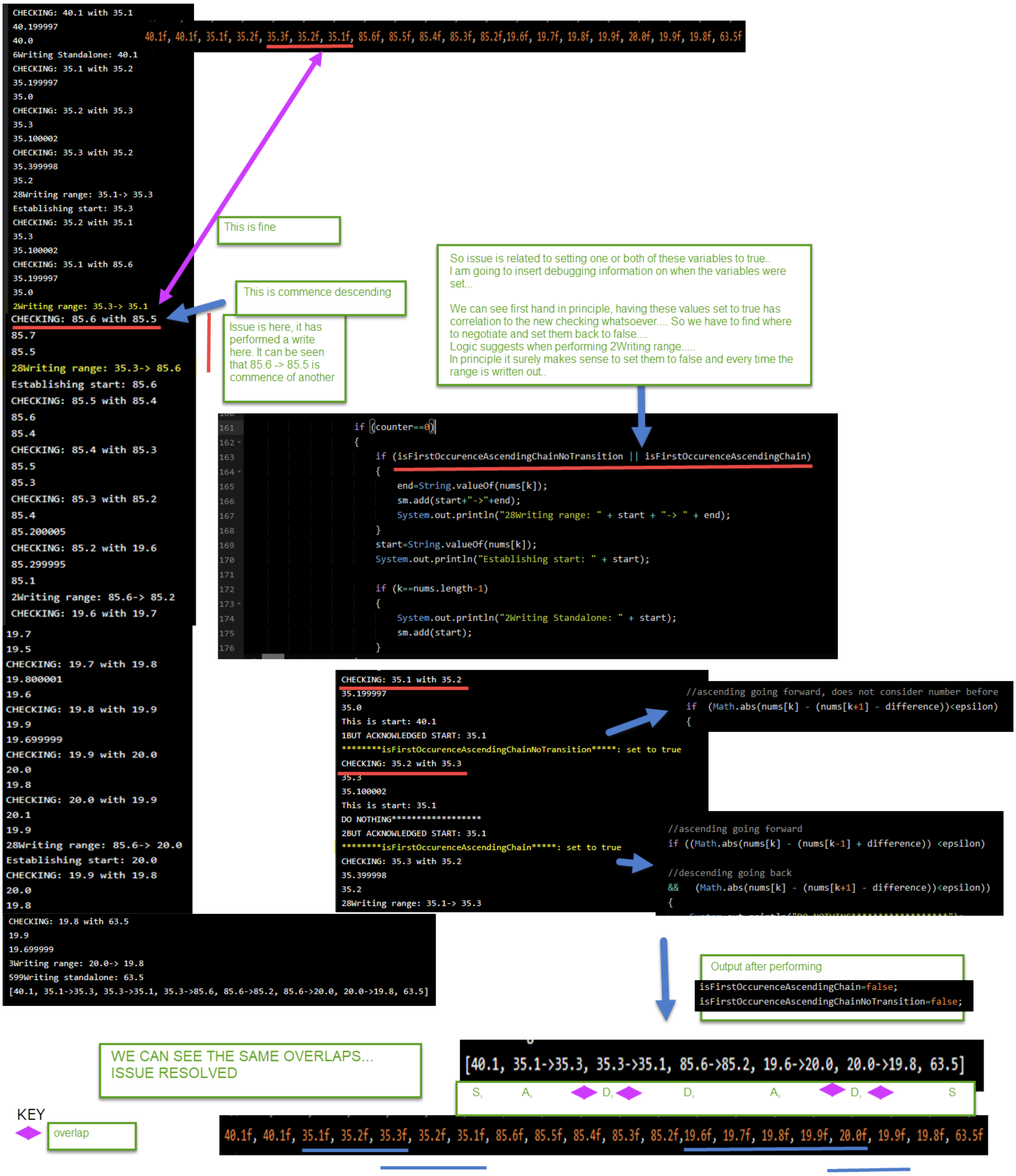
Further extension to new chatGPT data set

Inline with the failed execution of data (as per excel document: 30032025/SummaryRange/5/4/Output.xlsx

<u>TEST CASE: Due to failed test case such as below, I revisited logic in my code</u>



```
40.1f, 40.1f, 35.1f, 35.2f, 35.3f, 35.2f, 35.1f, 85.6f, 85.5f, 85.4f, 85.3f, 85.2f,19.6f, 19.7f, 19.8f, 19.9f, 20.0f, 19.9f, 19.8f, 63.5f
```

```
CHECKING: 40.1 with 35.1
40.199997
40.0
6Writing Standalone: 40.1
CHECKING: 35.1 with 35.2
35.199997
35.0
CHECKING: 35.2 with 35.3
35.3
35.100002
CHECKING: 35.3 with 35.2
35.399998
35.2
28Writing range: 35.1-> 35.3
Establishing start: 35.3
CHECKING: 35.2 with 35.1
35.3
35.100002
CHECKING: 35.1 with 85.6
35.199997
35.0
2Writing range: 35.3-> 35.1
CHECKING: 85.6 with 85.5
85.7
85.5
28Writing range: 35.3-> 85.6
Establishing start: 85.6
CHECKING: 85.5 with 85.4
85.6
85.4
CHECKING: 85.4 with 85.3
85.5
85.3
CHECKING: 85.3 with 85.2
85.4
85.200005
CHECKING: 85.2 with 19.6
85.299995
85.1
2Writing range: 85.6-> 85.2
CHECKING: 19.6 with 19.7
19.7
19.5
CHECKING: 19.7 with 19.8
19.800001
19.6
CHECKING: 19.8 with 19.9
19.9
19.699999
CHECKING: 19.9 with 20.0
20.0
19.8
CHECKING: 20.0 with 19.9
20.1
19.9
28Writing range: 85.6-> 20.0
Establishing start: 20.0
CHECKING: 19.9 with 19.8
20.0
19.8
CHECKING: 19.8 with 63.5
19.9
19.699999
3Writing range: 20.0-> 19.8
599Writing standalone: 63.5
[40.1, 35.1->35.3, 35.3->35.1, 35.3->85.6, 85.6->85.2, 85.6->20.0, 20.0->19.8, 63.5]
```

This is fine

This is commence descending

Issue is here, it has performed a write here. It can be seen that 85.6 -> 85.5 is commence of another

So issue is related to setting one or both of these variables to true..
I am going to insert debugging information on when the variables were set...

We can see first hand in principle, having these values set to true has correlation to the new checking whatsoever.... So we have to find where to negotiate and set them back to false....
Logic suggests when performing 2Writing range.....
In principle it surely makes sense to set them to false and every time the range is written out..

```java
if (counter==0)
{
    if (isFirstOccurenceAscendingChainNoTransition || isFirstOccurenceAscendingChain)
    {
        end=String.valueOf(nums[k]);
        sm.add(start+"->"+end);
        System.out.println("28Writing range: " + start + "-> " + end);
    }
    start=String.valueOf(nums[k]);
    System.out.println("Establishing start: " + start);

    if (k==nums.length-1)
    {
        System.out.println("2Writing Standalone: " + start);
        sm.add(start);
    }
```

```
CHECKING: 35.1 with 35.2
35.199997
35.0
This is start: 40.1
1BUT ACKNOWLEDGED START: 35.1
********isFirstOccurenceAscendingChainNoTransition*****: set to true
CHECKING: 35.2 with 35.3
35.3
35.100002
This is start: 35.1
DO NOTHING******************
2BUT ACKNOWLEDGED START: 35.1
********isFirstOccurenceAscendingChain*****: set to true
CHECKING: 35.3 with 35.2
35.399998
35.2
28Writing range: 35.1-> 35.3
```

```java
//ascending going forward, does not consider number before
if  (Math.abs(nums[k] - (nums[k+1] - difference))<epsilon)
{
```

```java
//ascending going forward
if ((Math.abs(nums[k] - (nums[k-1] + difference)) <epsilon)

//descending going back
&&  (Math.abs(nums[k] - (nums[k+1] - difference))<epsilon))
{
```

Output after performing

```java
isFirstOccurenceAscendingChain=false;
isFirstOccurenceAscendingChainNoTransition=false;
```

```
[40.1, 35.1->35.3, 35.3->35.1, 85.6->85.2, 19.6->20.0, 20.0->19.8, 63.5]
```

```
40.1f, 40.1f, 35.1f, 35.2f, 35.3f, 35.2f, 35.1f, 85.6f, 85.5f, 85.4f, 85.3f, 85.2f,19.6f, 19.7f, 19.8f, 19.9f, 20.0f, 19.9f, 19.8f, 63.5f
```

WE CAN SEE THE SAME OVERLAPS...
ISSUE RESOLVED

KEY

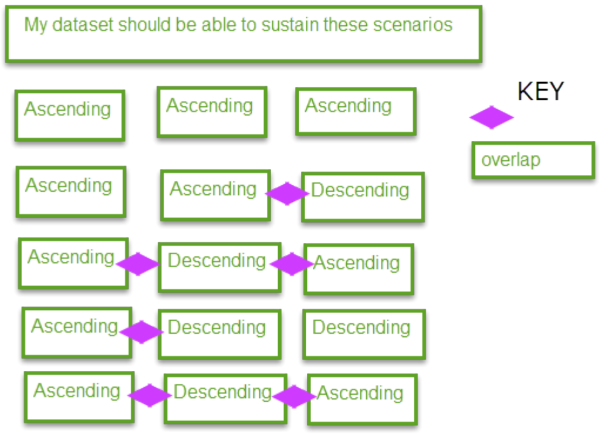
◆ overlap

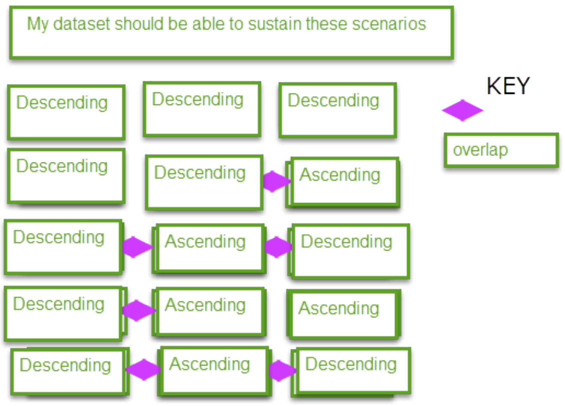
As oppose to make any other changes to my code, I will re-run the code again and populate excel workbook

TEST CASE: Check Excel data (PASS)

Now my only other concern was the following, so I will quickly generate a few personal test cases with these..... And then I will just merge them up in various order and see the outcome...
Since neither of my ChatGPT data consisted of two consecutive ascending or descending, I will put some emphasis on this also...

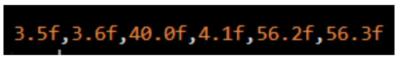
TEST CASES DERIVED FROM HERE:



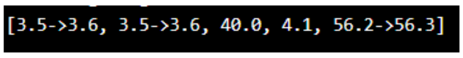


TEST CASE:

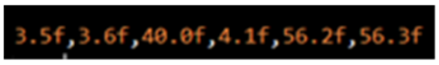
Ascending Ascending Ascending



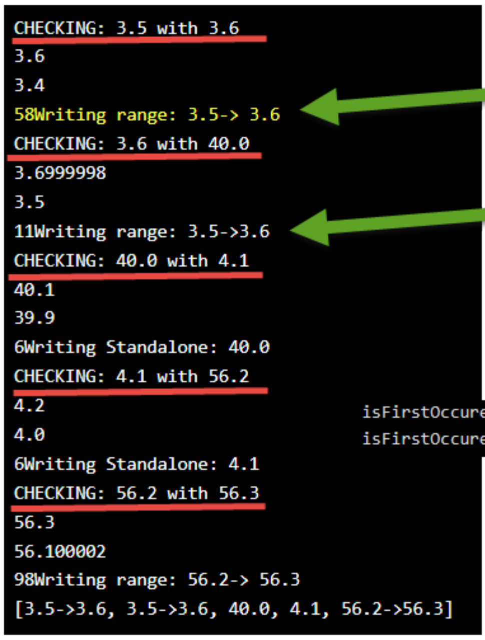
```
3.5f,3.6f,40.0f,4.1f,56.2f,56.3f
```



```
[3.5->3.6, 3.5->3.6, 40.0, 4.1, 56.2->56.3]
```



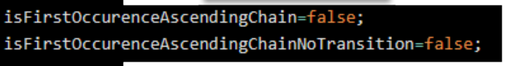
```
3.5f,3.6f,40.0f,4.1f,56.2f,56.3f
```



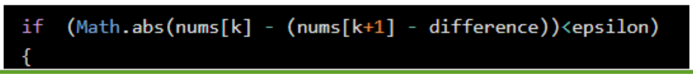
```
CHECKING: 3.5 with 3.6
3.6
3.4
58Writing range: 3.5-> 3.6
CHECKING: 3.6 with 40.0
3.6999998
3.5
11Writing range: 3.5->3.6
CHECKING: 40.0 with 4.1
40.1
39.9
6Writing Standalone: 40.0
CHECKING: 4.1 with 56.2
4.2
4.0
6Writing Standalone: 4.1
CHECKING: 56.2 with 56.3
56.3
56.100002
98Writing range: 56.2-> 56.3
[3.5->3.6, 3.5->3.6, 40.0, 4.1, 56.2->56.3]
```

This is ok

Issue starts here... So I need to contemplate as to whether this is related to the boolean resets or not. I will follow the

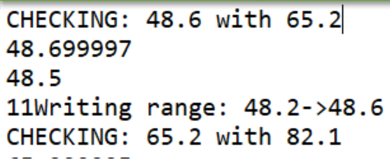


```
isFirstOccurenceAscendingChain=false;
isFirstOccurenceAscendingChainNoTransition=false;
```

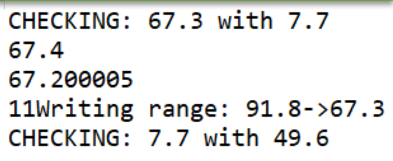


```
if  (Math.abs(nums[k] - (nums[k+1] - difference))<epsilon)
{
```

So neither of these evaluate to the above, hence enter else section. It can be seen that difference between nums[k] and nums[k] is not within specified range

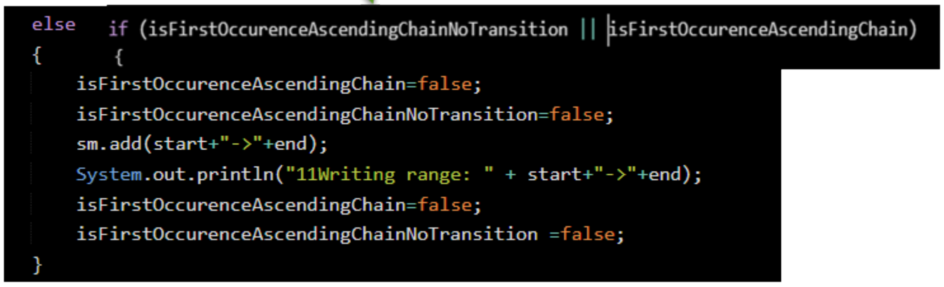


```
CHECKING: 48.6 with 65.2
48.699997
48.5
11Writing range: 48.2->48.6
CHECKING: 65.2 with 82.1
```

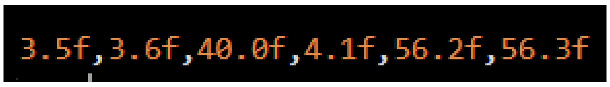


```
CHECKING: 67.3 with 7.7
67.4
67.200005
11Writing range: 91.8->67.3
CHECKING: 7.7 with 49.6
```

