappingArea(new int[]{-7, -3}, new int[]{-2, -1},new int[]{-8, -4},  new int[]{-2, -1}));
//
```

```
//TEST CASE 15  - explore nested arrangement 6
  System.out.println("The overlapping area is: " + overlappingArea(new int[]{-8, -4},  new int[]{-2, -1},new int[]{-7, -3}, new int[]{-2, -1}));
//
```

```
rect1Topright: -1
rect2TopRight: -1
rect1BottomLeft: -3
rect2BottomLeft: -4
**********************************************
AREA: 0
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 0
```

Same issue

TEST CASE 14 and TEST CASE 15 explore nested arrangement 7:




```
//TEST CASE 14  - explore nested arrangement 7
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-8, -3},  new int[]{-2, 6},new int[]{-7, 4}, new int[]{-2, 6}));
//
```

```
rect1Topright: 6
rect2TopRight: 6
rect1BottomLeft: -3
rect2BottomLeft: 4
MUST26
MUST28
test
The overlapping area is: 10
```

```
//TEST CASE 15  - explore nested arrangement 7
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-7, 4}, new int[]{-2, 6},new int[]{-8, -3},  new int[]{-2, 6}));
//
```

```
rect1Topright: 6
rect2TopRight: 6
rect1BottomLeft: 4
rect2BottomLeft: -3
MUST1
5
-2
MUST23
MUST25
The overlapping area is: 10
```

So it appears all the issues that are occurring are below the X axis.
So I will take a look to understand the rationale a bit better.

I completed the following observation immediately.

```
//all in positive quadrant
if ((rect1BottomLeft[1]>=0 && rect2BottomLeft[1]>0
&& rect1TopRight[1]>=0 && rect2TopRight[1]>=0)

|| (rect1BottomLeft[0]<0 && rect1TopRight[0]>0)  //rectangle1 width crosses the x axis
|| (rect1BottomLeft[1]<0 && rect1TopRight[1]>0)  //rectangle1 height crosses the y axis
 || (rect2BottomLeft[0]<0 && rect2TopRight[0]>0)  //rectangle2 width crosses the x axis
|| (rect2BottomLeft[1]<0 && rect2TopRight[1]>0))  //rectangle2 height crosses the y axis
```

It can be seen that none of these criteria are fulfilled to enter this main section of the if statement...

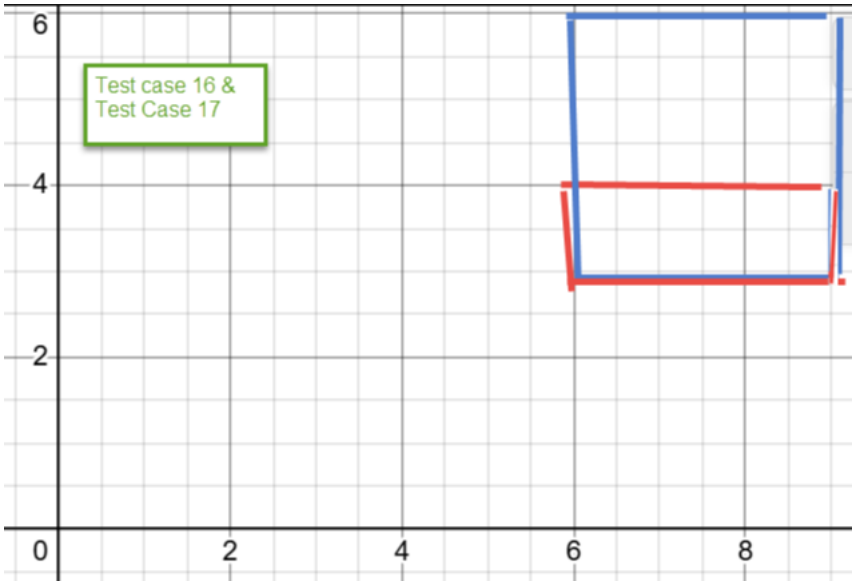
It is seeming really likely that the whole if structure needs to be stripped off.
Since we want to enter the code in all circumstances...
We know the exceptions are if exact overlap or no overlap..
Both of these have return statements and code blocks appear prior to this..

So I have chosen to remove this outer if expression altogether..
I am just going to quickly run through all test cases 14 and 15.
It shouldn't have any bearing on the test cases that already entered. I expect there to be different outcome to

TEST CASE 14 and TEST CASE 15: PASS
TEST CASE 14new and TEST CASE 15new: FAIL (to be revisited, overlap over Y axis)
TEST CASE 14 and TEST CASE 15 explore nested arrangement 1: PASS
TEST CASE 14 and TEST CASE 15 explore nested arrangement 2: PASS
TEST CASE 14 and TEST CASE 15 explore nested arrangement 3: PASS
TEST CASE 14 and TEST CASE 15 explore nested arrangement 4: PASS
TEST CASE 14 and TEST CASE 15 explore nested arrangement 5: PASS
TEST CASE 14 and TEST CASE 15 explore nested arrangement 6: PASS
TEST CASE 14 and TEST CASE 15 explore nested arrangement 7: PASS

I am now going over all my test cases from 16 onwards.

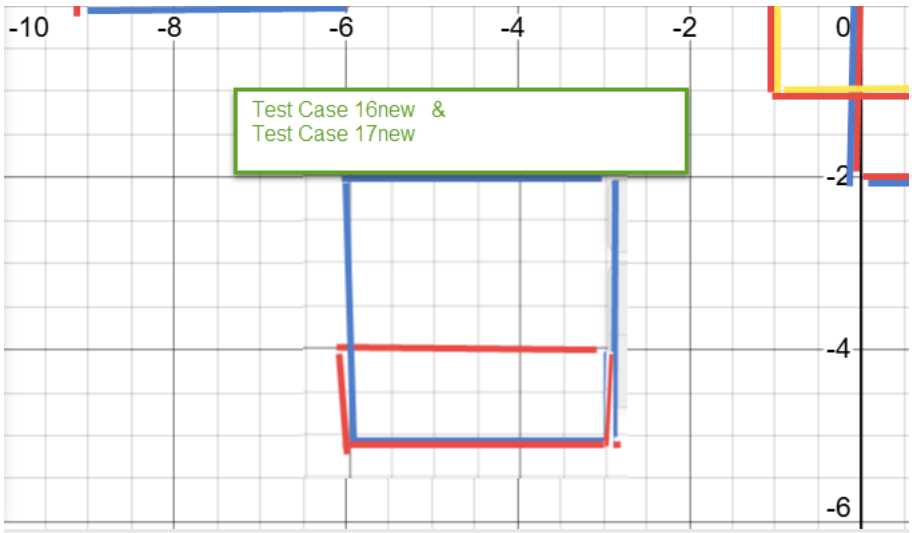
TEST CASE 16 and TEST CASE 17:




```
rect1Topright: 4
rect2TopRight: 6
rect1BottomLeft: 3
rect2BottomLeft: 3
-3
-1
MUST29
**********************************************
AREA: 3
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 3
```

New test exploring in negative quadrant

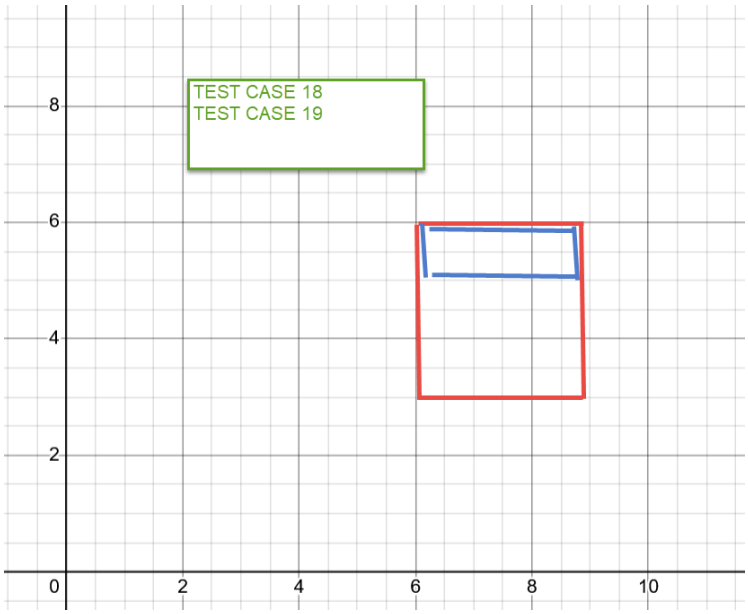
TEST CASE 16new and TEST CASE 17new:



```
//TEST CASE 16new  -
  //System.out.println("The overlapping area is: " + overlappingArea(new int[]{-6, -5}, new int[]{-3, -4},new int[]{-6, -5},  new int[]{-3, -2}));
//

//TEST CASE 17new  -
  System.out.println("The overlapping area is: " + overlappingArea(new int[]{-6, -5},  new int[]{-3, -2},new int[]{-6, -5}, new int[]{-3, -4}));
//
```

```
rect1Topright: -2
rect2TopRight: -4
rect1BottomLeft: -5
rect2BottomLeft: -5
MUST3
3
1
**********************************************
AREA: 3
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 3
```

TEST CASE 18 and TEST CASE 19:



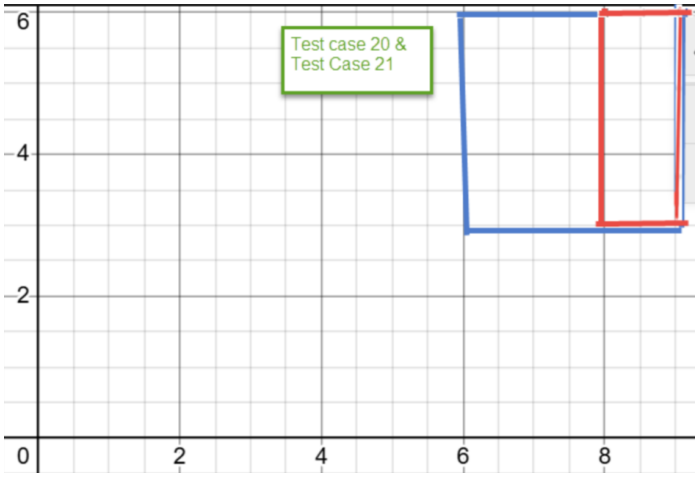

```
//TEST CASE 18
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{6, 3}, new int[]{9, 6},new int[]{6, 5},  new int[]{9, 6}));
```

```
rect1Topright: 6
rect2TopRight: 6
rect1BottomLeft: 3
rect2BottomLeft: 5
MUST26
MUST28
test
The overlapping area is: 3
```

```
//TEST CASE 19, same as above but flipped
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{6, 5},  new int[]{9, 6},new int[]{6, 3}, new int[]{9, 6}));
//
```

```
rect1Topright: 6
rect2TopRight: 6
rect1BottomLeft: 5
rect2BottomLeft: 3
-3
-1
MUST29
MUST23
MUST25
The overlapping area is: 3
```

TEST CASE 20 and TEST CASE 21:




```
//TEST CASE 20
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{6, 3},  new int[]{9, 6},new int[]{8, 3}, new int[]{9, 6}));
//
```

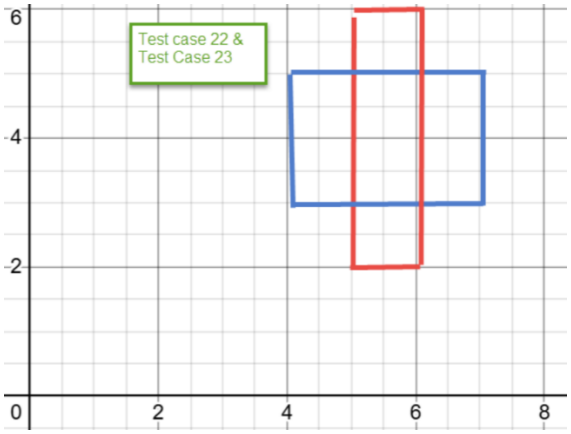
```
rect1Topright: 6
rect2TopRight: 6
rect1BottomLeft: 3
rect2BottomLeft: 3
MUST2
-1
-3
MUST3
-1
-3
**********************************************
AREA: 3
AREA: 3
AREA: 0
AREA: 0
The overlapping area is: 3
```

We can see it has triggered two areas of code... Since both the values are the same, perhaps I can decide on a return value on the first occurrence..

```
//TEST CASE 21, same as above but flipped
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{8, 3}, new int[]{9, 6},new int[]{6, 3},  new int[]{9, 6}));
 //
```

```
rect1Topright: 6
rect2TopRight: 6
rect1BottomLeft: 3
rect2BottomLeft: 3
MUST1
-1
-3
MUST4
-1
-3
**********************************************
AREA: 3
AREA: 3
AREA: 0
AREA: 0
The overlapping area is: 3
```

TEST CASE 22 and TEST CASE 23:




```
//TEST CASE 22
    System.out.println("The overlapping area is: " + overlappingArea(new int[]{5, 2}, new int[]{6, 6},new int[]{4, 3},  new int[]{7, 5}));
    //
```

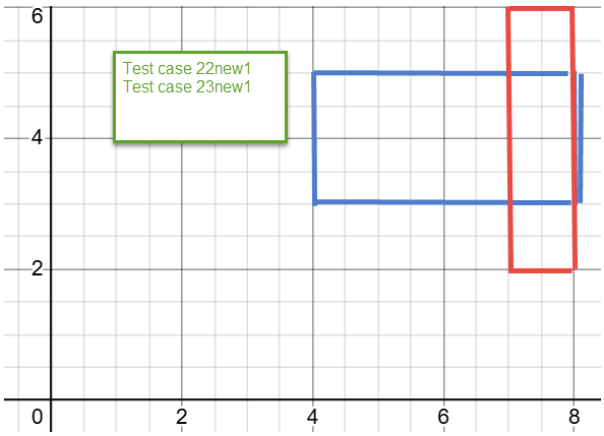
```
rect1Topright: 6
rect2TopRight: 5
rect1BottomLeft: 2
rect2BottomLeft: 3
HERE9
-1
-2
The overlapping area is: 2
```

```
//TEST CASE 23 - flip of the above
System.out.println("The overlapping area is: " + overlappingArea(new int[]{4, 3},  new int[]{7, 5},new int[]{5, 2}, new int[]{6, 6}));
//
```

```
rect1Topright: 5
rect2TopRight: 6
rect1BottomLeft: 3
rect2BottomLeft: 2
HERE9
-1
-2
The overlapping area is: 2
```

I will now experiment slightly and move the red rectangle across.

TEST CASE 22new1 and TEST CASE 23new1:




```
//TEST CASE 22new1
  System.out.println("The overlapping area is: " + overlappingArea(new int[]{7, 2}, new int[]{8, 6},new int[]{4, 3},  new int[]{8, 5}));
  //
```

```
rect1Topright: 6
rect2TopRight: 5
rect1BottomLeft: 2
rect2BottomLeft: 3
HERE2
-1
-3
The overlapping area is: 3
```

THIS IS INCORRECT... I WILL TRY THE FLIP OF THE RECTANGLES

```
 //TEST CASE 23new1 - flip of the above
System.out.println("The overlapping area is: " + overlappingArea(new int[]{7, 2}, new int[]{8, 6},new int[]{7, 2}, new int[]{8, 6}));
 //
```

```
rect1Topright: 5
rect2TopRight: 6
rect1BottomLeft: 3
rect2BottomLeft: 2
HERE1
-1
-3
The overlapping area is: 3
```

THIS IS INCORRECT... I WILL INVESTIGATE BOTH

NOTE, THERE WERE NO ISSUES WHEN RED RECTANGLE RAN THROUGH THE MIDDLE.

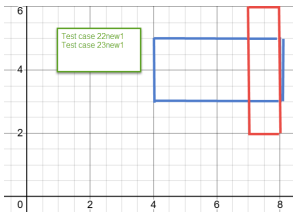
I have identified the following resolution:

```java
//TEST CASE 23new1 - flip of the above
System.out.println("The overlapping area is: " + overlappingArea(new int[]{4, 3},  new int[]{8, 5},new int[]{7, 2}, new int[]{8, 6}));
//
```

```java
        //in this else area, we know that rectangle does not cross the Y axis
        //we know in this area. we know the the blue rectangle is rectangle1.....
        //We also know that following statement is not true....  rect1TopRight[0]>rect2TopRight[0])
        //top right X coordinate of the blue rectangle1 is 8
        //top right X coordinate of the red rectangle 2 is also 8
        //but in else statement, we need to consider if both are the same....
        //in which case the area can be calculated different..

        else
        {
            if (rect1TopRight[0]==rect2TopRight[0])
            {
                //red is rectangle2
                width = Math.abs(0-rect2BottomLeft[0]) - Math.abs(0-rect2TopRight[0]);
                System.out.println(width);
                height = Math.abs(0-rect1TopRight[1]) - Math.abs(0-rect1BottomLeft[1]);
                System.out.println(width);
                System.out.println("MUST51");
            }

            else
            {
            System.out.println("HERE1");
            width = Math.abs(0-rect2BottomLeft[0]) - Math.abs(0-rect1TopRight[0]);
            System.out.println(width);
            height = Math.abs(0-rect2BottomLeft[1]) - Math.abs(0-rect1TopRight[1]);
            System.out.println(height);
            }
```

This change is for test case 23new1

As per usual, I will apply the same in the 22new1 section of the code...



The output is now correct:

```
rect1Topright: 5
rect2TopRight: 6
rect1BottomLeft: 3
rect2BottomLeft: 2
-1
-1
MUST51
The overlapping area is: 2
```

I will now implement the code in other section and it has resolved issue..

```java
//TEST CASE 22new1
  System.out.println("The overlapping area is: " + overlappingArea(new int[]{7, 2}, new int[]{8, 6},new int[]{4, 3},  new int[]{8, 5}));
  //
```

```
rect1Topright: 6
rect2TopRight: 5
rect1BottomLeft: 2
rect2BottomLeft: 3
-1
-1
MUST52
The overlapping area is: 2
```

It would be interesting to devise a case and analyse if the red rectangle cuts through the first part of the other rectangle....

TEST CASE 22new2 and TEST CASE 23new2:


```
//TEST CASE 22new2 - flip of the above
System.out.println("The overlapping area is: " + overlappingArea(new int[]{4, 0},  new int[]{5, 6},new int[]{4, 2}, new int[]{8, 4}));
//
```

```
rect1Topright: 6
rect2TopRight: 4
rect1BottomLeft: 0
rect2BottomLeft: 2
MUST3
-4
-2
**********************************************
AREA: 8
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 8
```

This is clearly wrong.. So it appears where the rectangles edge has coincided with top right or bottom left, its code that requires addressing...

But firstly, I will perform 23new2

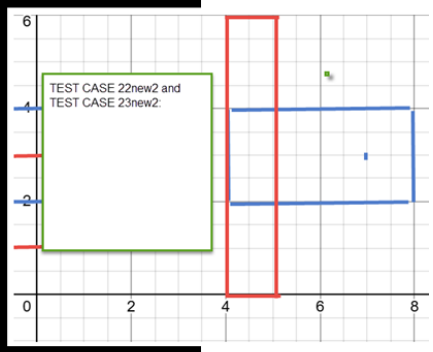
```
//TEST CASE 23new2 - flip of the above
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{4, 2}, new int[]{8, 4},new int[]{4, 0},  new int[]{5, 6}));
  //
```

```
rect1Topright: 4
rect2TopRight: 6
rect1BottomLeft: 2
rect2BottomLeft: 0
MUST1
-4
-2
**********************************************
AREA: 8
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 8
```

This is also wrong, we are expecting 2 overlap

I have remediated as below..

