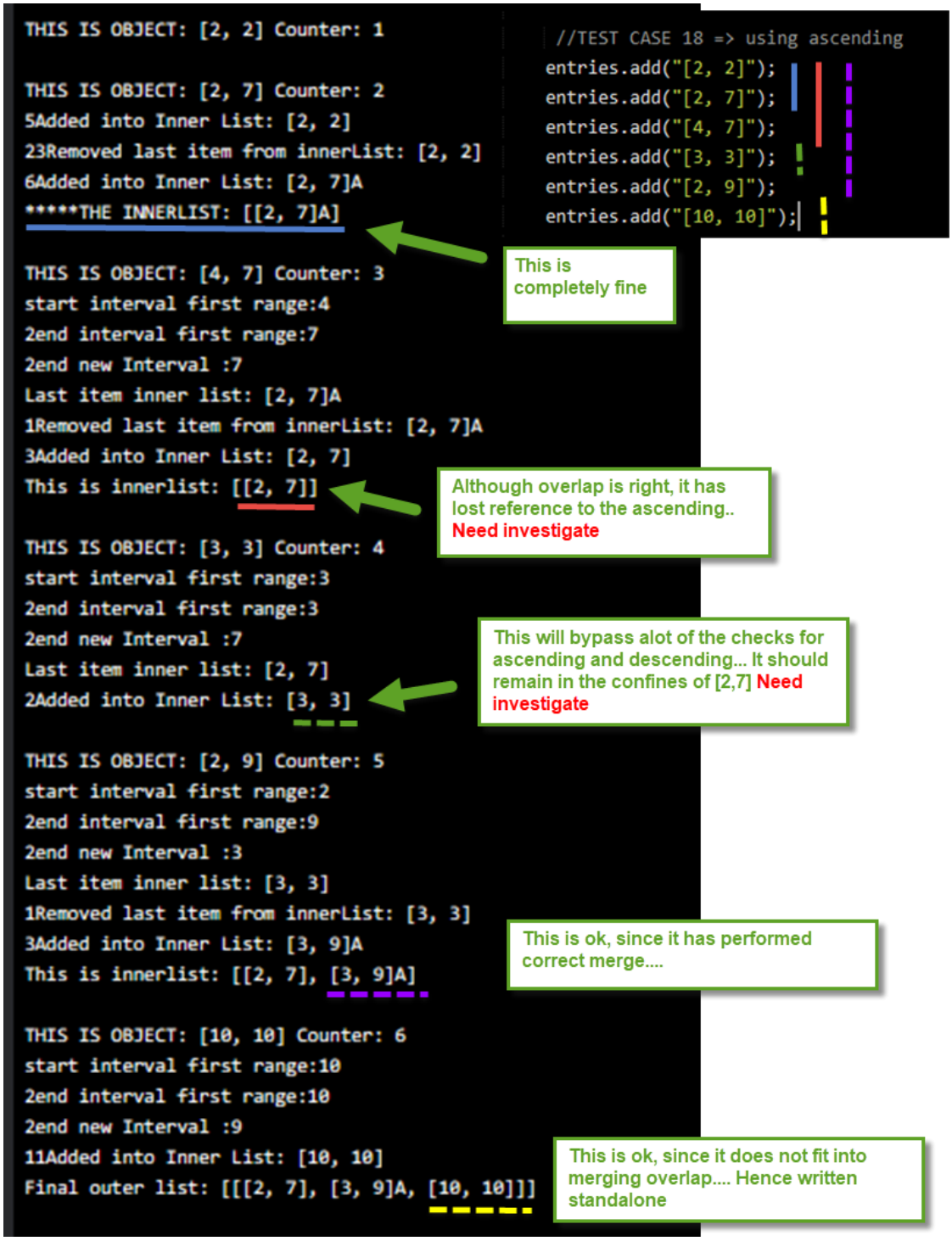
I have seen my first error here in a complex scenario. It shows how much the code has been evolved from Programiz version:




```
//TEST CASE 18 => using ascending
entries.add("[2, 2]");
entries.add("[2, 7]");
entries.add("[4, 7]");
entries.add("[3, 3]");
entries.add("[2, 9]");
entries.add("[10, 10]");
```

