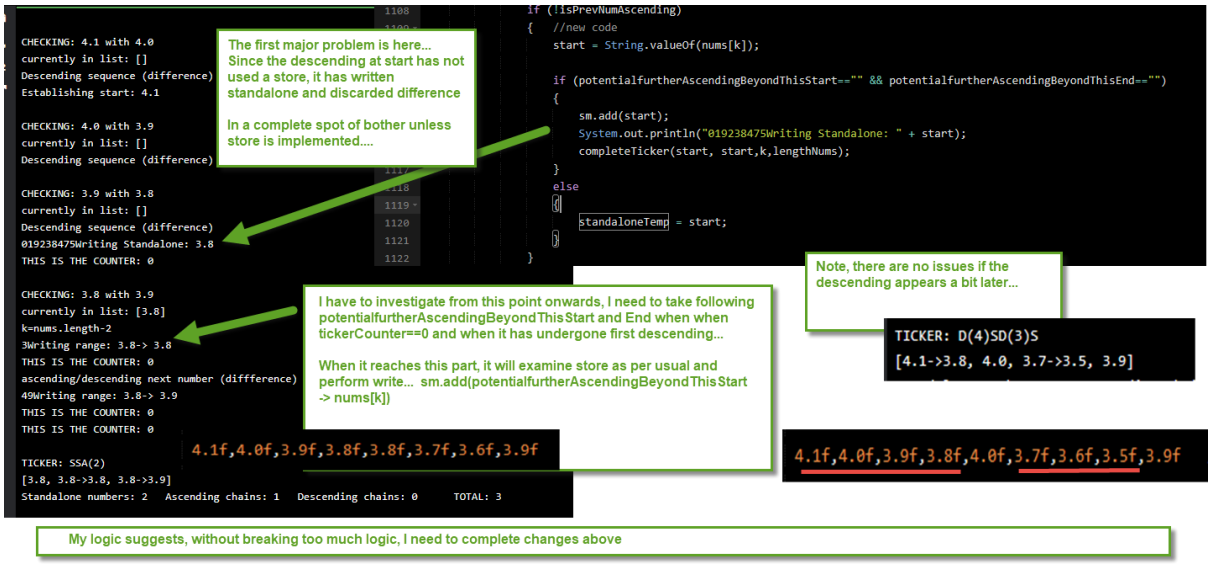
Examining this test case:

```
4.1f,4.0f,3.9f,3.8f,3.8f,3.9f
```

It has completely misinterpreted.. It looks to be related to having two entries for 3.8f

```
TICKER: SSA(2)
[3.8, 3.8->3.8, 3.8->3.9]
Standalone numbers: 2   Ascending chains: 1   Descending chains: 0      TOTAL: 3
```



```
413             if (Math.abs(nums[k] - (nums[k+1] + difference)) <epsilon)
414             {
415                 System.out.println("Descending sequence (difference)");
416                 descendingCounter++;
417 
418                 if (k==0)
419                 {
420                     end=String.valueOf(nums[k+1]);
421                     potentialfurtherAscendingBeyondThisStart = start;
422                     potentialfurtherAscendingBeyondThisEnd = end;
423                     System.out.println("-----------------------------------Stored start -> end: "
424                     + potentialfurtherAscendingBeyondThisStart + "->" + potentialfurtherAscendingBeyondThisEnd);
425                 }
```

