

We know in order to incorporate the summary range for ascending and descending sequence code together, there has to be a decision point in the code.

```
int[] nums = new int[]{0,1,2,4,5,7,6,10,14,17,19,23,24,25,26,35,42,47};
```

If the above code is executed with ascending summary range checker, we see at 7 and 6 it has regarded it as standalone.

```
[0->2, 4->5, 7, 6, 10, 14, 17, 19, 23->26, 35, 42, 47]
```

So now if I ran a slightly more sophisticated array with descending code, we can see that it makes correct decision....

```
int[] nums = new int[]{5,4,4,4,5,7,6,10,14,17,19,23,24,25,26,35,42,47};
```

```
[5->4, 4, 4, 5, 7->6, 10, 14, 17, 19, 23, 24, 25, 26, 35, 42, 47]
```

For now, I am just going to examine this from perspective of whole prices for the stocks as per the challenge.

Every time it makes a standalone decision, I will call the alternate class and await the outcome.

I will build my first test case simply from this.

TEST CASE: Configure code to call alternate class in which standalone decision is made. Use slight code adjustment to ensure that it is processing alternate code at same index location.....

For now I have introduced a much simpler array to get started....

```
//Merging both ascending and descending code  
int[] nums = new int[]{5,4,3,4,5,7,6};
```

Based on the above and current logic, we know that it would attempt a standalone input into the list for 5,4...

So I have identified this junction, and inserted the if code to check if descending...

```
144         if(k==0)
145         {
146             if ((nums[k]-1)==(nums[k+1]))
147             {
148                 System.out.println("-----NEW JUNCTION 15");
149                 //we know at this point we can definitely determine the start...
150                 //we are not sure if the descending will continue...
151                 start=String.valueOf(nums[k]);
152                 //break;
153             }
154             //it would have made this decision normally
155             //to write the 5 as standalone in {
156             else
157             {
158                 System.out.println("JUNCTION 12");
159                 System.out.println("5Writing standalone: " + start);
160                 sm.add(start);
161             }
162         }
```

The code now reaches this overall output,

```
JUNCTION 8
HERE
3Writing Standalone: 7
4Writing Standalone: 6
[4, 3->5, 7, 6]
```

So I need to check where it has decided to check between Junction 15 and at the point where it has decided to compare 4 with 3, since it clearly requires additional code.

Junctions of interest are 6,10,11,13



```

else
{
    System.out.println("JUNCTION 10");
    System.out.println("*****" + counter);
    //endConsecutive=true;
    end=String.valueOf(nums[k]);

    if (counter==0)
    {
        System.out.println("JUNCTION 11");

        if(k==0)
        {
            if ((nums[k]-1)==(nums[k+1]))
            {
                System.out.println("-----NEW JUNCTION 15");
            }
        }
    }
}


```

Upon doing this

```

146         if ((nums[k]-1)==(nums[k+1]))
147         {
148             System.out.println("-----NEW JUNCTION 15");
149             //we know at this point we can definitely determine the start...
150             //we are not sure if the descending will continue...
151             start=String.valueOf(nums[k]);
152             counter++;
153             //break;

```



My outputs were automatically looking to take on more meaning, however now I had to address the disconnect between 4 and 3.

[5->4, 3->5, 7, 6]

## TEST CASE: Investigating the pathway between 4 and 3

The following need to be traced

```
-----NEW JUNCTION 15
value k: 1
length nums: 7
value of counter: 1
CHECKING: 4 with 3
JUNCTION 6
JUNCTION 10
*****1
JUNCTION 14
Writing range: 5-> 4
value k: 2
length nums: 7
value of counter: 0
CHECKING: 3 with 4
```

Junction can be avoided since it is the main block of code where there are no influences on the start and end....