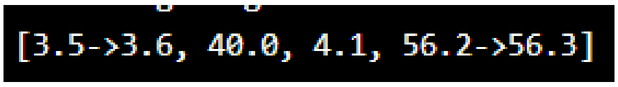
I need to include logic in here that if either of the booleans are set to true, then do not write the range... My instinct tells me that as I go through all the Test cases in the documentation, I will have to tweak most areas where it performs a write



```
else    if (isFirstOccurenceAscendingChainNoTransition || isFirstOccurenceAscendingChain)
{         {
        isFirstOccurenceAscendingChain=false;
        isFirstOccurenceAscendingChainNoTransition=false;
        sm.add(start+"->"+end);
        System.out.println("11Writing range: " + start+"->"+end);
        isFirstOccurenceAscendingChain=false;
        isFirstOccurenceAscendingChainNoTransition =false;
}
```



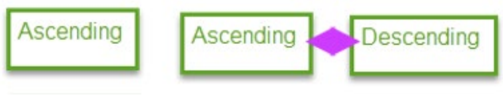
```
3.5f,3.6f,40.0f,4.1f,56.2f,56.3f
```

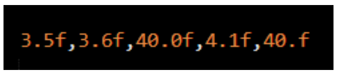


```
[3.5->3.6, 40.0, 4.1, 56.2->56.3]
```

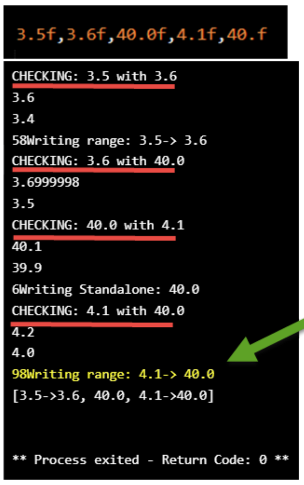
I will now try the next scenario:

TEST CASE:





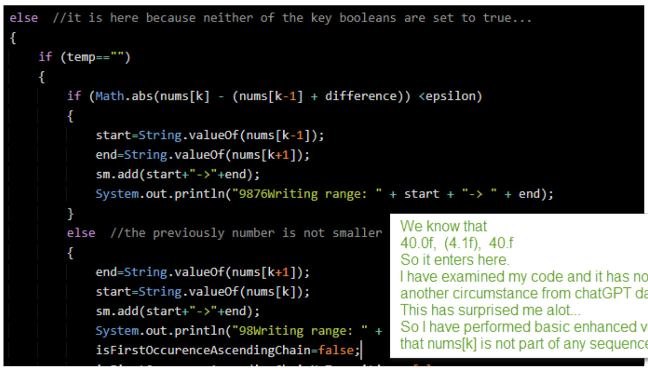
```
3.5f,3.6f,40.0f,4.1f,40.f
```



```
3.5f,3.6f,40.0f,4.1f,40.f
```

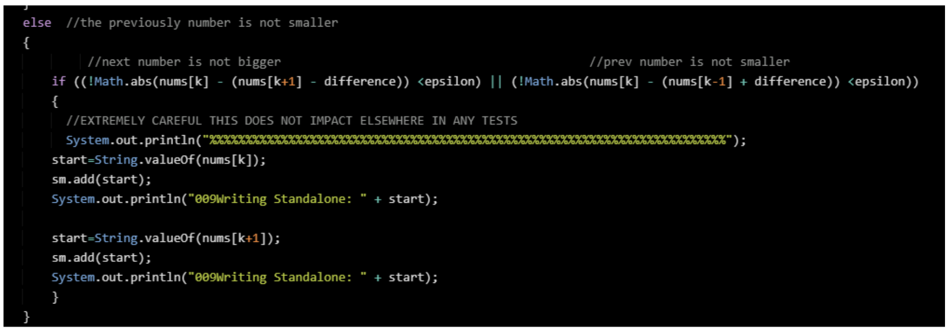
```
CHECKING: 3.5 with 3.6
3.6
3.4
58Writing range: 3.5-> 3.6
CHECKING: 3.6 with 40.0
3.6999998
3.5
CHECKING: 40.0 with 4.1
40.1
39.9
6Writing Standalone: 40.0
CHECKING: 4.1 with 40.0
4.2
4.0
98Writing range: 4.1-> 40.0
[3.5->3.6, 40.0, 4.1->40.0]


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

This is incorrect, I am just going to try analyse the area of code....
We were expecting it to write 40.0 ->40.1..

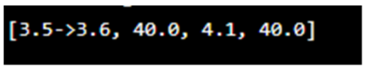


```
else  //it is here because neither of the key booleans are set to true...
{
    if (temp=="")
    {
        if (Math.abs(nums[k] - (nums[k-1] + difference)) <epsilon)
        {
            start=String.valueOf(nums[k-1]);
            end=String.valueOf(nums[k+1]);
            sm.add(start+"->"+end);
            System.out.println("9876Writing range: " + start + "-> " + end);
        }
        else  //the previously number is not smaller
        {
            end=String.valueOf(nums[k+1]);
            start=String.valueOf(nums[k]);
            sm.add(start+"->"+end);
            System.out.println("98Writing range: " +
            isFirstOccurenceAscendingChain=false;
```

We know that
40.0f, (4.1f), 40.f
So it enters here.
I have examined my code and it has not entered here for another circumstance from chatGPT data.,,,,
This has surprised me alot...
So I have performed basic enhanced validation to check that nums[k] is not part of any sequence

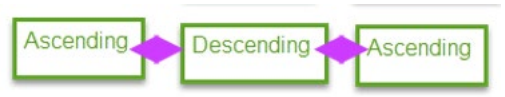


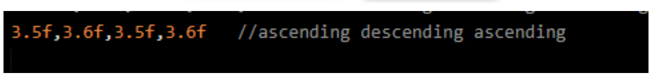
```
else  //the previously number is not smaller
{
        //next number is not bigger                                          //prev number is not smaller
    if ((!Math.abs(nums[k] - (nums[k+1] - difference)) <epsilon) || (!Math.abs(nums[k] - (nums[k-1] + difference)) <epsilon))
    {
      //EXTREMELY CAREFUL THIS DOES NOT IMPACT ELSEWHERE IN ANY TESTS
      System.out.println("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%");
    start=String.valueOf(nums[k]);
    sm.add(start);
    System.out.println("009Writing Standalone: " + start);

    start=String.valueOf(nums[k+1]);
    sm.add(start);
    System.out.println("009Writing Standalone: " + start);
    }
}
```



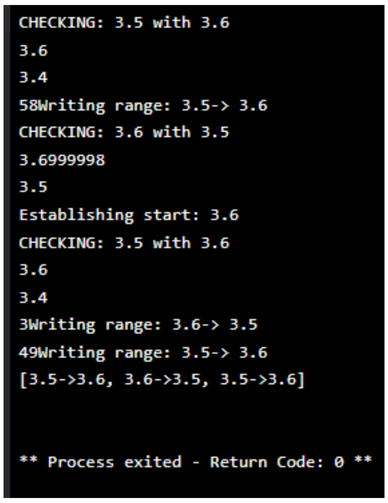
```
[3.5->3.6, 40.0, 4.1, 40.0]
```

TEST CASE:



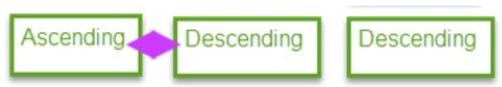


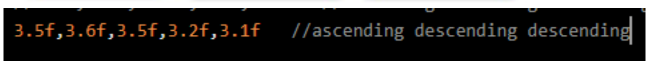
```
3.5f,3.6f,3.5f,3.6f    //ascending descending ascending
```



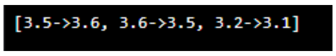
```
CHECKING: 3.5 with 3.6
3.6
3.4
58Writing range: 3.5-> 3.6
CHECKING: 3.6 with 3.5
3.6999998
3.5
Establishing start: 3.6
CHECKING: 3.5 with 3.6
3.6
3.4
3Writing range: 3.6-> 3.5
49Writing range: 3.5-> 3.6
[3.5->3.6, 3.6->3.5, 3.5->3.6]


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

TEST CASE:



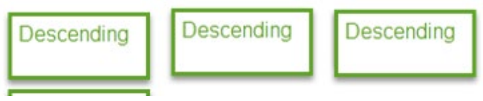


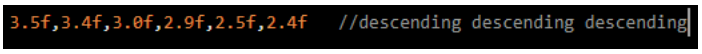
```
3.5f,3.6f,3.5f,3.2f,3.1f    //ascending descending descending
```



```
[3.5->3.6, 3.6->3.5, 3.2->3.1]
```

TEST CASE:

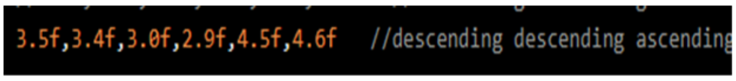




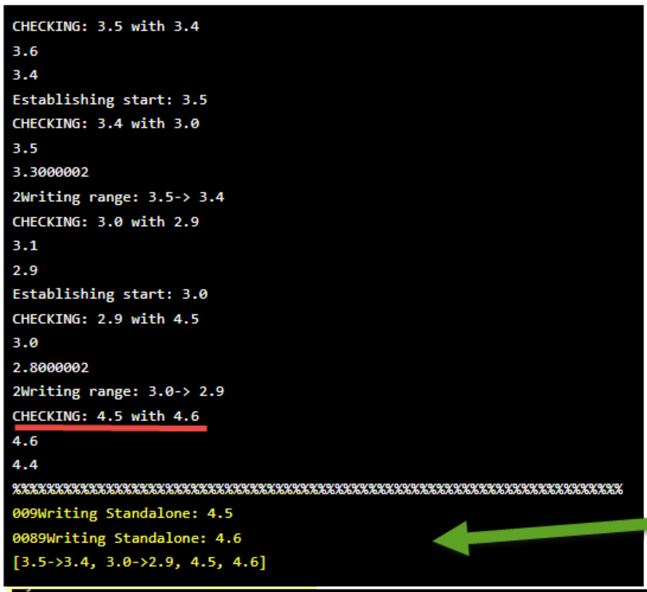
```
3.5f,3.4f,3.0f,2.9f,2.5f,2.4f    //descending descending descending
```

TEST CASE:

Descending
Descending
Ascending



```
3.5f,3.4f,3.0f,2.9f,4.5f,4.6f   //descending descending ascending
```



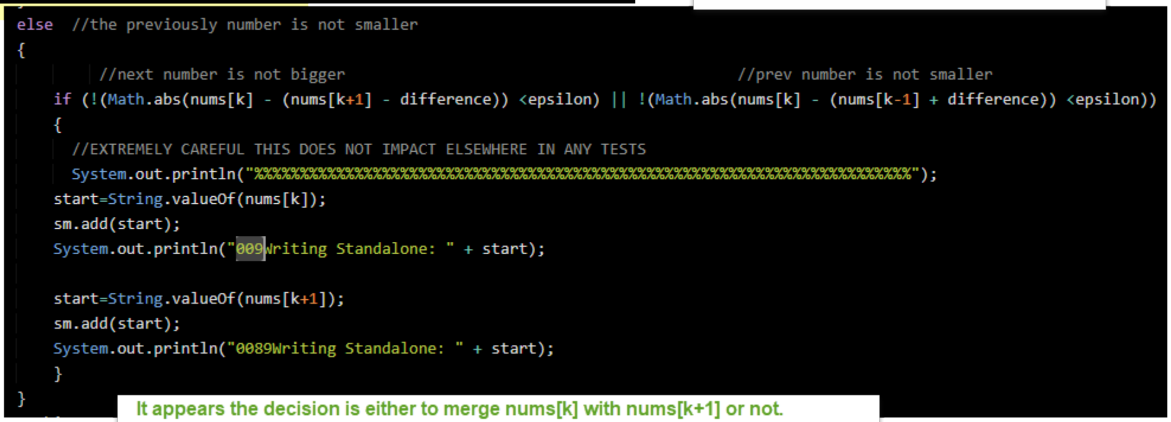
```
CHECKING: 3.5 with 3.4
3.6
3.4
Establishing start: 3.5
CHECKING: 3.4 with 3.0
3.5
3.3000002
2Writing range: 3.5-> 3.4
CHECKING: 3.0 with 2.9
3.1
2.9
Establishing start: 3.0
CHECKING: 2.9 with 4.5
3.0
2.8000002
2Writing range: 3.0-> 2.9
CHECKING: 4.5 with 4.6
4.6
4.4
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
009Writing Standalone: 4.5
0089Writing Standalone: 4.6
[3.5->3.4, 3.0->2.9, 4.5, 4.6]
```

This is the code that I implemented earlier as part of

We decided as can be seen below that if nums[k] is not part of a sequence with previous and number after, to perform standalone write of nums[k] and nums[k+1]

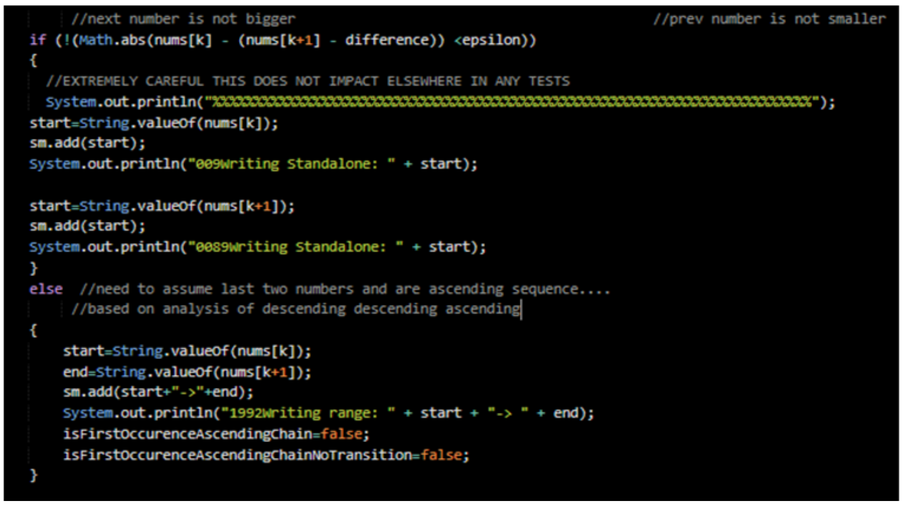
We can see 2.9f, (4.5f) validates this loop, so it writes 4.5, 4.6

So we need to adjust this logic... It is still not clear the logic to implement....
BUT WE KNOW IT REACHES HERE ON PENULTIMATE NUMBER

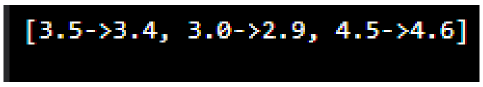


```
else  //the previously number is not smaller
{
    //next number is not bigger                                    //prev number is not smaller
    if (!(Math.abs(nums[k] - (nums[k+1] - difference)) <epsilon) || !(Math.abs(nums[k] - (nums[k-1] + difference)) <epsilon))
    {
      //EXTREMELY CAREFUL THIS DOES NOT IMPACT ELSEWHERE IN ANY TESTS
      System.out.println("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%");
    start=String.valueOf(nums[k]);
    sm.add(start);
    System.out.println("009Writing Standalone: " + start);

    start=String.valueOf(nums[k+1]);
    sm.add(start);
    System.out.println("0089Writing Standalone: " + start);
    }
}
```

It appears the decision is either to merge nums[k] with nums[k+1] or not.
Since the above shows 2.9f should not have any influence on processing last two numbers...
i have changed the loop to as follows:

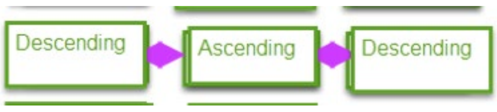


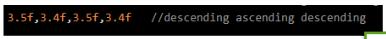
```
    //next number is not bigger                                    //prev number is not smaller
if (!(Math.abs(nums[k] - (nums[k+1] - difference)) <epsilon))
{
  //EXTREMELY CAREFUL THIS DOES NOT IMPACT ELSEWHERE IN ANY TESTS
  System.out.println("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%");
start=String.valueOf(nums[k]);
sm.add(start);
System.out.println("009Writing Standalone: " + start);

start=String.valueOf(nums[k+1]);
sm.add(start);
System.out.println("0089Writing Standalone: " + start);
}
else  //need to assume last two numbers and are ascending sequence....
    //based on analysis of descending descending ascending
{
    start=String.valueOf(nums[k]);
    end=String.valueOf(nums[k+1]);
    sm.add(start+"->"+end);
    System.out.println("1992Writing range: " + start + "-> " + end);
    isFirstOccurenceAscendingChain=false;
    isFirstOccurenceAscendingChainNoTransition=false;
}
```



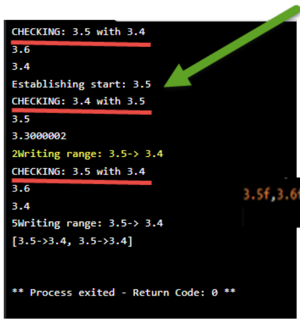
```
[3.5->3.4, 3.0->2.9, 4.5->4.6]
```

TEST CASE:





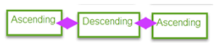
```
3.5f,3.4f,3.5f,3.4f   //descending ascending descending
```



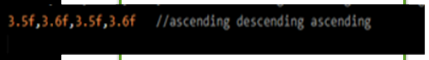
```
CHECKING: 3.5 with 3.4
3.6
3.4
Establishing start: 3.5
CHECKING: 3.4 with 3.5
3.5
3.3000002
2Writing range: 3.5-> 3.4
CHECKING: 3.5 with 3.4
3.6
3.4
5Writing range: 3.5-> 3.4
[3.5->3.4, 3.5->3.4]


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

It has failed to write range 3.5 -> 3.4

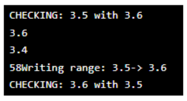
in principle it is a reflection of this scenario, so i would expect mirror logic is required that supported the first transition here



I am going to quickly revisit



```
3.5f,3.6f,3.5f,3.6f   //ascending descending ascending
```



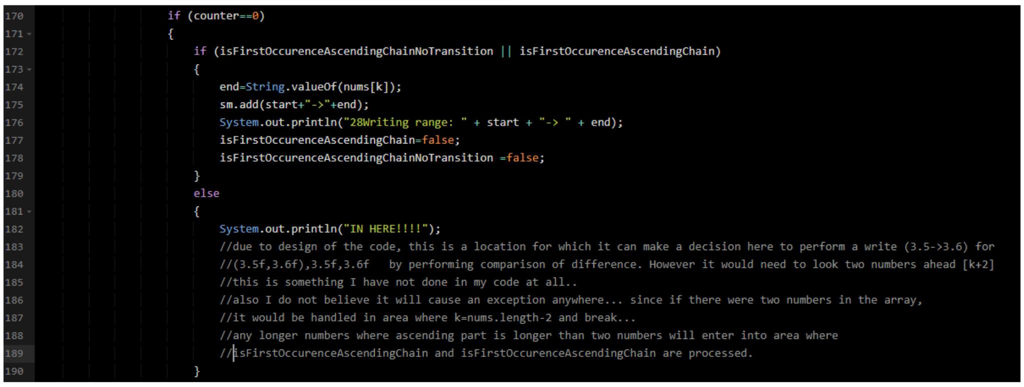
```
CHECKING: 3.5 with 3.6
3.6
3.4
58Writing range: 3.5-> 3.6
CHECKING: 3.6 with 3.5
```

It has entered 58writing range....

I have a feeling the code on the right has been nowhere near since it establishes start and processes k+1

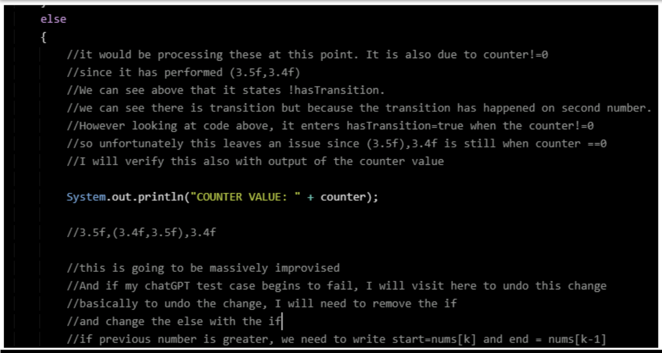
So I will need to investigate logic right from start of CHECKING 3.5 with 3.4

I am massively caught in two minds in where to handle this scenario...

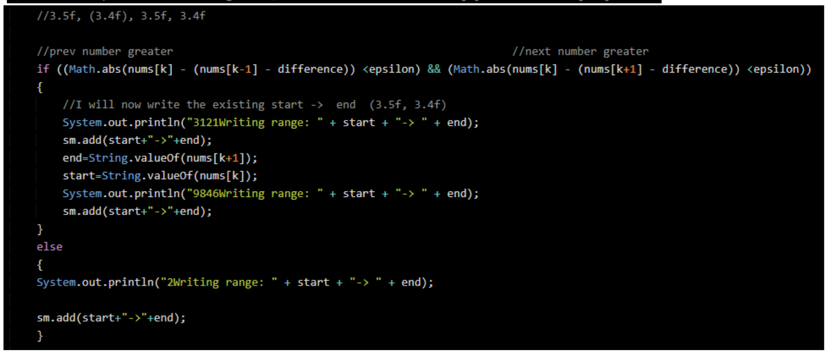


```
if (counter==0)
{
    if (isFirstOccurenceAscendingChainNoTransition || isFirstOccurenceAscendingChain)
    {
        end=String.valueOf(nums[k]);
        sm.add(start+"->"+end);
        System.out.println("28Writing range: " + start + "-> " + end);
        isFirstOccurenceAscendingChain=false;
        isFirstOccurenceAscendingChainNoTransition =false;
    }
    else
    {
        System.out.println("IN HERE!!!!");
        //due to design of the code, this is a location for which it can make a decision here to perform a write (3.5->3.6) for
        //(3.5f,3.6f),3.5f,3.6f   by performing comparison of difference. However it would need to look two numbers ahead [k+2]
        //this is something I have not done in my code at all..
        //also I do not believe it will cause an exception anywhere... since if there were two numbers in the array,
        //it would be handled in area where k=nums.length-2 and break...
        //any longer numbers where ascending part is longer than two numbers will enter into area where
        //isFirstOccurenceAscendingChain and isFirstOccurenceAscendingChain are processed.
    }
```

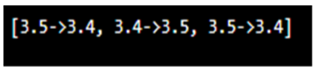
The other option is inline with where it performs 2writing range. I need to be extremely careful since this is a heavily reached area in the code..
Basically we are here since first two numbers are descending sequence and then we are moving into ascending sequence. Performed the safe bet would be to hardwire something to state if k==2 make a certain decision. This way it will not interfere with the existing code in here.....
But at same time, it might be worth examining which area the code would reach if created a scenario in which descending sequence (two numbers) occurs at later point in code following by overlapping ascending



```
else
{
    //it would be processing these at this point. It is also due to counter!=0
    //since it has performed (3.5f,3.4f)
    //We can see above that it states !hasTransition.
    //we can see there is transition but because the transition has happened on second number.
    //However looking at code above, it enters hasTransition=true when the counter!=0
    //so unfortunately this leaves an issue since (3.5f),3.4f is still when counter ==0
    //I will verify this also with output of the counter value

    System.out.println("COUNTER VALUE: " + counter);

    //3.5f,(3.4f,3.5f),3.4f

    //this is going to be massively improvised
    //And if my chatGPT test case begins to fail, I will visit here to undo this change
    //basically to undo the change, I will need to remove the if
    //and change the else with the if
    //if previous number is greater, we need to write start=nums[k] and end = nums[k-1]
```



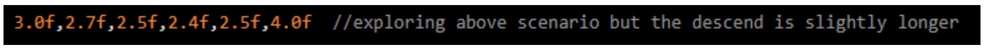
```
//3.5f, (3.4f), 3.5f, 3.4f

//prev number greater                                                    //next number greater
if ((Math.abs(nums[k] - (nums[k-1] - difference)) <epsilon) && (Math.abs(nums[k] - (nums[k+1] - difference)) <epsilon))
{
    //I will now write the existing start ->  end  (3.5f, 3.4f)
    System.out.println("3121Writing range: " + start + "-> " + end);
    sm.add(start+"->"+end);
    end=String.valueOf(nums[k+1]);
    start=String.valueOf(nums[k]);
    System.out.println("9846Writing range: " + start + "-> " + end);
    sm.add(start+"->"+end);
}
else
{
System.out.println("2Writing range: " + start + "-> " + end);

sm.add(start+"->"+end);
}
```



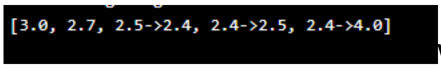
```
[3.5->3.4, 3.4->3.5, 3.5->3.4]
```

Due to the above extremely awkward adjustment, I have now created a test case below which delays the inception of the two digit descending which moves into ascending and then descending.

TEST CASE:



```
3.0f,2.7f,2.5f,2.4f,2.5f,4.0f  //exploring above scenario but the descend is slightly longer
```

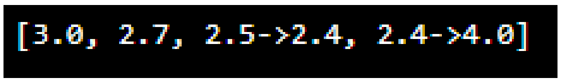


```
[3.0, 2.7, 2.5->2.4, 2.4->2.5, 2.4->4.0]
```

We can see things have not gone to plan..

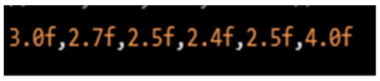
So I want to quickly undo the change I did above... It will give me idea straight away of the root cause

Undoing change of previous test case... We are worse off so I have re-instated the logic again...

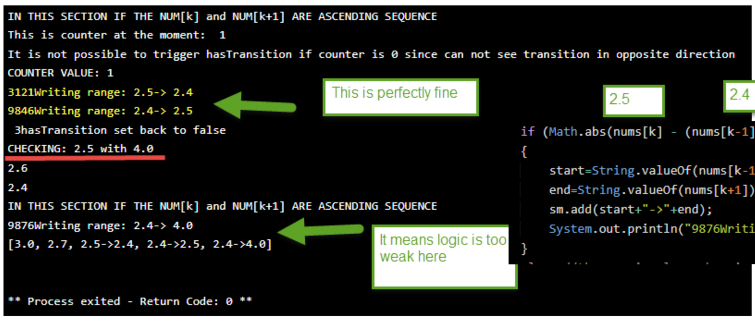


```
[3.0, 2.7, 2.5->2.4, 2.4->4.0]
```

Although it looks improvised, I still think it’s a controlled change given unique circumstances



```
3.0f,2.7f,2.5f,2.4f,2.5f,4.0f
```



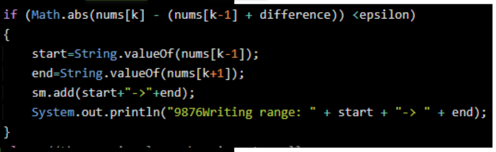
```
IN THIS SECTION IF THE NUM[k] and NUM[k+1] ARE ASCENDING SEQUENCE
This is counter at the moment:  1
It is not possible to trigger hasTransition if counter is 0 since can not see transition in opposite direction
COUNTER VALUE: 1
3121Writing range: 2.5-> 2.4
9846Writing range: 2.4-> 2.5
 3hasTransition set back to false
CHECKING: 2.5 with 4.0
2.6
2.4
IN THIS SECTION IF THE NUM[k] and NUM[k+1] ARE ASCENDING SEQUENCE
9876Writing range: 2.4-> 4.0
[3.0, 2.7, 2.5->2.4, 2.4->2.5, 2.4->4.0]


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

This is perfectly fine

It means logic is too weak here

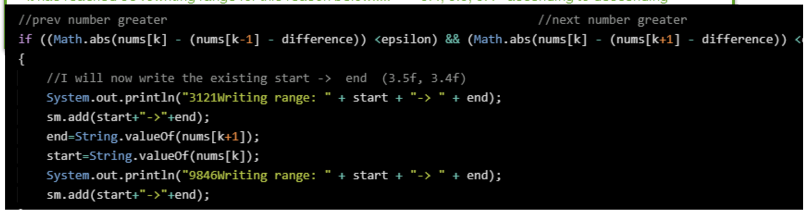
2.5

2.4



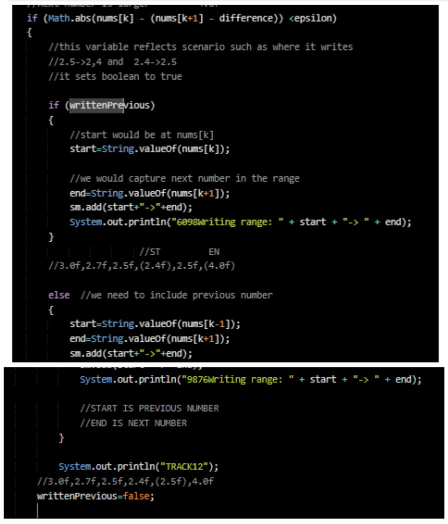
```
if (Math.abs(nums[k] - (nums[k-1] + difference)) <epsilon)
{
    start=String.valueOf(nums[k-1]);
    end=String.valueOf(nums[k+1]);
    sm.add(start+"->"+end);
    System.out.println("9876Writing range: " + start + "-> " + end);
}
```

I will track back through my test cases to ascertain when I introduced this code.. But I had something prompting me to assume that if previous number was smaller, then I would need to configure start -> end for nums[k-1] -> nums[k]
THERE WILL BE NO LOGICAL CODE IN HERE UNTIL I UNDERSTAND THIS CASE FIRST.
FORTUNATELY it has not reached here for any numbers in chatGPT data, so I know I have exclusive rights to get correct solution.....(without affecting my test case in this document)....
The biggest doubt is to whether to set start to nums[k-1], which I know is currently wrong
or nums[k]
It has reached 9846writing range for this reason below.....   3.4, 3.5, 3.4   ascending to descending



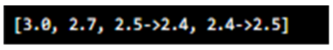
```
//prev number greater                                              //next number greater
if ((Math.abs(nums[k] - (nums[k-1] - difference)) <epsilon) && (Math.abs(nums[k] - (nums[k+1] - difference)) <
{
    //I will now write the existing start ->  end  (3.5f, 3.4f)
    System.out.println("3121Writing range: " + start + "-> " + end);
    sm.add(start+"->"+end);
    end=String.valueOf(nums[k+1]);
    start=String.valueOf(nums[k]);
    System.out.println("9846Writing range: " + start + "-> " + end);
    sm.add(start+"->"+end);
```

I set up this code so that it identifies if a boolean is set when it performs 9846writing. This will ensure previous number is not included again



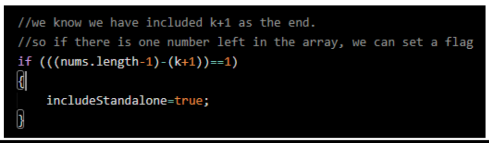
```
if (Math.abs(nums[k] - (nums[k+1] - difference)) <epsilon)
{
    //this variable reflects scenario such as where it writes
    //2.5->2,4 and  2.4->2.5
    //it sets boolean to true

    if (writtenPrevious)
    {
        //start would be at nums[k]
        start=String.valueOf(nums[k]);

        //we would capture next number in the range
        end=String.valueOf(nums[k+1]);
        sm.add(start+"->"+end);
        System.out.println("6098Writing range: " + start + "-> " + end);
    }
                     //ST         EN
    //3.0f,2.7f,2.5f,(2.4f),2.5f,(4.0f)

    else  //we need to include previous number
    {
        start=String.valueOf(nums[k-1]);
        end=String.valueOf(nums[k+1]);
        sm.add(start+"->"+end);
          System.out.println("9876Writing range: " + start + "-> " + end);

          //START IS PREVIOUS NUMBER
          //END IS NEXT NUMBER
      }

      System.out.println("TRACK12");
    //3.0f,2.7f,2.5f,2.4f,(2.5f),4.0f
    writtenPrevious=false;
```

It is almost perfect but it dropped off the standalone number at end 4.0f. I believe the way to achieve this is to create an associated else statement.. But there is already one configured...