We are in this complex area of code when it performs the addition of [2,7] into the innerList... This is correct but its worth trying to follow this logic and ascertain why it has not picked up the A from the mergeDirection.....

```
if ((Integer.valueOf(endIntervalFirstRange)>=Integer.valueOf(endInterval)  && Integer.valueOf(endIntervalFirstRange)>=Integer.valueOf(startInterval))
||  (Integer.valueOf(endIntervalFirstRange)<=Integer.valueOf(endInterval)  && Integer.valueOf(endIntervalFirstRange)<=Integer.valueOf(startInterval))
                {
                    mergeDirection="";

                    if (Integer.valueOf(endIntervalFirstRange)>Integer.valueOf(endIntervalSecondRange)  && (endIntervalFirstRange!=startIntervalF
                    {
                        mergeDirection="A";
                    }

                    if (Integer.valueOf(endIntervalFirstRange)<Integer.valueOf(endIntervalSecondRange)&& (endIntervalFirstRange!=startIntervalFir
                    {
                        mergeDirection="D";
                    }

                    System.out.println("1Removed last item from innerList: " + innerList.get(innerList.size()-1));
                    innerList.remove(innerList.size()-1);
                    newInterval = "["+startInterval+","+ " " + endIntervalFirstRange+"]"+mergeDirection;
                    innerList.add(newInterval);
                    System.out.println("3Added into Inner List: " + newInterval);
                    System.out.println("This is innerlist: " + innerList);
                }
```

ISSUE IS HERE:
We can see that comparison is being made against endIntervalSecondRange.... We know we should be looking at the values that are in the innerList, and this is EndInterval...

To be replaced with endInterval

This effectively ensures that the range is suitable to replace the predecessor (last item in innerList). There is no concern here since [2,7],**[4,7]** is suitable to overwrite [2,7].

Unfortunately the error has persisted, so I have taken opportunity to open up this area further and also display output of variables....

```
272                 {
273                     mergeDirection="";
274                     System.out.println("HERE");
275                     System.out.println("start interval: " + startInterval);
276         System.out.println("end interval: " + endInterval);
277         System.out.println("start interval second range: " + startIntervalSecondRange);
278         System.out.println("end interval second range: " + endIntervalSecondRange);
279         System.out.println("start interval first range: " + startIntervalFirstRange);
280         System.out.println("end interval first range: " + endIntervalFirstRange);
281
282             //[2,7], [4,7]
283
284                     if (Integer.valueOf(endIntervalFirstRange)>Integer.valueOf(endInterval)  && (endIntervalFirstRange!=startIntervalFirstRange)
285                     {
286                         mergeDirection="A";
```

Ln: 282, Col: 19

Run | Share | Command Line Arguments

```
*****THE INNERLIST: [[2, 7]A]

THIS IS OBJECT: [4, 7] Counter: 3
start interval first range:4
2end interval first range:7
2end new Interval :7
Last item inner list: [2, 7]A
HERE
start interval: 2
end interval: 7
start interval second range: 2
end interval second range: 7
start interval first range: 4
end interval first range: 7
1Removed last item from innerList: [2, 7]A
3Added into Inner List: [2, 7]
```

[2,7]A is in the innerList and [4,7] is the incoming

7 7 7 4

I have found issue here... endInterval does not need to be greater. Even if it is equal, its a valid scenario for merging... I will change this for >=

I have applied this same logic across all the three loops in my code that deals with this scenario..

So far the code has functioned correctly.

```
THIS IS OBJECT: [4, 7] Counter: 3
start interval first range:4
2end interval first range:7
2end new Interval :7
Last item inner list: [2, 7]A
HERE
start interval: 2
end interval: 7
start interval second range: 2
end interval second range: 7
start interval first range: 4
end interval first range: 7
1Removed last item from innerList: [2, 7]A
3Added into Inner List: [2, 7]A
This is innerlist: [[2, 7]A]
```