I have introduced this code above Junction 10, however it has disrupted the output:

```
System.out.println("-----NEW JUNCTION 18");
//we know at this point we can definitely determine the start as 5
//we know descending has continued with 4 and 3...
//so we need to include the descending for check

if ((nums[k]-1)==(nums[k+1]))
{
    //start=String.valueOf(nums[k]);
    counter++;
}
else
{
    System.out.println("JUNCTION 10");
    System.out.println("*****" + counter);
    //endConsecutive=true;
    end=String.valueOf(nums[k]);
}
```

[5->, 5->, 3->5, 7, 6]

TEST CASE: Checking the junctions again:

So I will now try to investigate again between the Junction 15 (note I have renamed this to new junction 17 since I had counted other existing incorrectly) and up to the point where it performs 'Checking 3 with 4'

It can be seen that I performed counter++ in junction 18, it has prevented it from entering here since it relies on counter==0

```
if (counter==0)
{
    System.out.println("JUNCTION 11");

    if(k==0)
    {
        if ((nums[k]-1)==(nums[k+1]))
        {
            System.out.println("-----NEW JUNCTION 17");
            //we know at this point we can definitely determine the start...
            //we are not sure if the descending will continue...
            start=String.valueOf(nums[k]);
            counter++;
            //break;
        }
    }
}
```

So I will remove the counter increment as part of Junction 18 (Note this appears higher in the code than junction 17).

I have also had to include in Junction 19....

```
System.out.println("-----NEW JUNCTION 19");
end=String.valueOf(nums[k+1]);
```

I can see my outcome is looking better. I just need to investigate now the start for 3->5. We know this would occur once it has written 5->3 onwards....

[5->3, 3->5, 7, 6]

But in reality there is absolutely nothing wrong with this, since it is investigating from the same point onwards....

So I will try extending the array above to something more adventurous:

TEST CASE:

```
int[] nums = new int[]{5,4,3,4,5,6,9,8};
```

[5→3, 3→6, 9, 8]

I can see it has adapted to the longer flow of ascending numbers....

I can see it has not handled the 9 and 8 correctly.

It has kept them as standalone.

My logic immediately tells me there are few areas I have not addressed.

I know I included condition `nums.length()-2` (this would include the number 9)....

So once again, I will not overcomplicate anything and just follow the junctions (once it has written 3-> 6 onwards).

TEST CASE: Investigating impact of Junction 6, 7 and 8 on the incorrect writing of standalone numbers

```
2Writing range: 3-> 6  
value k: 6  
length nums: 8  
value of counter: 0  
CHECKING: 9 with 8  
JUNCTION 6  
JUNCTION 7  
counter:0-----LAST TO LAST ITEM :9  
JUNCTION 8  
HERE  
3Writing Standalone: 9  
4Writing Standalone: 8  
[5->3, 3->6, 9, 8]
```

We can see all the execution has been done here.

We can see it is in right area since the else statement at top states its not consecutive match.... This is correct analysis of numbers 9 and 8.

And we know if we had the standalone code of ascending, it would have performed the same operation of writing two standalone numbers.

However if we had the standalone code of descending numbers, it would have written 9->8.

So we need to place this logic of descending in this area of code..... Since the else statement has recognised that its not ascending sequence at moment.

```
else    ///not consecutive match
{
    System.out.println("JUNCTION 6");
    if (k==nums.length-2)
    {
        System.out.println("JUNCTION 7");
        System.out.println("counter:" + counter + "-----LAST TO LAST");

        if (counter==0)
        {
            System.out.println("JUNCTION 8");
            end = String.valueOf(nums[k]);
            System.out.println("HERE");
            System.out.println("3Writing Standalone: " + end);
            sm.add(end);
            //System.out.println("6Writing range: " + start + "-> " + end);
            start=String.valueOf(nums[k+1]);
            System.out.println("4Writing Standalone: " + start);
            sm.add(start);
        }
    }
}
```