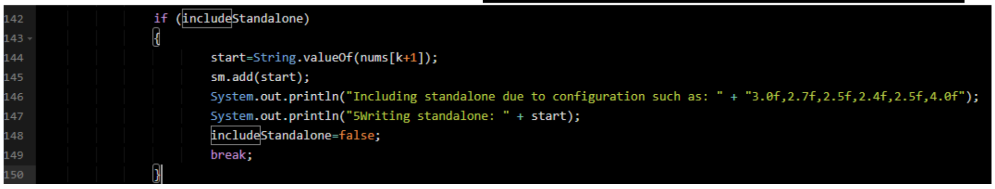


```
[3.0, 2.7, 2.5->2.4, 2.4->2.5]
```

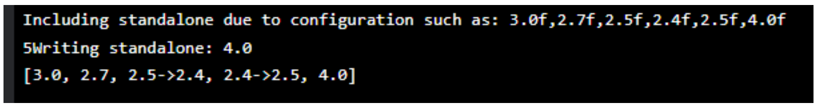
Change made added code



```
//we know we have included k+1 as the end.
//so if there is one number left in the array, we can set a flag
if (((nums.length-1)-(k+1))==1)
{
    includeStandalone=true;
}
```

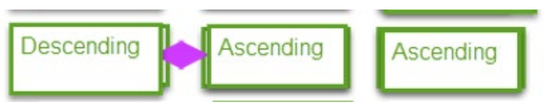


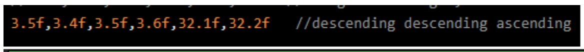
```
if (includeStandalone)
{
    start=String.valueOf(nums[k+1]);
    sm.add(start);
    System.out.println("Including standalone due to configuration such as: " + "3.0f,2.7f,2.5f,2.4f,2.5f,4.0f");
    System.out.println("5Writing standalone: " + start);
    includeStandalone=false;
    break;
}
```



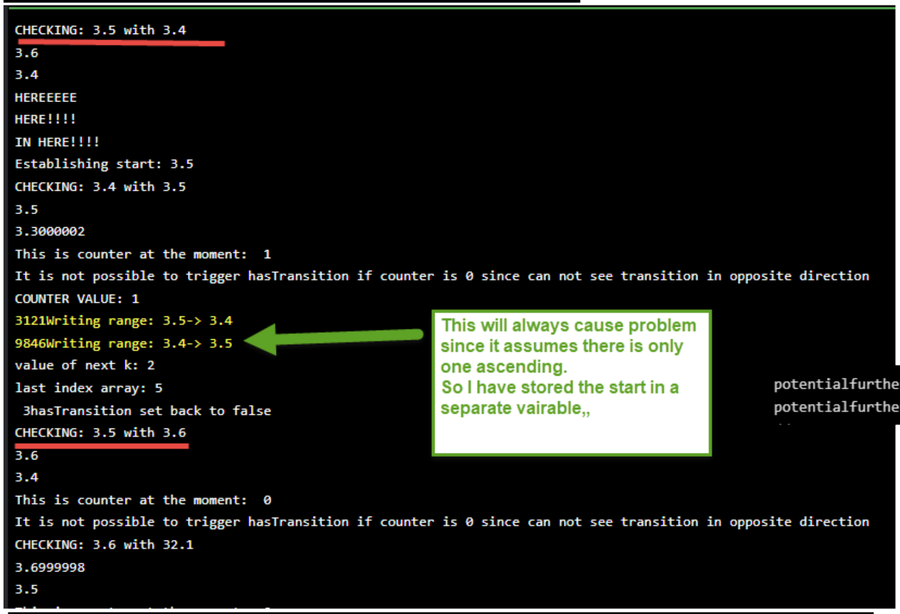
```
Including standalone due to configuration such as: 3.0f,2.7f,2.5f,2.4f,2.5f,4.0f
5Writing standalone: 4.0
[3.0, 2.7, 2.5->2.4, 2.4->2.5, 4.0]
```

TEST CASE:



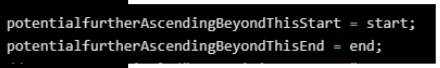


```
3.5f,3.4f,3.5f,3.6f,32.1f,32.2f    //descending descending ascending
```

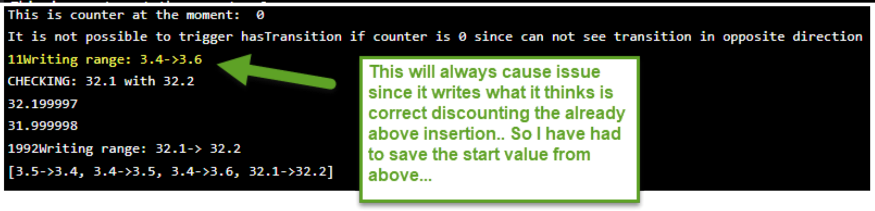


```
CHECKING: 3.5 with 3.4
3.6
3.4
HEREEEEE
HERE!!!!
IN HERE!!!!
Establishing start: 3.5
CHECKING: 3.4 with 3.5
3.5
3.3000002
This is counter at the moment:  1
It is not possible to trigger hasTransition if counter is 0 since can not see transition in opposite direction
COUNTER VALUE: 1
3121Writing range: 3.5-> 3.4
9846Writing range: 3.4-> 3.5
value of next k: 2
last index array: 5
 3hasTransition set back to false
CHECKING: 3.5 with 3.6
3.6
3.4
This is counter at the moment:  0
It is not possible to trigger hasTransition if counter is 0 since can not see transition in opposite direction
CHECKING: 3.6 with 32.1
3.6999998
3.5
```

This will always cause problem since it assumes there is only one ascending.
So I have stored the start in a separate vairable,,



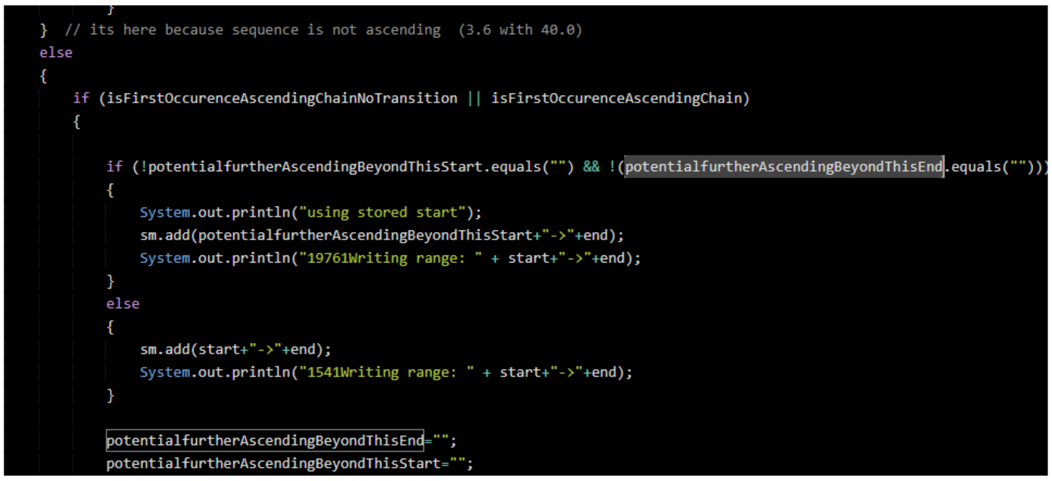
```
potentialfurtherAscendingBeyondThisStart = start;
potentialfurtherAscendingBeyondThisEnd = end;
```



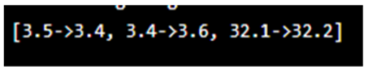
```
This is counter at the moment:  0
It is not possible to trigger hasTransition if counter is 0 since can not see transition in opposite direction
11Writing range: 3.4->3.6
CHECKING: 32.1 with 32.2
32.199997
31.999998
1992Writing range: 32.1-> 32.2
[3.5->3.4, 3.4->3.5, 3.4->3.6, 32.1->32.2]
```

This will always cause issue since it writes what it thinks is correct discounting the already above insertion.. So I have had to save the start value from above...

And if there is a circumstance like this that the ascending streak continues.... it will get stored variable start... And we know the end it would calculate correctly



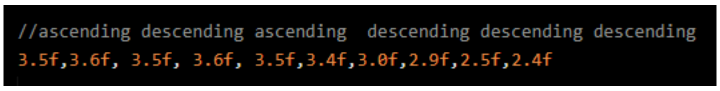
```
}  // its here because sequence is not ascending  (3.6 with 40.0)
else
{
    if (isFirstOccurenceAscendingChainNoTransition || isFirstOccurenceAscendingChain)
    {

        if (!potentialfurtherAscendingBeyondThisStart.equals("") && !(potentialfurtherAscendingBeyondThisEnd.equals("")))
        {
            System.out.println("using stored start");
            sm.add(potentialfurtherAscendingBeyondThisStart+"->"+end);
            System.out.println("19761Writing range: " + start+"->"+end);
        }
        else
        {
            sm.add(start+"->"+end);
            System.out.println("1541Writing range: " + start+"->"+end);
        }

        potentialfurtherAscendingBeyondThisEnd="";
        potentialfurtherAscendingBeyondThisStart="";
```



```
[3.5->3.4, 3.4->3.6, 32.1->32.2]
```

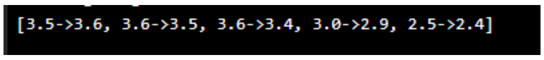
I have now gone through all those diagrammatic interpretation of ascending and descending..
I will just merge a few now quickly and see the outcome.

TEST CASE:



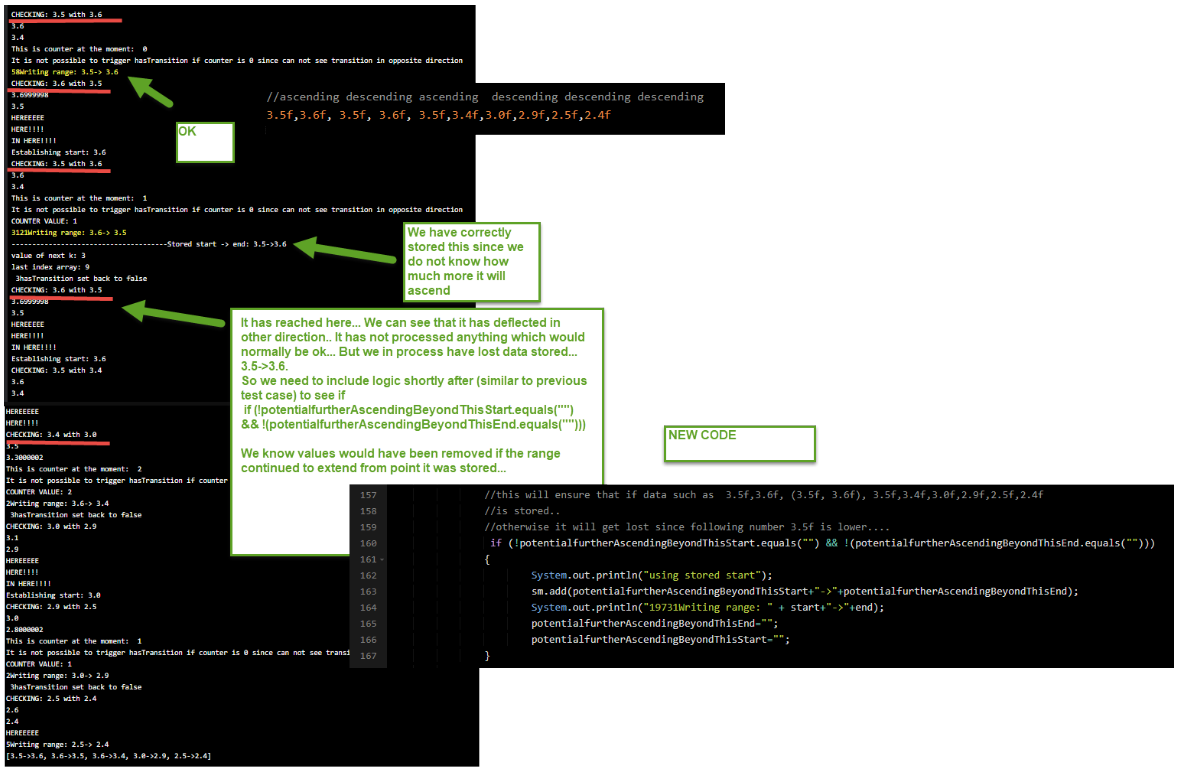
```
//ascending descending ascending  descending descending descending
3.5f,3.6f, 3.5f, 3.6f, 3.5f,3.4f,3.0f,2.9f,2.5f,2.4f
```

We can see it has missing   3.5->3.6 representing the third transition.



```
[3.5->3.6, 3.6->3.5, 3.6->3.4, 3.0->2.9, 2.5->2.4]
```

This is undoubtedly related to my code change above...



```
CHECKING: 3.5 with 3.6
3.6
3.4
This is counter at the moment:  0
It is not possible to trigger hasTransition if counter is 0 since can not see transition in opposite direction
58Writing range: 3.5-> 3.6
CHECKING: 3.6 with 3.5
3.6999998
3.5
HEREEEEE
HERE!!!!
IN HERE!!!!
Establishing start: 3.6
CHECKING: 3.5 with 3.6
3.6
3.4
This is counter at the moment:  1
It is not possible to trigger hasTransition if counter is 0 since can not see transition in opposite direction
COUNTER VALUE: 1
3121Writing range: 3.6-> 3.5
-------------------------------------Stored start -> end: 3.5->3.6
value of next k: 3
last index array: 9
 3hasTransition set back to false
CHECKING: 3.6 with 3.5
3.6999998
3.5
HEREEEEE
HERE!!!!
IN HERE!!!!
Establishing start: 3.6
CHECKING: 3.5 with 3.4
3.6
3.4
HEREEEEE
HERE!!!!
CHECKING: 3.4 with 3.0
3.5
3.3000002
This is counter at the moment:  2
It is not possible to trigger hasTransition if counter
COUNTER VALUE: 2
2Writing range: 3.6-> 3.4
 3hasTransition set back to false
CHECKING: 3.0 with 2.9
3.1
2.9
HEREEEEE
HERE!!!!
IN HERE!!!!
Establishing start: 3.0
CHECKING: 2.9 with 2.5
3.0
2.8000002
This is counter at the moment:  1
It is not possible to trigger hasTransition if counter is 0 since can not see transi
COUNTER VALUE: 1
2Writing range: 3.0-> 2.9
 3hasTransition set back to false
CHECKING: 2.5 with 2.4
2.6
2.4
HEREEEEE
5Writing range: 2.5-> 2.4
[3.5->3.6, 3.6->3.5, 3.6->3.4, 3.0->2.9, 2.5->2.4]
```

```
//ascending descending ascending  descending descending descending
3.5f,3.6f, 3.5f, 3.6f, 3.5f,3.4f,3.0f,2.9f,2.5f,2.4f
```

OK

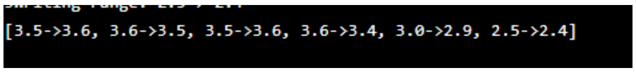
We have correctly stored this since we do not know how much more it will ascend

It has reached here... We can see that it has deflected in other direction.. It has not processed anything which would normally be ok... But we in process have lost data stored... 3.5->3.6.
So we need to include logic shortly after (similar to previous test case) to see if
if (!potentialfurtherAscendingBeyondThisStart.equals("") && !(potentialfurtherAscendingBeyondThisEnd.equals("")))

We know values would have been removed if the range continued to extend from point it was stored...

