So as usual, I am in this documentation again to try and trace through my code...
I have taken a small extract of the output (OutputToleranceLimit.txt)...

//PROVIDED INPUT

//float[] nums = {

// 4.9f, 4.8f, 4.7f, 4.6f, 4.7f, 4.8f, 4.9f, 5.0f, 48.5f, 91.7f, 82.9f, 57.6f, 57.5f, 57.4f, 57.3f, 57.2f,

//OUTPUT

[4.9->4.6, 4.6->4.7, 4.7->4.8, 4.8->4.9, 4.9->5.0, 48.5, 91.7, 82.9, 57.6->57.2

TEST CASE: Identifying pattern

It can be seen that 4.9f, 4.8f, 4.7f, 4.6f has been summarised as 4.9->4.6
Now there is a change in direction..... 4.6f, 4.7f as 4.6->4.7
But it continues to separate 4.7->4.8, 4.8->4.9, 4.9->5.0
So my first point of investigating will be checking 4.7 with 4.8 and 4.8 with 4.9 in my text file....

TEST CASE: Determining rationale for issue

```
CHECKING: 4.6 with 4.7
4.7
4.5
2Writing range: 4.9-> 4.6

CHECKING: 4.7 with 4.8
4.7999997
4.6
11Writing range: 4.6->4.7
CHECKING: 4.8 with 4.9
4.9
4.7000003
11Writing range: 4.7->4.8
CHECKING: 4.9 with 5.0
5.0
4.8
11Writing range: 4.8->4.9
CHECKING: 5.0 with 48.5
5.1
4.9
11Writing range: 4.9->5.0
CHECKING: 48.5 with 91.7
48.6
48.4
6Writing Standalone: 48.5
CHECKING: 91.7 with 82.9
```

This is perfectly fine since the next check (4.7->4.8) is moving in opposite direction

Issue starts here since it has not accepted the tolerance between 4.7 ->4.8...
Hence it has written the range 4.6 -> 4.7
We can see that tolerance is 4.8 - 4.7999997 = 0.000003. This is well within 0.00005

The ironic area is that the summary ranges are not failing:

If we review the data again in parallel:

So my focus will go straight to check 4.7 => 4.8

```
//float[] nums = {
    //    4.9f, 4.8f, 4.7f, 4.6f, 4.7f, 4.8f, 4.9f, 5.0f, 48.5f, 91.7f, 82.9f, 57.6f, 57.5f, 57.4f, 57.3f, 57.2f, 57.3f, 57.4f, 57.5f, 57.6f, 26.1f, 25.4f, 21.2f, 83.5f, 56.3f, 56.2f, 56.1f
98.9f, 63.8f, 63.9f, 64.0f, 64.1f, 64.2f, 64.1f, 64.0f, 63.9f, 63.8f, 54.8f, 22.6f, 41.1f, 91.4f, 47.9f, 47.8f, 47.7f, 47.6f, 47.5f, 47.6f, 47.7f, 47.8f, 47.9f, 58.5f, 70.8f, 11.3f, 58.6f,
.9f, 49.8f, 49.9f, 50.0f, 50.1f, 50.2f, 58.0f, 67.8f, 41.3f, 96.8f, 72.3f, 72.2f, 72.1f, 72.0f, 71.9f, 71.8f, 71.7f, 71.6f, 71.5f, 29.7f, 81.9f, 18.0f, 46.4f, 34.3f, 34.4f, 34.5f, 34.6f, 34
f, 28.0f, 28.1f, 70.7f, 54.5f, 13.5f, 77.1f, 41.2f, 41.1f, 41.0f, 40.9f, 40.8f, 40.7f, 40.6f, 40.5f, 40.4f, 59.4f, 90.0f,  6.7f, 85.4f, 69.5f, 69.6f, 69.7f, 69.8f, 69.9f, 70.0f, 70.1f, 70.2
27.4f, 56.1f, 88.7f, 50.8f, 50.9f, 51.0f, 51.1f, 51.2f, 51.1f, 51.0f, 50.9f, 50.8f, 12.9f, 72.1f, 33.3f, 67.9f, 74.9f, 74.8f, 74.7f, 74.6f, 74.5f, 74.6f, 74.7f, 74.8f, 74.9f, 49.0f, 21.9f,
    //    };

//OUTPUT

[4.9->4.6, 4.6->4.7, 4.7->4.8, 4.8->4.9, 4.9->5.0, 48.5, 91.7, 82.9, 57.6->57.2, 57.2->57.3, 57.3->57.4, 57.4->57.5, 57.5->57.6, 26.1, 25.4, 21.2, 83.5, 56.3->55.6, 13.9, 35.7, 54.8->54.5,
, 44.2, 96.7, 52.0->51.2, 20.3, 93.8, 37.8, 61.8, 30.6, 30.6->30.7, 30.7->30.8, 30.8->30.9, 31.0->30.6, 12.1, 84.4, 2.9, 63.2, 69.7->69.3, 69.3->69.4, 69.4->69.5, 69.5->69.6, 69.6->69.7, 5
1.5, 84.7, 51.0->50.6, 50.6->50.7, 50.7->50.8, 50.8->50.9, 50.9->51.0, 78.7, 36.4, 59.9, 95.3, 44.9->44.1, 83.6, 25.9, 32.1, 11.1, 67.5->67.1, 67.1->67.2, 67.2->67.3, 67.3->67.4, 67.4->67.5
.3, 67.9, 74.9->74.5, 74.5->74.6, 74.6->74.7, 74.7->74.8, 74.8->74.9, 49.0, 21.9, 61.2, 93.6, 55.3->55.0, 55.0->55.1, 55.1->55.2, 55.2->55.3, 55.3->55.4, 55.4->55.5, 43.4, 61.0, 47.3, 47.4
```

it seems to perform at small increments

it has handled descending in one block. Could this be due to code being derrived from ascending challenge?

It appears for any ascending scenario where it spans 3 or more numbers, it tends to isolate them

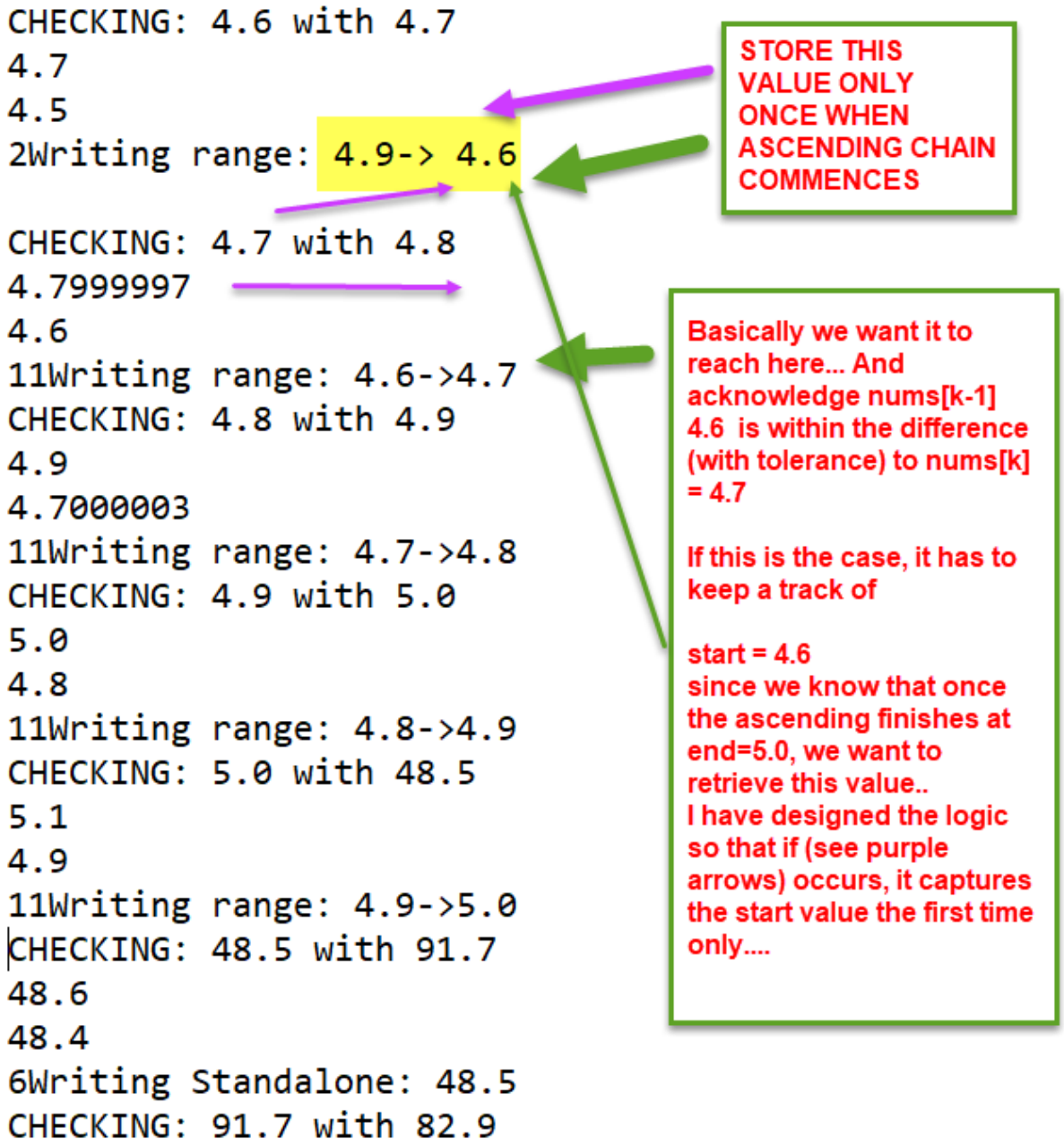
TEST CASE:

This has required a severe amount of mental resource...

Firstly I have broken down the above into something more manageable and relevant to my investigation:

4.9f, 4.8f, 4.7f, 4.6f, 4.7f, 4.8f, 4.9f, 5.0f, 48.5f, 91.7f

In simple terms... see below:



The code translates to similar:

```
239                 else
240                 {
241                     if (Math.abs(nums[k] - (nums[k-1] + difference)) <epsilon)
242                     //if ((nums[k-1]-difference)==(nums[k]))
243                     {
244                         //we know it would enter here for cases such as  4.6->4.7
245 
246                         //String backupEnd = end;            //this would be 4.6 value before
247                         end = String.valueOf(nums[k]);
248 
249                         //since for ascending, we are getting entries such as
250                         //4.6->4.7, 4.7->4.8, 4.8->4.9, 4.9->5.0
251                         //we need to merge summary range together again
252 
253                          //4.9f, 4.8f, 4.7f, 4.6f,        4.7f, 4.8f, 4.9f, 5.0f, 48.5f, 91.7f
254 
255                         System.out.println("This is backupEnd: " + backupEnd);
256                         System.out.println("This is start: " + start);
```

```
255 System.out.println("This is backupEnd: " + backupEnd);
256 System.out.println("This is start: " + start);
257 
258 //here we know we are in ascending area....
259 //so we need to check using epsilon if the prev is less than current
260 //within margin of error
261 
262 //this logic suggests previous number is lower since addding difference
263 //it also has to ensure the front is increasing by difference, in which case we have to subtract difference from
264 //also need to find a way of not keep obtaining the start...
265 //it would only get the start on the first instance of the below if loop
266 //for instance we want start to remain as 4.6f and not be adjusted everytime nums[k] increases
267 
268 //4.6f,       4.7f, 4.8f, 4.9f, 5.0f,
269 
270 if ((Math.abs(nums[k] - (nums[k-1] + difference)) <epsilon)
271 &&   (Math.abs(nums[k] - (nums[k+1] - difference))<epsilon))
272 {
273     System.out.println("DO NOTHING*****************");
274 
```

```
275                         if (!isFirstOccurenceAscendingChain)
276                         {
277                             start=String.valueOf(nums[k-1]);  //4.6
278                             System.out.println("BUT ACKNOWLEDGED START: " + start);
279 
280                             //need to lock this loop until else has reached
281                             isFirstOccurenceAscendingChain=true;
282                         }
283                     }
284 
285                     else
286                     {
287                         isFirstOccurenceAscendingChain=false;
288                         System.out.println("*************************");
289                     sm.add(start+"->"+end);
290                     System.out.println("11Writing range: " + backupEnd+"->"+end);
291                     }
292                 }
```

It can be seen as I run the execution, the output is successful

```
float[] nums = {
4.9f, 4.8f, 4.7f, 4.6f, 4.7f, 4.8f, 4.9f, 5.0f, 48.5f, 91.7f
```

```
[4.9->4.6, 4.6->5.0, 48.5, 91.7]
```

I am extremely pleased with this outcome since it was a challenging addition to my code.

TEST CASE:

My next intuition was to change the values from this point onwards....

```
float[] nums = {
//4.9f, 4.8f, 4.7f, 4.6f, 4.7f, 4.8f, 4.9f, 5.0f, 48.5f, 91.7f  //in test documentation
  4.9f, 4.8f, 4.7f, 4.5f, 4.6f, 4.7f, 4.9f, 5.0f, 48.5f, 91.7f
```

So now exploring ascending from 4.5f onwards but there is no overlap with previous summary range (as per commented line above)

FAIL

```
[4.9->4.7, 4.5, 4.5->4.7, 4.9, 4.9->5.0, 48.5, 91.7]
```

We can see two errors here of a standalone write. This is without a doubt related to my added logic... So I will just follow the code around

```
Establishing start: 4.9
CHECKING: 4.8 with 4.7
4.9
4.7000003
CHECKING: 4.7 with 4.5
4.7999997
4.6
2Writing range: 4.9-> 4.7
CHECKING: 4.5 with 4.6
4.6
4.4
6Writing Standalone: 4.5
CHECKING: 4.6 with 4.7
```

The issue starts exactly here, so I will check the area of code surrounding 6Writing. Note I have lost all the junctions in this code, but I still have kept references

```
float[] nums = {
//4.9f, 4.8f, 4.7f, 4.6f, 4.7f, 4.8f, 4.9f, 5.0f, 48.5f, 91.7f  //in test documentation
  4.9f, 4.8f, 4.7f, 4.5f, 4.6f, 4.7f, 4.9f, 5.0f, 48.5f, 91.7f
```

```
[4.9->4.7, 4.5, 4.5->4.7, 4.9, 4.9->5.0, 48.5, 91.7]
```

And fortunately it is exactly in the same area of code that I made changes, which in some respect will make my life much easier to fault find!!

It performs this operation, and this is stating that nums[k]  4.5 is greater than nums[k-1] 4.7 by  (difference + epsilon)
We know difference has been configured to 0.1....  So, clearly this is not the case,

```
240                             else
241                             {
242                                 if (Math.abs(nums[k] - (nums[k-1] + difference)) <epsilon)
243                                 //if ((nums[k-1]-difference)==(nums[k]))
244                                 {
                                        //we know it would enter here for cases such as  4.6->4.7
```

So it would correctly entered associated else and write the standalone number....

```
else
{
    start = String.valueOf(nums[k]);
    sm.add(start);
    System.out.println("6Writing Standalone: " + start);
}
```

But question remains, has the newly entered code really had an impact? Because I introduced logic

```
240                             else
241                             {
242                                 if (Math.abs(nums[k] - (nums[k-1] + difference)) <epsilon)
243                                 //if ((nums[k-1]-difference)==(nums[k]))
244                                 {
245                                     //we know it would enter here for cases such as  4.6->4.7
246
247                                     //String backupEnd = end;           //this would be 4.6 value before
248                                     end = String.valueOf(nums[k]);
249
250                                     //since for ascending, we are getting entries such as
251                                     //4.6->4.7, 4.7->4.8, 4.8->4.9, 4.9->5.0
252                                     //we need to merge summary range together again
253
254                                      //4.9f, 4.8f, 4.7f, 4.6f,       4.7f, 4.8f, 4.9f, 5.0f, 48.5f, 91.7f
255
256                                     System.out.println("This is start: " + start);
257
258                                     //here we know we are in ascending area....
259                                     //so we need to check using epsilon if the prev is less than current
260                                     //within margin of error
261
262                                     //this logic suggests previous number is lower since addding difference
263                                     //it also has to ensure the front is increasing by difference, in which case we have to subtract difference from nums[k+1]
264                                     //also need to find a way of not keep obtaining the start...
265                                     //it would only get the start on the first instance of the below if loop
266                                     //for instance we want start to remain as 4.6f and not be adjusted everytime nums[k] increases
267
268                                     //4.6f,       4.7f, 4.8f, 4.9f, 5.0f,
269
270                                     if ((Math.abs(nums[k] - (nums[k-1] + difference)) <epsilon)
271                                     &&   (Math.abs(nums[k] - (nums[k+1] - difference))<epsilon))
```

But it passed as false here... This makes me believe the fault must be before  I provided implementation.. So I will try this test case in my original code in which I configured the epsilon.