```
                    //we are here because of issues in test case 22new2 and 23new2
                    //at this point we know that rectangle 1 is blue,   rectangle 2 is red

                    //overlappingArea(new int[]{4, 2}, new int[]{8, 4},new int[]{4, 0},  new int[]{5, 6}));

                    //we know the blue has a higher or equal  Y coordinate than red triangle
                    //we also know that blue has a higher or equal  X coordinate than red triangle
                    //but really we are concerned with a different calculation if the x coordinates of the two rectangles are the same
                    if (rect1BottomLeft[0]==rect2BottomLeft[0])
                    {
                        width = Math.abs(0-rect2BottomLeft[0]) - Math.abs(0-rect2TopRight[0]);
                        System.out.println(width);
                        height = Math.abs(0-rect1BottomLeft[1]) - Math.abs(0-rect1TopRight[1]);
                        System.out.println(height);
                        System.out.println("MUST29");
                        store[count]=(Math.abs(width) * Math.abs(height));
                        count++;
                    }
                    else
                    {
                    System.out.println("MUST1");
                    width = Math.abs(0-rect1BottomLeft[0]) - Math.abs(0-rect1TopRight[0]);
                    System.out.println(width);
                    height = Math.abs(0-rect1BottomLeft[1]) - Math.abs(0-rect1TopRight[1]);
                    System.out.println(height);

                    store[count]=(Math.abs(width) * Math.abs(height));
                    count++;
                    }
```




```
//TEST CASE 23new2 - flip of the above
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{4, 2}, new int[]{8, 4},new int[]{4, 0},  new int[]{5, 6}));
  //
```

```
rect1Topright: 4
rect2TopRight: 6
rect1BottomLeft: 2
rect2BottomLeft: 0
-1
-2
MUST29
**********************************************
AREA: 2
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 2
```

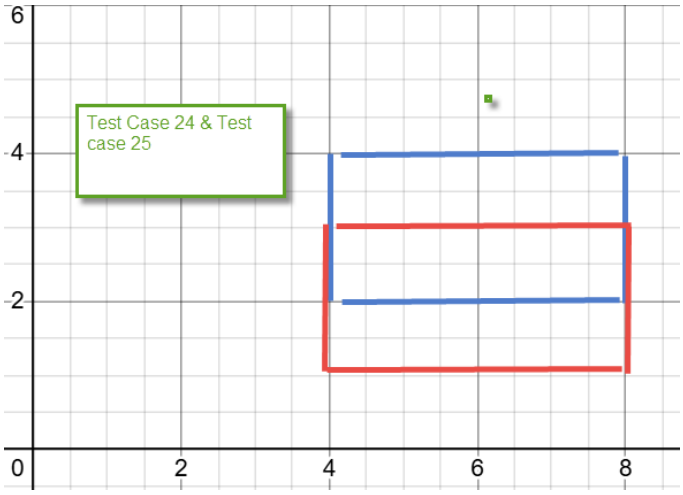
I will now address the other area of code..

```
 //TEST CASE 22new2 - flip of the above
System.out.println("The overlapping area is: " + overlappingArea(new int[]{4, 0},  new int[]{5, 6},new int[]{4, 2}, new int[]{8, 4}));
 //
```

```
rect1Topright: 6
rect2TopRight: 4
rect1BottomLeft: 0
rect2BottomLeft: 2
-1
-2
MUST30
**********************************************
AREA: 2
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 2
```

I will contine with remaining test cases

TEST CASE 24 and TEST CASE 25




```
    //TEST CASE 24
System.out.println("The overlapping area is: " + overlappingArea(new int[]{4, 1},  new int[]{8, 3},new int[]{4, 2}, new int[]{8, 4}));
//
```

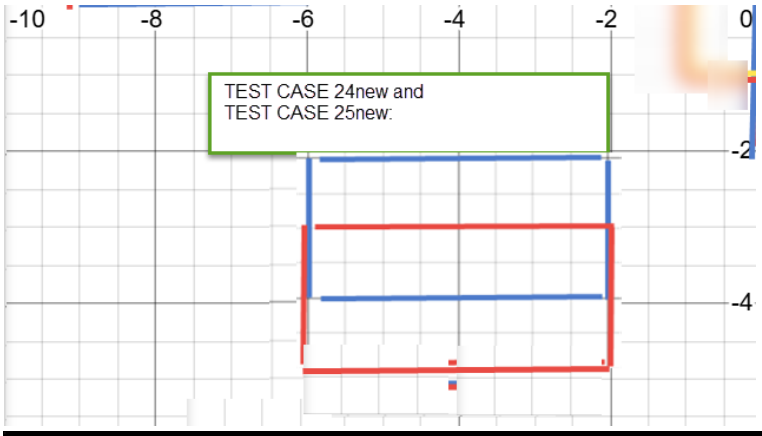
```
rect1Topright: 3
rect2TopRight: 4
rect1BottomLeft: 1
rect2BottomLeft: 2
-4
-1
MUST7
test
The overlapping area is: 4
```

```
    //TEST CASE 25  - flip of the above
System.out.println("The overlapping area is: " + overlappingArea(new int[]{4, 2}, new int[]{8, 4},new int[]{4, 1},  new int[]{8, 3}));
//
```

```
rect1Topright: 4
rect2TopRight: 3
rect1BottomLeft: 2
rect2BottomLeft: 1
-4
-1
MUST8
The overlapping area is: 4
```

I will now move the shapes into negative x and y axis

TEST CASE 24new and TEST CASE 25new:



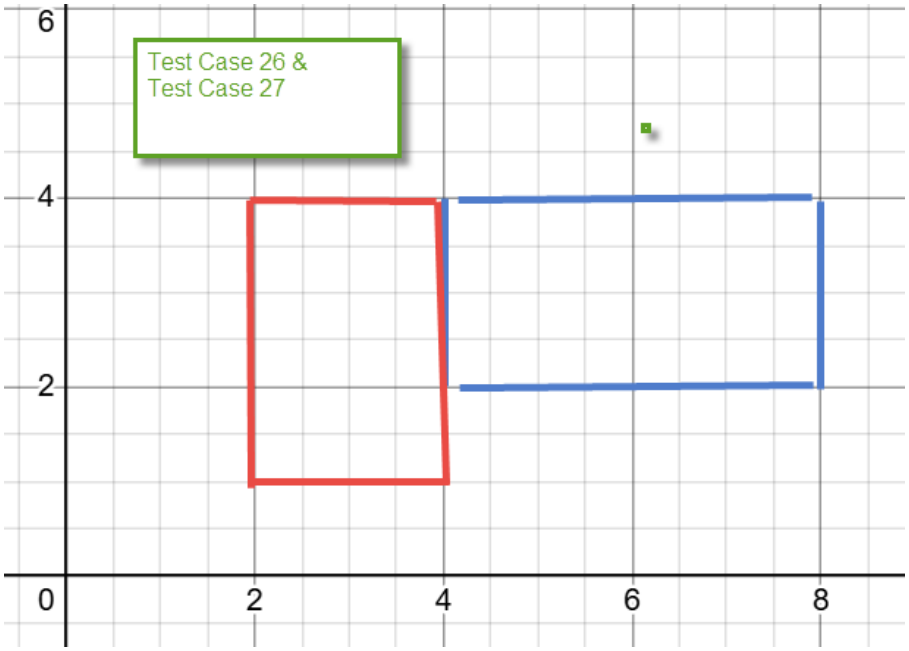

```
 //TEST CASE 24new
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-6, -5}, new int[]{-2, -3},new int[]{-6, -4},  new int[]{-2, -2}));
//
```

```
rect1Topright: -3
rect2TopRight: -2
rect1BottomLeft: -5
rect2BottomLeft: -4
4
1
MUST7
test
The overlapping area is: 4
```

```
    //TEST CASE 25new  - flip of the above
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-6, -4},  new int[]{-2, -2},new int[]{-6, -5}, new int[]{-2, -3}));
//
```

```
rect1Topright: -2
rect2TopRight: -3
rect1BottomLeft: -4
rect2BottomLeft: -5
4
1
MUST8
The overlapping area is: 4
```

TEST CASE 26 and TEST CASE 27:



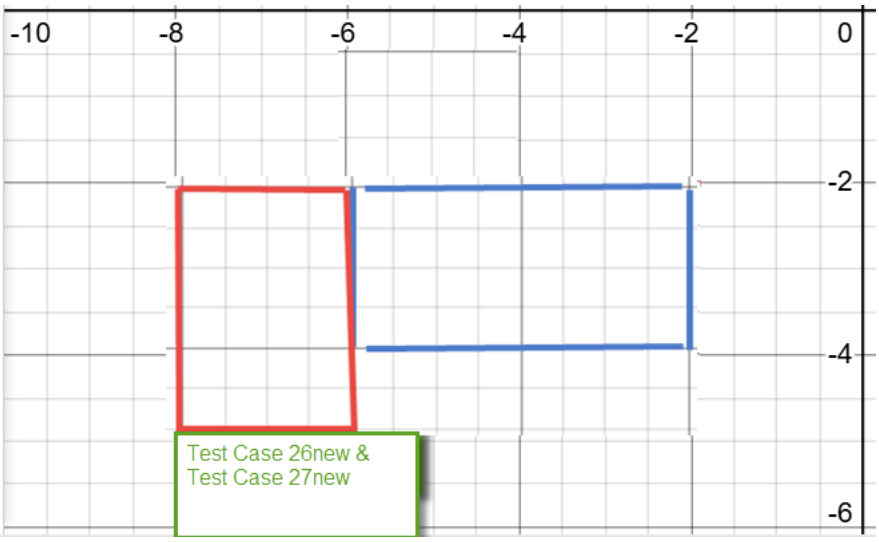

```
 //TEST CASE 26
System.out.println("The overlapping area is: " + overlappingArea(new int[]{2, 1}, new int[]{4, 4},new int[]{4, 2},  new int[]{8, 4}));
//
```

```
NO OVERLAP FOUND
```

```
 //TEST CASE 27 - flip of the above
System.out.println("The overlapping area is: " + overlappingArea(new int[]{4, 2}, new int[]{8, 4},new int[]{2, 1},  new int[]{4, 4}));
//
```

```
NO OVERLAP FOUND


** Process exited - Return Code: 0 **
```

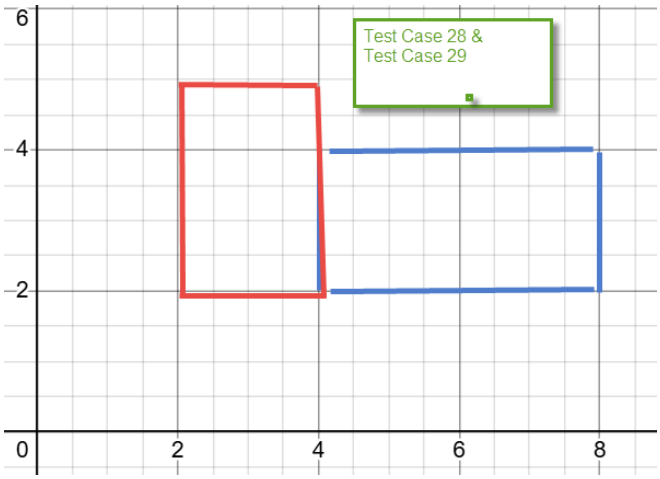
TEST CASE 26new and TEST CASE 27new:




```
//TEST CASE 26new
//System.out.println("The overlapping area is: " + overlappingArea(new int[]{-8, -5}, new int[]{-6, -2},new int[]{-6, -4},  new int[]{-2, -2}));
//

//TEST CASE 27new
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-6, -4},  new int[]{-2, -2},new int[]{-8, -5}, new int[]{-6, -2}));
//
```

```
NO OVERLAP FOUND
```

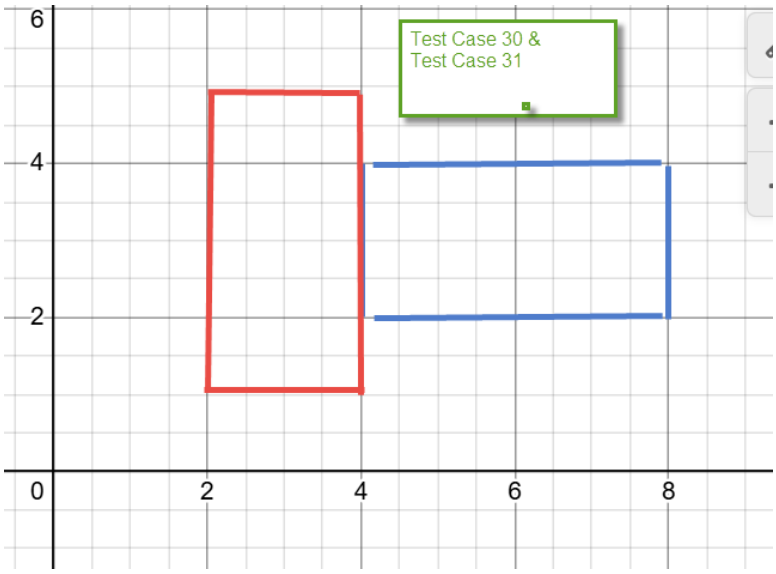
TEST CASE 28 and TEST CASE 29:




```
//TEST CASE 28
//System.out.println("The overlapping area is: " + overlappingArea(new int[]{2, 2}, new int[]{4, 5},new int[]{4, 2},  new int[]{8, 4}));
//

 //TEST CASE 29 - flip of the above
System.out.println("The overlapping area is: " + overlappingArea(new int[]{4, 2},  new int[]{8, 4},new int[]{2, 2}, new int[]{4, 5}));
//
```

```
NO OVERLAP FOUND
```

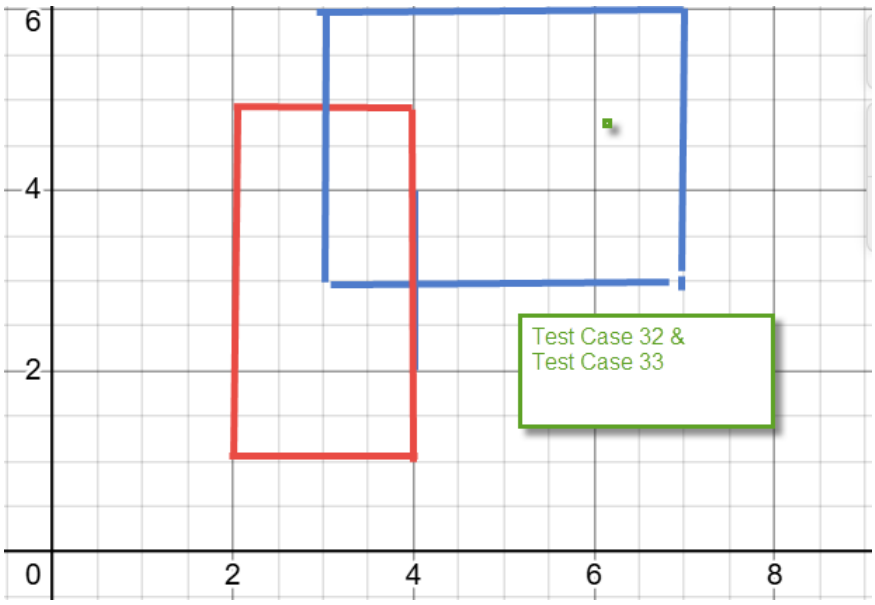
TEST CASE 30 and TEST CASE 31:




```
//TEST CASE 30
//System.out.println("The overlapping area is: " + overlappingArea(new int[]{2, 1},  new int[]{4, 5},new int[]{4, 2}, new int[]{8, 4}));
//

//TEST CASE 31 - flip of the above
System.out.println("The overlapping area is: " + overlappingArea(new int[]{4, 2}, new int[]{8, 4},new int[]{2, 1},  new int[]{4, 5}));
//
```

```
NO OVERLAP FOUND
```

TEST CASE 32 and TEST CASE 33:



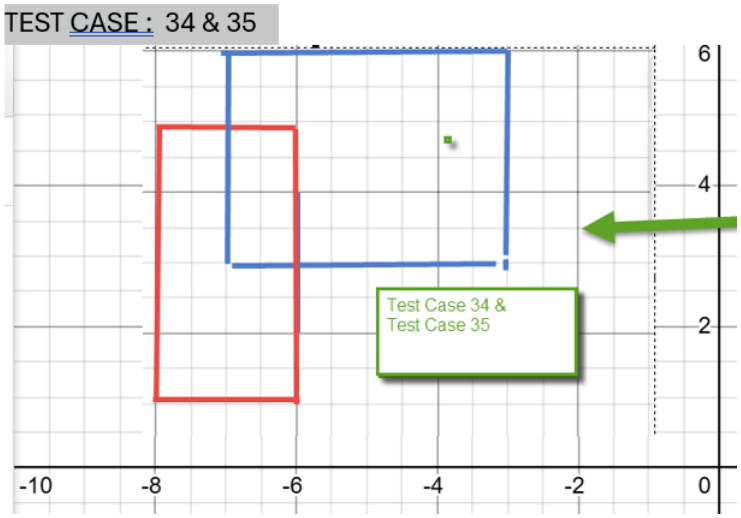

```
//TEST CASE 32
System.out.println("The overlapping area is: " + overlappingArea(new int[]{2, 1}, new int[]{4, 5},new int[]{3, 3},  new int[]{7, 6}));
//
```

```
rect1Topright: 5
rect2TopRight: 6
rect1BottomLeft: 1
rect2BottomLeft: 3
HERE1
-1
-2
The overlapping area is: 2
```

```
//TEST CASE 33 - flip of the above
System.out.println("The overlapping area is: " + overlappingArea(new int[]{3, 3},  new int[]{7, 6},new int[]{2, 1}, new int[]{4, 5}));
//
```

```
rect1Topright: 6
rect2TopRight: 5
rect1BottomLeft: 3
rect2BottomLeft: 1
HERE2
-1
-2
The overlapping area is: 2
```

TEST CASE 34 and TEST CASE 35:




```
//TEST CASE 34
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-8, 1},  new int[]{-6, 5},new int[]{-7, 3}, new int[]{-3, 6}));
//
```

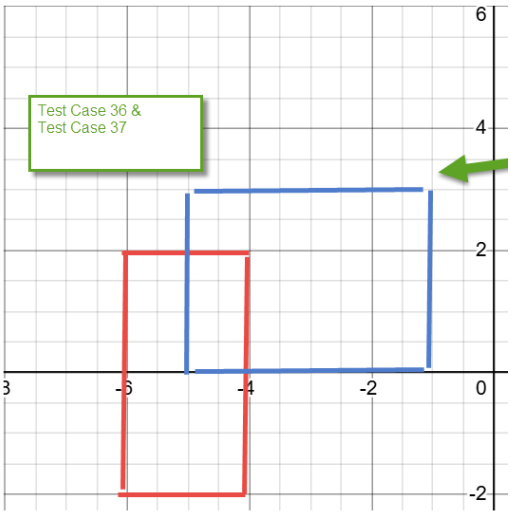
```
rect1Topright: 5
rect2TopRight: 6
rect1BottomLeft: 1
rect2BottomLeft: 3
HERE1
1
-2
The overlapping area is: 2
```

```
//TEST CASE 35 - flip of the above
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-7, 3}, new int[]{-3, 6},new int[]{-8, 1},  new int[]{-6, 5}));
//
```

```
rect1Topright: 6
rect2TopRight: 5
rect1BottomLeft: 3
rect2BottomLeft: 1
HERE2
1
-2
The overlapping area is: 2
```

<u>TEST CASE 36 and TEST CASE 37:</u>

Just to re-iterate this passed earlier in the document also. It is passing through the x-axis and we ascertained in our documentation that challenges will arise when rectangles pass through Y-axis. I will do these tests at end of the existing test cases....