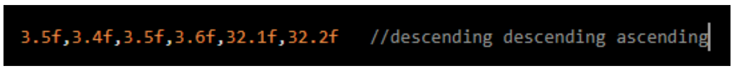
NEW CODE

```
				//this will ensure that if data such as  3.5f,3.6f, (3.5f, 3.6f), 3.5f,3.4f,3.0f,2.9f,2.5f,2.4f
				//is stored..
				//otherwise it will get lost since following number 3.5f is lower....
				 if (!potentialfurtherAscendingBeyondThisStart.equals("") && !(potentialfurtherAscendingBeyondThisEnd.equals("")))
				{
					System.out.println("using stored start");
					sm.add(potentialfurtherAscendingBeyondThisStart+"->"+potentialfurtherAscendingBeyondThisEnd);
					System.out.println("19731Writing range: " + start+"->"+end);
					potentialfurtherAscendingBeyondThisEnd="";
					potentialfurtherAscendingBeyondThisStart="";
				}
```

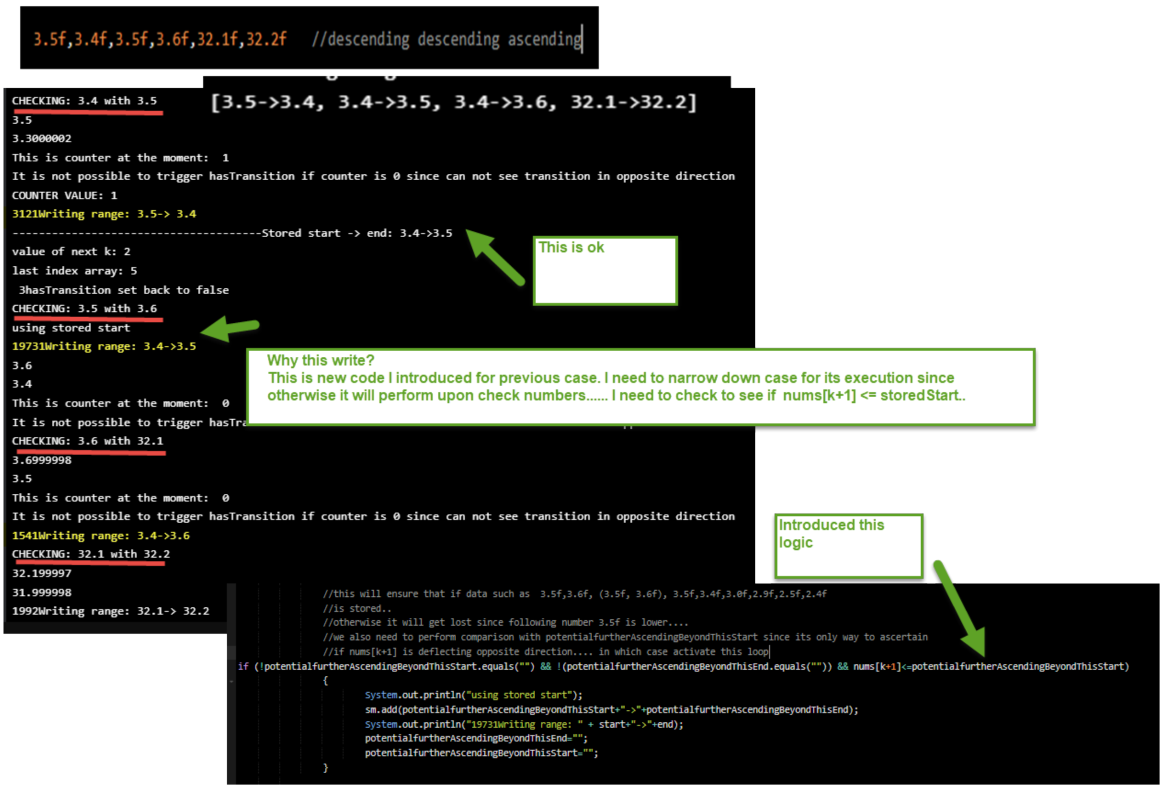


```
[3.5->3.6, 3.6->3.5, 3.5->3.6, 3.6->3.4, 3.0->2.9, 2.5->2.4]
```

TEST CASE:
I think its sensible to try the sample test cases with overlaps..
If satisfied, I can then try similar style..



```
3.5f,3.4f,3.5f,3.6f,32.1f,32.2f    //descending descending ascending
```



```
3.5f,3.4f,3.5f,3.6f,32.1f,32.2f   //descending descending ascending
```

```
[3.5->3.4, 3.4->3.5, 3.4->3.6, 32.1->32.2]
```

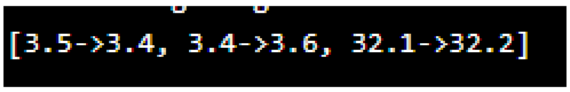
```
CHECKING: 3.4 with 3.5
3.5
3.3000002
This is counter at the moment:  1
It is not possible to trigger hasTransition if counter is 0 since can not see transition in opposite direction
COUNTER VALUE: 1
3121Writing range: 3.5-> 3.4
------------------------------------Stored start -> end: 3.4->3.5
value of next k: 2
last index array: 5
 3hasTransition set back to false
CHECKING: 3.5 with 3.6
using stored start
19731Writing range: 3.4->3.5
3.6
3.4
This is counter at the moment:  0
It is not possible to trigger hasTra
CHECKING: 3.6 with 32.1
3.6999998
3.5
This is counter at the moment:  0
It is not possible to trigger hasTransition if counter is 0 since can not see transition in opposite direction
1541Writing range: 3.4->3.6
CHECKING: 32.1 with 32.2
32.199997
31.999998
1992Writing range: 32.1-> 32.2
```

This is ok

Why this write?
This is new code I introduced for previous case. I need to narrow down case for its execution since otherwise it will perform upon check numbers...... I need to check to see if nums[k+1] <= storedStart..

Introduced this logic

```
                //this will ensure that if data such as  3.5f,3.6f, (3.5f, 3.6f), 3.5f,3.4f,3.0f,2.9f,2.5f,2.4f
                //is stored..
                //otherwise it will get lost since following number 3.5f is lower....
                //we also need to perform comparison with potentialfurtherAscendingBeyondThisStart since its only way to ascertain
                //if nums[k+1] is deflecting opposite direction.... in which case activate this loop
if (!potentialfurtherAscendingBeyondThisStart.equals("") && !(potentialfurtherAscendingBeyondThisEnd.equals("")) && nums[k+1]<=potentialfurtherAscendingBeyondThisStart)
                {
                        System.out.println("using stored start");
                        sm.add(potentialfurtherAscendingBeyondThisStart+"->"+potentialfurtherAscendingBeyondThisEnd);
                        System.out.println("19731Writing range: " + start+"->"+end);
                        potentialfurtherAscendingBeyondThisEnd="";
                        potentialfurtherAscendingBeyondThisStart="";
                }
```

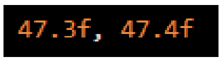


```
[3.5->3.4, 3.4->3.6, 32.1->32.2]
```

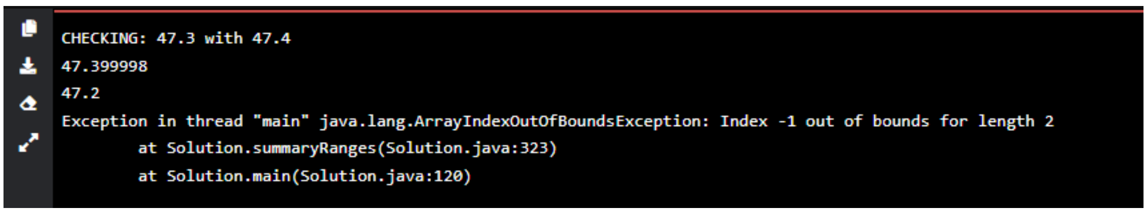
I am going through all my test cases right from the top of declarations...
These are all scenarios related to 0.1f
I will document failed instances and aim to fix it

<u>TEST CASE</u>:
//it performs 47.3->47.4 , 47.3->47.5
Note this test case has even failed on my most original code which passed with ChatGPT, so its clear this sequence was not even present in my data
//47.3f, 47.4f,47.5f //late fixes in code - FAIL (this is straight forward ascending) - FAIL************
<u>NOW FIXED AS BELOW</u>

I will now quickly try quicker ascending



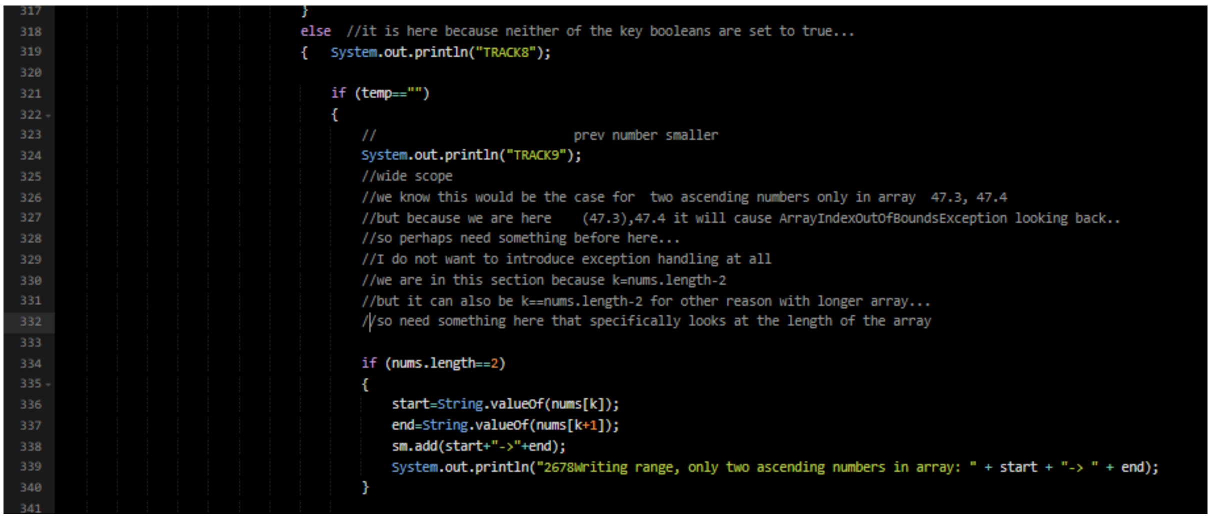
```
47.3f, 47.4f
```



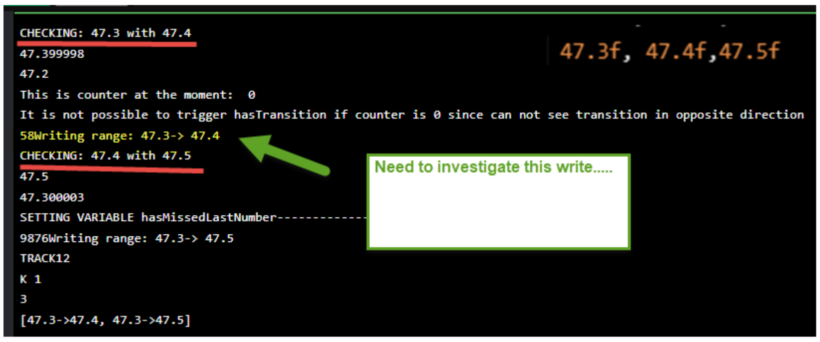
```
CHECKING: 47.3 with 47.4
47.399998
47.2
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index -1 out of bounds for length 2
        at Solution.summaryRanges(Solution.java:323)
        at Solution.main(Solution.java:120)
```

This should hopefully be a straightforward fix

I have accommodated in this section.



```
}
else  //it is here because neither of the key booleans are set to true...
{   System.out.println("TRACK8");

    if (temp=="")
    {
        //                            prev number smaller
        System.out.println("TRACK9");
        //wide scope
        //we know this would be the case for  two ascending numbers only in array  47.3, 47.4
        //but because we are here    (47.3),47.4 it will cause ArrayIndexOutOfBoundsException looking back..
        //so perhaps need something before here...
        //I do not want to introduce exception handling at all
        //we are in this section because k=nums.length-2
        //but it can also be k==nums.length-2 for other reason with longer array...
        //so need something here that specifically looks at the length of the array

        if (nums.length==2)
        {
            start=String.valueOf(nums[k]);
            end=String.valueOf(nums[k+1]);
            sm.add(start+"->"+end);
            System.out.println("2678Writing range, only two ascending numbers in array: " + start + "-> " + end);
        }
```

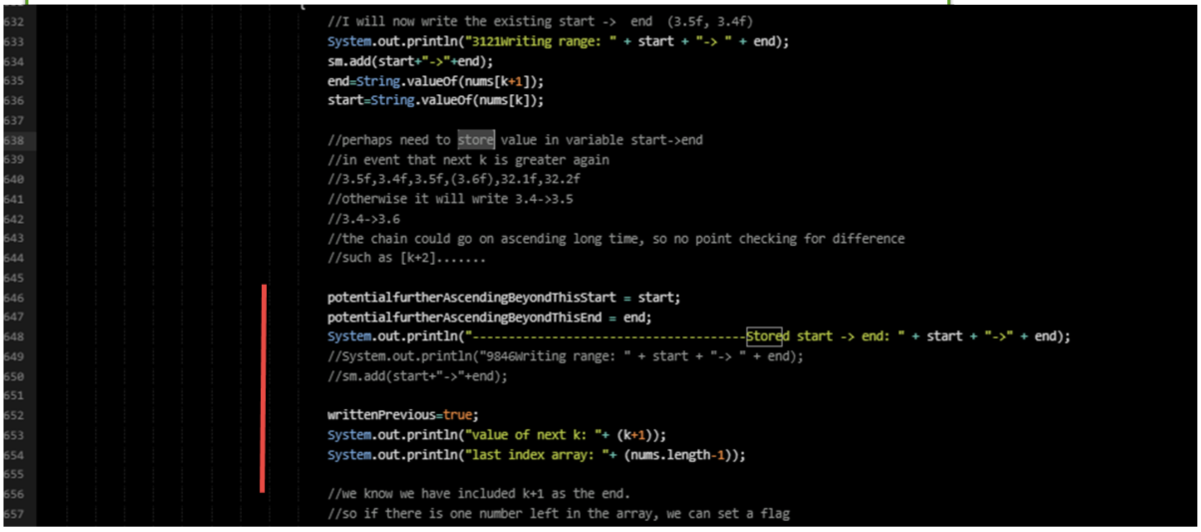


```
CHECKING: 47.3 with 47.4
47.399998
47.2
This is counter at the moment:  0
It is not possible to trigger hasTransition if counter is 0 since can not see transition in opposite direction
58Writing range: 47.3-> 47.4
CHECKING: 47.4 with 47.5
47.5
47.300003
SETTING VARIABLE hasMissedLastNumber------------
9876Writing range: 47.3-> 47.5
TRACK12
K 1
3
[47.3->47.4, 47.3->47.5]
```

```
47.3f, 47.4f,47.5f
```

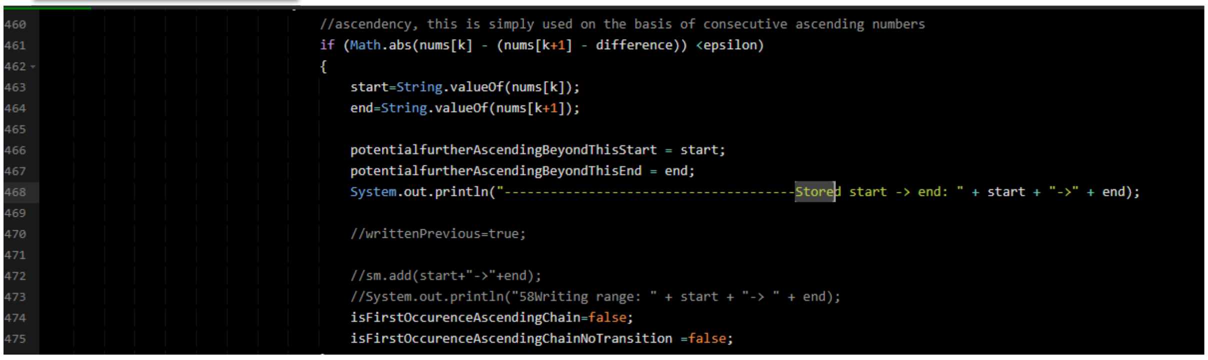
Need to investigate this write.....