I introduced logic here...

And it can be seen that it was an issue in my previous code:

```
CHECKING: 4.5 with 4.6
4.6
4.4
6Writing Standalone: 4.5
CHECKING: 4.6 with 4.7
4.7
4.5
11Writing range: 4.5->4.6
CHECKING: 4.7 with 4.9
```

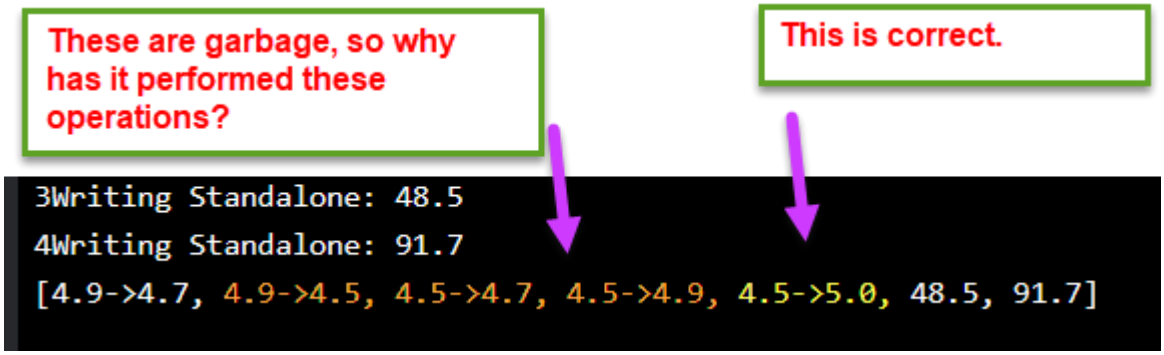
Effectively what it suggests is that before I write standalone  4.5, I need to check if there is an ascending or descending in front....
If there is, we are only interested in the start until chain stops...
This would be start  = nums[k]

I am rather surprised this has not flagged up earlier anywhere before given that I have completed standalone write  six times in my code...

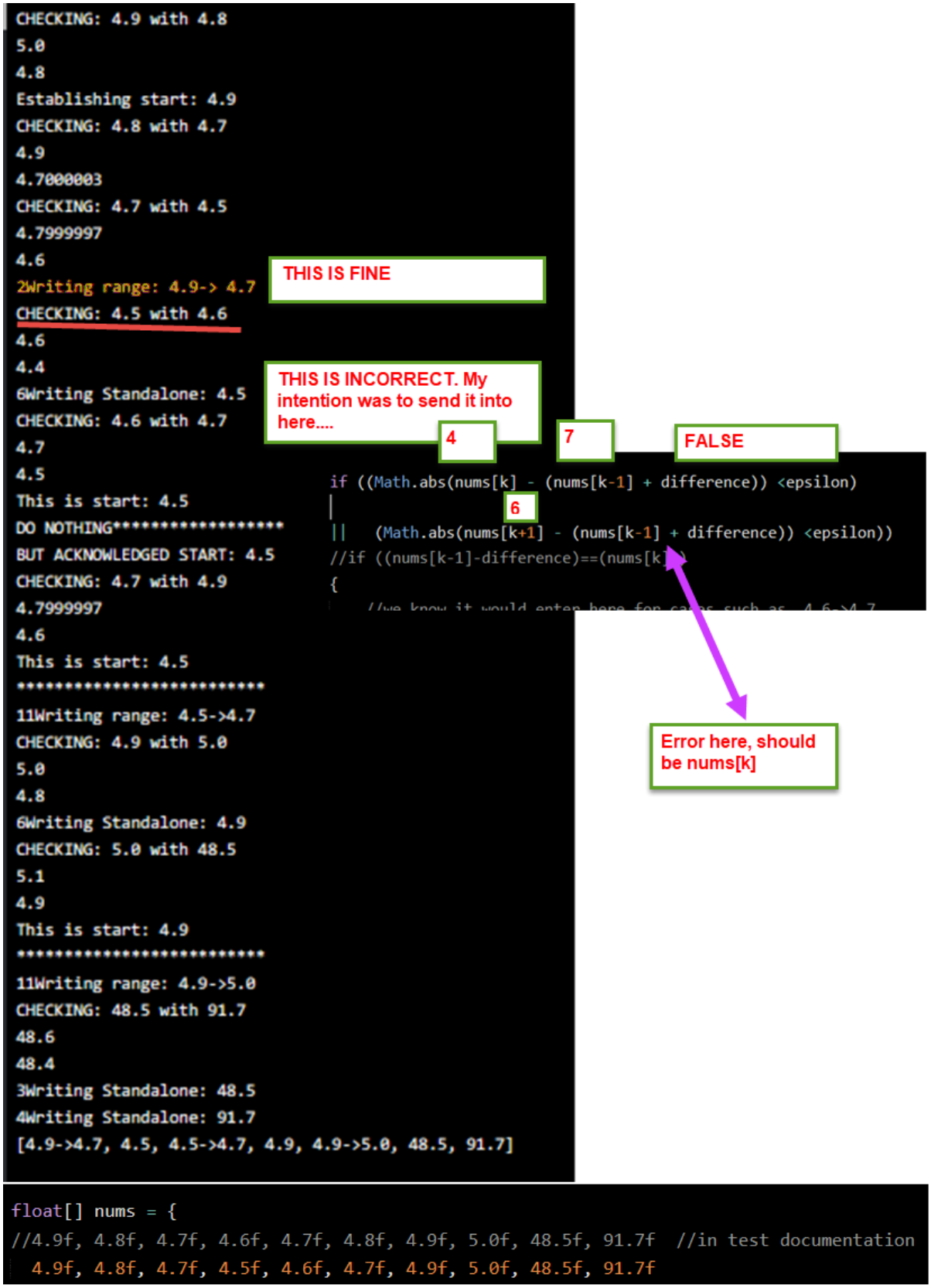
For now, I will just target if loop.
And unfortunately it has totally messed up the outcome...

```
240             else
241             {
242                 //to ensure a standalone does not enter the else loop, the first part determines relationship
243                 //with nums[k-1] but we need to forge connection with nums[k+1]
244                 //ie if nums[k] + difference is  within tolerance of nums[k+1]  ie still ascending
245
246                 if ((Math.abs(nums[k] - (nums[k-1] + difference)) <epsilon)
247                 ||   (Math.abs(nums[k+1] - (nums[k-1] + difference)) <epsilon))
248                 //if ((nums[k-1]-difference)==(nums[k]))
249                 {
250                     //we know it would enter here for cases such as  4.6->4.7
251
```

```
float[] nums = {
//4.9f, 4.8f, 4.7f, 4.6f, 4.7f, 4.8f, 4.9f, 5.0f, 48.5f, 91.7f  //in test documentation
  4.9f, 4.8f, 4.7f, 4.5f, 4.6f, 4.7f, 4.9f, 5.0f, 48.5f, 91.7f
```




Before I move further, I need to understand my false reasoning for this.
We can see in the photo above that the correct output at yellow.