```
//TEST CASE 36
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-6, -2}, new int[]{-4, 2},new int[]{-5, 0},  new int[]{-1, 3}));
//
```

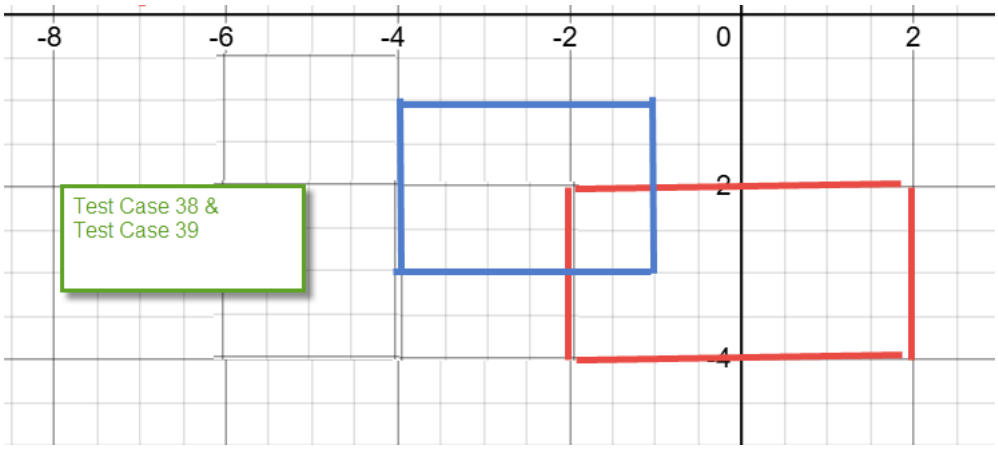
```
rect1Topright: 2
rect2TopRight: 3
rect1BottomLeft: -2
rect2BottomLeft: 0
HERE1
1
-2
The overlapping area is: 2
```

```
//TEST CASE 37 - flip of the above
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-5, 0},  new int[]{-1, 3},new int[]{-6, -2}, new int[]{-4, 2}));
//
```

```
rect1Topright: 3
rect2TopRight: 2
rect1BottomLeft: 0
rect2BottomLeft: -2
HERE2
1
-2
The overlapping area is: 2
```

TEST CASE 38 and TEST CASE 39:

Also note, there overlap does not occur over the y axis



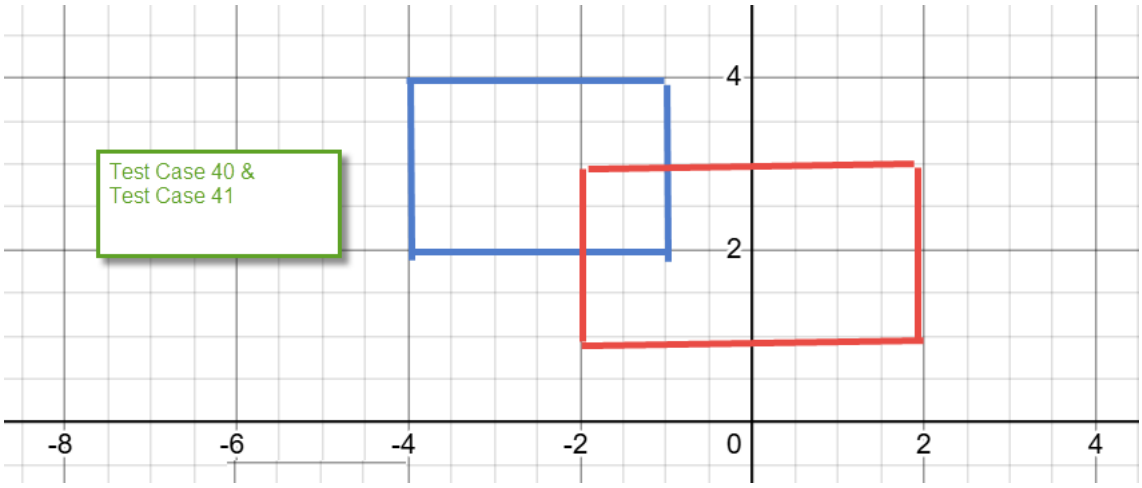

```
//TEST CASE 38
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-4, -3},  new int[]{-1, -1},new int[]{-2, -4}, new int[]{2, 2}));
//
```

```
rect1Topright: -1
rect2TopRight: 2
rect1BottomLeft: -3
rect2BottomLeft: -4
1
1
MUST15
The overlapping area is: 1
```

```
    //TEST CASE 39 - flip of the above
  System.out.println("The overlapping area is: " + overlappingArea(new int[]{-2, -4}, new int[]{2, 2},new int[]{-4, -3},  new int[]{-1, -1}));
  //
```

```
rect1Topright: 2
rect2TopRight: -1
rect1BottomLeft: -4
rect2BottomLeft: -3
1
1
MUST16
The overlapping area is: 1
```

TEST CASE 40 and TEST CASE 41:



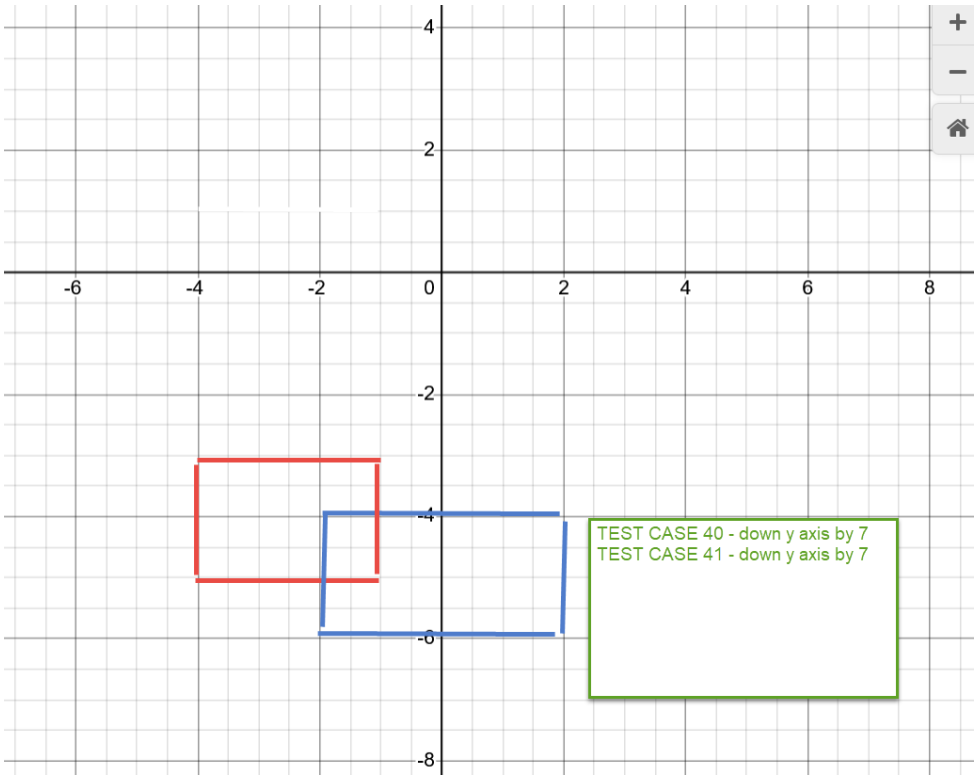

```
 //TEST CASE 40
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-4, 2}, new int[]{-1, 4},new int[]{-2, 1},  new int[]{2, 3}));
```

```
rect1Topright: 2
rect2TopRight: -1
rect1BottomLeft: -4
rect2BottomLeft: -3
1
1
MUST16
The overlapping area is: 1
```

```
 //TEST CASE 41 - flip of the above
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-2, 1},  new int[]{2, 3},new int[]{-4, 2}, new int[]{-1, 4}));
//
```

```
rect1Topright: 3
rect2TopRight: 4
rect1BottomLeft: 1
rect2BottomLeft: 2
1
-1
MUST7
test
The overlapping area is: 1
```

TEST CASE 40 and TEST CASE 41: down Y axis by 7



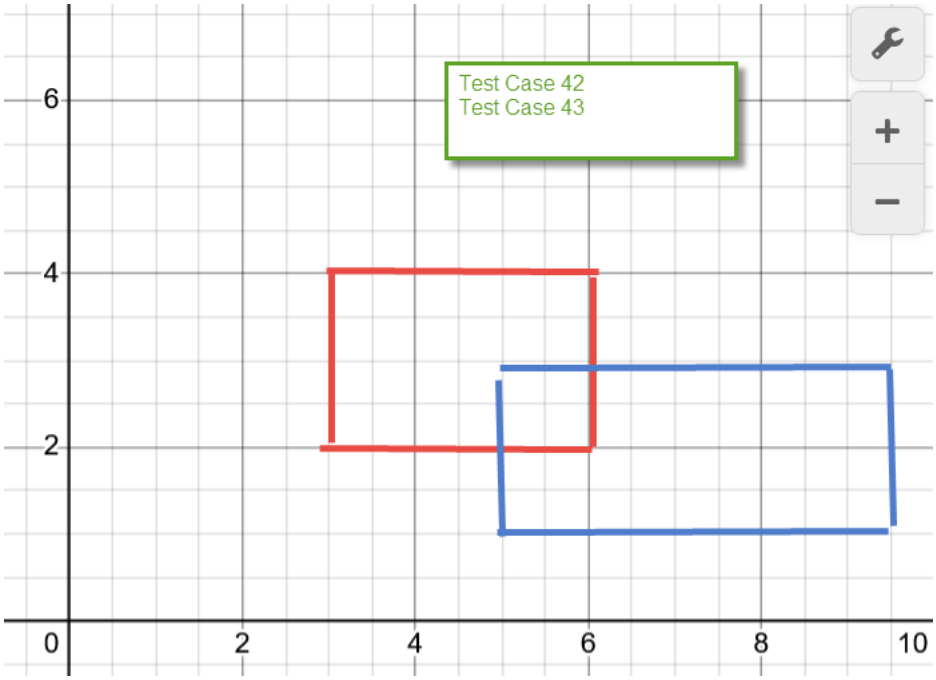

```
//TEST CASE 40 - down y axis by 7
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-4, -5}, new int[]{-1, -3},new int[]{-2, -6},  new int[]{2, -4}));
//
```

```
rect1Topright: -3
rect2TopRight: -4
rect1BottomLeft: -5
rect2BottomLeft: -6
REACH NOW!!!!!
1
1
MUST501
The overlapping area is: 1
```

```
//TEST CASE 41 - down y axis by 7
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-2, -6},  new int[]{2, -4},new int[]{-4, -5}, new int[]{-1, -3}));
//
```

```
rect1Topright: -4
rect2TopRight: -3
rect1BottomLeft: -6
rect2BottomLeft: -5
REACH THIS AREA!!!!!!!!!
1
1
MUST501
test
The overlapping area is: 1
```

TEST CASE 42 and TEST CASE 43:



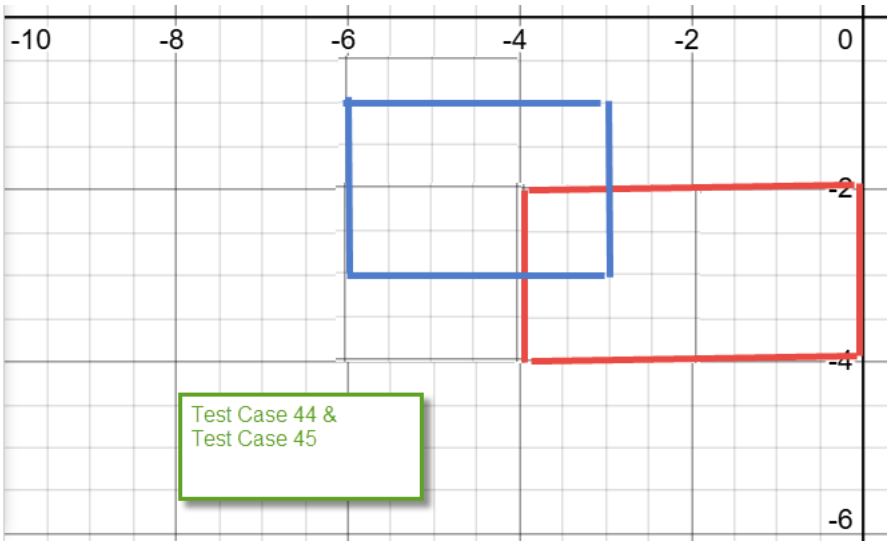

```
//TEST CASE 42
System.out.println("The overlapping area is: " + overlappingArea(new int[]{3, 2},  new int[]{6, 4},new int[]{5, 1}, new int[]{9, 3}));
//
```

```
rect1Topright: 4
rect2TopRight: 3
rect1BottomLeft: 2
rect2BottomLeft: 1
-1
-1
MUST8
The overlapping area is: 1
```

```
//TEST CASE 43 - flip of the above
System.out.println("The overlapping area is: " + overlappingArea(new int[]{5, 1}, new int[]{9, 3},new int[]{3, 2},  new int[]{6, 4}));
//
```

```
rect1Topright: 3
rect2TopRight: 4
rect1BottomLeft: 1
rect2BottomLeft: 2
-1
-1
MUST7
test
The overlapping area is: 1
```

TEST CASE 44 and TEST CASE 45:



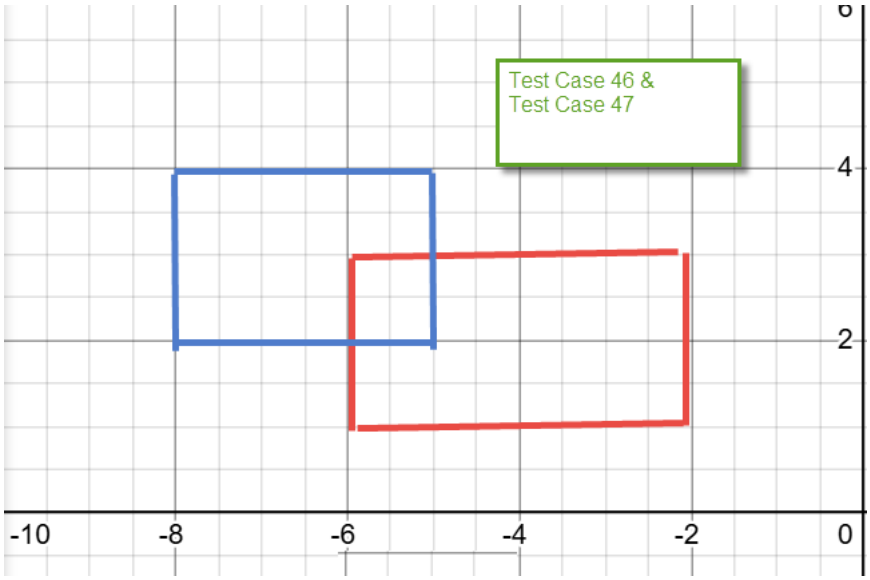

```
//TEST CASE 44
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-4, -4}, new int[]{0, -2},new int[]{-6, -3},  new int[]{-3, 1}));
//
```

```
rect1Topright: -2
rect2TopRight: 1
rect1BottomLeft: -4
rect2BottomLeft: -3
1
1
MUST7
test
The overlapping area is: 1
```

```
//TEST CASE 45 - flip of the above
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-6, -3},  new int[]{-3, 1},new int[]{-4, -4}, new int[]{0, -2}));
//
```

```
rect1Topright: 1
rect2TopRight: -2
rect1BottomLeft: -3
rect2BottomLeft: -4
1
1
MUST8
The overlapping area is: 1
```

TEST CASE 46 and TEST CASE 47:




```
//TEST CASE 46
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-6, 1},  new int[]{-2, 3},new int[]{-8, 2}, new int[]{-5, 4}));
//
```

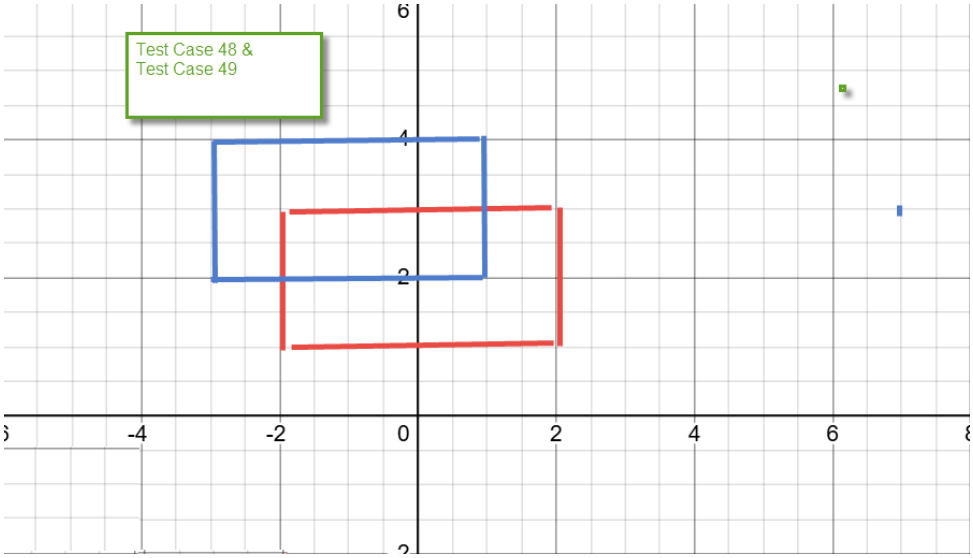
```
rect1Topright: 3
rect2TopRight: 4
rect1BottomLeft: 1
rect2BottomLeft: 2
1
-1
MUST7
test
The overlapping area is: 1
```

```
//TEST CASE 47 - flip of the above
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-8, 2}, new int[]{-5, 4},new int[]{-6, 1},  new int[]{-2, 3}));
//
```

```
rect1Topright: 4
rect2TopRight: 3
rect1BottomLeft: 2
rect2BottomLeft: 1
1
-1
MUST8
The overlapping area is: 1
```

TEST CASE 48 and TEST CASE 49:

This is a typical test case in which I expect issue and code revision:




```
rect1Topright: 3
rect2TopRight: 4
rect1BottomLeft: 1
rect2BottomLeft: 2
1
-3
MUST20
test
The overlapping area is: 3
```

```
//TEST CASE 48
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-2, 1}, new int[]{2, 3},new int[]{-3, 2},  new int[]{1, 4}));
//
```

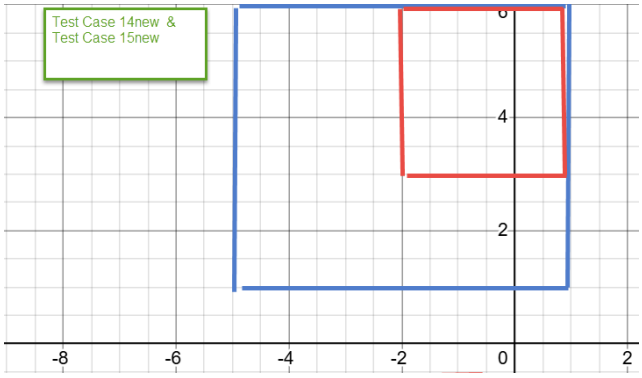
```
rect1Topright: 3
rect2TopRight: 4
rect1BottomLeft: 1
rect2BottomLeft: 2
1
-3
MUST20
test
The overlapping area is: 3
```

```
//TEST CASE 49
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-3, 2},  new int[]{1, 4},new int[]{-2, 1}, new int[]{2, 3}));
//
```

```
rect1Topright: 3
rect2TopRight: 4
rect1BottomLeft: 1
rect2BottomLeft: 2
1
-3
MUST20
test
The overlapping area is: 3
```

TEST CASE 14new and TEST CASE 15new:

My objectives are to look at the area of code surrounding no overlap and correcting it so that it accepts this configuration..

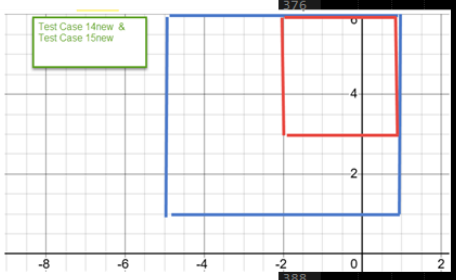



```
//TEST CASE 14new   same as above, but flipped
  System.out.println("The overlapping area is: " + overlappingArea(new int[]{-5, 6},  new int[]{1, 6},new int[]{-2, 3}, new int[]{1, 6}));
//
```

```
//TEST CASE 14new
  System.out.println("The overlapping area is: " + overlappingArea(new int[]{-5, 1},  new int[]{1, 6},new int[]{-2, 3}, new int[]{1, 6}));
//
```

```
rect1Topright: 6
rect2TopRight: 6
rect1BottomLeft: 1
rect2BottomLeft: 3
1
-5
MUST20
test
The overlapping area is: 5


** Process exited - Return Code: 0 **
```

I have created this logic as per usual on both scenarios...
Note I realised that issue persisted even if the red rectangle was moved down the y axis...
Hence I chose not to compare the Y coordinates at the top right location




```
371
372
373
374
375
376
388
389
390
```

```
if ((rect1BottomLeft[0]<0) && (rect1TopRight[0]>0)
    && (rect2BottomLeft[0]<0) && (rect2TopRight[0]>0))
{
    //in cases such as test case 14new and test case 15 new we still have where the
    //x coordinates are same for rect2TopRight for both rectangles...
    //this has to be handled separately in comparison to having two rectangles spanning across the Y axis
    //(test case 48 and 49) where the rectTopRight are not same for both rectangles
    //overlappingArea(new int[]{-5, 1},  new int[]{1, 6},new int[]{-2, 3}, new int[]{1, 6}));
    //blue is rectangle1

     //if they share same x coordinate
      if (rect1TopRight[0]==rect2TopRight[0])
      {
          //we also need to add width as oppose to subtract
          width = Math.abs(0-rect2BottomLeft[0]) + Math.abs(0-rect1TopRight[0]);
          System.out.println(width);
          height = Math.abs(0-rect2BottomLeft[1]) - Math.abs(0-rect1TopRight[1]);
          System.out.println(height);
     System.out.println("MUST31");
      }

     else
     {

    width = Math.abs(0-rect2BottomLeft[0]) - Math.abs(0-rect1TopRight[0]);
    System.out.println(width);
    height = Math.abs(0-rect1BottomLeft[1]) - Math.abs(0-rect2TopRight[1]);
    System.out.println(height);
    System.out.println("MUST20");
     }
```