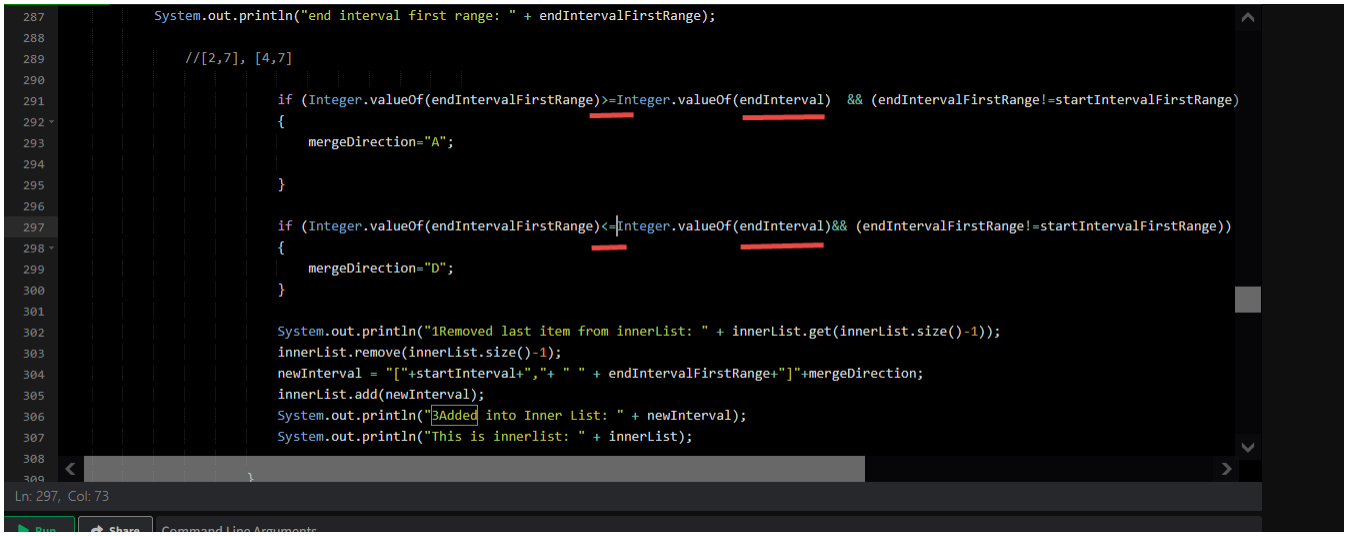
but I need to examine from the perspective of descending...

So I will create a scenario as follows:

[7,7], [7,4], [3,4] and see how the code behaves....

As to whether same change is required...

TEST CASE: Adapting the code with changes described (in every relevant section) as above and trying



[7,7], [7,4], [3,4]

```
1Removed last item from innerList: [7, 4]D
3Added into Inner List: [7, 4]D
This is innerlist: [[7, 4]D]
Final outer list: [[[7, 4]D]]
```

I will now run all my test cases again to see the impact since it is an extremely area of the code modified...

For some reason, the following is now passing through as descending...

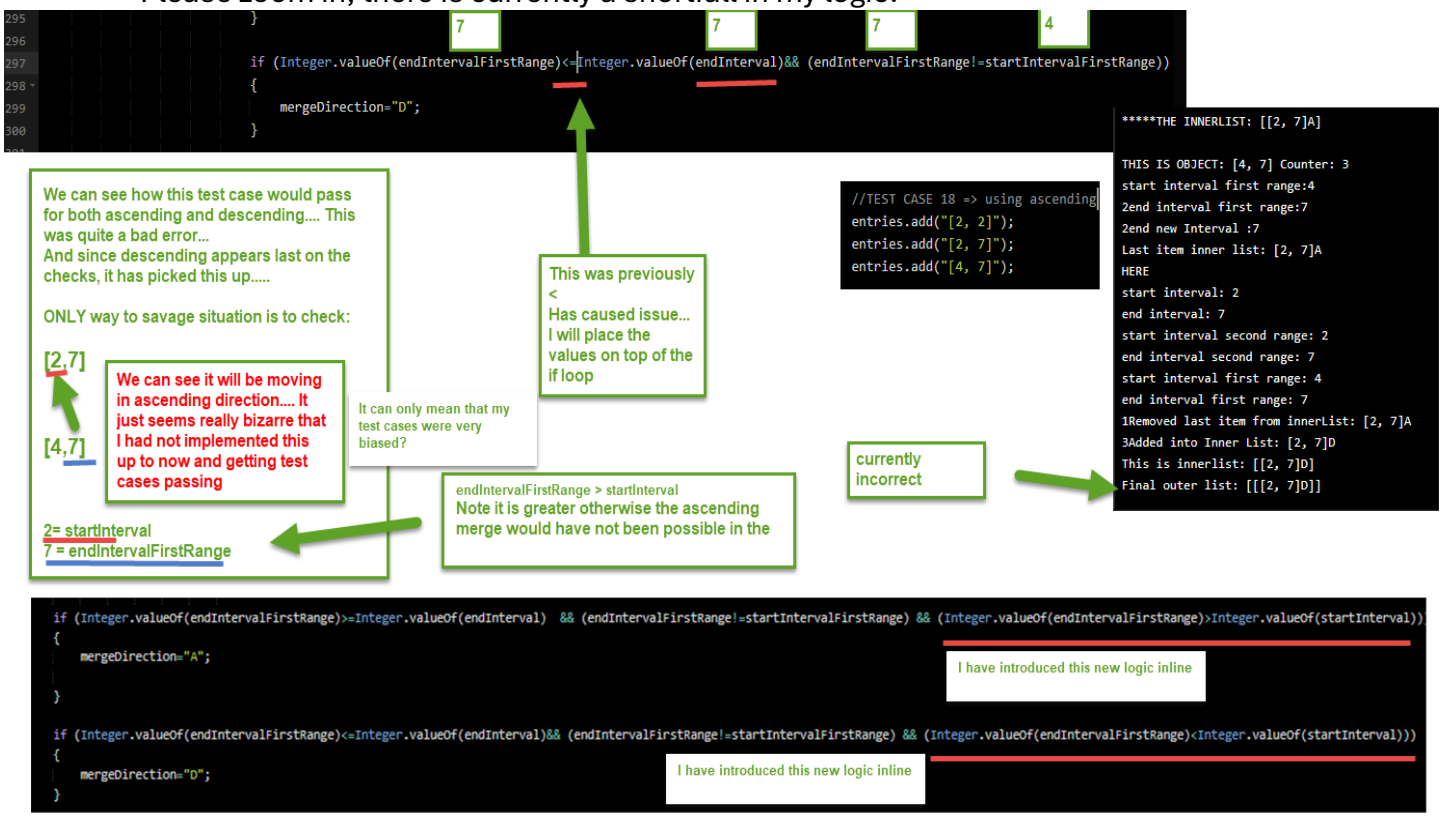
```
//TEST CASE 18 => using ascending
entries.add("[2, 2]");
entries.add("[2, 7]");
entries.add("[4, 7]");
```