I find it extremely odd that one of the most simplest arrangements is now failing... So I will try to resolve this..... Infact having tried this scenario on all my codes, it fails for same issue irrespective of the length of ascending..
I believe I have completed this logic on another area of code... I will just need to make a copy in this section...

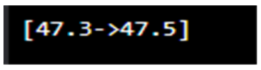


```java
            //I will now write the existing start ->  end  (3.5f, 3.4f)
            System.out.println("3121Writing range: " + start + "-> " + end);
            sm.add(start+"->"+end);
            end=String.valueOf(nums[k+1]);
            start=String.valueOf(nums[k]);

            //perhaps need to store value in variable start->end
            //in event that next k is greater again
            //3.5f,3.4f,3.5f,(3.6f),32.1f,32.2f
            //otherwise it will write 3.4->3.5
            //3.4->3.6
            //the chain could go on ascending long time, so no point checking for difference
            //such as [k+2].......

            potentialfurtherAscendingBeyondThisStart = start;
            potentialfurtherAscendingBeyondThisEnd = end;
            System.out.println("-------------------------------------Stored start -> end: " + start + "->" + end);
            //System.out.println("9846Writing range: " + start + "-> " + end);
            //sm.add(start+"->"+end);

            writtenPrevious=true;
            System.out.println("value of next k: "+ (k+1));
            System.out.println("last index array: "+ (nums.length-1));

            //we know we have included k+1 as the end.
            //so if there is one number left in the array, we can set a flag
```

I have now modified this section and prevented the write.....



```java
        //ascendency, this is simply used on the basis of consecutive ascending numbers
        if (Math.abs(nums[k] - (nums[k+1] - difference)) <epsilon)
        {
            start=String.valueOf(nums[k]);
            end=String.valueOf(nums[k+1]);

            potentialfurtherAscendingBeyondThisStart = start;
            potentialfurtherAscendingBeyondThisEnd = end;
            System.out.println("-------------------------------------Stored start -> end: " + start + "->" + end);

            //writtenPrevious=true;

            //sm.add(start+"->"+end);
            //System.out.println("58Writing range: " + start + "-> " + end);
            isFirstOccurenceAscendingChain=false;
            isFirstOccurenceAscendingChainNoTransition =false;
```



```
[47.3->47.5]
```

It is now fixed, and I have tried several sets of ascending numbers and it has passed

Since this is a critical check, I will run through all my test cases below and see the impact...

//FAILED TO WRITE THE STANDALONE AT END

  //this is second chatGPT extract

30032025/SummaryRange/5/4/ChatGPTgeneratedNumbersBetterDataSet.txt

75.0f, 75.0f, 95.6f, 95.7f, 95.8f, 95.9f, 96.0f, 96.1f, 40.1f, 40.1f

//it has missed out the middle ascending

3.5f,3.6f,40.0f,40.1f,40.f   //ascending ascending descending FAIL

It has not written what it has stored (2.4->2.5) and also 4.0f

3.0f,2.7f,2.5f,2.4f,2.5f,4.0f //exploring above scenario but the descend is slightly longer

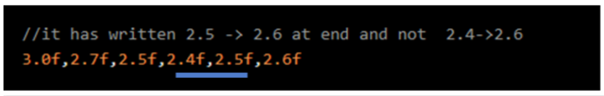
***FAIL

//it has written 2.5 -> 2.6 at end and not 2.4->2.6

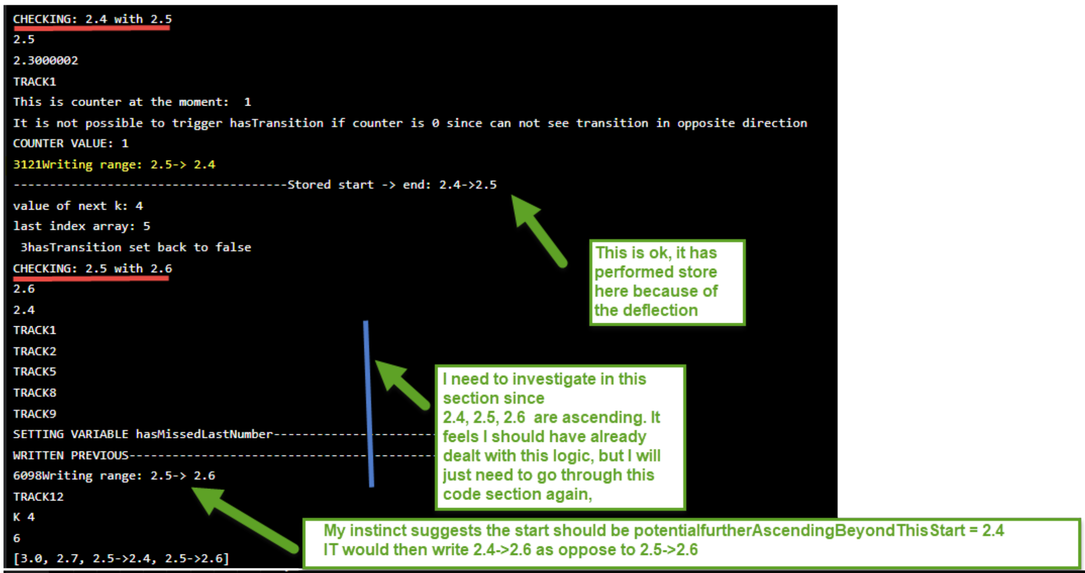
3.0f,2.7f,2.5f,2.4f,2.5f,2.6f

I have now resolved but will once again go through all failed test cases....

infact it might be good idea to go through all test cases at critical change...



```
//it has written 2.5 -> 2.6 at end and not  2.4->2.6
3.0f,2.7f,2.5f,2.4f,2.5f,2.6f
```

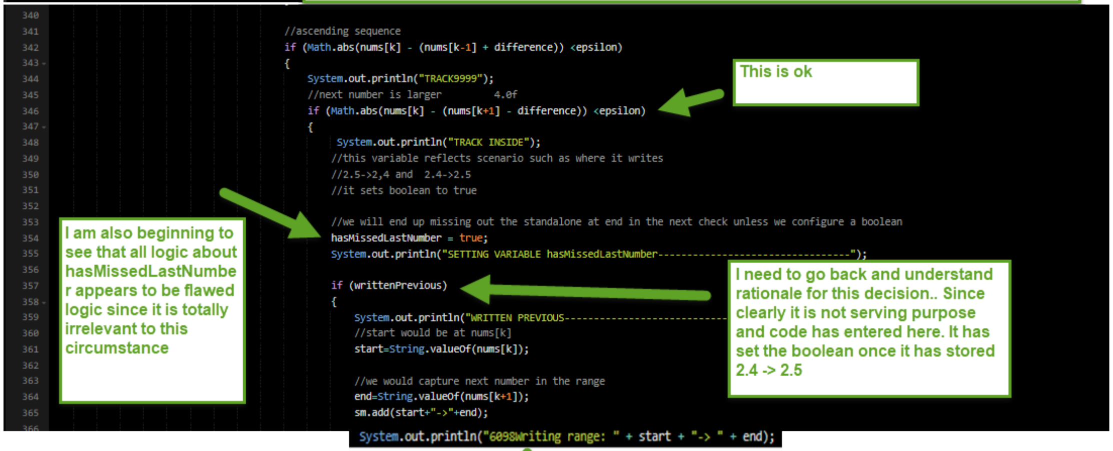


```
CHECKING: 2.4 with 2.5
2.5
2.3000002
TRACK1
This is counter at the moment:  1
It is not possible to trigger hasTransition if counter is 0 since can not see transition in opposite direction
COUNTER VALUE: 1
3121Writing range: 2.5-> 2.4
-------------------------------------Stored start -> end: 2.4->2.5
value of next k: 4
last index array: 5
 3hasTransition set back to false
CHECKING: 2.5 with 2.6
2.6
2.4
TRACK1
TRACK2
TRACK5
TRACK8
TRACK9
SETTING VARIABLE hasMissedLastNumber--------------------------------
WRITTEN PREVIOUS--------------------------------------------
6098Writing range: 2.5-> 2.6
TRACK12
K 4
6
[3.0, 2.7, 2.5->2.4, 2.5->2.6]
```

This is ok, it has performed store here because of the deflection

I need to investigate in this section since 2.4, 2.5, 2.6 are ascending. It feels I should have already dealt with this logic, but I will just need to go through this code section again,

My instinct suggests the start should be potentialfurtherAscendingBeyondThisStart = 2.4
IT would then write 2.4->2.6 as oppose to 2.5->2.6

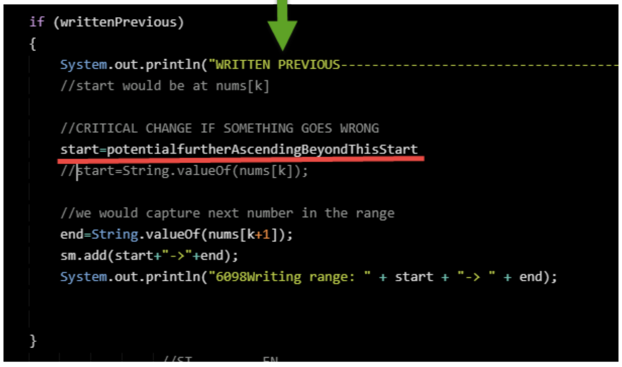


```
                //ascending sequence
                if (Math.abs(nums[k] - (nums[k-1] + difference)) <epsilon)
                {
                    System.out.println("TRACK9999");
                    //next number is larger        4.0f
                    if (Math.abs(nums[k] - (nums[k+1] - difference)) <epsilon)
                    {
                        System.out.println("TRACK INSIDE");
                        //this variable reflects scenario such as where it writes
                        //2.5->2,4 and  2.4->2.5
                        //it sets boolean to true

                        //we will end up missing out the standalone at end in the next check unless we configure a boolean
                        hasMissedLastNumber = true;
                        System.out.println("SETTING VARIABLE hasMissedLastNumber--------------------------------");

                        if (writtenPrevious)
                        {
                            System.out.println("WRITTEN PREVIOUS--------------------------
                            //start would be at nums[k]
                            start=String.valueOf(nums[k]);

                            //we would capture next number in the range
                            end=String.valueOf(nums[k+1]);
                            sm.add(start+"->"+end);
                            System.out.println("6098Writing range: " + start + "-> " + end);
```

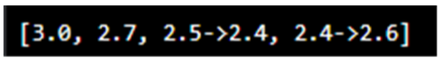
This is ok

I am also beginning to see that all logic about hasMissedLastNumbe r appears to be flawed logic since it is totally irrelevant to this circumstance

I need to go back and understand rationale for this decision.. Since clearly it is not serving purpose and code has entered here. It has set the boolean once it has stored 2.4 -> 2.5

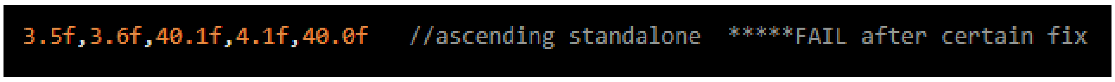


```
if (writtenPrevious)
{
    System.out.println("WRITTEN PREVIOUS------------------------------------
    //start would be at nums[k]

    //CRITICAL CHANGE IF SOMETHING GOES WRONG
    start=potentialfurtherAscendingBeyondThisStart
    //start=String.valueOf(nums[k]);

    //we would capture next number in the range
    end=String.valueOf(nums[k+1]);
    sm.add(start+"->"+end);
    System.out.println("6098Writing range: " + start + "-> " + end);


}
```

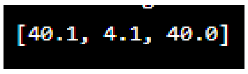


```
[3.0, 2.7, 2.5->2.4, 2.4->2.6]
```

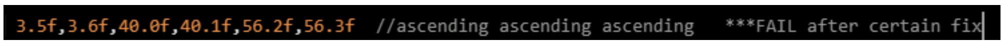
I have unfortunately seen this test case fail as a result:



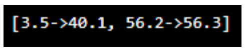
```
3.5f,3.6f,40.1f,4.1f,40.0f   //ascending standalone   *****FAIL after certain fix
```



```
[40.1, 4.1, 40.0]
```

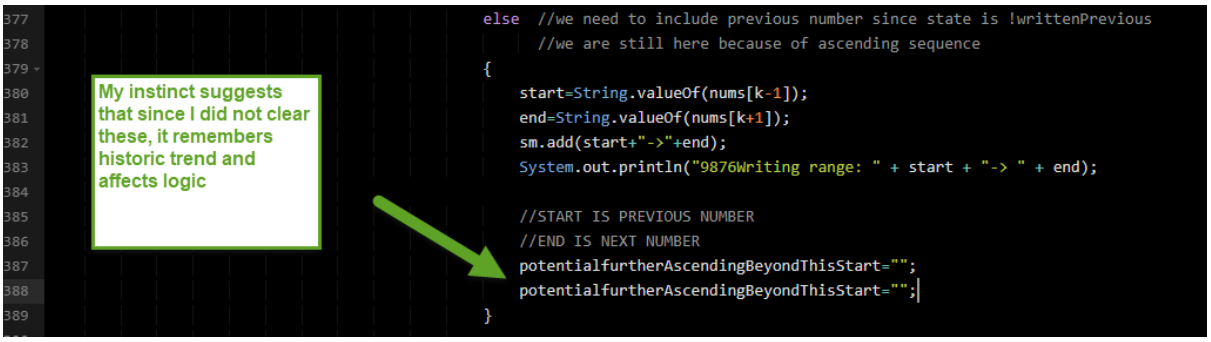


```
3.5f,3.6f,40.0f,40.1f,56.2f,56.3f  //ascending ascending ascending   ***FAIL after certain fix
```



```
[3.5->40.1, 56.2->56.3]
```

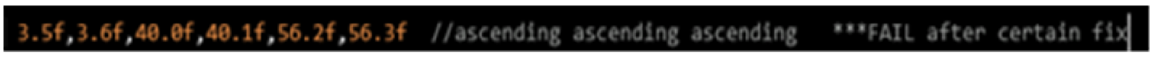
My logic tells me straight away that when I added new code such as:



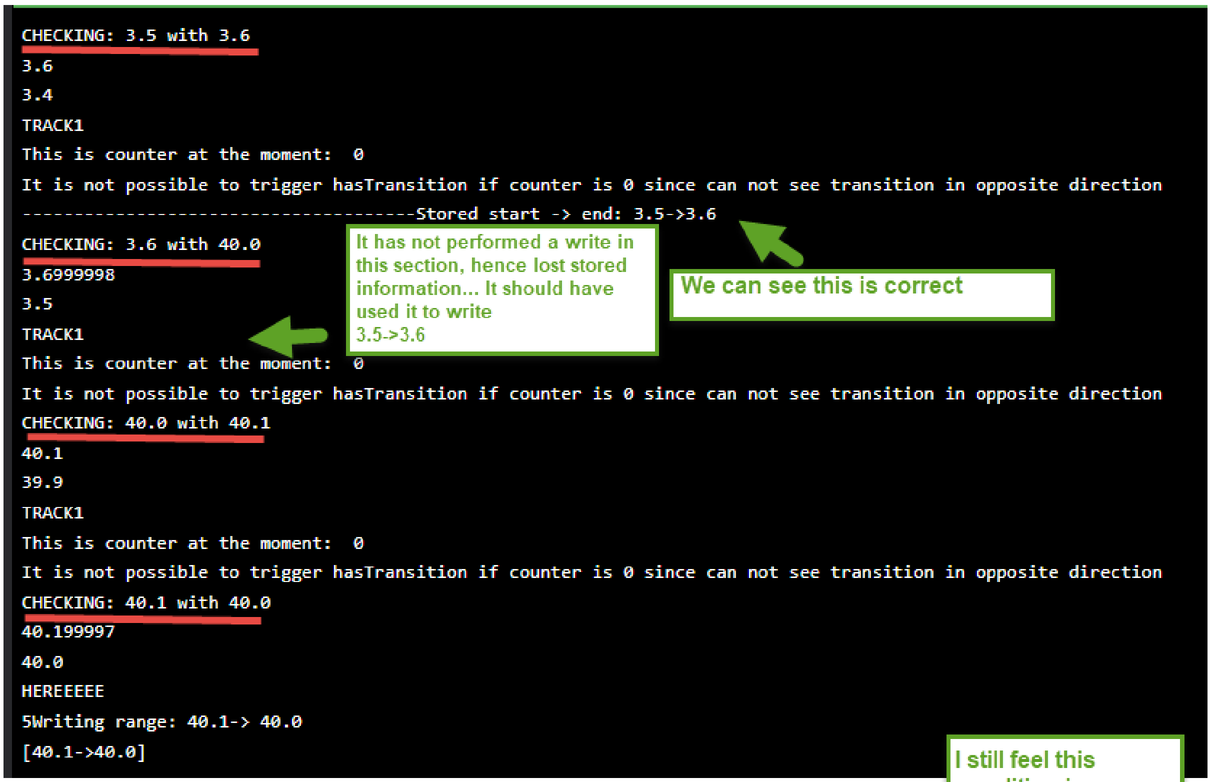
My instinct suggests that since I did not clear these, it remembers historic trend and affects logic

```
else  //we need to include previous number since state is !writtenPrevious
      //we are still here because of ascending sequence
{
    start=String.valueOf(nums[k-1]);
    end=String.valueOf(nums[k+1]);
    sm.add(start+"->"+end);
    System.out.println("9876Writing range: " + start + "-> " + end);

    //START IS PREVIOUS NUMBER
    //END IS NEXT NUMBER
    potentialfurtherAscendingBeyondThisStart="";
    potentialfurtherAscendingBeyondThisStart="";
}
```

This is totally unrelated, I have explored this area and it has fixed the issue:



```
3.5f,3.6f,40.0f,40.1f,56.2f,56.3f  //ascending ascending ascending   ***FAIL after certain fix
```



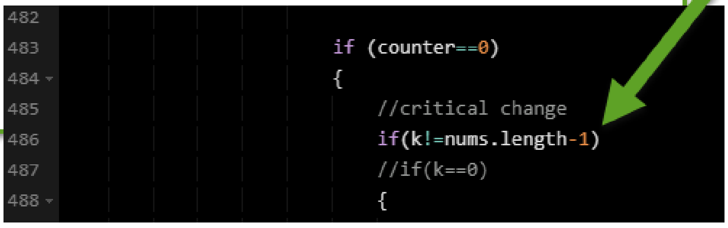
```
CHECKING: 3.5 with 3.6
3.6
3.4
TRACK1
This is counter at the moment:  0
It is not possible to trigger hasTransition if counter is 0 since can not see transition in opposite direction
--------------------------------------Stored start -> end: 3.5->3.6
CHECKING: 3.6 with 40.0
3.6999998
3.5
TRACK1
This is counter at the moment:  0
It is not possible to trigger hasTransition if counter is 0 since can not see transition in opposite direction
CHECKING: 40.0 with 40.1
40.1
39.9
TRACK1
This is counter at the moment:  0
It is not possible to trigger hasTransition if counter is 0 since can not see transition in opposite direction
CHECKING: 40.1 with 40.0
40.199997
40.0
HEREEEEE
5Writing range: 40.1-> 40.0
[40.1->40.0]
```

It has not performed a write in this section, hence lost stored information... It should have used it to write 3.5->3.6

We can see this is correct

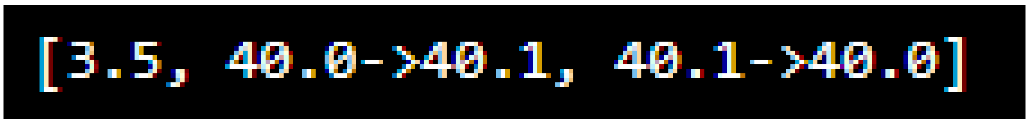
I still feel this condition is wrong... since we should be able to use the stored values at any point.. For now, I could not think of any other condition since there is lots of code depending on this

I have now completed one of the biggest changes to my logic.. I found it was not going into area to use stored values because I had loop set to if (k==0)... we know this is not the case..



```
                    if (counter==0)
                    {
                        //critical change
                        if(k!=nums.length-1)
                        //if(k==0)
                        {
```

I think it is now extremely critical I go through all my devised cases also include the ChatGPT extract or even try larger section of this code....



```
[3.5, 40.0->40.1, 40.1->40.0]
```

I have rolled the change back:

I have gone through all my test cases again...

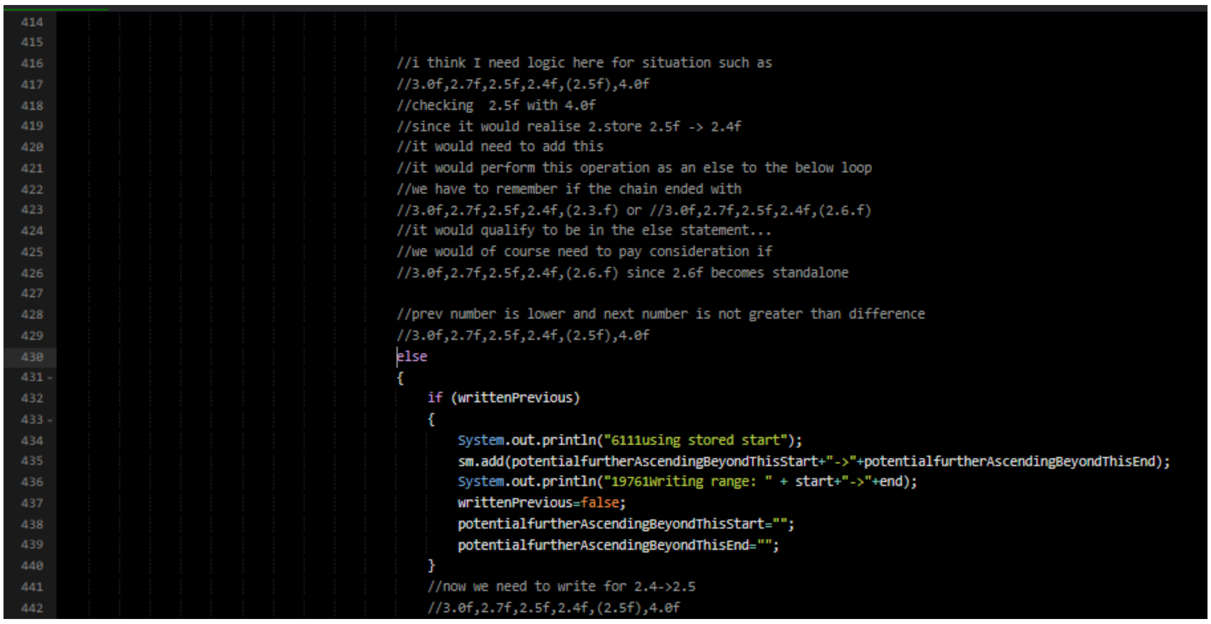
These are failing occurrences...

I am still very sure these can be modified without impacting the main flow in chatGPT code.....

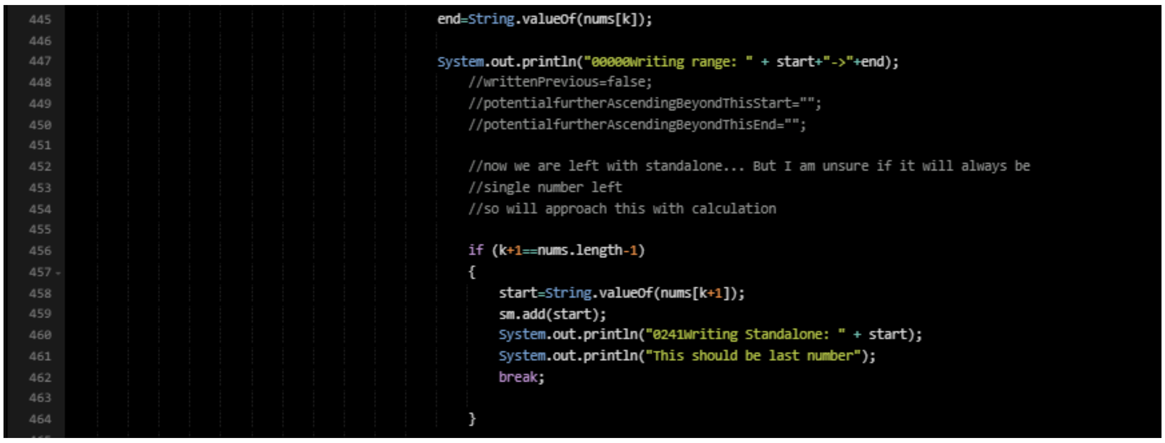
I will take each turn by turn...... and I really need to understand why it differs from the passing ones... otherwise it will be a total spiral.

//ADDRESS*************************************

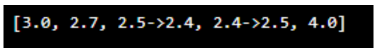
//3.0f,2.7f,2.5f,2.4f,2.5f,4.0f //exploring above scenario but the descend is slightly longer ***FAIL**********



```
//i think I need logic here for situation such as
//3.0f,2.7f,2.5f,2.4f,(2.5f),4.0f
//checking  2.5f with 4.0f
//since it would realise 2.store 2.5f -> 2.4f
//it would need to add this
//it would perform this operation as an else to the below loop
//we have to remember if the chain ended with
//3.0f,2.7f,2.5f,2.4f,(2.3.f) or //3.0f,2.7f,2.5f,2.4f,(2.6.f)
//it would qualify to be in the else statement...
//we would of course need to pay consideration if
//3.0f,2.7f,2.5f,2.4f,(2.6.f) since 2.6f becomes standalone

//prev number is lower and next number is not greater than difference
//3.0f,2.7f,2.5f,2.4f,(2.5f),4.0f
else
{
    if (writtenPrevious)
    {
        System.out.println("6111using stored start");
        sm.add(potentialfurtherAscendingBeyondThisStart+"->"+potentialfurtherAscendingBeyondThisEnd);
        System.out.println("19761Writing range: " + start+"->"+end);
        writtenPrevious=false;
        potentialfurtherAscendingBeyondThisStart="";
        potentialfurtherAscendingBeyondThisEnd="";
    }
    //now we need to write for 2.4->2.5
    //3.0f,2.7f,2.5f,2.4f,(2.5f),4.0f
```



```
    end=String.valueOf(nums[k]);

    System.out.println("00000Writing range: " + start+"->"+end);
        //writtenPrevious=false;
        //potentialfurtherAscendingBeyondThisStart="";
        //potentialfurtherAscendingBeyondThisEnd="";

        //now we are left with standalone... But I am unsure if it will always be
        //single number left
        //so will approach this with calculation

        if (k+1==nums.length-1)
        {
            start=String.valueOf(nums[k+1]);
            sm.add(start);
            System.out.println("0241Writing Standalone: " + start);
            System.out.println("This should be last number");
            break;

        }
```

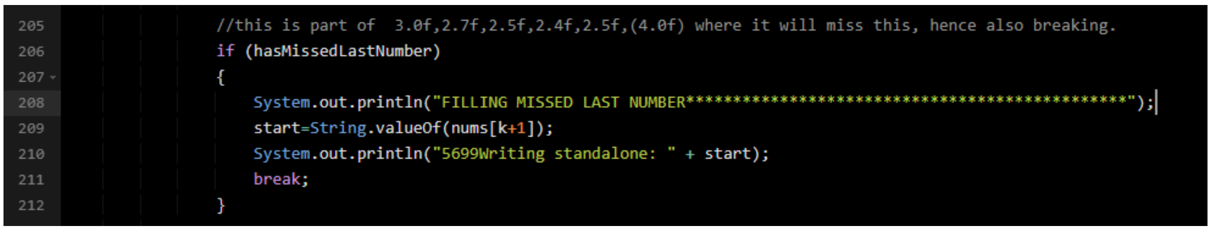


```
[3.0, 2.7, 2.5->2.4, 2.4->2.5, 4.0]
```

Before I address the rest, I can see I have used technique to accommodate for the last number as standalone..

A few test cases earlier, I had code for this circumstance. And ironically we can see it was to handle the situation that I just fixed...

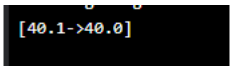
I am going to quickly run through my test cases and ascertain how many cases are actually relying on this. It now seems like poor practice given how I managed it better above....