I have introduced this code and created another if else

```
109         if (counter==0)
110         {
111             if ((nums[k]-1)==(nums[k+1]))
112             {
113                 System.out.println("-----NEW JUNCTION 20");
114                 start=String.valueOf(nums[k]);
115                 end = String.valueOf(nums[k+1]);
116                 sm.add(start+"->" + end);
117                 System.out.println("7Writing range: " + start + "-> " + end);
118             }
119             else
120             {
121                 System.out.println("JUNCTION 8");
122                 end = String.valueOf(nums[k]);
123                 System.out.println("HERE");
124                 System.out.println("3Writing Standalone: " + end);
125                 sm.add(end);
126                 //System.out.println("6Writing range: " + start + "-> " + end);
127                 start=String.valueOf(nums[k+1]);
128                 System.out.println("4Writing Standalone: " + start);
129                 sm.add(start);
130             }
131         }
```

```
7Writing range: 9-> 8
[5->3, 3->6, 9->8]
```

This is now looking very functional...

The next logical extension would be to place the last two numbers as ascending.

TEST CASE: Switching last two numbers as ascending.....

```
//Merging both ascending and descending code
int[] nums = new int[]{5,4,3,4,5,6,8,9};
```

```
[5->3, 3->6, 8->9]
```

So now we know if number at `nums.length()-2` is the start of the counter.. It

will always perform a merge UNLESS if both numbers are the same.....  
I will quickly give this a try.....

TEST CASE:

```
int[] nums = new int[]{5,4,3,4,5,6,8,8};  
[5->3, 3->6, 8, 8]
```

It is all looking good, so now I think I am in a position where I can be more explorative:

TEST CASE:

```
int[] nums = new int[]{5,4,3,4,5,7,6,10,14,17,19,23,24,25,26,35,34,47};
```

It has started ok, but then it has failed 7->6

35->34

```
[5->3, 3->5, 7, 6, 10, 14, 17, 19, 23->26, 35, 34, 47]
```

I am now going to explain less, but use the same troubleshooting techniques

It can be seen that it has only performed two standalone single write into the list....

```

CHECKING: 7 with 6
JUNCTION 6
-----NEW JUNCTION 18
7
6
-----NEW JUNCTION 19
JUNCTION 11
JUNCTION 13
6Writing Standalone: 7
value k: 6
length nums: 18
value of counter: 0
CHECKING: 6 with 10
JUNCTION 6
-----NEW JUNCTION 18
6
10
JUNCTION 10
*****0
JUNCTION 11
JUNCTION 13
6Writing Standalone: 6
value k: 7
length nums: 18
value of counter: 0
CHECKING: 10 with 14

```

I expected it to have taken action to write 7->6 once it had performed 6->10

We have to remember in above test cases which passed, I focussed on the descending scenarios at the start and at end of the array.

```
int[] nums = new int[]{5,4,3,4,5,6,9,8};|
```

```
[5->3, 3->6, 9, 8]
```

Now I am presented with 7->6 and 35-> 34 which occurs between mid way and before `nums.length-2`

Logic above tells me to look at Junction 13. I do not believe I had taken remediation coding here before....

I believe I need to perform a check here for descendance:

```
192
193 //if k!=0
194 else
195 {
196     System.out.println("JUNCTION 13");
197     start = String.valueOf(nums[k]);
198     sm.add(start);
199     System.out.println("6Writing Standalone: " + start);
200 }
```

I applied this code:

```
195 //if k!=0 && k!=nums.length-2)
196 else
197 {
198     if ((nums[k]-1)==(nums[k+1]))
199     {
200         System.out.println("-----NEW JUNCTION 21");
201         start = String.valueOf(nums[k]);
202     }
203     else
204     {
205         System.out.println("JUNCTION 13");
206         start = String.valueOf(nums[k]);
207         sm.add(start);
208         System.out.println("6Writing Standalone: " + start);
209     }
210 }
```

But it has had a massive impact on the outcome, we can see that it has removed numbers off the output altogether (7,35)

```
int[] nums = new int[]{5,4,3,4,5,7,6,10,14,17,19,23,24,25,26,35,34,47};
[5->3, 3->5, 6, 10, 14, 17, 19, 23->26, 34, 47]
```

For the moment, I am rolling back this change....