So it shows that performing the following operation has had an issue....
But I clearly know from all my coding a lot of the code is reflective...
And this can never be the root cause.

```
if (Integer.valueOf(endIntervalFirstRange)<=Integer.valueOf(endInterval)
```

So I have opened the code up again:

Please zoom in, there is currently a shortfall in my logic:



I am now trying the test cases again and hopefully they should pass through correct area...

TEST CASE:

```
//TEST CASE 18 => using ascending
entries.add("[2, 2]");
entries.add("[2, 7]");
entries.add("[4, 7]");
```

```
1Removed last item from innerList: [2, 7]A
3Added into Inner List: [2, 7]A
This is innerlist: [[2, 7]A]
Final outer list: [[[2, 7]A]]
```

TEST CASE:

```
//TEST CASE 18 => using DESCENDING.....
entries.add("[7, 7]");
entries.add("[7, 4]");
entries.add("[3, 4]");
```

```
1Removed last item from innerList: [7, 4]D
3Added into Inner List: [7, 4]D
This is innerlist: [[7, 4]D]
Final outer list: [[[7, 4]D]]
```

I now feel I am largely on right track.. I need to replicate the underlined red code in all sections of my code in which it makes decision ascending and descending....

I had to be extremely careful since we know the some areas of code reference startInterval and endInterval, other areas reference startIntervalSecondRange, endIntervalSecondRange.

I also notice one stray variable during my testing... I already reached so many ambiguous ones, so if there is an opportunity I will phase out. But for now, it serves just sufficient logic to leave it aside..

TEST CASE:

```
//TEST CASE 19 => mixed
entries.add("[2, 2]");
entries.add("[2, 7]");
entries.add("[4, 7]");
entries.add("[2, 2]");
entries.add("[2, 7]");
entries.add("[4, 7]");
```

It has now passed...

THIS IS OBJECT: [2, 2] Counter: 1

THIS IS OBJECT: [2, 7] Counter: 2

5Added into Inner List: [2, 2]

23Removed last item from innerList: [2, 2]

6Added into Inner List: [2, 7]A

*****THE INNERLIST: [[2, 7]A]

THIS IS OBJECT: [4, 7] Counter: 3

start interval first range:4

2end interval first range:7

2end new Interval :7

Last item inner list: [2, 7]A

1Removed last item from innerList: [2, 7]A

3Added into Inner List: [2, 7]A

This is innerlist: [[2, 7]A]

THIS IS OBJECT: [2, 2] Counter: 4

start interval first range:2

2end interval first range:2

2end new Interval :7

Last item inner list: [2, 7]A

1Removed last item from innerList: [2, 7]A

3Added into Inner List: [2, 2]