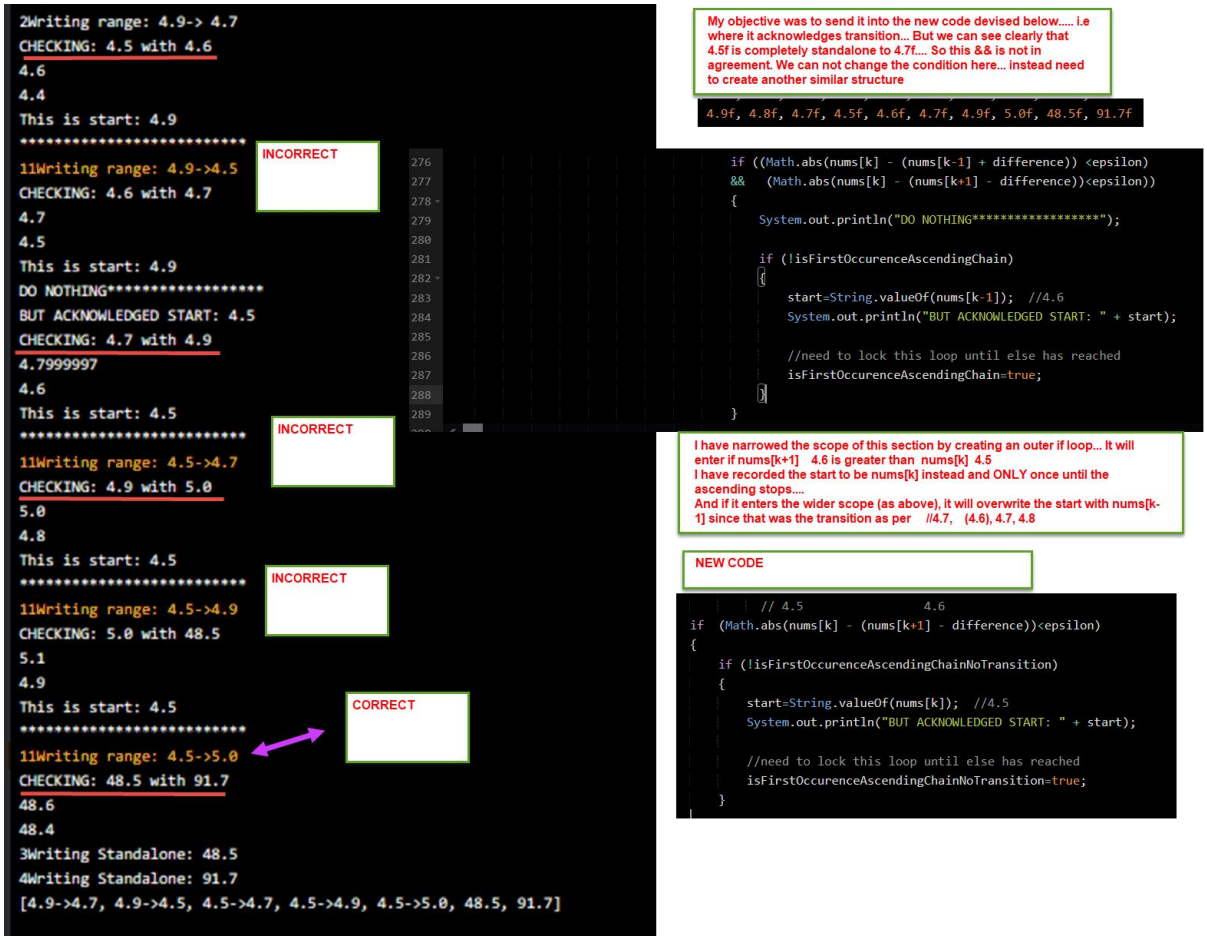



```
float[] nums = {
//4.9f, 4.8f, 4.7f, 4.6f, 4.7f, 4.8f, 4.9f, 5.0f, 48.5f, 91.7f  //in test documentation
  4.9f, 4.8f, 4.7f, 4.5f, 4.6f, 4.7f, 4.9f, 5.0f, 48.5f, 91.7f
```

I immediately found flaw in my logic.. I have remediated this.

The excellent news below is that it has not performed a standalone write for 5.0f or 4.5f, so I am definitely getting closer...

```
[4.9->4.7, 4.9->4.5, 4.5->4.7, 4.5->4.9, 4.5->5.0, 48.5, 91.7]
```




TEST CASE: I will now try the two test cases and see if they both pass.

```
float[] nums = {
//4.9f, 4.8f, 4.7f, 4.6f, 4.7f, 4.8f, 4.9f, 5.0f, 48.5f, 91.7f  //in test documentation
  4.9f, 4.8f, 4.7f, 4.5f, 4.6f, 4.7f, 4.9f, 5.0f, 48.5f, 91.7f
```

```
[4.9->4.7, 4.5->4.7, 4.9->5.0, 48.5, 91.7]
```

This has stayed untouched...

```
float[] nums = {
4.9f, 4.8f, 4.7f, 4.6f, 4.7f, 4.8f, 4.9f, 5.0f, 48.5f, 91.7f  //in test documentation
 // 4.9f, 4.8f, 4.7f, 4.5f, 4.6f, 4.7f, 4.9f, 5.0f, 48.5f, 91.7f
```

```
[4.9->4.6, 4.6->5.0, 48.5, 91.7]
```

So this is positive news given level of tweaking..

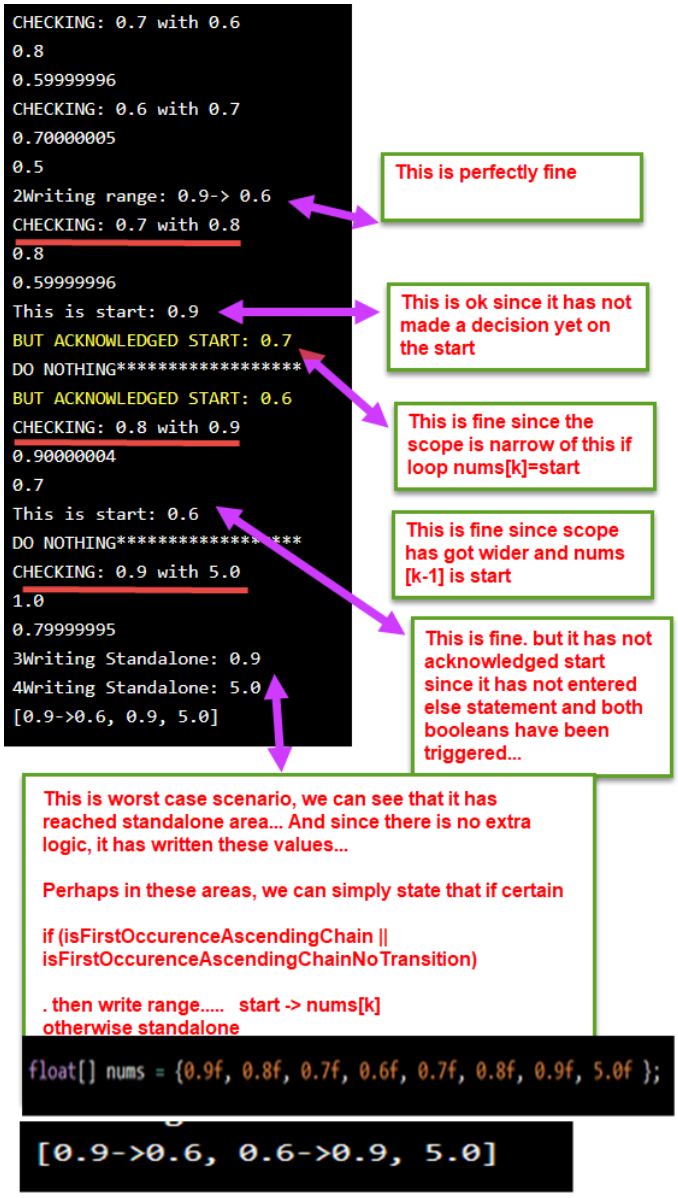
Now it is massively in my interest to run through all the test cases in the program before I explore with 1000 digits...