### TEST CASE:

I desperately need to understand what is really going on in the else statement on line 196...

I will need to add a written summary on top.....

I can safely write the following:

k!=0

k!=nums.length-2

counter==0

not consecutive number (nums[k+1] ascending by 1 from nums[k]

**It gives me every reason to believe I should be able to implement logic in here.**

This only gives me one option without sabotaging my entire code, I am forced to write the start->end and await the outcome.

I have now performed the following operation:

```
if ((nums[k]-1)==(nums[k+1]))
{
    //System.out.println("-----NEW JUNCTION 21");
    start = String.valueOf(nums[k]);
    end = String.valueOf(nums[k+1]);
    sm.add(start+"->" + end);
}
```

```
[5->3, 3->5, 7->6, 6, 10, 14, 17, 19, 23->26, 35->34, 34, 47]
```

I can see there are still standalone being written (6) and (34) upon closing the range...

Before I try to remediate this, since my approach above was improvised, I will try to modify the array so that it extends the descending summary range.

TEST CASE: Using similar array but extending the descending sequence

```
int[] nums = new int[]{5,4,3,4,5,7,6,5,10,14,17,19,23,24,25,26,35,34,33,47};
```

[5->3, 3->5, 7->6, 6->5, 5, 10, 14, 17, 19, 23->26, 35->34, 34->33, 33, 47]

It can be seen that my choice of writing the start and end was not the correct choice AT ALL in Junction 21... Perhaps I can avoid writing the values. Instead I can just obtain new start and end values..... This is of no use...

Ultimately I do not want the logic to check for any values already written in the list since these are taken to be final...

So I have chosen to store this range (7->6) and (35->34) in a temp variable..  
I then have to see the flow in code which performs (34->33)

My mindset suggests I will need to keep updating the temp variable until it is ready to write the correct descending range...

I am really unsure how else to remediate...

TEST CASE: Creating temp variable to store value and also ascertain when it has to finalise the write into the list

```
-----NEW JUNCTION 21  
8Storing range: 7-> 6  
value k: 6  
length nums: 20  
value of counter: 0  
ÂÊÃÄÅÆÇÈÉÊËÌÍÎÏÐÑRADIOCHECKING: 6 with 5  
JUNCTION 6  
-----NEW JUNCTION 18  
6  
5  
-----NEW JUNCTION 19  
JUNCTION 11  
-----NEW JUNCTION 21  
8Storing range: 6-> 5  
value k: 7
```

Note it is in the exact same junction 21....

We can see that the start and end get overwritten. The start will become 6 and

This is total wrong...

If there is none... Then:

It is difficult to know end, since there could be another descending after 5.

TEST CASE: Running with above logic implemented....

```
8Storing range: 7->6->5->
value k: 7
length nums: 20
value of counter: 0
CHECKING: 5 with 10
```

-----NEW JUNCTION 18

10

\*\*\*\*\*9

## JUNCTION 13

6Writing Standalone: 5

```
value k: 8
```

```
length nums: 20
```

```
value of counter: 0
```

CHECKING: 10 with 14

## JUNCTION 6

-----NEW JUNCTION 18

10

14

## JUNCTION 10

I have implemented the following code, and it appears to have given correct output:

```

227         else
228         {
229             if (temp!="")
230             {
231                 sm.add(temp);
232                 System.out.println("8Writing range: " + temp);
233                 temp="";
234             }
235             else
236             {
237                 System.out.println("JUNCTION 13");
238                 start = String.valueOf(nums[k]);
239                 sm.add(start);
240                 System.out.println("6Writing Standalone: " + start);
241             }
242         }
243     }

```

For now, I will just accept that the entire range is written out in full. But main worry is the disappearance of the following (see underlined red below)...