```
rect1Topright: 6
rect2TopRight: 6
rect1BottomLeft: 1
rect2BottomLeft: 3
3
-3
MUST31
test
The overlapping area is: 9
```

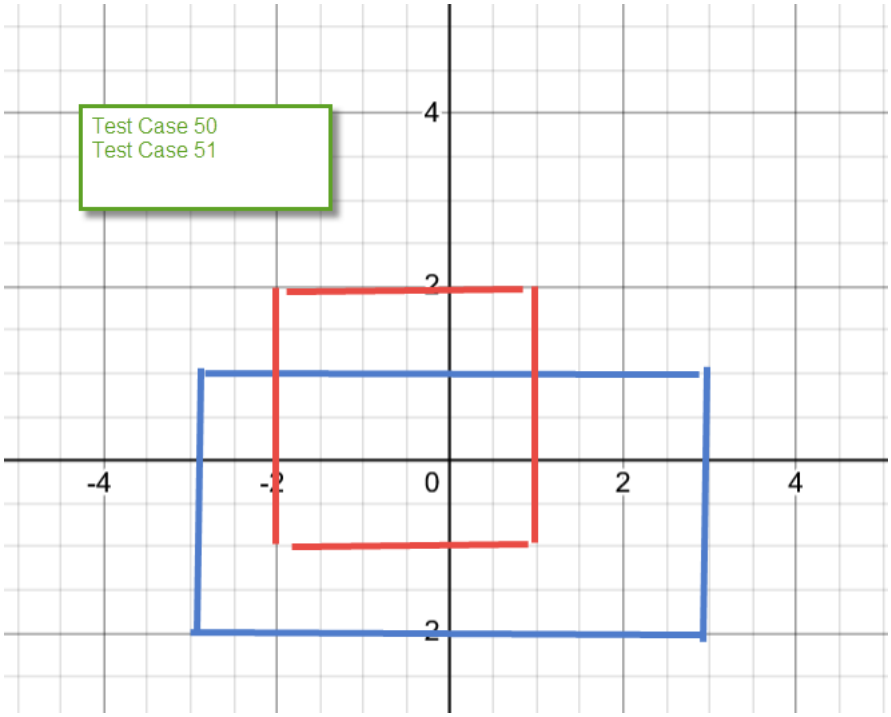
I will now reflect same code in the other scenario

```
//TEST CASE 15new   same as above, but flipped
  System.out.println("The overlapping area is: " + overlappingArea(new int[]{-2, 3}, new int[]{1, 6},new int[]{-5, 1},  new int[]{1, 6}));
//
```

```
rect1Topright: 6
rect2TopRight: 6
rect1BottomLeft: 3
rect2BottomLeft: 1
MUST1
1
-3
3
-3
MUST32
The overlapping area is: 9
```

I have now passed all my test cases, but I am quite certain I need to try some test cases in which there is interaction in all the quadrants... This might be too adventurous, but its ultimately a valid scenario

<u>TEST CASE 50 and TEST CASE 51: FAIL</u>




```
//TEST CASE 50
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-2, -1},  new int[]{1, 2},new int[]{-3, -2}, new int[]{3, 1}));
//
```

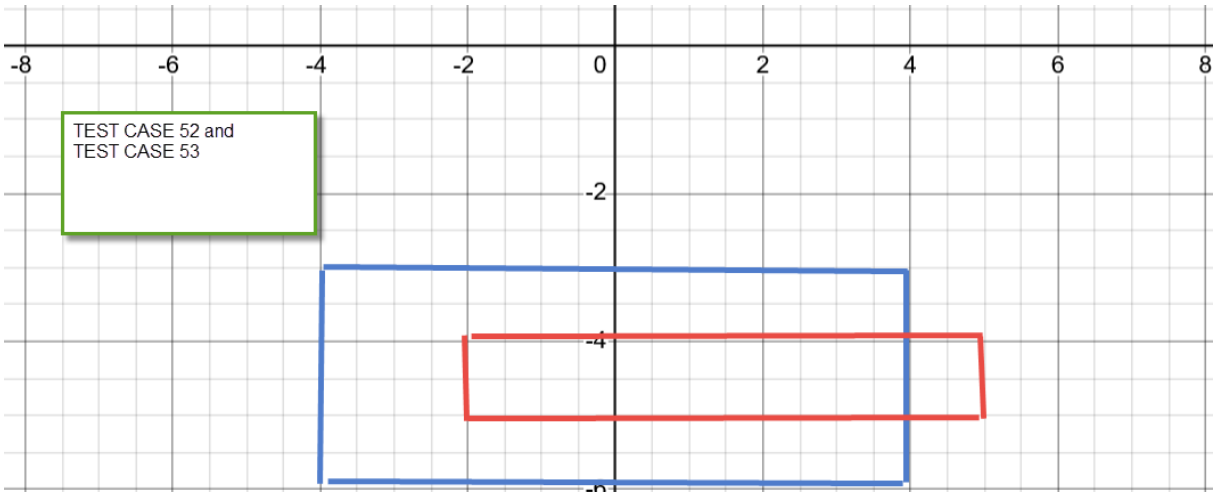
```
rect1Topright: 2
rect2TopRight: 1
rect1BottomLeft: -1
rect2BottomLeft: -2
HERE10
1
1
The overlapping area is: 1
```

```
//TEST CASE 51 - flip of the above
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-3, -2}, new int[]{3, 1},new int[]{-2, -1},  new int[]{1, 2}));
//
```

```
rect1Topright: 1
rect2TopRight: 2
rect1BottomLeft: -2
rect2BottomLeft: -1
HERE9
1
1
The overlapping area is: 1
```

We can see lots work is required to resolve this. Perhaps its best if a simpler example such as overlap on the y axis... (in negative quadrant). Also consider a more simplistic overlap also...

<u>TEST CASE 52 and TEST CASE 53: FAIL</u>

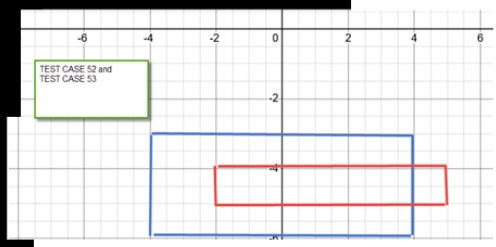



```
//TEST CASE 52
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-4, -6}, new int[]{4, -3},new int[]{-2, -5},  new int[]{5, -4}));
//
```

```
rect1Topright: -3
rect2TopRight: -4
rect1BottomLeft: -6
rect2BottomLeft: -5
MUST3
-3
1
**********************************************
AREA: 3
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 3
```

I can see it has gone into Must3 screen output.

```java
else
{
    //We are here because of test case 52.  Blue is rectangle1
    //overlappingArea(new int[]{-4, -6}, new int[]{4, -3},new int[]{-2, -5},  new int[]{5, -4}))
    //It is important to understand what we know so far.....
    //We know the X coordinates of the bottom left of the rectangles are not the same....
    //but we know that the rectangle botton left coordinate of rectangle2 (red) is greater than blue
    //we also know that Y coordinate (top right) of blue rectangle is greater than equal to red..
    //we are also here because
    //x coordinate of blue rectangle(rect2) (bottom left ) is greater than or equal to red(rect1)
    //OR
    //x coordinate of red rectangle(rect1) (bottom left ) is greater than or equal to blue(rect2)
    //In this area of code, we have not touched upon the shapes spanning the Y axis
    //AND MOST IMPORTANTLY THE OVERLAP SPANNING ACROSS THE Y AXIS

    //I will get the if expression from above and re-use it here with caution (both rectangles spanning across Y axis)

    if ((rect1BottomLeft[0]<0) && (rect1TopRight[0]>0)
   && (rect2BottomLeft[0]<0) && (rect2TopRight[0]>0))
   {
        //note addition of the width
        //rect1 = blue
        width = Math.abs(0-rect1TopRight[0]) + Math.abs(0-rect2BottomLeft[0]);
        System.out.println(width);
        //note since the shape has not crossed height wise across the X axis,
        //we are not changing this....
        //but it will potentially become an issue when I further devise test cases
        //in which overlap is on both side of the X axis
        height = Math.abs(0-rect2BottomLeft[1]) - Math.abs(0-rect2TopRight[1]);
        System.out.println(height);
        System.out.println("MUST33");
         store[count]=(Math.abs(width) * Math.abs(height));
        count++;
   }

        else
        {

         System.out.println("MUST3");
         width = Math.abs(0-rect2BottomLeft[0]) - Math.abs(0-rect2TopRight[0]);
         System.out.println(width);
         height = Math.abs(0-rect2BottomLeft[1]) - Math.abs(0-rect2TopRight[1]);
         System.out.println(height);
         //return (Math.abs(width) * Math.abs(height));
         store[count]=(Math.abs(width) * Math.abs(height));
         count++;
```




```
rect1Topright: -3
rect2TopRight: -4
rect1BottomLeft: -6
rect2BottomLeft: -5
6
1
MUST33
*********************************************
AREA: 6
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 6
```

```
//TEST CASE 53- flip of the above
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-2, -5},  new int[]{5, -4},new int[]{-4, -6}, new int[]{4, -3}));
//
```

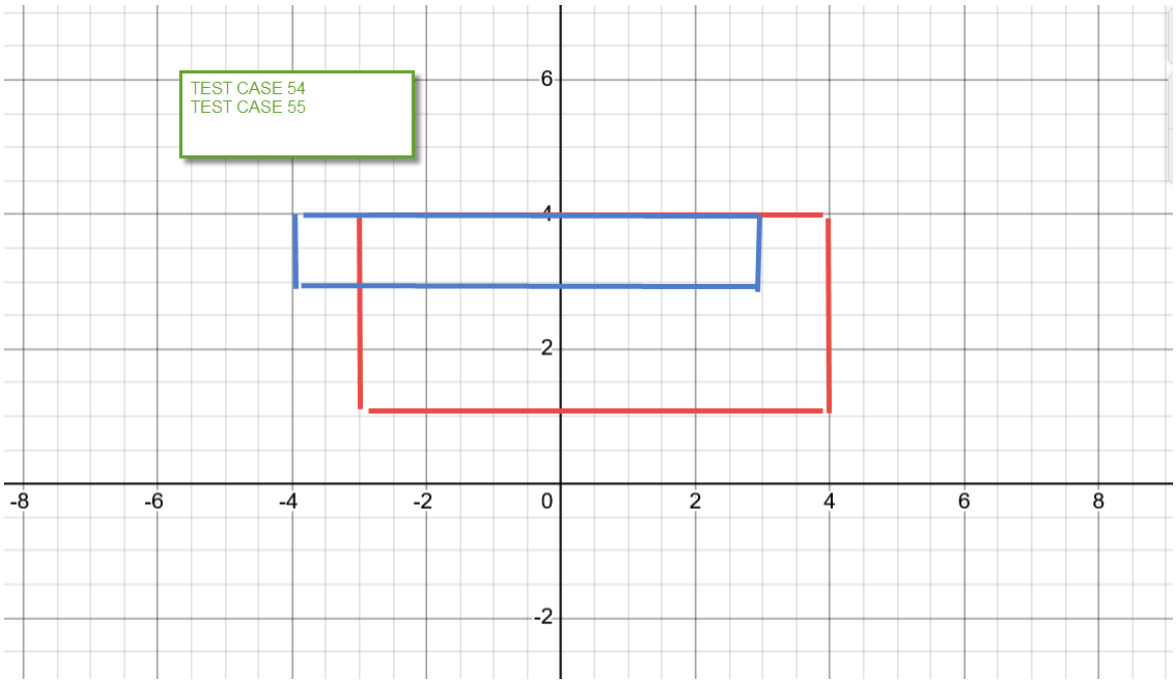
```
rect1Topright: -4
rect2TopRight: -3
rect1BottomLeft: -5
rect2BottomLeft: -6
MUST1
-3
1
**********************************************
AREA: 3
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 3
```

Again, I can see that it has entered Must1 area...
I will remediate this exactly same as above

```
rect1Topright: -4
rect2TopRight: -3
rect1BottomLeft: -5
rect2BottomLeft: -6
6
1
MUST34
**********************************************
AREA: 6
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 6
```

For now, I am still leaving the storage and method call to process minimum overlap. I will clear this as final phase once I have checked all cases again and ascertained it is not making a decision from the storage array..

I am now moving into a test case in which overlap is above the x axis and spanning two quadrants. This is feeling excessively repetitive but unfortunately I could not devise a strategic based test driven approach..




```
//TEST CASE 54
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-3, 1},  new int[]{4, 4},new int[]{-4, 3}, new int[]{3, 4}));
//
```

```
rect1Topright: 4
rect2TopRight: 4
rect1BottomLeft: 1
rect2BottomLeft: 3
0
-3
MUST20
test
The overlapping area is: 0
```

```
//TEST CASE 55
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-4, 3}, new int[]{3, 4}, new int[]{-3, 1},  new int[]{4, 4}));
//
```

```
rect1Topright: 4
rect2TopRight: 4
rect1BottomLeft: 3
rect2BottomLeft: 1
MUST2
-1
-3
0
-3
MUST21
The overlapping area is: 0
```

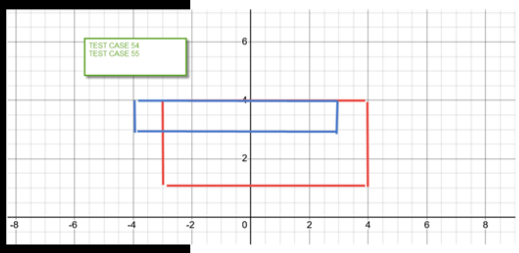
We can see both dimensions are incorrect. Based on understanding of the coding, I envisage it will require segregation of if / else best on circumstances..

Remediating this issue:

```
//TEST CASE 54
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-3, 1},  new int[]{4, 4},new int[]{-4, 3}, new int[]{3, 4}));
//
```

```
rect1Topright: 4
rect2TopRight: 4
rect1BottomLeft: 1
rect2BottomLeft: 3
0
-3
MUST20
test
The overlapping area is: 0
```

```
//we are here due to test cases 54 and 55
//rect1 = red
//we know that top right do not share same x coordinates for the rectangles
//we know both rectangles span across Y axis
//We know that Y coordinate of blue bottom left is greater than red bottom left
//we know that it is not a shape within a shape
//But it appears there needs to be reference to top right of both rectangles being on the same
//Y coordinate
//So this naturally leads me towards issue that might arise if the botom left of the rectangles share
//same X coordinate (this will be potentially test cases 56 and 57)

if (rect1TopRight[1]==rect2TopRight[1])
{
    //again we need to add due to crossing the Y axis
    width = Math.abs(0-rect1BottomLeft[0]) + Math.abs(0-rect2TopRight[0]);
System.out.println(width);
 height = Math.abs(0-rect2BottomLeft[1]) - Math.abs(0-rect2TopRight[1]);
 System.out.println(height);
 System.out.println("MUST35");
}
else
{

width = Math.abs(0-rect2BottomLeft[0]) - Math.abs(0-rect1TopRight[0]);
System.out.println(width);
height = Math.abs(0-rect1BottomLeft[1]) - Math.abs(0-rect2TopRight[1]);
System.out.println(height);
System.out.println("MUST20");
```




```
rect1Topright: 4
rect2TopRight: 4
rect1BottomLeft: 1
rect2BottomLeft: 3
6
-1
MUST35
test
The overlapping area is: 6
```

I will now perform same remediation in other section of code (“MUST21”)

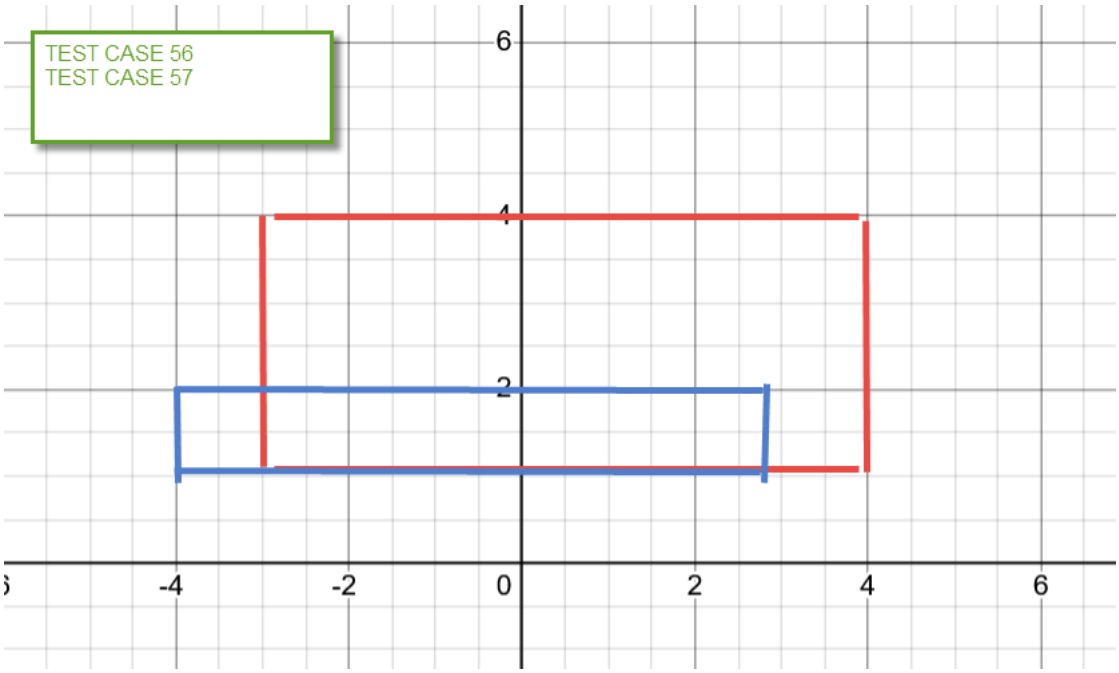
```
//TEST CASE 55
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-4, 3}, new int[]{3, 4}, new int[]{-3, 1},  new int[]{4, 4}));
//
```

```
rect1Topright: 4
rect2TopRight: 4
rect1BottomLeft: 3
rect2BottomLeft: 1
MUST2
-1
-3
6
-1
MUST36
The overlapping area is: 6
```

As discussed in my code comments, I will move the blue block so that y coordinates coincide with bottom left...

I will also then move the blue block so that the corners align exactly.. And I will devise several test cases in advance to try any scenario suitable.....

NOTE I am still following the principle of spanning across the Y axis



We can see both outputs are incorrect, I will trace the code as per usual

```
//TEST CASE 56
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-4, 1}, new int[]{3, 2}, new int[]{-3, 1},  new int[]{4, 4}));
//
```

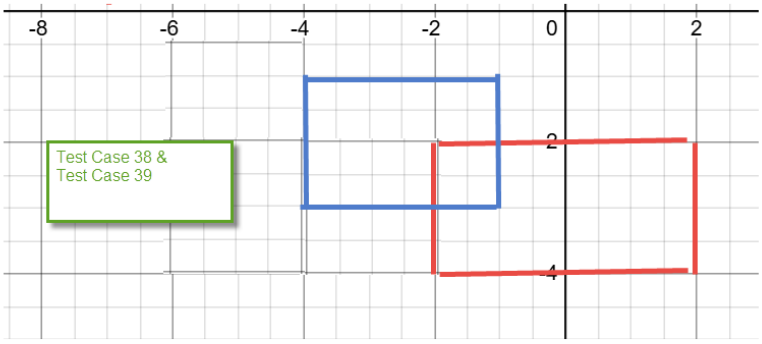
```
rect1Topright: 2
rect2TopRight: 4
rect1BottomLeft: 1
rect2BottomLeft: 1
0
0
MUST15
The overlapping area is: 0
```

```java
//TEST CASE 57
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-3, 1},  new int[]{4, 4},new int[]{-4, 1}, new int[]{3, 2}));
//
```

```
rect1Topright: 4
rect2TopRight: 2
rect1BottomLeft: 1
rect2BottomLeft: 1
0
0
MUST16
The overlapping area is: 0
```

TEST CASE 38 and TEST CASE 39:

Also note, there overlap does not occur over the y axis



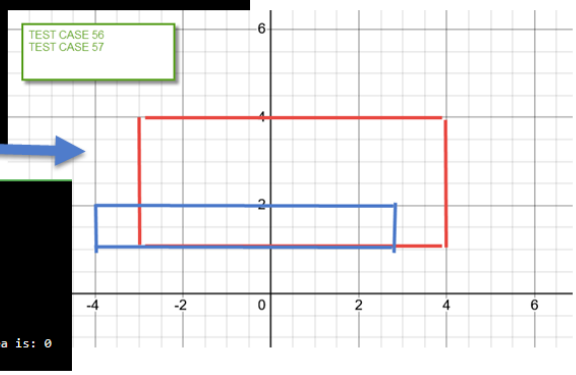
We can see in my existing notes, I made reference to Test case 38 and 39... It went and passed this section of code since the red rectangle had X coordinates in the negative and positive

```java
    }

    else
    {
        //This will be new code here to ascertain if a rectangle appears on both sides of the Y axis.
        //Based on the above statement, we know that rectangle1 top right coordinate is less than rect2 top right
        //we will know check if rect2 runs on both sides of the Y axis
        //At moment, we do not know implications if both rectangles cross the Y axis.. But it is ignored at this moment in time
        //This is in reference to test case 38 and 39
        if (rect2BottomLeft[0]<0 && rect2TopRight[0]>0)
        {
            width = Math.abs(0-rect2BottomLeft[0]) - Math.abs(0-rect1TopRight[0]);
            System.out.println(width);
            height = Math.abs(0-rect2TopRight[1]) - Math.abs(0-rect1BottomLeft[1]);
            System.out.println(width);
             System.out.println("MUST15");
        }
```

The fact that this configuration entered the same area of code, it clearly suggests that this logical block is unrelated to both shapes crossing the Y axis (given that Test case 38 and 39 rectangles did not cross the axis)..

I will verify all the nested loops to be 100% sure first before amending code...