TEST CASE:  FAIL

```
float[] nums = {0.9f, 0.8f, 0.7f, 0.6f, 0.7f, 0.8f, 0.9f, 5.0f };
```

```
[0.9->0.6, 0.9, 5.0]
```

I am going to investigate quite quickly since its an error..
I expected this to be similar to transition example above...
Once again I have followed the debugging




```
152                 }
153                 else
154                 {
155                     end = String.valueOf(nums[k]);
156
157                     if (isFirstOccurenceAscendingChainNoTransition || isFirstOccurenceAscendingChain)
158                     {
159                         end=String.valueOf(nums[k]);
160                 sm.add(start+"->"+end);
161                 System.out.println("27Writing range: " + start + "-> " + end);
162                 //sm.add(String.valueOf(nums[k+1]));
163
164                     }
165                     else
```

NEW CODE

TEST CASE:

I will now try all three small test cases again that deal with ascending and descending by 0.1

```
float[] nums = {
//4.9f, 4.8f, 4.7f, 4.6f, 4.7f, 4.8f, 4.9f, 5.0f, 48.5f, 91.7f  //in test documentation,  used to test the transition 4.6
 4.9f, 4.8f, 4.7f, 4.5f, 4.6f, 4.7f, 4.9f, 5.0f, 48.5f, 91.7f  //in test documentation , used to test 4.5f standalone merging upwards into asce
//0.9f, 0.8f, 0.7f, 0.6f, 0.7f, 0.8f, 0.9f, 5.0f   //final test case in documentation where I resolved end of range with standalone....
```

It is a positive sign, now I will try the 1000 digit ChatGPT array.
In effect it was this that assisted me ascertain errors.

I also need to be extremely careful when I check the debugging for any instances in which it adds standalone numbers...
It might be the same tweak that I completed above....
This will be a slow process for me to check if I am 100% satisfied...

Unfortunately there are errors in the initial part.... Good news is that it is related to the direction of the written start -> end
And it is missing numbers only in the case of a sequence.

The first thing I did was reduce the dataset since working with 1000 numbers is extremely difficult...

We can see that although I went long extent to fix the issues, the original epsilon code was consistent in some areas which the Fixed code failed... However I will continue moving forward since I feel I am on the right track... I believe I have a good sample of information to finish this challenge.

ORIGINAL DATA

4.9f, 4.8f, 4.7f, 4.6f, 4.7f, 4.8f, 4.9f, 5.0f, 48.5f, 91.7f, 82.9f,
57.6f, 57.5f, 57.4f, 57.3f, 57.2f, 57.3f, 57.4f, 57.5f, 57.6f,
26.1f, 25.4f, 21.2f, 83.5f, **56.3f, 56.2f, 56.1f, 56.0f, 55.9f, 55.8f, 55.7f, 55.6f**, 13.9f, 35.7f,
54.8f, 54.7f, 54.6f, 54.5f,
54.6f, 54.7f, 54.8f, 54.9f, 49.9f, 9.6f, 86.0f,
77.7f, 77.8f, 77.9f, 78.0f, 78.1f,
**78.0f, 77.9f, 77.8f, 77.7f, (we can see this data is missing below)**
25.3f, 72.8f, 42.5f,

74.5f, 74.4f, 74.3f, 74.2f, 74.1f, 74.2f, 74.3f, 74.4f, 74.5f,
90.6f, 30.0f, 66.5f, 11.2f,
64.2f, 64.3f, 64.4f, 64.5f, 64.6f, 64.5f, 64.4f, 64.3f, 64.2f,

---

FIXED CODE
[4.9->4.6, 4.6->5.0, 48.5, 91.7, 82.9, 57.6->57.2, 57.2->57.6, 26.1, 25.4, 21.2, 83.5, 56.3->55.6, 13.9, 35.7, 54.8->54.5, 54.5->54.9, 49.9, 9.6, 86.0,

78.1->77.7 (we can see the direction is back to front to the above), but it is missing the next range above (see bigger font)
25.3, 72.8, 42.5, 74.5->74.1, 74.5->74.5 **(we can see this is back to front to above, and why is start 74.5 and not 74.1).**
90.6, 30.0, 66.5, 11.2,
64.6->64.2 (we can see that it is back to front) and also it is missing 64.2 – 64.5)
18.8, 49.3, 60.3, 51.1,

---

This is the original epsilon code

**This code never had issue of the reversing ranges as above..**

**Red areas are areas with issues**

[**4.9->4.6, 4.6->4.7, 4.7->4.8, 4.8->4.9, 4.9->5.0**, 48.5, 91.7, 82.9,
**57.6->57.2, 57.2->57.3, 57.3->57.4, 57.4->57.5, 57.5->57.6**, 26.1, 25.4, 21.2, 83.5,
**56.3->55.6,** 13.9, 35.7, **54.8->54.5**, **54.5->54.6, 54.6->54.7, 54.7->54.8, 54.8->54.9**,
49.9, 9.6, 86.0,
77.7 (this is the first real error, all other areas above are due to no collapsing. Standalone entry not required.. This has been fixed for example...)
77.7->77.8, 77.8->77.9, 77.9->78.0,
78.1->77.7 (we can see it should include the overlap with 78.0 but it has not done this... Even in my fixed code, it has dropped this, which I will investigate).

25.3, 72.8, 42.5,
**74.5->74.1, 74.1->74.2, 74.2->74.3, 74.3->74.4, 74.4->74.5**, 90.6, 30.0, 66.5, 11.2,
64.2, **64.2->64.3, 64.3->64.4, 64.4->64.5,**
64.6->64.2 (again start should be 64.5, this is an area I have fixed in code above).
18.8, 49.3,
---

# TEST CASE:

Currently

FIXED CODE
[4.9->4.6, 4.6->5.0, 48.5, 91.7, 82.9, 57.6->57.2, 57.2->57.6, 26.1, 25.4, 21.2, 83.5, 56.3->55.6, 13.9, 35.7, 54.8->54.5, 54.5->54.9, 49.9, 9.6, 86.0,

78.1->77.7 (we can see the direction is back to front to the above), but it is missing the next range above (see bigger font)

54.8f, 54.7f, 54.6f, 54.5f,
54.6f, 54.7f, 54.8f, 54.9f, 49.9f, 9.6f, 86.0f,
77.7f, 77.8f, 77.9f, 78.0f, 78.1f,
**78.0f, 77.9f, 77.8f, 77.7f, (we can see this data is missing below)**

EXPECTED BASED ON ORIGINAL DATA

EXPECTED BASED ON ORIGINAL DATA

FIXED CODE

[4.9->4.6, 4.6->5.0, 48.5, 91.7, 82.9, 57.6->57.2, 57.2->57.6, 26.1, 25.4, 21.2, 83.5, 56.3->55.6, 13.9, 35.7, 54.8->54.5, 54.5->54.9, 49.9, 9.6, 86.0,

78.1->77.7 (we can see the direction is back to front to the above), but it is missing the next range above (see bigger font)

Ascending sequence

```
        // 4.5                4.6       narrow scope

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

So I need to go into my debugging in which it completes CHECK 86.0 with 77.7

```
CHECKING: 86.0 with 77.7
86.1
85.9
6Writing Standalone: 86.0
CHECKING: 77.7 with 77.8
77.799995
77.6
This is start: 86.0
1BUT ACKNOWLEDGED START: 77.7
CHECKING: 77.8 with 77.9
77.9
77.700005
This is start: 77.7
DO NOTHING*****************
2BUT ACKNOWLEDGED START: 77.7
CHECKING: 77.9 with 78.0
78.0
77.8
This is start: 77.7
DO NOTHING*****************
CHECKING: 78.0 with 78.1
78.1
77.9
This is start: 77.7
DO NOTHING*****************
CHECKING: 78.1 with 78.0
78.2
78.0
Establishing start: 78.1
CHECKING: 78.0 with 77.9
78.1
77.9
```

This is fine

This is fine, but why has it flipped this with end above? To be investigated.
It has entered here since the narrow scope recognises 77.7<77.8
So all OK

We can NOW SEE FIRST TIME.. It is not a case of writing information back to front.. We expected at this point for it to have performed Write (77.8->78.1). But it has missed this...

And when it has completed the below, it has written.....