[5->3, 3->5, 7->6->5, 10, 14, 17, 19, 23->26, 33, 47]

I will quickly check in Junction 8 to ascertain if there is a value in temp and if there is, ensure it is written into the list.....

```

8Storing range: 35->34->33
value k: 18
length nums: 20
value of counter: 0
CHECKING: 33 with 47
JUNCTION 6
JUNCTION 7
counter:0-----LAST TO LAST ITEM :33
JUNCTION 8
HERE
3Writing Standalone: 33
4Writing Standalone: 47

```



I have now added this code...

```
23         else
24         {
25             if (temp!="")
26             {
27                 sm.add(temp);
28                 System.out.println("9Writing range: " + temp);
29             }
30
31         System.out.println("JUNCTION 8");
```

Output is as follows, I am getting extremely close to the finish line now... However this is the only real issue, so I will identify where the standalone number is written...

```
[5->3, 3->5, 7->6->5, 10, 14, 17, 19, 23->26, 35->34->33, 33, 47]
```

I have added the following code:

```
System.out.println("JUNCTION 8");
end = String.valueOf(nums[k]);
System.out.println("HERE");

if (temp=="")
{
    System.out.println("3Writing Standalone: " + end);
    sm.add(end);
}

//System.out.println("6Writing range: " + start + "-> " + end);
start=String.valueOf(nums[k+1]);
System.out.println("4Writing Standalone: " + start);
sm.add(start);
}
```

Now before I run exhaustive tests, my only concern is to tidy up how temp is growing since the summary range should be in notation X->Y  
I have performed this by tidying up code in lots of area...  
I am now ready to run an exhaustive test...

TEST CASE:

This is almost a perfect execution

```
int[] nums = new int[]{5,6,7,6,5,9,10,11,44,45,46,47,46,45,44,43,25,24,24,23,23,22};
```

[5->6, 6->5, 9->11, 44->46, 46->43, 25->24, 24->23, 23->22]

However we can see that it has dropped the 7 and it has also dropped the 47

I am hoping these can be identified quite quickly...

```
int[] nums = new int[]{5,6,7,6,5,9,10,11,44,45,46,47,46,45,44,43,25,24,24,23,23,22};
```

CHECKING: 7 with 6

## JUNCTION 6

-----NEW JUNCTION 18

7

6

-----NEW JUNCTION 19

## JUNCTION 14

2Writing range: 5→ 6

```
value k: 3
```

```
length nums: 22
```

```
value of counter: 0
```

CHECKING: 6 with 5

I have first quickly changed the 7 to a 9, and the output captures the standalone 9....

**So the issue is clearly with it interfering with the ascendancy and descendancy transition ONLY**

I have had to implement logic to identify if there is a transition number, it will clearly be shown in my code.....

This will ensure that

```
int[] nums = new int[]{5,6,7,6,5,9,10,11,44,45,46,47,46,45,44,43,25,24,24,23,23,22};
```

5->7 and 7->5 is captured and

44->47    and   47->44

#### TEST CASE:

I have made a significant amount of changes and including this as new code.  
called transition number..

It appears it has captured all the transitions and lost no data....

```
[5->7, 7->5, 9->11, 44->47, 47->43, 25->24, 24->23, 23->22]
```

Only way forward now is to include more standalone numbers in between and try this a bit more, mixing up the ascending and descending.....