```
rect1Topright: 2
rect2TopRight: 4
rect1BottomLeft: 1
rect2BottomLeft: 1
0
0
MUST15
The overlapping area is: 0
```



I implemented following change:

```java
if (rect2BottomLeft[0]<0 && rect2TopRight[0]>0)
{
    //This is in regarding test case 56 and 57
    //it does not appear that any form of reference has been made above in relation to both shapes spanning Y axis
    //the only areas of outer if loops include:

     //if(rect1BottomLeft[0]<rect2BottomLeft[0])   //X cordinate larger on rectangle 2 (bottom left)
     //if (rect1TopRight[1]<rect2TopRight[1])     //Y cordinate larger on rectangle 2 (top right)
    //rect2 = red
    //rect1=blue

    //X coordinate on top right of rectangle2 (red) is larger than blue
    //We also know that red top right is larger than blue top right

    //this now tackles both rectangles span across the Y axis (test case 56 and 57)
    if ((rect1BottomLeft[0]<0) && (rect1TopRight[0]>0)
        && (rect2BottomLeft[0]<0) && (rect2TopRight[0]>0))
    {
        //note addition
        width = Math.abs(0-rect2BottomLeft[0]) + Math.abs(0-rect1TopRight[0]);
        System.out.println(width);
        height = Math.abs(0-rect1TopRight[1]) - Math.abs(0-rect2BottomLeft[1]);
        System.out.println(height);

    }
  else
  {
   width = Math.abs(0-rect2BottomLeft[0]) - Math.abs(0-rect1TopRight[0]);
   System.out.println(width);
   height = Math.abs(0-rect2TopRight[1]) - Math.abs(0-rect1BottomLeft[1]);
   System.out.println(width);
    System.out.println("MUST15");
  }
```

I do not believe I need to concern myself that both rectangles have bottom left on the same Y coordinate...
If I ever find I have issues in future, it it a logical expression which I need to make a bit more stringent

```java
 //TEST CASE 56
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-4, 1}, new int[]{3, 2}, new int[]{-3, 1},  new int[]{4, 4}));
//
```

```
rect1Topright: 2
rect2TopRight: 4
rect1BottomLeft: 1
rect2BottomLeft: 1
6
1
MUST37
The overlapping area is: 6
```

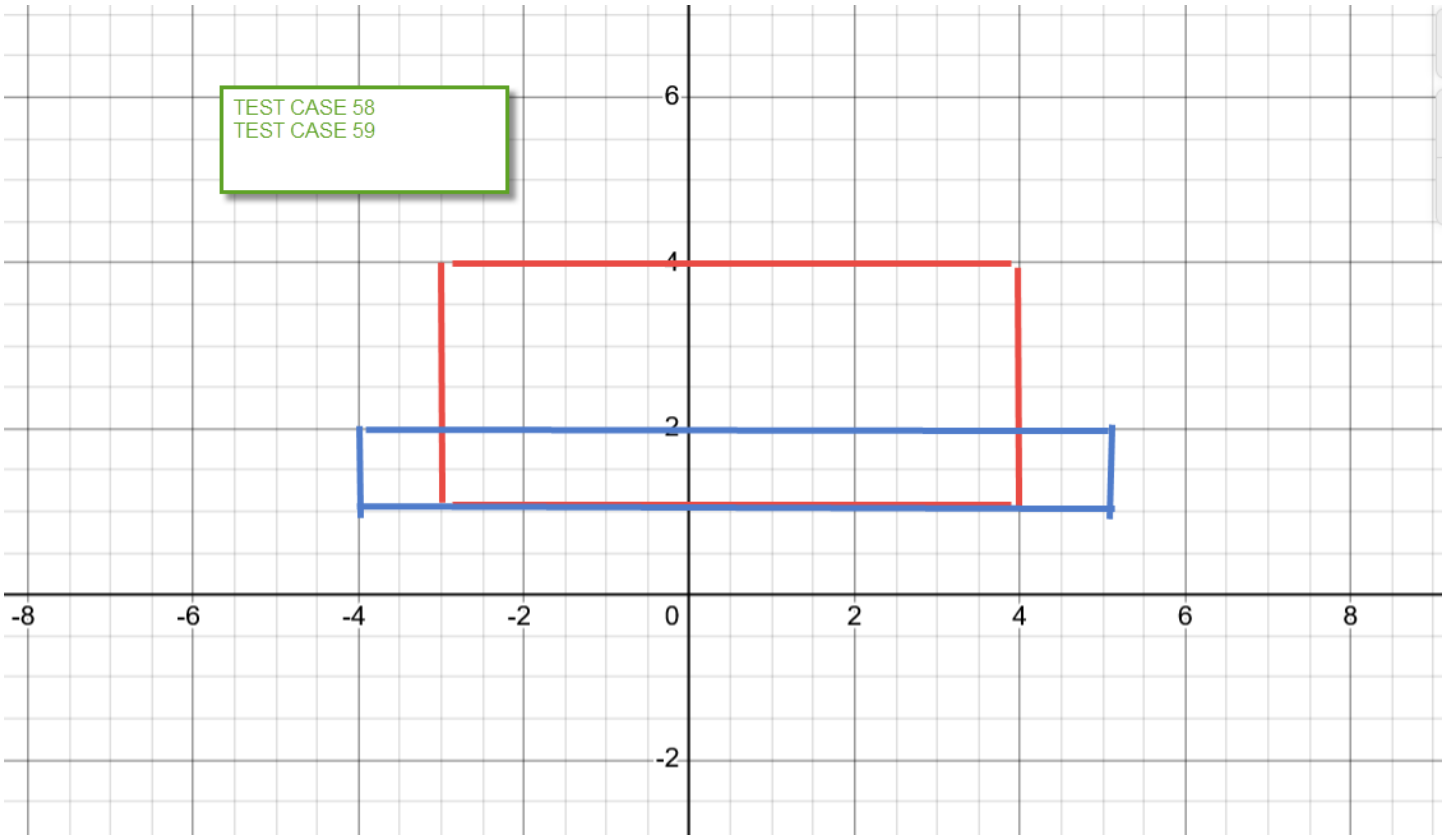
I will now amend the other rectangle scenario (MUST15 area)

```java
//TEST CASE 57
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-3, 1},  new int[]{4, 4},new int[]{-4, 1}, new int[]{3, 2}));
//
```

```
rect1Topright: 4
rect2TopRight: 2
rect1BottomLeft: 1
rect2BottomLeft: 1
6
1
MUST38
The overlapping area is: 6
```

TEST CASE 58 AND TEST CASE 59:

Both are incorrect




```
//TEST CASE 58
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-4, 1},  new int[]{5, 2},new int[]{-3, 1}, new int[]{4, 4}));
//
```

```
rect1Topright: 2
rect2TopRight: 4
rect1BottomLeft: 1
rect2BottomLeft: 1
HERE9
-1
-1
The overlapping area is: 1
```

```
//TEST CASE 59
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-3, 1}, new int[]{4, 4},new int[]{-4, 1},  new int[]{5, 2}));
//
```

```
rect1Topright: 4
rect2TopRight: 2
rect1BottomLeft: 1
rect2BottomLeft: 1
HERE10
-1
-1
The overlapping area is: 1
```

I have made similar code changes to above test cases, in which it required provision for spanning the Y axis both shapes

```
//TEST CASE 58
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-4, 1},  new int[]{5, 2},new int[]{-3, 1}, new int[]{4, 4}));
//
```

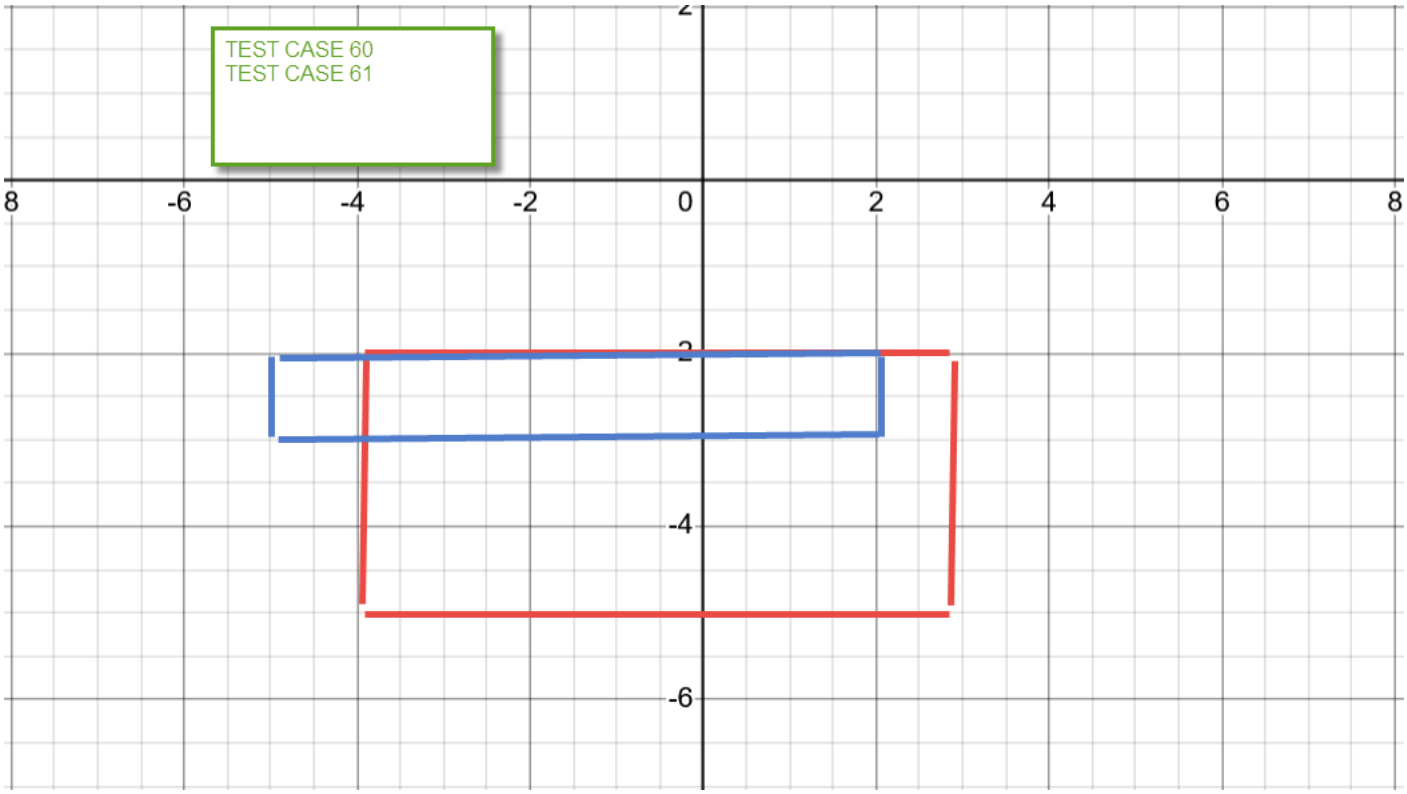
```
rect1Topright: 2
rect2TopRight: 4
rect1BottomLeft: 1
rect2BottomLeft: 1
7
-1
MUST39
The overlapping area is: 7
```

I will implement same code in Here10 section

```
rect1Topright: 4
rect2TopRight: 2
rect1BottomLeft: 1
rect2BottomLeft: 1
7
-1
MUST40
The overlapping area is: 7
```

<u>TEST CASE 60 AND TEST CASE 61:</u>

It appears the above resolutions have taken care of this test case



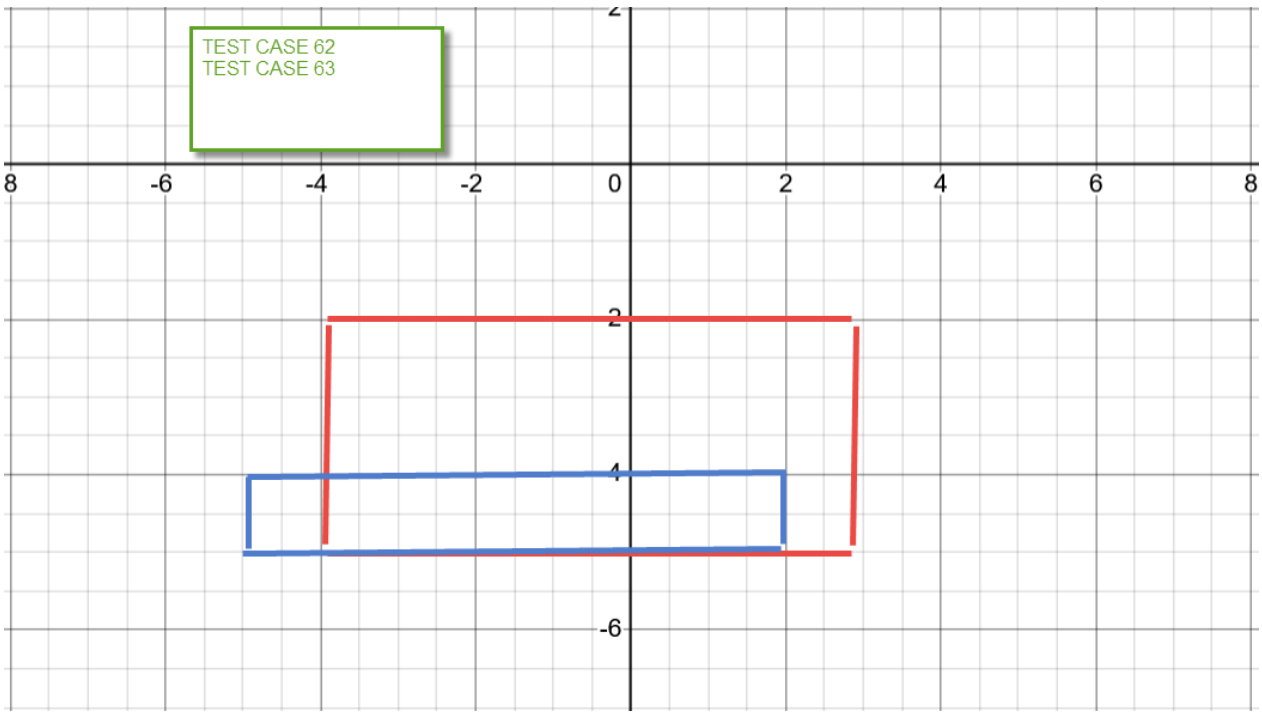
```
//TEST CASE 60
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-5, -3}, new int[]{2, -2},new int[]{-4, -5},  new int[]{3, -2}));
//
```

```
rect1Topright: -2
rect2TopRight: -2
rect1BottomLeft: -3
rect2BottomLeft: -5
MUST2
1
3
6
1
MUST36
The overlapping area is: 6
```

```
//TEST CASE 61
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-4, -5},  new int[]{3, -2},new int[]{-5, -3}, new int[]{2, -2}));
//
```

```
rect1Topright: -2
rect2TopRight: -2
rect1BottomLeft: -5
rect2BottomLeft: -3
6
1
MUST35
test
The overlapping area is: 6
```

TEST CASE 62 AND TEST CASE 63:



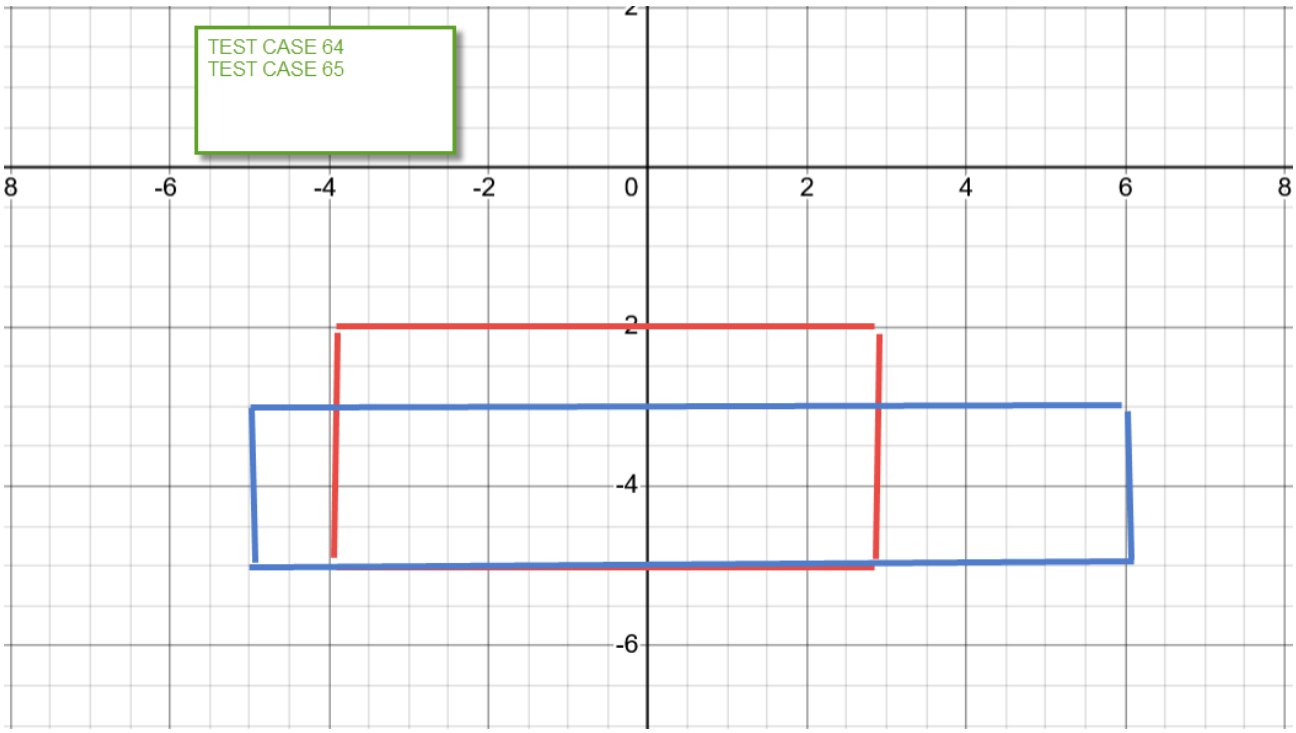
```
//TEST CASE 62
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-5, -5},  new int[]{2, -4},new int[]{-4, -5}, new int[]{3, -2}));
//
```

```
rect1Topright: -4
rect2TopRight: -2
rect1BottomLeft: -5
rect2BottomLeft: -5
6
-1
MUST37
The overlapping area is: 6
```

```
//TEST CASE 63
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-4, -5}, new int[]{3, -2},new int[]{-5, -5},  new int[]{2, -4}));
//
```