```
CHECKING: 78.1 with 78.0
78.2
78.0
Establishing start: 78.1
CHECKING: 78.0 with 77.9
78.1
77.9
CHECKING: 77.9 with 77.8
78.0
77.8
CHECKING: 77.8 with 77.7
77.9
77.700005
CHECKING: 77.7 with 25.3
77.799995
77.6
2Writing range: 78.1-> 77.7
CHECKING: 25.3 with 72.8
25.4
```

What we need is that like similar, one or both of the booleans will be in true state. If this is the case, we need to write start->end once it establishes the start here.....

isFirstOccurenceAscending Chain
isFirstOccurenceAscending ChainNoTransition

```
128
129             if (counter==0)
130             {
131                 start=String.valueOf(nums[k]);
132                 System.out.println("Establishing start: " + start);
133
134                 if (isFirstOccurenceAscendingChainNoTransition || isFirstOccurenceAscendingChain)
135                 {
136                     end=String.valueOf(nums[k-1]);
137                     sm.add(start+"->"+end);
138                     System.out.println("28Writing range: " + start + "-> " + end);
139                 }
```

Once implemented the above code, the output is as follows:

```
4Writing Standalone: 42.5
[86.0, 78.1->78.0, 78.1->77.7, 25.3, 25.3->72.8, 42.5]
```

I was expecting 77.1f => 78.1

Analysing my logic:

```
128
129             if (counter==0)
130             {
131                 start=String.valueOf(nums[k]);
132                 System.out.println("Establishing start: " + start);
133
134                 if (isFirstOccurenceAscendingChainNoTransition || isFirstOccurenceAscendingChain)
135                 {
136                     end=String.valueOf(nums[k-1]);
137                     sm.add(start+"->"+end);
138                     System.out.println("28Writing range: " + start + "-> " + end);
139                 }
```

```
CHECKING: 78.1 with 78.0
78.2
78.0
Establishing start: 78.1
CHECKING: 78.0 with 77.9
78.1
77.9
CHECKING: 77.9 with 77.8
78.0
77.8
CHECKING: 77.8 with 77.7
77.9
77.700005
CHECKING: 77.7 with 25.3
77.799995
77.6
2Writing range: 78.1-> 77.7
CHECKING: 25.3 with 72.8
25.4
```

I can see few errors. This piece of code should have been before it configured the new start above.
Also, we are here in the code. We are passed here because the numbers are swinging from ascending to descending... So the end has to be nums[k] = 78.1

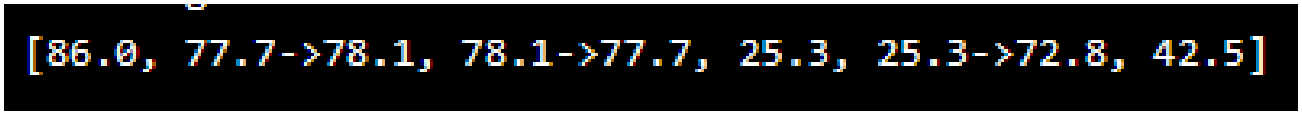
```
This is start: 77.7
DO NOTHING**********
CHECKING: 78.0 with 78.1
78.1
77.9
This is start: 77.7
DO NOTHING**********
CHECKING: 78.1 with 78.0
78.2
```

I have amended the code as below:

```
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);
```

And all seems well.. I do not think I have jeopardised any other test cases since it highly controlled modifications...



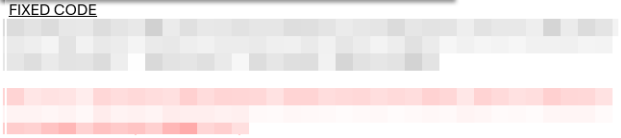

Now, I will place my focus on remediating the next section that failed in my new code...
I will create another test case to just focus on this section....

TEST CASE:

```
25.3f, 72.8f,42.5f, 74.5f, 74.4f, 74.3f, 74.2f, 74.1f, 74.2f, 74.3f, 74.4f,74.5f
```

It can be seen that my previous analysis was similar to the last one, but I now expect different outcome

FIXED CODE



25.3, 72.8, 42.5, 74.5->74.1, 74.5->74.5 **(we can see this is back to front to above, and why is start 74.5 and not 74.1).**

It now all seems to be resolved..... But why is 75.5 excluded from the last summary range, it appears to be a blatant continuation

```
4Writing Standalone: 74.5
[25.3, 72.8, 42.5, 74.5->74.1, 74.1->74.4, 74.5]
```

It can be seen that I speculated earlier that the writing standalone areas of the code are required to have some knowledge of the ascending / descending and it has proved to be the case again.. For now, I am just going to fix this area in 27Writing range..

```
CHECKING: 74.4 with 74.5
74.5
74.3
27Writing range: 74.1-> 74.4
4Writing Standalone: 74.5
[25.3, 72.8, 42.5, 74.5->74.1, 74.1->74.4, 74.5]
```

It is in the same area of code which I implemented, but really its hard to understand why it is differentiated from CHECK 74.3 -> 74.4 unless I understand it again

```
176                     else
177                     {
178                         end = String.valueOf(nums[k]);
179
180                         if (isFirstOccurenceAscendingChainNoTransition || isFirstOccurenceAscendingChain)
181                         {
182                             end=String.valueOf(nums[k]);
183                     sm.add(start+"->"+end);
184                     System.out.println("27Writing range: " + start + "-> " + end);
185                     //sm.add(String.valueOf(nums[k+1]));
186
```

```
CHECKING: 74.1 with 74.2
74.2
74.0
2Writing range: 74.5-> 74.1
```

This is fine, it has performed closure

The data, all fine before this

```
74.1f, 74.2f, 74.3f, 74.4f,74.5f
```

```
CHECKING: 74.2 with 74.3
74.299995
74.1
This is start: 74.5
1BUT ACKNOWLEDGED START: 74.2
DO NOTHING*****************
2BUT ACKNOWLEDGED START: 74.1
CHECKING: 74.3 with 74.4
74.4
74.200005
This is start: 74.1
DO NOTHING*****************
CHECKING: 74.4 with 74.5
74.5
74.3
27Writing range: 74.1-> 74.4
4Writing Standalone: 74.5
[25.3, 72.8, 42.5, 74.5->74.1, 74.1->74.4, 74.5]
```

This is fine. We know one or both booleans are set to true since it has not entered this area of code:

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

I now understand the change that is required...
With this data, we can see that 5.0f breaks the chain so nums[k] was correct for start and 0.9f was correct end to terminate the range... And then write standalone...

```
0.9f, 0.8f, 0.7f, 0.6f, 0.7f, 0.8f, 0.9f, 5.0f
```

But for this data, we need to check if nums[k+1] is greater than nums[k] within difference and tolerance.... If this is the case....
end=nums[k+1]
sm.add(start->end).
Since this has processed 74.5f, we need to narrow down the scope of the following piece of code.....
I will unify both areas with another boolean

```
74.1f, 74.2f, 74.3f, 74.4f,74.5f
```

```
196                     }
197                         start=String.valueOf(nums[k+1]);
198                         System.out.println("4Writing Standalone: " + start);
199                         sm.add(start);
```

We can see it has fixed the issue.....

```
35Writing range: 74.1-> 74.5
[25.3, 72.8, 42.5, 74.5->74.1, 74.1->74.5]
```

I will quickly try all test cases below

TEST CASE:

```
float[] nums = {
//4.9f, 4.8f, 4.7f, 4.6f, 4.7f, 4.8f, 4.9f, 5.0f, 48.5f, 91.7f  //in test documentation,  used to test the transit
//4.9f, 4.8f, 4.7f, 4.5f, 4.6f, 4.7f, 4.9f, 5.0f, 48.5f, 91.7f  //in test documentation , used to test 4.5f standa
//0.9f, 0.8f, 0.7f, 0.6f, 0.7f, 0.8f, 0.9f, 5.0f   //test case in documentation where I resolved end of range with
86.0f,77.7f, 77.8f, 77.9f, 78.0f, 78.1f,78.0f, 77.9f, 77.8f, 77.7f, 25.3f, 72.8f,  42.5f   //this will be section
//25.3f, 72.8f,42.5f, 74.5f, 74.4f, 74.3f, 74.2f, 74.1f, 74.2f, 74.3f, 74.4f,74.5f   //this will be section of the
```

This has failed, here are my explanations below:

```
[86.0, 77.7->78.1, 78.1->77.7, 25.3, 25.3->72.8, 42.5]
```

Issue from here onwards

```
CHECKING: 77.7 with 25.3
77.799995
77.6
2Writing range: 78.1-> 77.7
CHECKING: 25.3 with 72.8
25.4
25.199999
6Writing Standalone: 25.3
CHECKING: 72.8 with 42.5
72.9
72.700005
27Writing range: 25.3-> 72.8
4Writing Standalone: 42.5
[86.0, 77.7->78.1, 78.1->77.7, 25.3, 25.3->72.8, 42.5]
```

DATA