This is innerlist: [[2, 2]]

THIS IS OBJECT: [2, 7] Counter: 5

start interval first range:2

2end interval first range:7

2end new Interval :2

Last item inner list: [2, 2]

1Removed last item from innerList: [2, 2]

3Added into Inner List: [2, 7]A

This is innerlist: [[2, 7]A]

THIS IS OBJECT: [4, 7] Counter: 6

start interval first range:4

2end interval first range:7

2end new Interval :7

Last item inner list: [2, 7]A

1Removed last item from innerList: [2, 7]A

3Added into Inner List: [2, 7]A

This is innerlist: [[2, 7]A]

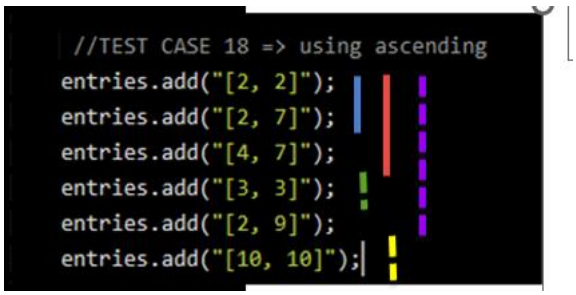
Final outer list: [[[2, 7]A]]

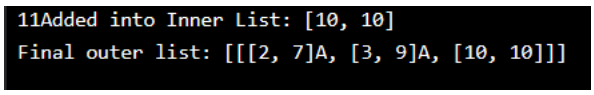
** Process exited - Return Code: 0 **

I am now going to go through all my test cases... Since the above passed, I have lots more confidence.. I will not record my findings anywhere, it will be visual analysis.

However more important is trying the exact test case at top of this document...

TEST CASE: FAIL

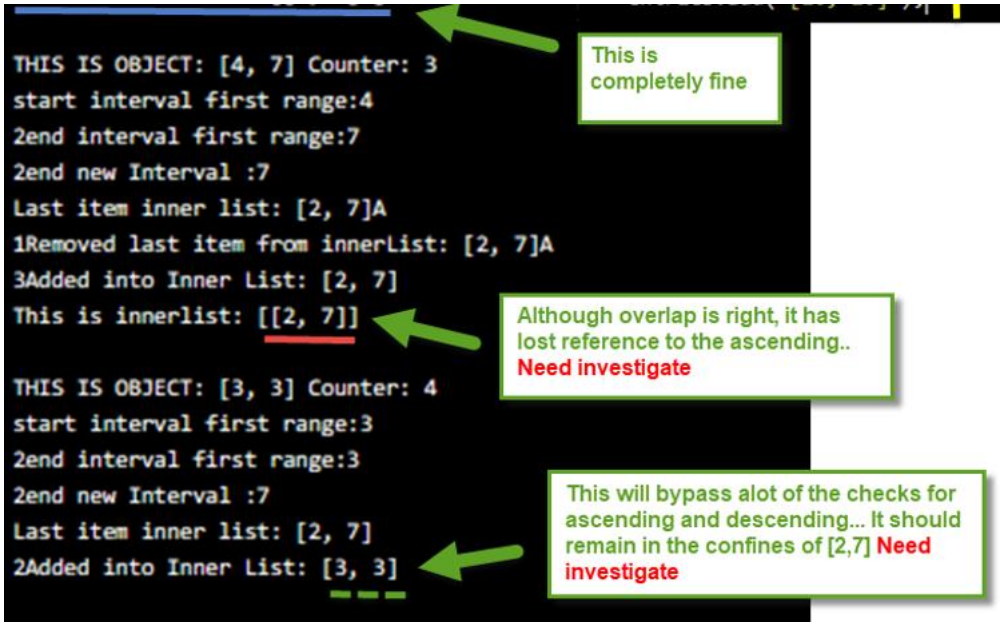




We can see that it has kept separate entry for [3,3] and when incoming [2,9] has arrived it has merged them...

But we can see that [3,3] should have remained inside [4,7].

This is logic that I have not completed at all..
But I know looking at Page 1 of this document, it was the secondary area that required investigation...



I will find the area of code which is 2Added and aim to resolve this..
I need to understand this better.