```
rect1Topright: -2
rect2TopRight: -4
rect1BottomLeft: -5
rect2BottomLeft: -5
6
-1
MUST38
The overlapping area is: 6
```

TEST CASE 64 AND TEST CASE 65:



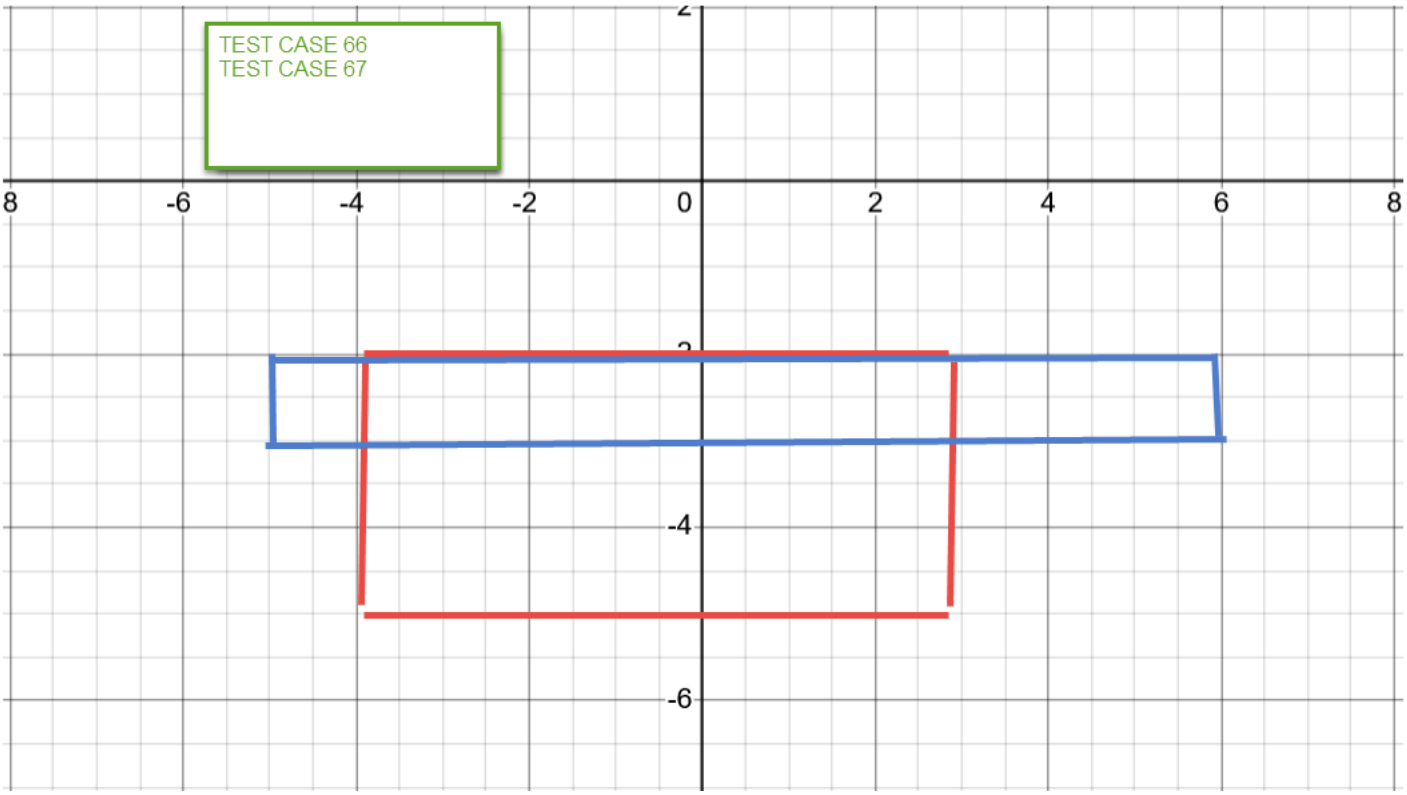

```
//TEST CASE 64
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-5, -5}, new int[]{6, -3},new int[]{-4, -5},  new int[]{3, -2}));
//
```

```
rect1Topright: -3
rect2TopRight: -2
rect1BottomLeft: -5
rect2BottomLeft: -5
7
2
MUST39
The overlapping area is: 14
```

```
//TEST CASE 65
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-4, -5},  new int[]{3, -2},new int[]{-5, -5}, new int[]{6, -3}));
```

```
rect1Topright: -2
rect2TopRight: -3
rect1BottomLeft: -5
rect2BottomLeft: -5
7
2
MUST40
The overlapping area is: 14
```

TEST CASE 66 AND TEST CASE 67:



```java
//TEST CASE 66
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-5, -3},  new int[]{6, -2},new int[]{-4, -5}, new int[]{3, -2}));
```

```
rect1Topright: -2
rect2TopRight: -2
rect1BottomLeft: -3
rect2BottomLeft: -5
MUST2
1
3
10
1
MUST36
The overlapping area is: 10
```

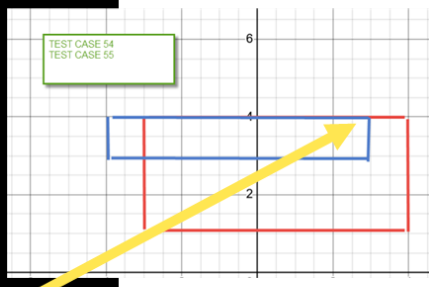
```java
//TEST CASE 67
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-4, -5}, new int[]{3, -2},new int[]{-5, -3},  new int[]{6, -2}));
```

```
rect1Topright: -2
rect2TopRight: -2
rect1BottomLeft: -5
rect2BottomLeft: -3
10
1
MUST35
test
The overlapping area is: 10
```

I found it odd that test case 66 entered two areas of code (MUST 2 and MUST36) whereas test case 67 entered one area (MUST35 and NOT MUST4). THE ONLY OTHER OPTION IS TO ROLL BACK THE CODE

This is practically impossible to remediate unless I compare code by code exactly in both sections...
I expected symmetry in both codes... Irrespective of this, both outcomes are also wrong..
So I will try to troubleshoot
In hindsight, I should have saved my code after each test case.....
I will still adhere to same code for both situations..

```java
//Y coordinate
//So this naturally leads me towards issue that might arise if the botom left of the rectangles share
//same X coordinate (this will be potentially test cases 56 and 57)

if (rect2TopRight[1]==rect1TopRight[1])
{
    //again we need to add due to crossing the Y axis

    //in this area due to test case 66 and 67..
    //this is already in an area of code that considers spanning Y axis both rectangles.
    //so what makes this scenario different to Test case 55 which passes through this area
    //we know they both have the same Y coordinate for the top right of the rectangle

    width = Math.abs(0-rect2BottomLeft[0]) + Math.abs(0-rect1TopRight[0]);
System.out.println(width);
 height = Math.abs(0-rect1BottomLeft[1]) - Math.abs(0-rect1TopRight[1]);
 System.out.println(height);
 System.out.println("MUST36");
}

else
{
```



Perhaps we need to use the following as part of the decision making..

We can conclude that if the top right if the top right of the rectangle exceeds the other, we then use the dimension of the red rectangle as the width

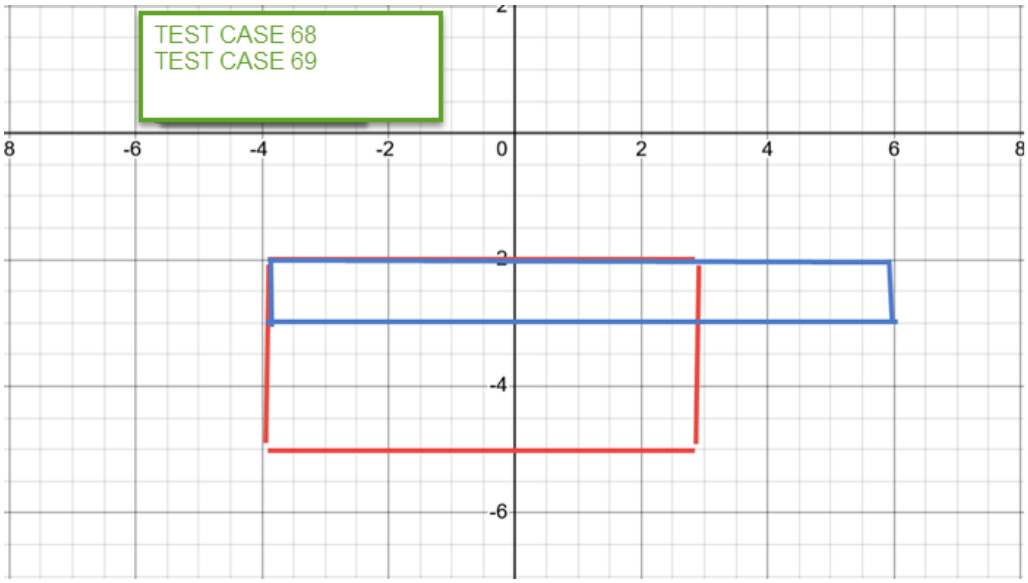
```java
//TEST CASE 66
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-5, -3},  new int[]{6, -2},new int[]{-4, -5}, new int[]{3, -2}));
//
```

```
rect1Topright: -2
rect2TopRight: -2
rect1BottomLeft: -3
rect2BottomLeft: -5
MUST2
1
3
7
1
MUST41
The overlapping area is: 7
```

```java
//TEST CASE 67
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-4, -5}, new int[]{3, -2},new int[]{-5, -3},  new int[]{6, -2}));
//
```

```
rect1Topright: -2
rect2TopRight: -2
rect1BottomLeft: -5
rect2BottomLeft: -3
7
1
MUST42
test
The overlapping area is: 7
```

TEST CASE 68 AND TEST CASE 69:




```
//TEST CASE 68
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-4, -3}, new int[]{6, -2},new int[]{-4, -5},  new int[]{3, -2}));
//
```

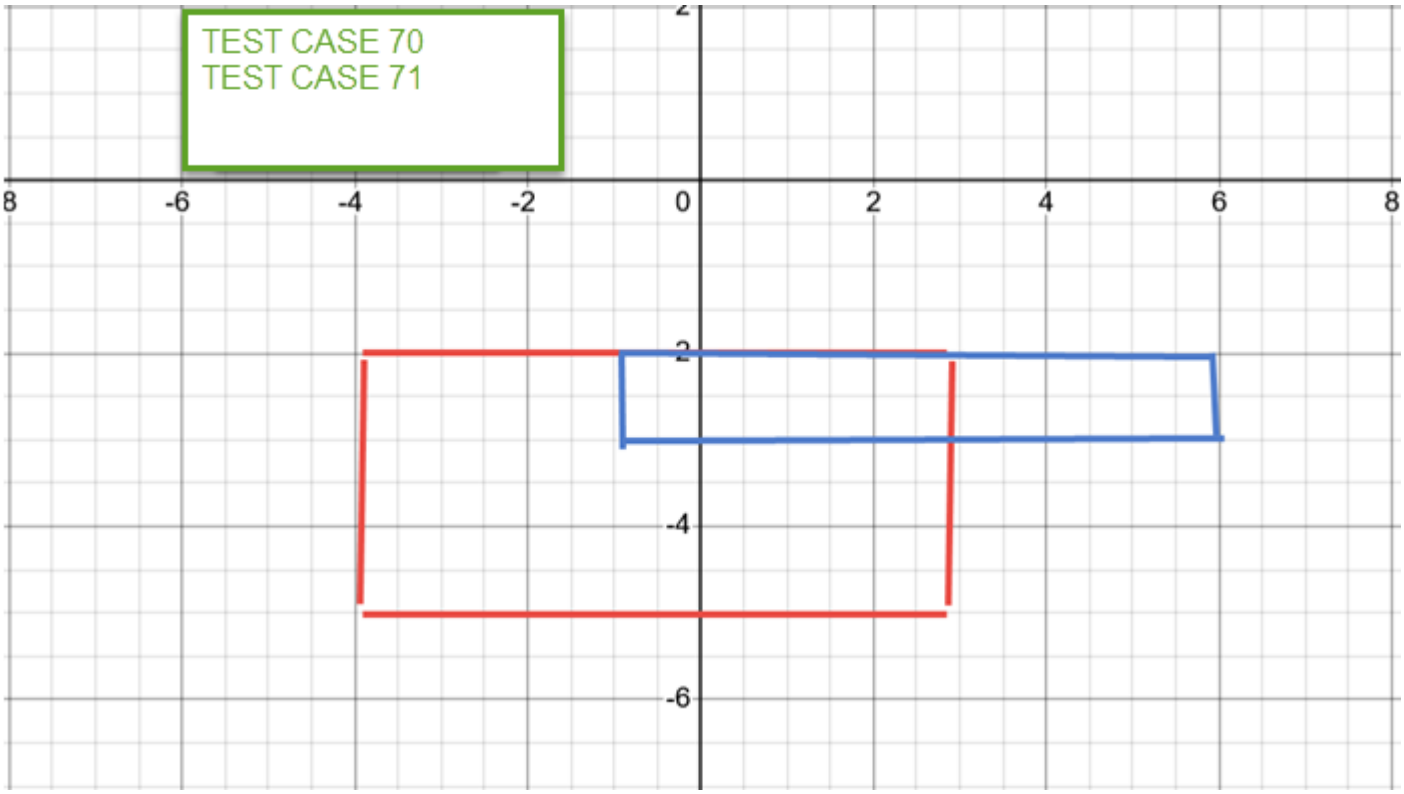
```
rect1Topright: -2
rect2TopRight: -2
rect1BottomLeft: -3
rect2BottomLeft: -5
1
1
MUST29
7
1
MUST41
The overlapping area is: 7
```

```
//TEST CASE 69
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-4, -5},  new int[]{3, -2},new int[]{-4, -3}, new int[]{6, -2}));
//
```

```
rect1Topright: -2
rect2TopRight: -2
rect1BottomLeft: -5
rect2BottomLeft: -3
7
1
MUST42
test
The overlapping area is: 7
```

Again, even though the outcomes are correct we can see that Test case 68 is entering two areas of code whereas Test case 69 enters one.. This is frustrating given my effort to replicate code....
But as long as it passes, we can consider the last entry to be correct...

TEST CASE 70 AND TEST CASE 71:



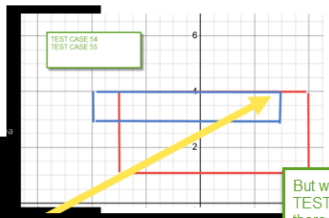

```
//TEST CASE 70
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-4, -5},  new int[]{3, -2},new int[]{-1, -3}, new int[]{6, -2}));
//
```

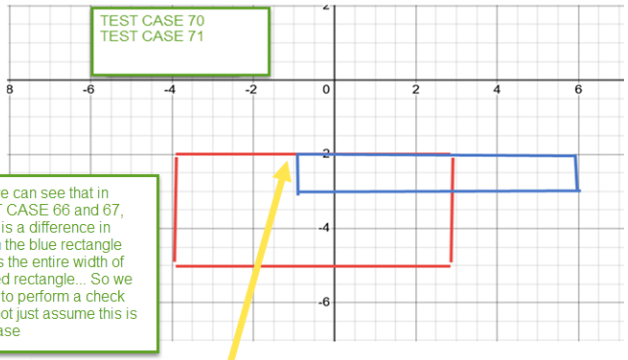
```
rect1Topright: -2
rect2TopRight: -2
rect1BottomLeft: -5
rect2BottomLeft: -3
7
1
MUST42
test
The overlapping area is: 7
```

```
//TEST CASE 71
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-1, -3}, new int[]{6, -2},new int[]{-4, -5},  new int[]{3, -2}));
//
```

```
rect1Topright: -2
rect2TopRight: -2
rect1BottomLeft: -3
rect2BottomLeft: -5
4
1
MUST34
7
1
MUST41
The overlapping area is: 7
```

I have performed following correction....

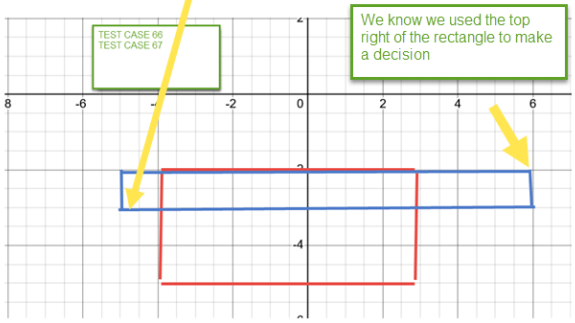





```
if (rect2TopRight[1]==rect1TopRight[1])
{
    //again we need to add due to crossing the Y axis

    //in this area due to test case 66 and 67..
    //this is already in an area of code that considers spanning Y axis both rectangles.
    //so what makes this scenario different to Test case 55 which passes through this area
    //we know they both have the same Y coordinate for the top right of the rectangle
    //rect1=blue

    if (rect1TopRight[0]>rect2TopRight[0])
    {
        //in this area of code due to test cases 68 and 69

          width = Math.abs(0-rect2BottomLeft[0]) + Math.abs(0-rect2TopRight[0]);
System.out.println(width);
 height = Math.abs(0-rect1BottomLeft[1]) - Math.abs(0-rect1TopRight[1]);
 System.out.println(height);
 System.out.println("MUST41");


    }
```

But we can see that in TEST CASE 66 and 67, there is a difference in which the blue rectangle spans the entire width of the red rectangle... So we need to perform a check and not just assume this is the case

TEST CASE 66 AND TEST CASE 67:



We know we used the top right of the rectangle to make a decision

I have implemented the following code changes:

```
    if (rect1TopRight[0]>rect2TopRight[0])
    {
        //we can not assume that the rectangle overlaps from the top left of the shape
        //this needs to be checked exactly to make a decision such as for test case 66 and 67
        //since it will have an effect on cases such as test case 70 and 71

        if (rect1BottomLeft[0]<=rect2BottomLeft[0])
        {

        //in this area of code due to test cases 68 and 69

          width = Math.abs(0-rect2BottomLeft[0]) + Math.abs(0-rect2TopRight[0]);
System.out.println(width);
 height = Math.abs(0-rect1BottomLeft[1]) - Math.abs(0-rect1TopRight[1]);
 System.out.println(height);
 System.out.println("MUST41");
        }

        else
        {
            //red=rect2
            //for cases such as test case 70 and 71,,,
            //the width will be different now
             width = Math.abs(0-rect1BottomLeft[0]) + Math.abs(0-rect2TopRight[0]);
             System.out.println(width);
              height = Math.abs(0-rect1BottomLeft[1]) - Math.abs(0-rect1TopRight[1]);
               System.out.println(height);
 System.out.println("MUST43");
```

```
//TEST CASE 70
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-4, -5},  new int[]{3, -2},new int[]{-1, -3}, new int[]{6, -2}));
//
```

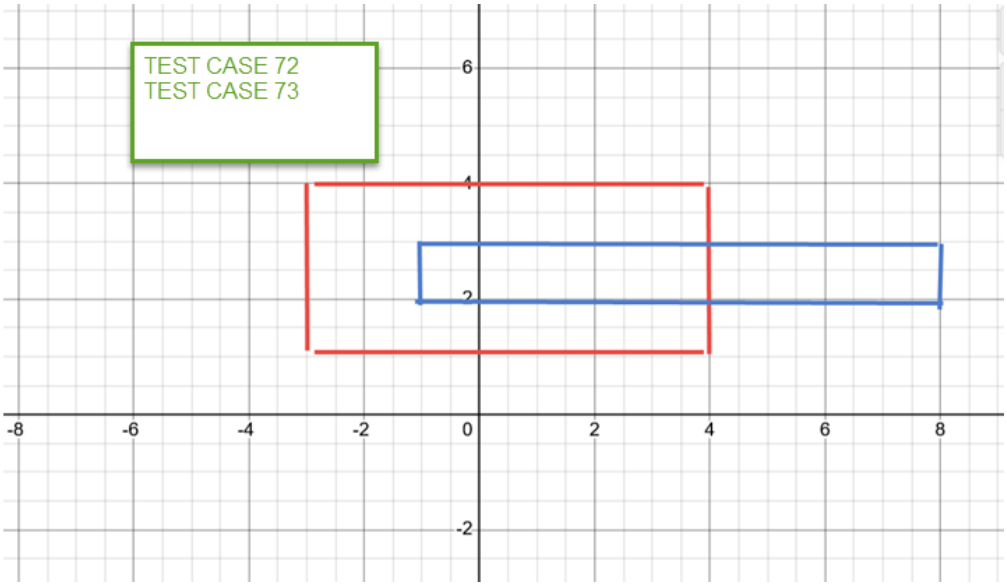
```
rect1Topright: -2
rect2TopRight: -2
rect1BottomLeft: -5
rect2BottomLeft: -3
MUST42
test
The overlapping area is: 4
```

I will now address for other area of code

```
//TEST CASE 71
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-1, -3}, new int[]{6, -2},new int[]{-4, -5},  new int[]{3, -2}));
//
```

But I tidied up the section of the code since I had extra else statement and logic was not brilliant....