#### TEST CASE:

```
int[] nums = new int[]{5,9,13,2,3,4,3,1,2,3,4,5,6,22,66,65,65,64,63,90,89,60};
```

It has made a mess of the situation, there are several ranges missing and standalone numbers missing

```
[5, 9, 13, 2->4, 3, 22, 4->65, 65->63, 90->89, 60]
```

I am going to comment out bits of my new code (in relation to transitionNumber) and see the outcome...

```
[5, 9, 13, 2->4, 2->4, 3, 1->6, 22, 66->65, 65->63, 90->89, 60]
```

We can see it is much closer.... So I need to take another approach... the errors are highlighted below.....

```
[5, 9, 13, 2->4, 2->4, 3, 1->6, 22, 66->65, 65->63, 90->89, 60]
```

I think I need to take another re-visit and identify the standalone storage for 3 and determine if sufficient validation to perform 4->3

I believe my code is fairly robust and I didn't think I really required all the additional logic, so I glad I had opportunity to comment out logic in relation to transition number.....

I can also aim to address the multiple storage for 2->4

I have roll back slightly and re-applied the logic again...

I am taking my code through ALL test cases in my code so far...

I also need label all the junctions in my code, I will NOT change any junction numbers labelled so far (otherwise it will make it difficult to follow my documentation so far), I will increment from the current highest.....

\*\*\*\*\*THESE ARE ASCENDING TEST CASES\*\*\*\*\*

TEST CASE:

```
int[] nums = new int[]{0,2,3,4,6,8,9};
```

Junctions 38,2,6,18,10,11,24,12,3,5,34,36,14,25,30,32,13,4

```
[0, 2->4, 6, 8->9]
```

TEST CASE:

```
int[] nums = new int[]{-1};
```

Junctions 1

```
[-1]
```

TEST CASE:

```
int[] nums = new int[]{0};
```

Junctions 1

```
[0]
```

TEST CASE:

```
int[] nums = new int[]{0,1,2,4,5,7};
```

Junctions 2,38,3,5,6,18,10,34,36,14,7,9

```
[0->2, 4->5, 7]
```

\*\*\*\*\*THESE ARE DESCENDING TEST CASES (REVERSE OF ABOVE\*\*\*\*\*

TEST CASE:

```
int[] nums = new int[]{9,8,6,4,3,2,0};
```

Junctions

2,6,18,19,11,24,17,6,18,10,34,36,14,25,30,32,13,25,21,26,28,29,7,20a,222,23,  
8

```
[9->8, 6, 4->2, 0]
```

TEST CASE:

```
int[] nums = new int[]{-1};
```

Junctions: 1

```
[-1]
```

TEST CASE:

```
int[] nums = new int[]{0};
```

Junctions: 1

```
[0]
```

TEST CASE: FAIL

It can be seen that it has missed out number 2 which should be standalone. I will quickly analyse the area of code.

```
int[] nums = new int[]{7,5,4,2,1,0};
```

Junctions: 2,6,18,10,11,24,12,18,19,25,21,26,28,25,30,31,7,20a,20

```
[7, 5->4, 1->0]
```

This all seems ok

```

CHECKING: 4 with 2
JUNCTION 6
-----NEW JUNCTION 18
4
2
JUNCTION 10
*****0
JUNCTION 11
-----NEW JUNCTION 25
-----NEW JUNCTION 30
-----NEW JUNCTION 31
8Writing range: 5->4
value k: 3
length nums: 6
```

It can be seen that it has stored the range in temp, but it has not had opportunity to write this before.... 1->0

I will follow all the junctions between these two points (highlighted in yellow)...

```

CHECKING: 2 with 1
JUNCTION 6
-----NEW JUNCTION 18
2
1
-----NEW JUNCTION 19
JUNCTION 11
-----NEW JUNCTION 25
-----NEW JUNCTION 21
-----NEW JUNCTION 26
-----NEW JUNCTION 28
*****2Value in temp: 2->1
8Storing range: 2->1
value k: 4
length nums: 6
value of counter: 0
CHECKING: 1 with 0
JUNCTION 6
JUNCTION 7
counter:0-----LAST TO LAST ITEM :1
-----NEW JUNCTION 20a
-----NEW JUNCTION 20
7Writing range: 1-> 0