It has to perform the insertion of [X,X] if X is greater than 7.
Otherwise it would naturally fall into [2,7]

Fortunately this occurrence is in the else statement, so much easier to anticipate

```
317     else
318     {
319         if ((endIntervalFirstRange==startIntervalFirstRange) && (Integer.valueOf(endIntervalFirstRange)>Integer.valueOf(endInterval)))
320         {
321     newInterval = "["+startIntervalFirstRange+","+ " " + endIntervalFirstRange+"]";
322     innerList.add(newInterval);
323     System.out.println("2Added into Inner List: " + newInterval);
```

I am hoping I do not need to reflect this logic elsewhere...
The only way to be sure if perform a new fresh test case as follows: [2,7], [3,3]. I do not expect an issue because it is in different section of code where the counter <=2 and managed.

I can examine this later once I have retested:

TEST CASE: Re-testing the original one in this documentation

```
//TEST CASE 18 => using ascending
entries.add("[2, 2]");
entries.add("[2, 7]");
entries.add("[4, 7]");
entries.add("[3, 3]");
entries.add("[2, 9]");
entries.add("[10, 10]");
```

```
THIS IS OBJECT: [3, 3] Counter: 4
start interval first range:3
2end interval first range:3
2end new Interval :7
Last item inner list: [2, 7]A

THIS IS OBJECT: [2, 9] Counter: 5
start interval first range:2
2end interval first range:9
2end new Interval :7
Last item inner list: [2, 7]A
ascending
1Removed last item from innerList: [2, 7]A
3Added into Inner List: [2, 9]A
This is innerlist: [[2, 9]A]

THIS IS OBJECT: [10, 10] Counter: 6
start interval first range:10
2end interval first range:10
2end new Interval :9
11Added into Inner List: [10, 10]
Final outer list: [[[2, 9]A, [10, 10]]]
```

It can be seen no action taken, which is correct

It now passes correctly

TEST CASE:

```
//TEST CASE 18 => using ascending
entries.add("[2, 7]");
entries.add("[3, 3]");
```

```
THIS IS OBJECT: [2, 7] Counter: 1

THIS IS OBJECT: [3, 3] Counter: 2
4Added into Inner List: [2, 3]A
4*****THE INNERLIST: [[2, 3]A]
Final outer list: [[[2, 3]A]]
```

In practice, this is incorrect since it should embed within [2,7]
So I will just follow the logic around into 4Added section

This was existing code in the section:

```
newInterval = "["+startIntervalFirstRange+","+ " " + endIntervalSecondRange+"]A";
innerList.add(newInterval);
System.out.println("4Added into Inner List: " + newInterval);
System.out.println("4*****THE INNERLIST: " + innerList);
```

Again concepts are very similar, but need to be careful and perform a write where into list if possible, since can see there is nothing in there at moment....

```
THIS IS OBJECT: [2, 7] Counter: 1

THIS IS OBJECT: [3, 3] Counter: 2
4Added into Inner List: [2, 3]A
4*****THE INNERLIST: [[2, 3]A]
Final outer list: [[[2, 3]A]]
```

I do not believe I need the repeat code in here at all. It is in a good place in the else section

It has to perform the insertion of [X,X] if X is greater than 7.
Otherwise it would naturally fall into [2,7]

```
if ((Integer.valueOf(endIntervalSecondRange)==Integer.valueOf(startIntervalSecondRange) && (Integer.valueOf(endIntervalFirstRange)>Integer.valueOf(endIntervalSecondRange))))
{
    newInterval = "["+startIntervalFirstRange+","+ " " + endIntervalSecondRange+"]";
    innerList.add(newInterval);
    System.out.println("4Added into Inner List: " + newInterval);
    System.out.println("4*****THE INNERLIST: " + innerList);
}
```

<u>TEST CASE: Repeating same one as above..</u>

```
//TEST CASE 18 => using ascending
entries.add("[2, 7]");
entries.add("[3, 3]");
```

```
THIS IS OBJECT: [2, 7] Counter: 1

THIS IS OBJECT: [3, 3] Counter: 2
start interval:
end interval:
start interval second range: 3
end interval second range: 3
start interval first range: 2
end interval first range: 7
4Added into Inner List: [2, 3]
4*****THE INNERLIST: [[2, 3]]
Final outer list: [[[2, 3]]]
```

I have now checked all points in my code where I perform innerList.add and seem content that all have received correct remediation…

I have one concern in that in lots of my loops, I performed equality checks on String as oppose to performing Integer.valueOf()… Of course Java informs me of > or <
But there are several with !=…….
I am going to cast all of the Strings and hopefully run through all my test cases again informally…