```
rect1Topright: -2
rect2TopRight: -2
rect1BottomLeft: -3
rect2BottomLeft: -5
4
1
MUST34
MUST41
The overlapping area is: 4
```



Both are totally incorrect from height perspective...

```
overlappingArea(new int[]{-3, 1}, new int[]{4, 4},new int[]{-1, -2},  new int[]{8, 3}));
```

```
rect1Topright: 4
rect2TopRight: 3
rect1BottomLeft: 1
rect2BottomLeft: -2
-5
-2
MUST21
The overlapping area is: 10
```

```
//TEST CASE 73
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-1, -2},  new int[]{8, 3},new int[]{-3, 1}, new int[]{4, 4}));
//
```

```
rect1Topright: 3
rect2TopRight: 4
rect1BottomLeft: -2
rect2BottomLeft: 1
-5
-2
MUST20
test
The overlapping area is: 10
```

Note: There is no issue if I transposed the arrangement into the positive quadrant

I have had to have a massive analysis..
This else area of code is associated the if where it deals with the top right not alligned on the Y axis
In principle all the criteria are the same as the if area (with exception of this logic).
So I had to copy the code from above into below section...

Now the most confusing aspect is that in the above if area, we focussed on comparing the top right X coordinates... Both were in the respective sections of code when rectangle1 and rectangle2 had coordinates switched...

However we also require a copy of each other's respective code to be available in both sections...
Reason is that there will be a gap in logic available to make correct decision...

```
if (rect2TopRight[0]>=rect1TopRight[0])
{
```

```
if (rect1TopRight[0]>=rect2TopRight[0])
```

I now have a feeling I could have completed the entire project based on this.
I would need to perform a similar structure:
if both rectangles resided in one quadrant
if rectangles crossed the y axis
But I know it is never as simple as this in reality.

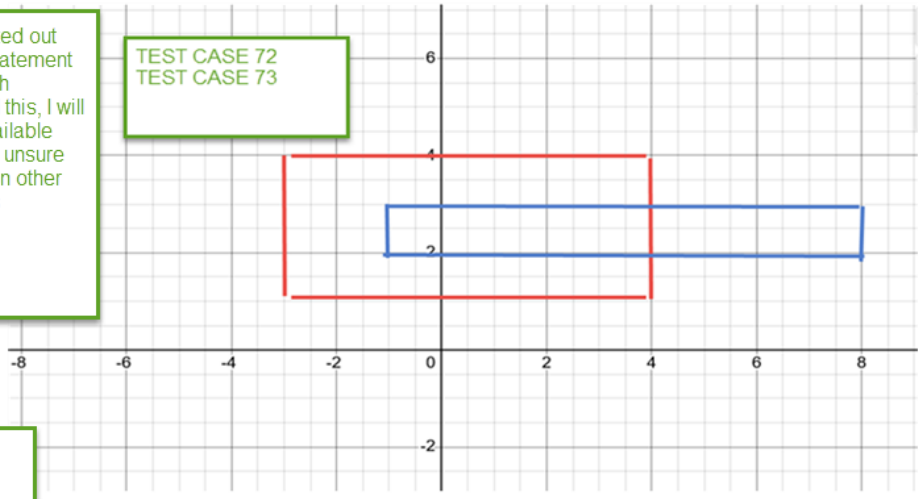
```
    if (rect2TopRight[0]>=rect1TopRight[0])
    {
        //we are here because of test case 70 and 71
        //
        if (rect2BottomLeft[0]<=rect1BottomLeft[0])
        {
            width = Math.abs(0-rect1BottomLeft[0]) + Math.abs(0-rect1TopRight[0]);
            height = Math.abs(0-rect2BottomLeft[1]) - Math.abs(0-rect2TopRight[1]);
        }
        else
        {
            width = Math.abs(0-rect2BottomLeft[0]) + Math.abs(0-rect1TopRight[0]);
            height = Math.abs(0-rect2BottomLeft[1]) - Math.abs(0-rect2TopRight[1]);
            System.out.println("MUST42");
        }
    }
```

```
if (rect1TopRight[0]>=rect2TopRight[0])
{
    //we are here because of test case 70 and 71
    //
    if (rect1BottomLeft[0]<=rect2BottomLeft[0])
    {
        width = Math.abs(0-rect2BottomLeft[0]) + Math.abs(0-rect2TopRight[0]);
        height = Math.abs(0-rect1BottomLeft[1]) - Math.abs(0-rect1TopRight[1]);
    }
    else
    {
        width = Math.abs(0-rect1BottomLeft[0]) + Math.abs(0-rect2TopRight[0]);
        height = Math.abs(0-rect1BottomLeft[1]) - Math.abs(0-rect1TopRight[1]);
        System.out.println("MUST41");
    }
}
```

```
//TEST CASE 72
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-3, 1}, new int[]{4, 4},new int[]{-1, -2},  new int[]{8, 3}));
//
```

```
rect1Topright: 4
rect2TopRight: 3
rect1BottomLeft: 1
rect2BottomLeft: -2
-5
-2
MUST21
The overlapping area is: 10
```

I commented out the else statement code which generated this, I will have it available since I am unsure its effect on other test cases



The outcome is as expected now

```
rect1Topright: 4
rect2TopRight: 3
rect1BottomLeft: 1
rect2BottomLeft: -2
IN THIS SECTION1
MUST42
The overlapping area is: 5
```

I commented out the else statement code which generated this, I will have it available since I am unsure its effect on other test cases

```
//TEST CASE 73
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-1, -2},  new int[]{8, 3},new int[]{-3, 1}, new int[]{4, 4}));
//
```

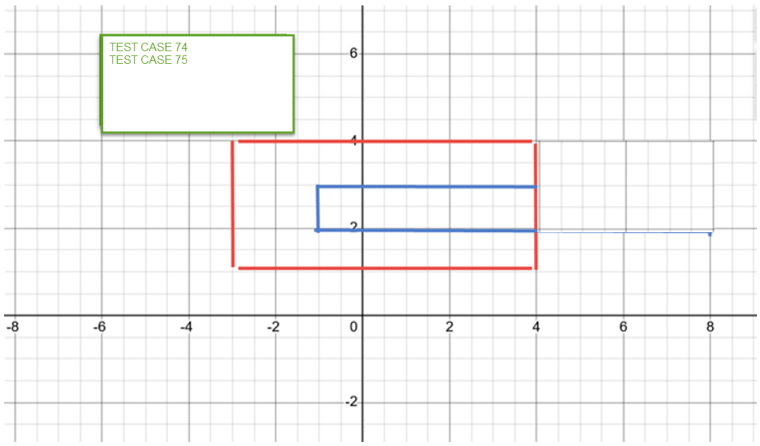
```
rect1Topright: 3
rect2TopRight: 4
rect1BottomLeft: -2
rect2BottomLeft: 1
-5
-2
MUST20
test
The overlapping area is: 10
```

```
rect1Topright: 3
rect2TopRight: 4
rect1BottomLeft: -2
rect2BottomLeft: 1
IN THIS SECTION2
MUST41
test
The overlapping area is: 5
```

The outcome is as expected now

I have in the process spawned a few more test cases to ensure things are all ok.

TEST CASE 74 AND TEST CASE 75



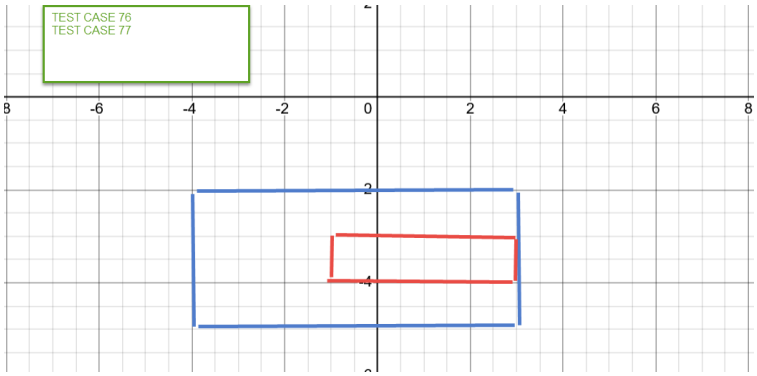

```
//TEST CASE 74
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-3, 1},  new int[]{4, 4},new int[]{-1, 2}, new int[]{4, 3}));
//
```

```
rect1Topright: 4
rect2TopRight: 3
rect1BottomLeft: 1
rect2BottomLeft: 2
5
-1
MUST33
**********************************************
AREA: 5
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 5
```

```
//TEST CASE 75
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-1, 2}, new int[]{4, 3},new int[]{-3, 1},  new int[]{4, 4}));
//
```

```
rect1Topright: 3
rect2TopRight: 4
rect1BottomLeft: 2
rect2BottomLeft: 1
5
-1
MUST34
**********************************************
AREA: 5
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 5
```

TEST CASE 76 AND TEST CASE 77



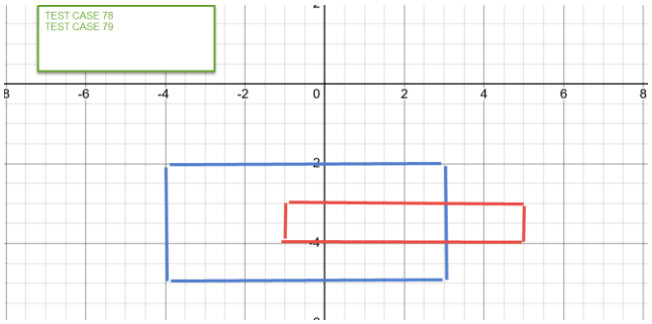

```
//TEST CASE 76
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-4, -5}, new int[]{3, -2},new int[]{-1, -4},  new int[]{3, -3}));
//
```

```
rect1Topright: -2
rect2TopRight: -3
rect1BottomLeft: -5
rect2BottomLeft: -4
4
1
MUST33
**********************************************
AREA: 4
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 4
```

```
//TEST CASE 77
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-1, -4},  new int[]{3, -3},new int[]{-4, -5}, new int[]{3, -2}));
//
```

```
rect1Topright: -3
rect2TopRight: -2
rect1BottomLeft: -4
rect2BottomLeft: -5
4
1
MUST34
**********************************************
AREA: 4
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 4
```

TEST CASE 78 AND TEST CASE 79



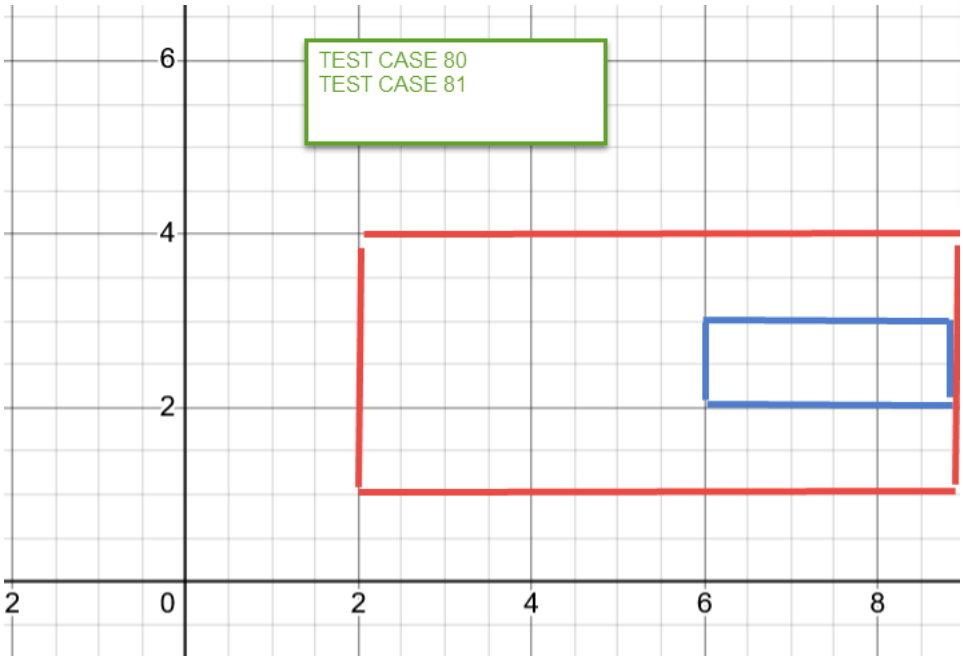

```
 //TEST CASE 78
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-4, -5},  new int[]{3, -2},new int[]{-1, -4}, new int[]{5, -3}));
//
```

```
rect1Topright: -2
rect2TopRight: -3
rect1BottomLeft: -5
rect2BottomLeft: -4
4
1
MUST33
**********************************************
AREA: 4
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 4
```

```
//TEST CASE 79
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-1, -4}, new int[]{5, -3},new int[]{-4, -5},  new int[]{3, -2}));
//
```

```
rect1Topright: -3
rect2TopRight: -2
rect1BottomLeft: -4
rect2BottomLeft: -5
4
1
MUST34
**********************************************
AREA: 4
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 4
```

TEST CASE 80 AND TEST CASE 81



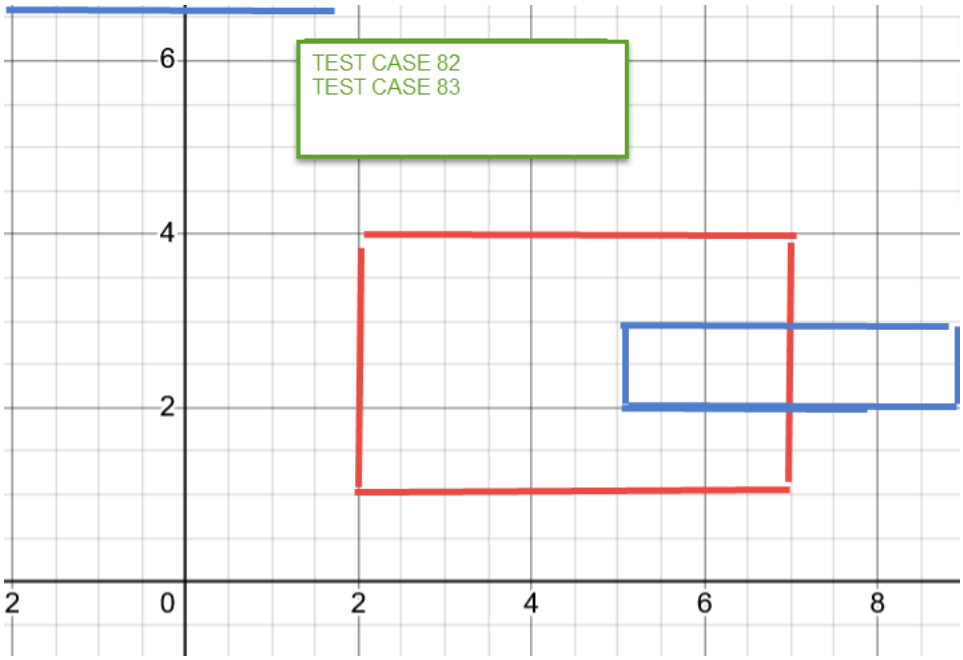

```
  //TEST CASE 80
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{2, 1}, new int[]{9, 4},new int[]{6, 2},  new int[]{9, 3}));
 //
```

```
rect1Topright: 4
rect2TopRight: 3
rect1BottomLeft: 1
rect2BottomLeft: 2
MUST3
-3
-1
**********************************************
AREA: 3
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 3
```

```
 //TEST CASE 81
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{6, 2},  new int[]{9, 3},new int[]{2, 1}, new int[]{9, 4}));
 //
```

```
rect1Topright: 3
rect2TopRight: 4
rect1BottomLeft: 2
rect2BottomLeft: 1
MUST1
-3
-1
**********************************************
AREA: 3
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 3
```

TEST CASE 82 AND TEST CASE 83:



These are incorrect

```
 //TEST CASE 82
System.out.println("The overlapping area is: " + overlappingArea(new int[]{2, 1},  new int[]{7, 4},new int[]{5, 2}, new int[]{9, 3}));
//
```

```
rect1Topright: 4
rect2TopRight: 3
rect1BottomLeft: 1
rect2BottomLeft: 2
MUST3
-4
-1
*********************************************
AREA: 4
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 4
```

```
    //TEST CASE 83
System.out.println("The overlapping area is: " + overlappingArea(new int[]{5, 2}, new int[]{9, 3},new int[]{2, 1},  new int[]{7, 4}));
//
```

```
rect1Topright: 3
rect2TopRight: 4
rect1BottomLeft: 2
rect2BottomLeft: 1
MUST1
-4
-1
*********************************************
AREA: 4
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 4
```

I completed several changes. I am presuming whilst I started challenge in positive quadrant, I was less stringent on some of the expressions.. It has been tidied now

I

I had to complete an extensive amount of tidy up to my code and re-understand several nested code blocks...

I

```
788
789                 else
790                 {
791                     //in here due to test case 82 and 83
792                     //the associated if is due to there being rectangles across the Y axis
793                     //this is clearly not the case, yet the logic seems to be lacking massively....
794                     //it needs spliting into if else
795                     //we know following at this point:   both rectangles have different X coordinate for bottom left
796                     //rect2bottomleft X coordinate is greater than or equal  rect1bottomleft (red)
797                     //rect1topright (red) Y coordinate is greater equal to  rect2TopRight
798
799                     //there is every possibility that is requires the exact same code as
800
801                     //red = rect2
802
803                     if (rect1TopRight[0]>rect2TopRight[0])
804                     {
805                         System.out.println("MUST60");
806                         width = Math.abs(0-rect1BottomLeft[0]) - Math.abs(0-rect2TopRight[0]);
807                         System.out.println(width);
808                         height = Math.abs(0-rect1BottomLeft[1]) - Math.abs(0-rect1TopRight[1]);
809                          System.out.println(height);
810                     }
811
812             //this would be shapes not spanning Y axis
813             //it has used the property of one shape..
814             //this is potentially my first bits of code since it assumes that rect1 and rect2 shares the same X coordinate...
815             //it would be false under any other circumstance.. But I am leaving it intact...
816             else
817             {
818                 if (rect1TopRight[0]==rect2TopRight[0])
819                 {
820             System.out.println("MUST3");
821             width = Math.abs(0-rect2BottomLeft[0]) - Math.abs(0-rect2TopRight[0]);
822             System.out.println(width);
823             height = Math.abs(0-rect2BottomLeft[1]) - Math.abs(0-rect2TopRight[1]);
824             System.out.println(height);
825             //return (Math.abs(width) * Math.abs(height));
826             store[count]=(Math.abs(width) * Math.abs(height));
827             count++;
828                 }
829                 else
830                 {
831                     System.out.println("MUST61");
832                     width = Math.abs(0-rect2BottomLeft[0]) - Math.abs(0-rect1TopRight[0]);
833                     System.out.println(width);
834                     height = Math.abs(0-rect2BottomLeft[1]) - Math.abs(0-rect2TopRight[1]);
835                     System.out.println(height);
836                     store[count]=(Math.abs(width) * Math.abs(height));
837                     count++;
838                 }
839
840
841             }
```

```
 //TEST CASE 82
System.out.println("The overlapping area is: " + overlappingArea(new int[]{2, 1},  new int[]{7, 4},new int[]{5, 2}, new int[]{9, 3}));
//
```

```
rect1Topright: 4
rect2TopRight: 3
rect1BottomLeft: 1
rect2BottomLeft: 2
MUST61
-2
-1
**********************************************
AREA: 2
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 2


** Process exited - Return Code: 0 **
```

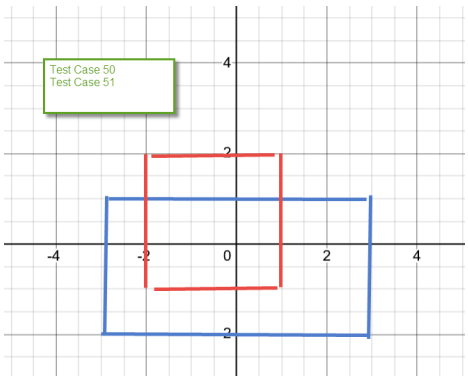
```
    //TEST CASE 83
System.out.println("The overlapping area is: " + overlappingArea(new int[]{5, 2}, new int[]{9, 3},new int[]{2, 1},  new int[]{7, 4}));
//
```

```
rect1Topright: 3
rect2TopRight: 4
rect1BottomLeft: 2
rect2BottomLeft: 1
MUST62
-2
-1
**********************************************
AREA: 2
AREA: 0
AREA: 0
AREA: 0
The overlapping area is: 2
```

I have now finished all my test cases... I will need to go through all of them from the start..
And now aim to phase out that method call. And tidy output...

There is still one test case which will cause lots issues and that is the overlap spanning all four quadrants...

<u>TEST CASE 50 AND TEST CASE 51:</u>



```java
//TEST CASE 50
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-2, -1},  new int[]{1, 2},new int[]{-3, -2}, new int[]{3, 1}));
//
```

```
rect1Topright: 2
rect2TopRight: 1
rect1BottomLeft: -1
rect2BottomLeft: -2
3
1
MUST40
The overlapping area is: 3
```

```java
//TEST CASE 51 - flip of the above
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-3, -2}, new int[]{3, 1},new int[]{-2, -1},  new int[]{1, 2}));
//
```

```
rect1Topright: 1
rect2TopRight: 2
rect1BottomLeft: -2
rect2BottomLeft: -1
3
1
MUST39
The overlapping area is: 3
```

I am now adding code into different areas and its first time I am identifying reusable code..
For now, I have repeated code blocks.... And as part of the tidy up process, will consolidate into methods..

For now, my priority is to ensure that both test cases enter correct area of code...