```

It can be seen that in this area of code (JUNCTION 6, 7, 20a, 20), there is currently no provision to check status of temp and write that range in... This has to be the priority before writing 1->0

```

System.out.println("JUNCTION 6");
if (k==nums.length-2)
{
    System.out.println("JUNCTION 7");
    System.out.println("counter:" + counter + "-----LAST TO LAST ITEM : " + nu

    if (counter==0)
    {
        System.out.println("-----NEW JUNCTION 20a");

        if ((nums[k]-1)==(nums[k+1]))
        {
            System.out.println("-----NEW JUNCTION 20");
            start=String.valueOf(nums[k]);
            end = String.valueOf(nums[k+1]);
            sm.add(start+"->" + end);
            System.out.println("7Writing range: " + start + "-> " + end);
        }
    }
}

```

My instinct suggests that this code (if statement below), is not really an association for the else of the if above, so I am going to move it above if counter==0

```

//if the next number is ascending
else
{
    System.out.println("-----NEW JUNCTION 22");
    if (temp!="")
    {
        System.out.println("-----NEW JUNCTION 23");
        sm.add(temp);
        System.out.println("9Writing range: " + temp);
    }
}

```

TEST CASE: Performed again with code adjustments: PASS

```
int[] nums = new int[]{7,5,4,2,1,0};
```

```
[7, 5->4, 2->1, 1->0]
```



TEST CASE: I will quickly run through test cases above again to ensure no disruptions... Also please note that Junction 22 has now been removed as a result of this move..

NOTE: THESE ARE ALL IN MY FINAL CODE

\*\*\*\* These were developed during my early testing to ensure it passed through Junction 8

TEST CASE:

```
int[] nums = new int[]{0,1,2,4,5,7,8,10,14};
```

Junctions 2,38,5,3,6,18,10,34,36,14,34,36,7,20a,8,24

```
[0->2, 4->5, 7->8, 10, 14]
```

TEST CASE:

```
int[] nums = new int[]{0,2,4,5,7,8,10,14,17,19,23,24,25,26,35,42,43,44};
```

Junctions.....

```
[0, 2, 4->5, 7->8, 10, 14, 17, 19, 23->26, 35, 42->44]
```

TEST CASE:

```
int[] nums = new int[]{5,4,3,4,5,7,6,10,14,17,19,23,24,25,26,35,42,47,60,61};
```

Junctions.....

```
[5->3, 3->5, 7->6, 10, 14, 17, 19, 23->26, 35, 42, 47, 60->61]
```

TEST CASE:

```
int[] nums = new int[]{7,6,9,3,5,8};
```

Junctions:.....

```
[7->6, 9, 3, 5, 8]
```

//Merging both ascending and descending code, there were used massively during this test documentation....

TEST CASE:

```
int[] nums = new int[]{5,4,3,4,5,6,8,8};
```

Junctions:.....

```
[5->3, 3->6, 8, 8]
```

TEST CASE:

```
int[] nums = new int[]{5,4,3,4,5,7,6,5,10,14,17,19,23,24,25,26,35,34,33,47};
```

Junctions:.....

```
[5->3, 3->5, 7->5, 10, 14, 17, 19, 23->26, 35->33, 47]
```

TEST CASE:

```
int[] nums = new int[]{5,6,7,6,5,9,10,11,44,45,46,47,46,45,44,43,25,24,24,23,23,22};
```

Junctions:.....

```
[5->7, 7->5, 9->11, 44->47, 47->43, 25->24, 24->23, 23->22]
```

TEST CASE:

```
int[] nums = new int[]{5,9,13,2,3,4,3,1,2,3,4,5,6,22,66,65,65,64,63,90,89,60};
```

Junctions: .....

```
[5, 9, 13, 2->4, 4->3, 1->6, 22, 66->65, 65->63, 90->89, 60]
```

TEST CASE:

```
//repeat numbers
```

```
int[] nums = new int[]{2,2,2,4,4,1,1,9,9};
```

```
[2, 2, 2, 4, 4, 1, 1, 9, 9]
```