```
, 25.3f, 72.8f, 42.5f
```

Issue starts here

This is ok

We wrote this area of code when dealing with the following

```
0.6f, 0.7f, 0.8f, 0.9f, 5.0f
```

```
if (isFirstOccurenceAscendingChainNoTransition || isFirstOccurenceAscendingChain)
{
    //shows next is ascending
              //72.8       //42.5
    //we wrote this section of code when
    if (Math.abs(nums[k] - (nums[k+1] - difference)) <epsilon)
    {
        end=String.valueOf(nums[k+1]);
          sm.add(start+"->"+end);
          System.out.println("35Writing range: " + start + "-> " + end);
          hasWrittenLastNumber=false;
    }
    else
    {
        end=String.valueOf(nums[k]);
        sm.add(start+"->"+end);
        System.out.println("27Writing range: " + start + "-> " + end);
        //sm.add(String.valueOf(nums[k+1]));
    }
```

We can see a valid case to enter into here due to the transitional summaries.. But for the case above, 25.3 has no connection to previous number... So it suggests boolean values are not cleared and set back to false

I have implemented this.... My instinct tells me that in every standalone situtation, I need to reset the variables to false

```
else
{
    start = String.valueOf(nums[k]);
    sm.add(start);
    System.out.println("6Writing Standalone: " + start);
    isFirstOccurenceAscendingChain=false;
    isFirstOccurenceAscendingChainNoTransition=false;
}
```

```
CHECKING: 0.9 with 5.0
1.0
0.79999995
27Writing range: 0.6-> 0.9
4Writing Standalone: 5.0
[0.9->0.6, 0.6->0.9, 5.0]
```

```
[0.9->0.6, 0.9, 5.0]
```

This was before the above fix applied

```
[0.9->0.6, 0.6->0.9, 5.0]
```

This is after fix applied

It can be seen that test case now passes.

```
[86.0, 77.7->78.1, 78.1->77.7, 25.3, 72.8, 42.5]
```

I identified that potentially in every area where standalone number is written, I should endeavour to perform. But I have no other test cases providing this otherwise. So it is not considered a control change.

isFirstOccurenceAscendingChain=false;
isFirstOccurenceAscendingChainNoTransition=false;

There is one test that was continuously playing on my mind during the development phase... This was the presence of identical standalone numbers...
So I have quickly added some at the start of one of my test cases and at the end.....

TEST CASE:

```
//this is chatGPT extract
4.9f,4.9f,4.8f, 4.7f, 4.6f, 4.7f, 4.8f,
);
```

```
[4.9, 4.9->4.6,
```

We can see that it has not merged the 4.9 with 4.9.

I also tried another case:

```
4.9f,4.9f,4.8f,4.8f, 4.7f, 4.6f,
);
```

```
[4.9, 4.9->4.8, 4.8->4.6,
```

And it can be seen that this is perhaps the easiest fix on paper... Where it performs check of consecutive numbers, it it finds nums[k] and nums[k+1] are equal with respect to difference and epsilon, simply move to next iteration...
Logic suggests it would do this for ascending and descending checks....

And after all this mental pounding, I believe I had overcomplicated the thinking process. It merely required the below.. We are not concerned with differences, we are only concerned if the numbers are equal and there is no error involved in this process!!

```
87          for (int k=0; k<nums.length-1;k++)
88          {
89              if (k==0)
90              {
91                  start=String.valueOf(nums[k]);
92              }
93
94              if (nums[k]!=nums[k+1])
```

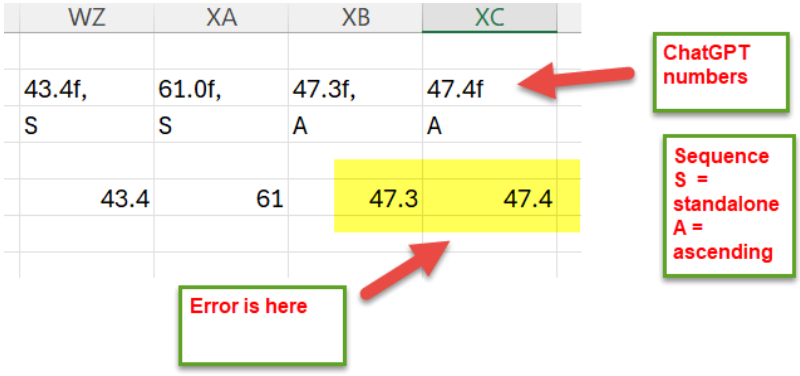
When getting ChatGPT to generate the sample it's the only input I did not specify, allow repetitive standalone numbers...

For now, I am trying all my test cases again... And if they all pass, I will run through the 1000 number case ChatGPT extract. But I will pay extra attention to writing standalone events...

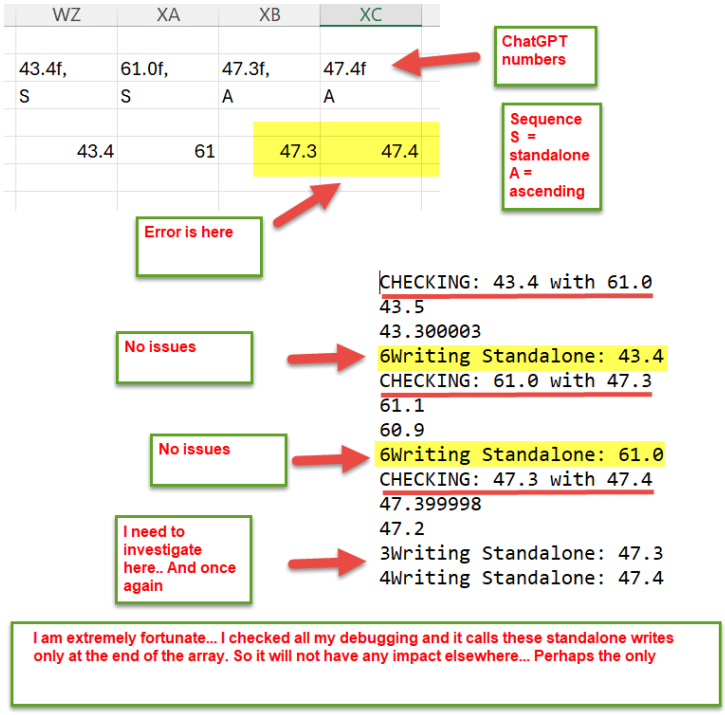
TEST CASE:

Fortunately I have moved fairly far into the numbers. I believe at this point, it requires human intervention and best technique is to present the information in an Excel spreadsheet to have full accountability and avoid human error at this critical stage...

And as I finished inputting the entries into Excel, I spotted one mistake at the end...




I am not entirely surprised that there was an error here, since there was so much negotiation done in my code... But I can clearly see something that it might be related to those two boolean variables... Although it is difficult to tell unless I look at the debugging aspect again...




However before I address above, I wanted to try few more arrays.. This has created a massive doubt in my head..

TEST CASE: (3 ascending numbers no issues)...

```
43.4f, 47.2f, 47.3f, 47.4f  /
```

```
[43.4, 47.2->47.4]
```

TEST CASE: 2 ascending numbers

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

```
47.4->47.3, 47.4->47.4,
```

This will be my priority.....

If I fail to find answer, I can do something analyse if X->X and use previous end..

But I do not want to enter this path.....