```java
if (rect2TopRight[0]>rect1TopRight[0])
{
    width = Math.abs(0-rect1BottomLeft[0]) - Math.abs(0-rect1TopRight[0]);
    height = Math.abs(0-rect2BottomLeft[1]) - Math.abs(0-rect2TopRight[1]);

    //this checks if spanning across the Y axis
    if ((rect2BottomLeft[0]<0) && (rect2TopRight[0]>0)
    && (rect1BottomLeft[0]<0) && (rect1TopRight[0]>0))
    {
        //in here due to test cases 50 and 51.
        //this is an extreme case in which it deals with overlap across all quadrants
        //we would firstly need a check to see if the rectangles also span across the X axis

                if ((rect2BottomLeft[1]<0) && (rect2TopRight[1]>0)
              && (rect1BottomLeft[1]<0) && (rect1TopRight[1]>0))
              {
                  //in this section, we will need the extensive code of if loops for both scenarios in each rectangle arrangement
                  //(similar to test case 72 and 73)
                  //Reason is there is the same large freedom of movement of the rectangles.
                  //But I will need to be extremely mindful of the impact of the absolutes and whether to add/subtract the dimensions!!

                   System.out.println("2OVERLAP 4 QUADRANTS");
```

```java

 if (rect1TopRight[0]>rect2TopRight[0])
 {
     //I am in this section due to Test case 58 and 59
     //we see the following
     //rect2=red
     //if(rect1BottomLeft[0]<rect2BottomLeft[0])  //X coordinate of blue rectangle bottom left is bigger
     //if (rect1TopRight[1]<rect2TopRight[1])  //Y coordinate of red rectangle top right is larger

     //Again there is no reference to spanning Y axis. This is created as below

     if ((rect1BottomLeft[0]<0) && (rect1TopRight[0]>0)
     && (rect2BottomLeft[0]<0) && (rect2TopRight[0]>0))
     {

             if ((rect1BottomLeft[1]<0) && (rect1TopRight[1]>0)
               && (rect2BottomLeft[1]<0) && (rect2TopRight[1]>0))
               {
                   //in this section, we will need the extensive code of if loops for both scenarios in each rectangle arrangement
                   //(similar to test case 72 and 73)
                   //Reason is there is the same large freedom of movement of the rectangles.
                   //But I will need to be extremely mindful of the impact of the absolutes and whether to add/subtract the dimensions!

                   System.out.println("1OVERLAP 4 QUADRANTS");
```

I will now begin the main process...

But I have just figured it will be incorrect process performing the checks in here..
Reason being that whole area of code is nested inside a section governed by loops which will contradict each others state.. For that given reason... I am just rolling back the code and testing all my test cases..
I will include validation to ensure termination of code if it crosses all four quadrants.

```java
//TEST CASE 50
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-2, -1},  new int[]{1, 2},new int[]{-3, -2}, new int[]{3, 1}));
//

//TEST CASE 51 - flip of the above
 System.out.println("The overlapping area is: " + overlappingArea(new int[]{-3, -2}, new int[]{3, 1},new int[]{-2, -1},  new int[]{1, 2}));
//
```

```
1CHECKING OVERLAP 4 QUADRANTS
VIOLATION
```

```java
        //this is now if there is violation and overlap spans four quadrants.
        //it would be quite impossible to calculate

        if ((rect1BottomLeft[0]<0) && (rect1TopRight[0]>0)
    && (rect2BottomLeft[0]<0) && (rect2TopRight[0]>0))
    {
        System.out.println("1CHECKING OVERLAP 4 QUADRANTS");

            if ((rect1BottomLeft[1]<0) && (rect1TopRight[1]>0)
              && (rect2BottomLeft[1]<0) && (rect2TopRight[1]>0))
              {
                  System.out.println("VIOLATION");
                  System.exit(0);

              }
    }
```

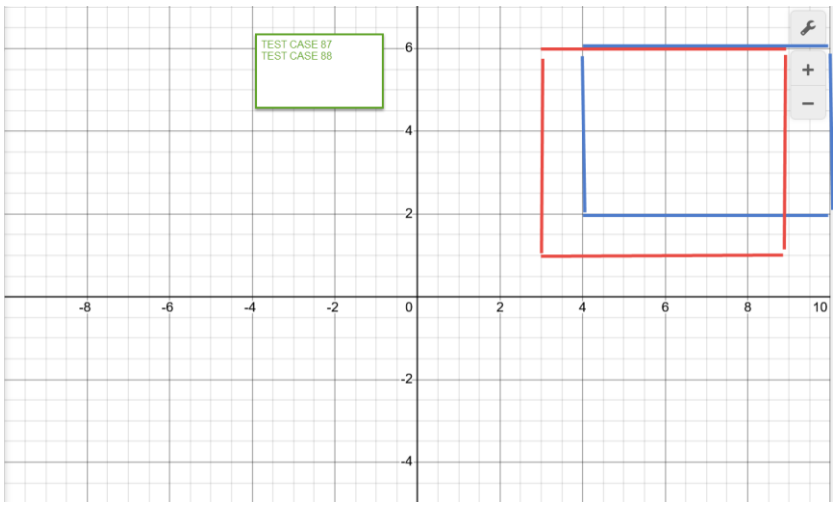
I am performing a few other test cases which I think are quite unique and also testing validation of the code....

TEST CASE 84

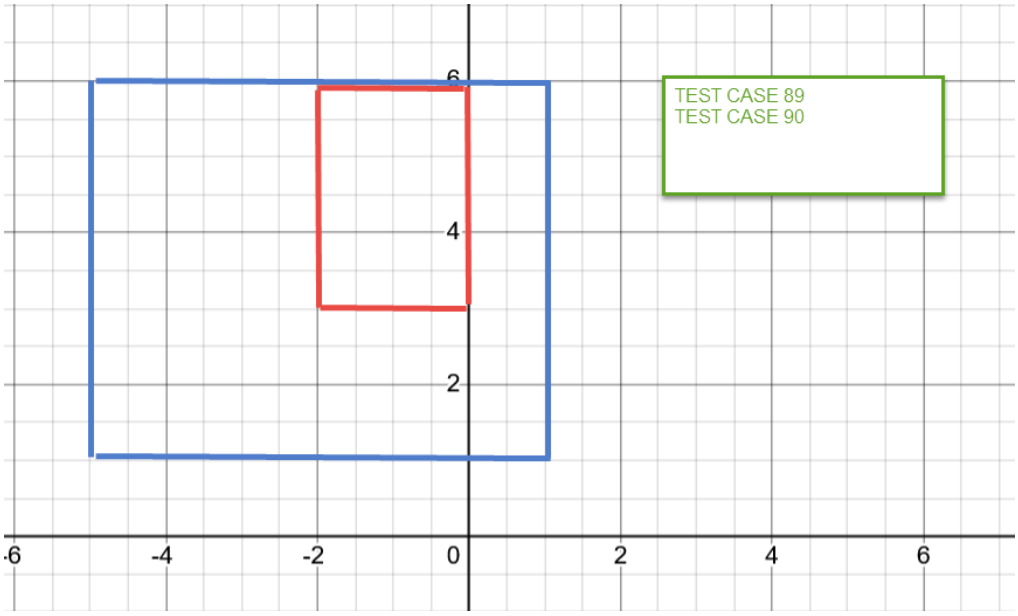
TEST CASE 85

TEST CASE 86

TEST CASE 87 & TEST CASE 88. It might be that I simply moved the shapes further across the X axis, but just performing to see if it validates...



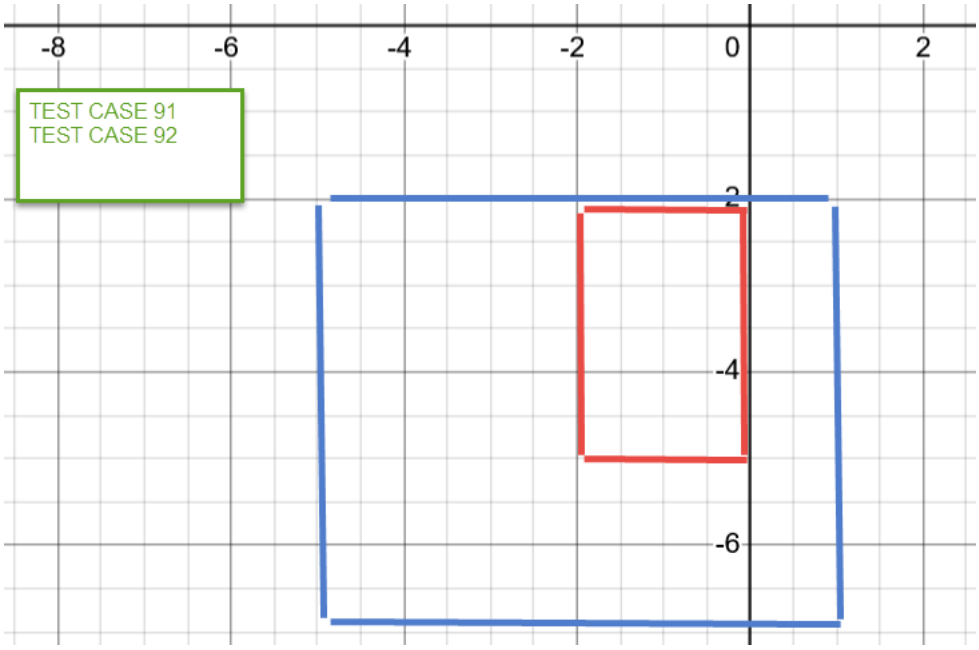
TEST CASE 89 & TEST CASE 90: FAIL




```
//TEST CASE 89
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-5, 1}, new int[]{1, 6},new int[]{-2, 3},  new int[]{0, 6}));
```

```
rect1Topright: 6
rect2TopRight: 6
rect1BottomLeft: 1
rect2BottomLeft: 3
1
-3
MUST7
test
The overlapping area is: 3
```

TEST CASE 91 & TEST CASE 92



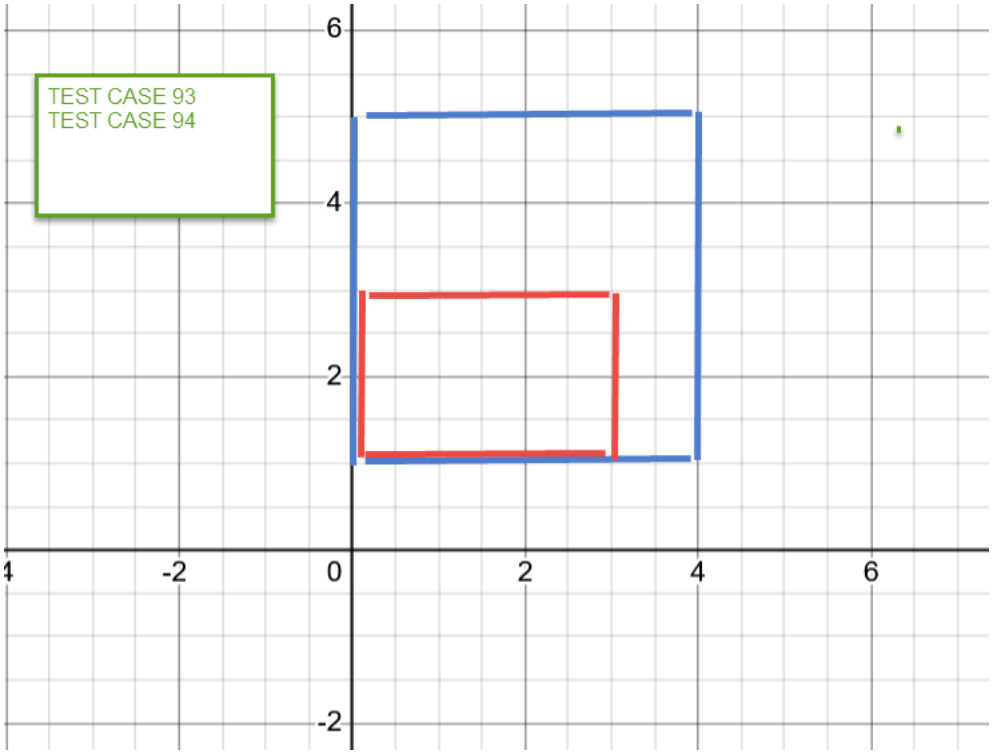

```
rect1Topright: -2
rect2TopRight: -2
rect1BottomLeft: -7
rect2BottomLeft: -5
2
3
MUST7a
test
The overlapping area is: 6
```

```
//TEST CASE 91
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-5, -7},  new int[]{1, -2},new int[]{-2, -5}, new int[]{0, -2}));
```

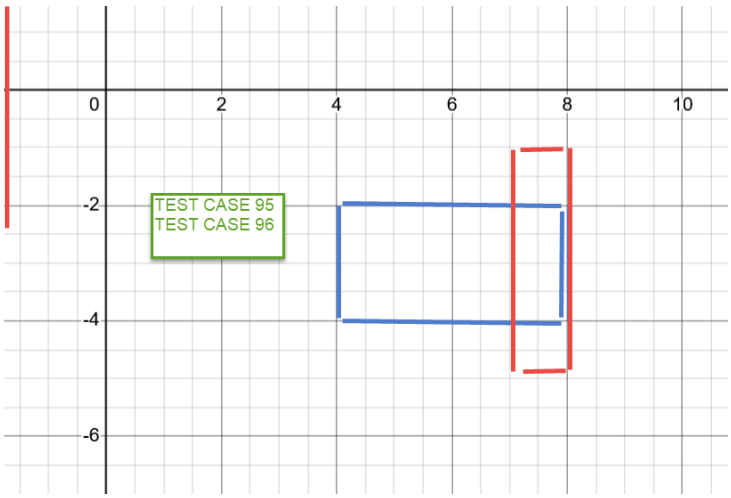
```
//TEST CASE 92
System.out.println("The overlapping area is: " + overlappingArea(new int[]{-2, -5}, new int[]{0, -2},new int[]{-5, -7},  new int[]{1, -2}));
```

```
rect1Topright: -2
rect2TopRight: -2
rect1BottomLeft: -5
rect2BottomLeft: -7
MUST60
2
3
2
3
MUST100
The overlapping area is: 6
```

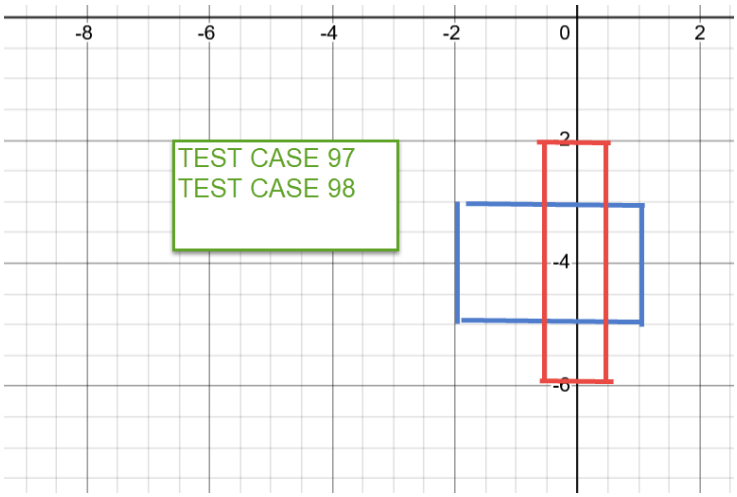
TEST CASE 93 & TEST CASE 94



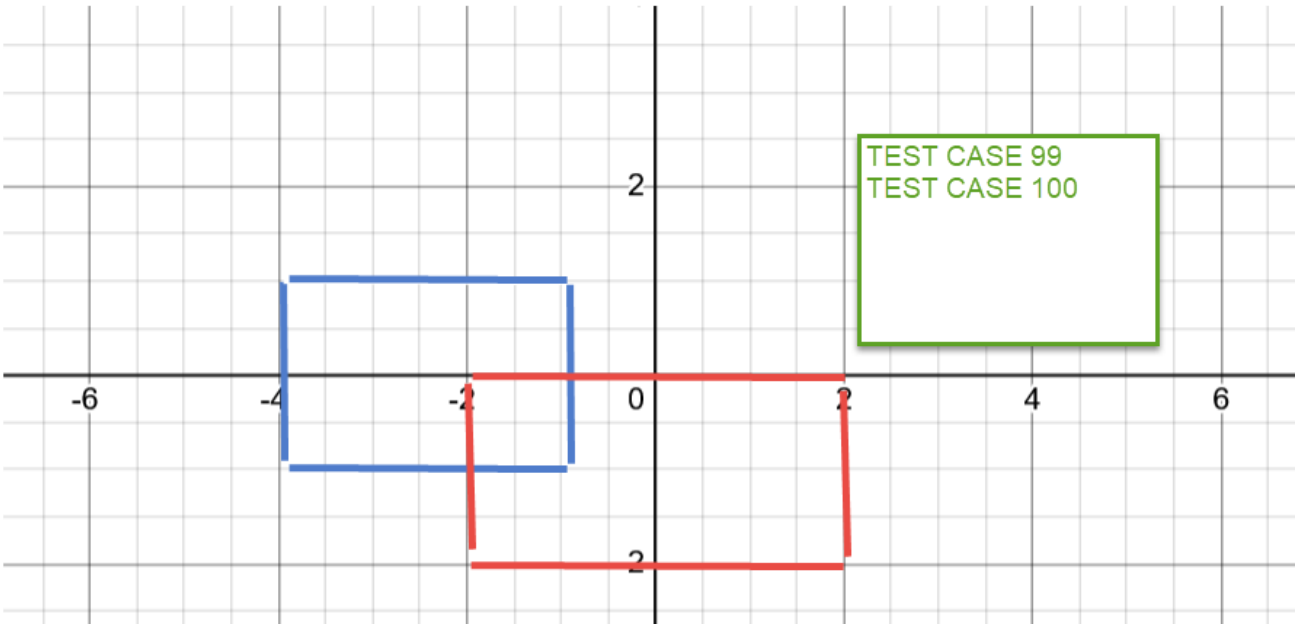
TEST CASE 95 & TEST CASE 96



TEST CASE 97 & TEST CASE 98



TEST CASE 99 & TEST CASE 100




```
//TEST CASE 99
    System.out.println("The overlapping area is: " + overlappingArea(new int[]{-4, -1},  new int[]{-1, 1},new int[]{-2, -2}, new int[]{2, 0}));
```

```
rect1Topright: 1
rect2TopRight: 0
rect1BottomLeft: -1
rect2BottomLeft: -2
REACH NOW!!!!!
1
1
MUST300
The overlapping area is: 1
```

```java
//TEST CASE 100 same as above but flipped
    System.out.println("The overlapping area is: " + overlappingArea(new int[]{-2, -2}, new int[]{2, 0},new int[]{-4, -1},  new int[]{-1, 1}));
```

```
rect1Topright: 0
rect2TopRight: 1
rect1BottomLeft: -2
rect2BottomLeft: -1
REACH THIS AREA!!!!!!!!!
1
1
MUST301
test
The overlapping area is: 1
```

TEST CASE 101