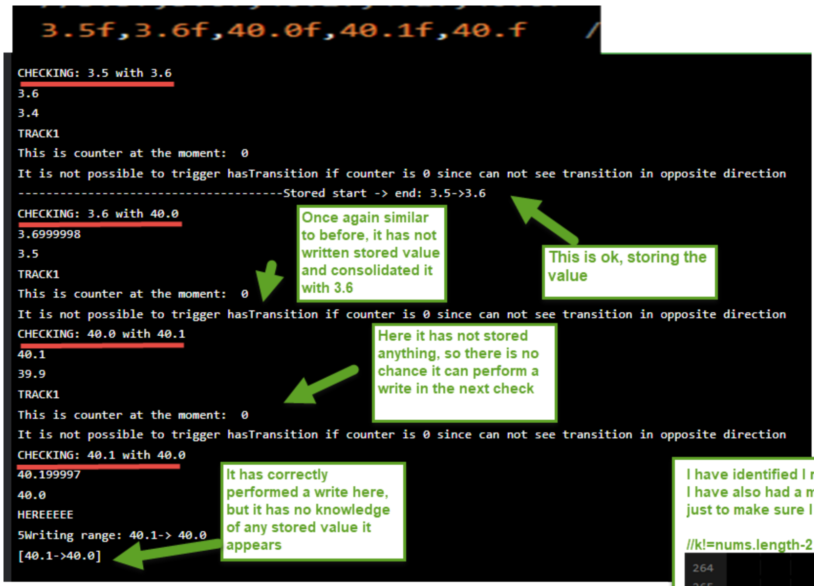
```
//this is part of  3.0f,2.7f,2.5f,2.4f,2.5f,(4.0f) where it will miss this, hence also breaking.
if (hasMissedLastNumber)
{
    System.out.println("FILLING MISSED LAST NUMBER************************************************");
    start=String.valueOf(nums[k+1]);
    System.out.println("5699Writing standalone: " + start);
    break;
}
```

There is no code reaching here, so I have removed all logic surrounding this

//3.5f,3.6f,40.1f,4.1f,40.0f   //ascending standalone *****FAIL after certain fix



```
[40.1->40.0]
```



```
3.5f,3.6f,40.0f,40.1f,40.f

CHECKING: 3.5 with 3.6
3.6
3.4
TRACK1
This is counter at the moment:  0
It is not possible to trigger hasTransition if counter is 0 since can not see transition in opposite direction
-----------------------------------Stored start -> end: 3.5->3.6
CHECKING: 3.6 with 40.0
3.6999998
3.5
TRACK1
This is counter at the moment:  0
It is not possible to trigger hasTransition if counter is 0 since can not see transition in opposite direction
CHECKING: 40.0 with 40.1
40.1
39.9
TRACK1
This is counter at the moment:  0
It is not possible to trigger hasTransition if counter is 0 since can not see transition in opposite direction
CHECKING: 40.1 with 40.0
40.199997
40.0
HEREEEEE
5Writing range: 40.1-> 40.0
[40.1->40.0]
```

Once again similar to before, it has not written stored value and consolidated it with 3.6

This is ok, storing the value

Here it has not stored anything, so there is no chance it can perform a write in the next check

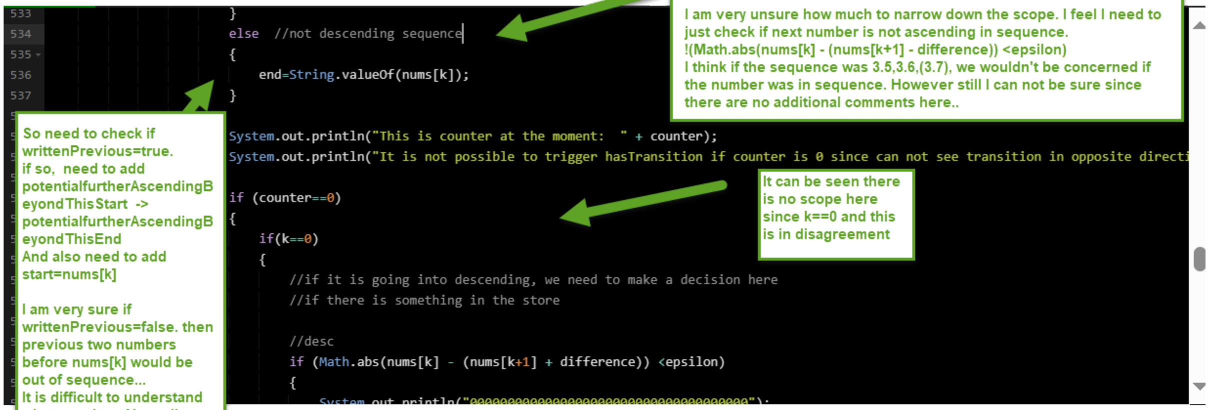
It has correctly performed a write here, but it has no knowledge of any stored value it appears

I have identified I require implementation either here
I have also had a massive look through all logic leading up to here..
just to make sure I am not adding logic on false grounds:

//k!=nums.length-2   Also we know not a descending sequence

```
                }  //not descending sequence
                else
                {
```

I am very unsure how much to narrow down the scope. I feel I need to just check if next number is not ascending in sequence.
!(Math.abs(nums[k] - (nums[k+1] - difference)) <epsilon)
I think if the sequence was 3.5,3.6,(3.7), we wouldn't be concerned if the number was in sequence. However still I can not be sure since there are no additional comments here..



```
                }
                else  //not descending sequence
                {
                    end=String.valueOf(nums[k]);
                }

                System.out.println("This is counter at the moment:  " + counter);
                System.out.println("It is not possible to trigger hasTransition if counter is 0 since can not see transition in opposite directi

                if (counter==0)
                {
                    if(k==0)
                    {
                        //if it is going into descending, we need to make a decision here
                        //if there is something in the store

                        //desc
                        if (Math.abs(nums[k] - (nums[k+1] + difference)) <epsilon)
                        {
                            System.out.println("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@");
```

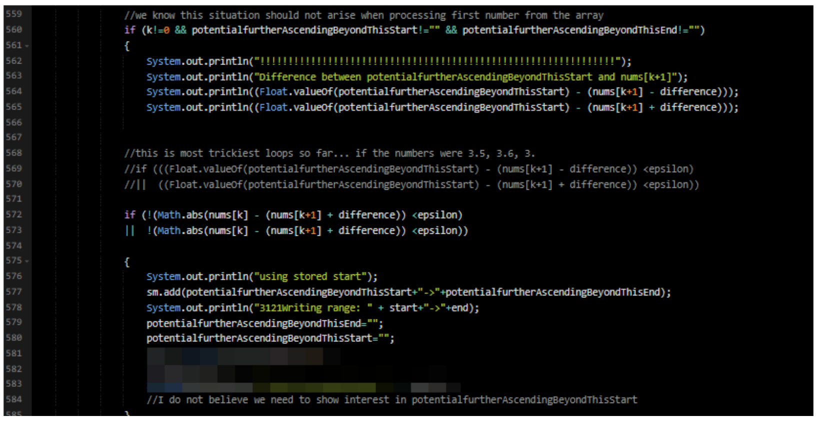
So need to check if writtenPrevious=true.
if so, need to add potentialfurtherAscendingBeyondThisStart -> potentialfurtherAscendingBeyondThisEnd
And also need to add start=nums[k]

I am very sure if writtenPrevious=false. then previous two numbers before nums[k] would be out of sequence...
It is difficult to understand what to write... Normally as can be seen above it would be writing end nums[k]=3.6

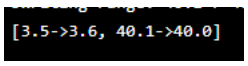
I would need to go down..
I am just not sure what the situation will be....

It can be seen there is no scope here since k==0 and this is in disagreement

Unfortunately my mental processing has been severely hindered since I needed to bring knowledge of lots code together...



```
            //we know this situation should not arise when processing first number from the array
            if (k!=0 && potentialfurtherAscendingBeyondThisStart!="" && potentialfurtherAscendingBeyondThisEnd!="")
            {
                System.out.println("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
                System.out.println("Difference between potentialfurtherAscendingBeyondThisStart and nums[k+1]");
                System.out.println((Float.valueOf(potentialfurtherAscendingBeyondThisStart) - (nums[k+1] - difference)));
                System.out.println((Float.valueOf(potentialfurtherAscendingBeyondThisStart) - (nums[k+1] + difference)));


            //this is most trickiest loops so far... if the numbers were 3.5, 3.6, 3.
            //if (((Float.valueOf(potentialfurtherAscendingBeyondThisStart) - (nums[k+1] - difference)) <epsilon)
            //||  ((Float.valueOf(potentialfurtherAscendingBeyondThisStart) - (nums[k+1] + difference)) <epsilon))

            if (!(Math.abs(nums[k] - (nums[k+1] + difference)) <epsilon)
            ||  !(Math.abs(nums[k] - (nums[k+1] + difference)) <epsilon))

            {
                System.out.println("using stored start");
                sm.add(potentialfurtherAscendingBeyondThisStart+"->"+potentialfurtherAscendingBeyondThisEnd);
                System.out.println("3121Writing range: " + start+"->"+end);
                potentialfurtherAscendingBeyondThisEnd="";
                potentialfurtherAscendingBeyondThisStart="";
                [redacted]
                //I do not believe we need to show interest in potentialfurtherAscendingBeyondThisStart
```



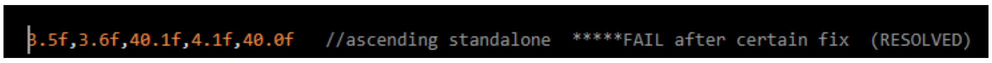
```
[3.5->3.6, 40.1->40.0]
```

We can see it has still failed to write 40.0->41.0

I am going to visit my outputs as usual

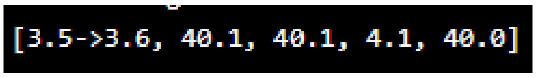
I am going to go through all my test cases again, the fact that I am outputting more screen outputs suggests it is becoming extremely difficult to remember the paths...

TEST CASE:

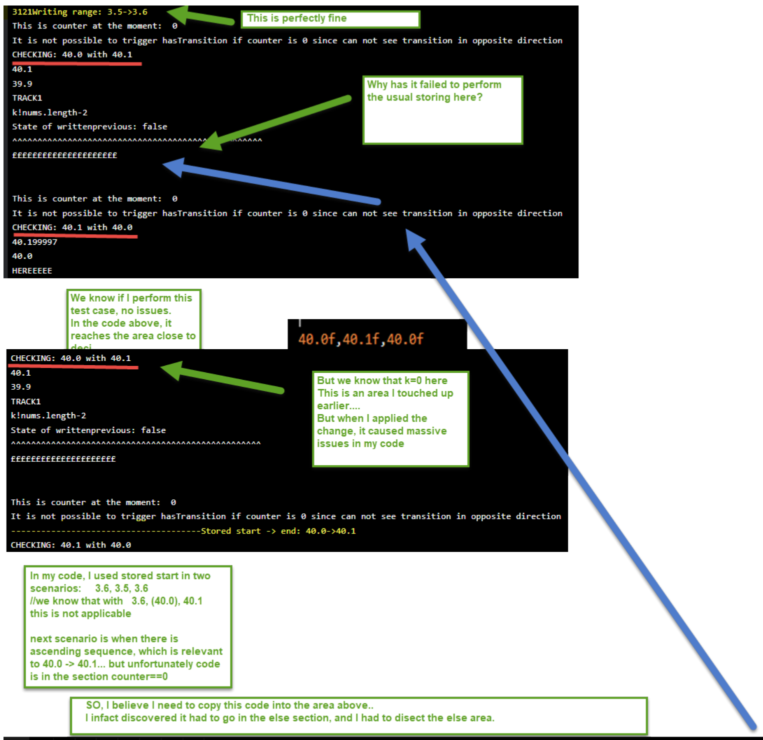


```
3.5f,3.6f,40.1f,4.1f,40.0f    //ascending standalone   *****FAIL after certain fix  (RESOLVED)
```

This has now failed, it is writing the standalone twice



```
[3.5->3.6, 40.1, 40.1, 4.1, 40.0]
```



```
3121Writing range: 3.5->3.6
This is counter at the moment:  0
It is not possible to trigger hasTransition if counter is 0 since can not see transition in opposite direction
CHECKING: 40.0 with 40.1
40.1
39.9
TRACK1
k!nums.length-2
State of writtenprevious: false
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
££££££££££££££££££££


This is counter at the moment:  0
It is not possible to trigger hasTransition if counter is 0 since can not see transition in opposite direction
CHECKING: 40.1 with 40.0
40.199997
40.0
HEREEEEE
```

This is perfectly fine

Why has it failed to perform the usual storing here?

We know if I perform this test case, no issues.
In the code above, it reaches the area close to deci

```
40.0f,40.1f,40.0f
```

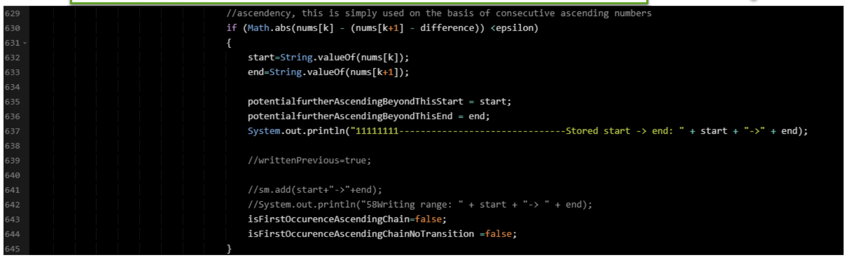
```
CHECKING: 40.0 with 40.1
40.1
39.9
TRACK1
k!nums.length-2
State of writtenprevious: false
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
££££££££££££££££££££


This is counter at the moment:  0
It is not possible to trigger hasTransition if counter is 0 since can not see transition in opposite direction
------------------------------------Stored start -> end: 40.0->40.1
CHECKING: 40.1 with 40.0
```

But we know that k=0 here
This is an area I touched up earlier....
But when I applied the change, it caused massive issues in my code

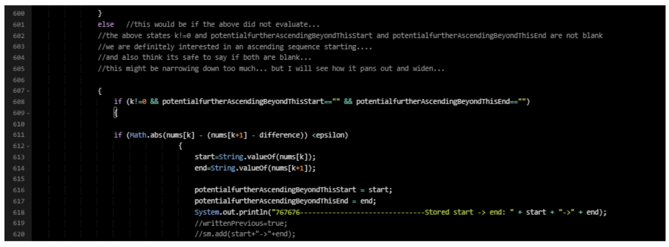
In my code, I used stored start in two scenarios: 3.6, 3.5, 3.6
//we know that with   3.6, (40.0), 40.1
this is not applicable

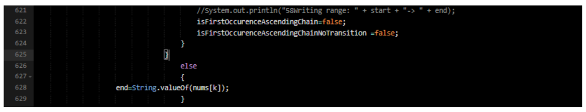
next scenario is when there is ascending sequence, which is relevant to 40.0 -> 40.1... but unfortunately code is in the section counter==0

SO, I believe I need to copy this code into the area above..
I infact discovered it had to go in the else section, and I had to disect the else area.

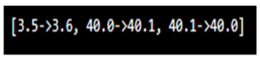


```java
                    //ascendency, this is simply used on the basis of consecutive ascending numbers
                    if (Math.abs(nums[k] - (nums[k+1] - difference)) <epsilon)
                    {
                        start=String.valueOf(nums[k]);
                        end=String.valueOf(nums[k+1]);

                        potentialfurtherAscendingBeyondThisStart = start;
                        potentialfurtherAscendingBeyondThisEnd = end;
                        System.out.println("11111111------------------------------Stored start -> end: " + start + "->" + end);

                        //writtenPrevious=true;

                        //sm.add(start+"->"+end);
                        //System.out.println("58Writing range: " + start + "-> " + end);
                        isFirstOccurenceAscendingChain=false;
                        isFirstOccurenceAscendingChainNoTransition =false;
                    }
```





```java
        }
        else   //this would be if the above did not evaluate...
        //the above states k!=0 and potentialfurtherAscendingBeyondThisStart and potentialfurtherAscendingBeyondThisEnd are not blank
        //we are definitely interested in an ascending sequence starting....
        //and also think its safe to say if both are blank...
        //this might be narrowing down too much... but I will see how it pans out and widen...

        {
            if (k!=0 && potentialfurtherAscendingBeyondThisStart=="" && potentialfurtherAscendingBeyondThisEnd=="")
            {

            if (Math.abs(nums[k] - (nums[k+1] - difference)) <epsilon)
                        {
                            start=String.valueOf(nums[k]);
                            end=String.valueOf(nums[k+1]);

                            potentialfurtherAscendingBeyondThisStart = start;
                            potentialfurtherAscendingBeyondThisEnd = end;
                            System.out.println("767676------------------------------Stored start -> end: " + start + "->" + end);
                            //writtenPrevious=true;
                            //sm.add(start+"->"+end);
                            //System.out.println("58Writing range: " + start + "-> " + end);
                            isFirstOccurenceAscendingChain=false;
                            isFirstOccurenceAscendingChainNoTransition =false;
                        }
                    }
                        else
                        {
            end=String.valueOf(nums[k]);
                        }
```



```
[3.5->3.6, 40.0->40.1, 40.1->40.0]
```

I am now going to visit the test cases below which were identified to fail.

And it has damaged my logic massively ......
So I need to roll back my documentation unfortunately....

```
    //3.5f,3.6f,40.0f,40.1f,40.f   //ascending ascending descending  FAIL

    //3.5f,3.6f,40.0f,40.1f,56.2f,56.3f  //ascending ascending ascending   ***FAIL after certain fix

    //75.0f, 75.0f, 95.6f, 95.7f, 95.8f, 95.9f, 96.0f, 96.1f, 40.1f, 40.1f //************************FAIL
```