```
CHECKING: 47.4 with 47.3
47.5
47.300003
Establishing start: 47.4
CHECKING: 47.3 with 47.4
47.399998
47.2
3Writing range: 47.4-> 47.3
[47.4->47.3, 47.4]


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

It is surprising that if I run this test case independently, the outcome is different to below.. I will examine it from both perspectives. There is every danger now that I might disrupt my existing logic

```
47.4->47.3, 47.4->47.4,
```

It has taken me into an area of code which I have not touched in this documentation, which suggests it has hit a conflict... I will study this area of code line by line to familiarize myself first

I created this code and it fixed issues for several failed cases...

```
//here we know there is only one value after current one... we need to check if it is descending or ascending
//and then formulate the closure
//

//it needs to pass one of these scenarios, no effect on start or end
//checking 47.3 with 47.4
            //descending                                        //ascending
if ( (Math.abs(nums[k] - (nums[k+1] - difference)) <epsilon)   ||
     (Math.abs(nums[k] - (nums[k+1] + difference)) <epsilon))
{
    start=String.valueOf(nums[k]);
    end=String.valueOf(nums[k+1]);
    sm.add(start+"->"+end);
    System.out.println("49Writing range: " + start + "-> " + end);

}
//NO IDEA WHY THIS WAS INCLUDED
//sm.add(String.valueOf(nums[k+1]));
```

My next straight forward test case will be reflection:

TEST CASE:

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

```
[47.3, 47.4->47.3]
```

I will just follow the screen output and performed this code update.

```
//descending
//I am extremely hesitant to start performing and ascending check here
//since I have no idea if my code has entered here before due to lack of screen outputs
//we know for an array such as  (47.3), 47.4 and 47.3, it will not enter here on the 1st element
if (Math.abs(nums[k] - (nums[k+1] + difference)) <epsilon)
//if ((nums[k]-difference)==(nums[k+1]))
{
    start=String.valueOf(nums[k]);
    counter++;
}
else   //since confident code has not entered here, I will place logic here
{
    //ascending
    if (Math.abs(nums[k] - (nums[k+1] - difference)) <epsilon)
    {
        start=String.valueOf(nums[k]);
        end=String.valueOf(nums[k+1]);
        sm.add(start+"->"+end);
        System.out.println("58Writing range: " + start + "-> " + end);
    }
    else
    {
    System.out.println("5Writing standalone: " + start);
    sm.add(start);
    }
```

The output is now correct:

```
5Writing range: 47.4-> 47.3
[47.3->47.4, 47.4->47.3]
```

TEST CASE: I will now examine the next logical test case

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

```
CHECKING: 47.4 with 47.3
47.5
47.300003
Establishing start: 47.4
CHECKING: 47.3 with 47.5
47.399998
47.2
MUST BE HERE SO TRANSITION PART NOT PRESENT
HAT IS START: 47.4
3Writing range: 47.4-> 47.3
[47.4->47.3]


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

As usual I will examine this area and check why 4.5 has not been added. it will be quite straight forward. It just requires else section....

```
[47.4->47.3, 47.5]
```

Now I will try slight adjustment

TEST CASE:

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

It can be seen it has failed..
It has required several tweaking, see areas in code.

```
CHECKING: 47.3 with 47.4
47.399998
47.2
58Writing range: 47.3-> 47.4
CHECKING: 47.4 with 47.5
47.5
47.300003
3Writing Standalone: 47.4
4Writing Standalone: 47.5
[47.3->47.4, 47.4, 47.5]
```

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

Issue here. I have changed my code logically around 3writing standalone area and 4writing standalone area

```
[47.3->47.4, 47.4->47.5]
```

I now need to explore more advanced test cases which I suspect would error also.

TEST CASE:

```
47.3f, 47.2f, 47.3f, 47.4f  //i
```

```
CHECKING: 47.3 with 47.2
47.399998
47.2
Establishing start: 47.3
CHECKING: 47.2 with 47.3
47.3
47.100002
2Writing range: 47.3-> 47.2
CHECKING: 47.3 with 47.4
47.399998
47.2
98Writing range: 47.3-> 47.4
[47.3->47.2, 47.3->47.4]
```

This is ok

This is incorrect, it has not taken knowledge of 47.2... I will investigate

```
47.3f, 47.2f, 47.3f, 47.4f  //i
```

It is in the area of code that I changed recently.. in the else statement. It can be seen that my thought process is perfectly correct to write start-> end (47.3-> 47.4) but it also needs to check if nums[k-1] is less than nums[k].. I will add this logic...
If it is, the start will be nums[k-1]

```
    }     //it is here because it has not performed boolean due to extended ascending or descending
          //but it has to
          //my logic suggests that it can be merged with nums[k+1]
          //and the area of code where it performs 4Writing standalone can be modified
          //I will set another boolean in code below so that it ensures that 47.5 is entered again...
    else
    {

    if (temp=="")
    {
        //LATE CODE REMOVAL
        //System.out.println("3Writing Standalone: " + end);
        //sm.add(end);
        end=String.valueOf(nums[k+1]);
        start=String.valueOf(nums[k]);
        sm.add(start+"->"+end);
        System.out.println("98Writing range: " + start + "-> " + end);
        combineMerge=true;
        //At this moment in time,  I am not sure when to set the variable back to false!
    }
```

Added this code

```
if (temp=="")
{
    //if prev number is less than nums k
    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);
        //combineMerge=true;
    }
```

```
[47.3->47.2, 47.2->47.4]
```

TEST CASE:  Makes no difference to above.. Number descending in front of ascending is irrelevant..

```
47.4f,47.3f, 47.2f, 47.3f, 47.4f
```

```
[47.4->47.2, 47.4]
```

TEST CASE:

```
47.3f, 47.2f, 47.3f, 47.4f,47.5f
```

```
[47.3->47.2, 47.2->47.5]
```

```
CHECKING: 47.3 with 47.2
47.399998
47.2
Establishing start: 47.3
CHECKING: 47.2 with 47.3
47.3
47.100002
2Writing range: 47.3-> 47.2
CHECKING: 47.3 with 47.4
47.399998
47.2
This is start: 47.3
1BUT ACKNOWLEDGED START: 47.3
DO NOTHING**********
2BUT ACKNOWLEDGED START: 47.2
CHECKING: 47.4 with 47.5
47.5
47.300003
35Writing range: 47.2-> 47.5
[47.3->47.2, 47.2->47.5]
```

```
47.2f,47.3f,47.2f
```

```
CHECKING: 47.2 with 47.3
47.3
47.100002
58Writing range: 47.2-> 47.3
CHECKING: 47.3 with 47.2
47.399998
47.2
5Writing range: 47.3-> 47.2
[47.2->47.3, 47.3->47.2]
```

TEST CASE:  Ascending then descending  (3 descending numbers) – No issues

```
47.2f, 47.3f, 47.2f,47.1f,
```

```
47.2->47.3, 47.3->47.1,
```

I am now going to embed these into the main ChatGPT array...
I hope it will translate similarly....

```
91.7f, 47.2f,47.3f,47.2f,47.1f,      47.4f, 47.3f,47.4f,
```

```
91.7, 47.2->47.3, 47.3->47.1, 47.3->47.4,
```

I will now add it to the end

```
47.2f, 47.3f, 47.4f,   47.2f,47.3f,47.2f,47.1f
```

```
>41.0, 58.9, 27.4, 56.1, 88.7, 50.8->51.2, 51.2->50
47.3->47.2, 47.2->47.4, 47.2->47.3, 47.3->47.1]
```

At this moment, I believe all the additional changes that I made are not violating anything else...
But I will need to run the entire chatGPT original array again to confirm this...
I will need to fill the Excel spreadsheet again also.

```
98Writing range: 47.3-> 47.4
[4.9->4.6, 4.6->5.0, 48.5, 91.7, 82.9, 57.6->57.2, 57.2->57.6, 26.1, 25.4, 21.2, 83.5, 56.3->55.6,
8.0, 67.8, 41.3, 96.8, 72.3->71.5, 29.7, 81.9, 18.0, 46.4, 34.3->34.7, 34.7->34.3, 85.7, 14.3, 26.5
, 55.3->55.0, 55.0->55.5, 43.4, 61.0, 47.3->47.4]

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

GOOD NEWS IS THAT IT HAS FIXED THE PREVIOUS ERROR THAT APPEARED

I have compared all values and they are all the same except for last two values which have been fixed....

The only code amendment in which I was totally unsure of when to set back to false with this boolean:

```
  //At this moment in time,  I am not sure when to set the variable back to false!
  }
  combineMerge=true;
}
```

But I am quite certain that once it is set to true, it will be referenced in this section once only so it is irrelevant.

```
if (hasWrittenLastNumber && !combineMerge)
{
    start=String.valueOf(nums[k+1]);
    System.out.println("4Writing Standalone: " + start);
    sm.add(start);
    hasWrittenLastNumber=false;
}
```

One other final area not performed is the calculation of epsilon…
It was a nice experience to explore this manually, but I chose the initial sample of numbers and quick visual examination…

TEST CASE:  Incorporating logic to calculate epsilon and set the epsilon variable IN FUTURE