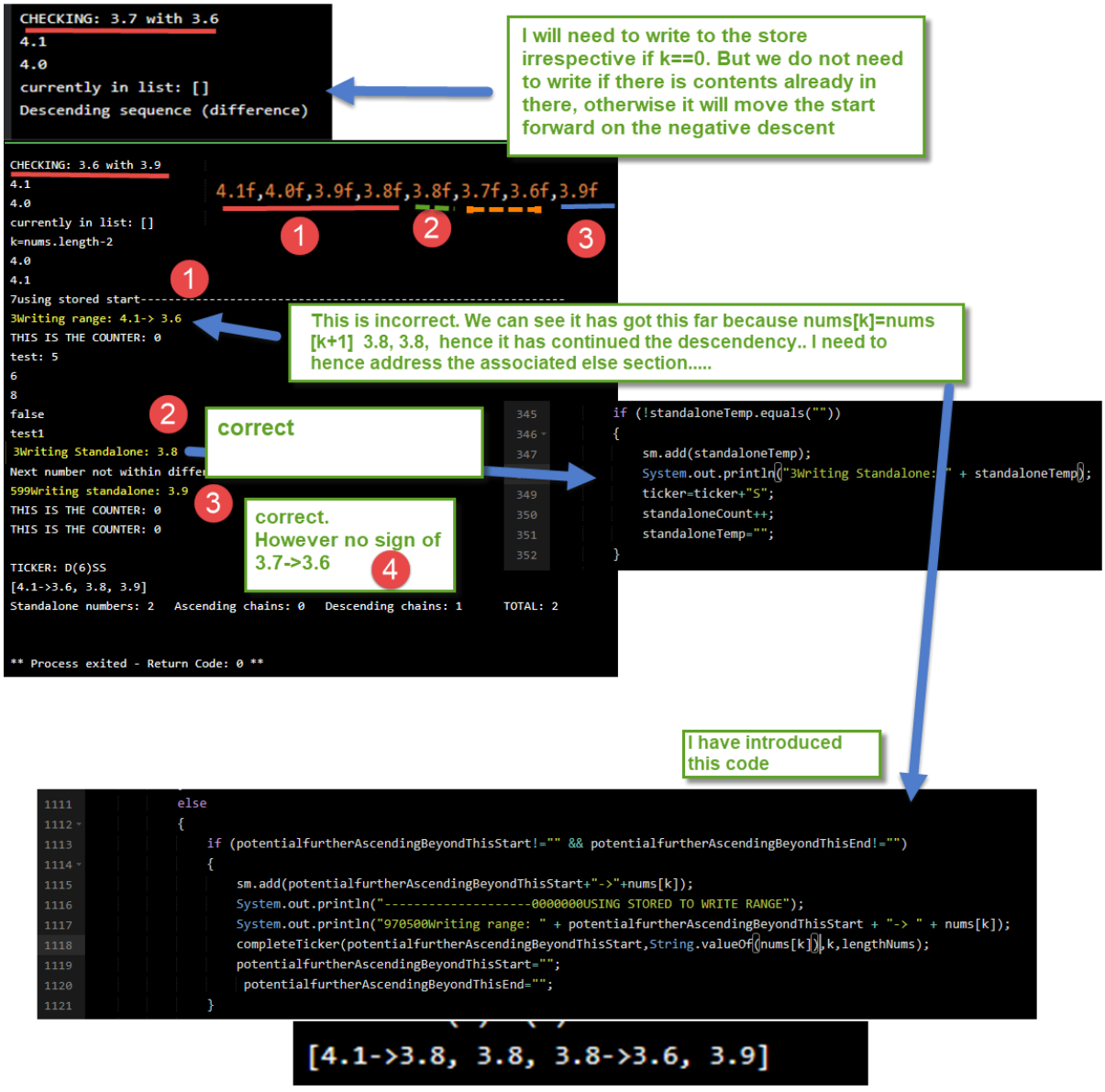
THIS IS THE NEW CODE

```
769 
770                     if (!(potentialfurtherAscendingBeyondThisEnd.equals("")) && !(potentialfurtherAscendingBeyondThisStart.equals("")))
771                     {
772                         System.out.println("7using stored start----------------------------------------------------------------");
773                         start=potentialfurtherAscendingBeyondThisStart;
774                         sm.add(start+"->"+end);
775                         System.out.println("3Writing range: " + start + "-> " + end);
776                         potentialfurtherAscendingBeyondThisStart="";
777                         potentialfurtherAscendingBeyondThisEnd="";
```

```
4.1f,4.0f,3.9f,3.8f,3.8f,3.7f,3.6f,3.9f

TICKER: D(6)SS
[4.1->3.6, 3.8, 3.9]
Standalone numbers: 2   Ascending chains: 0   Descending chains: 1      TOTAL: 2
```

I can see this has been missed out



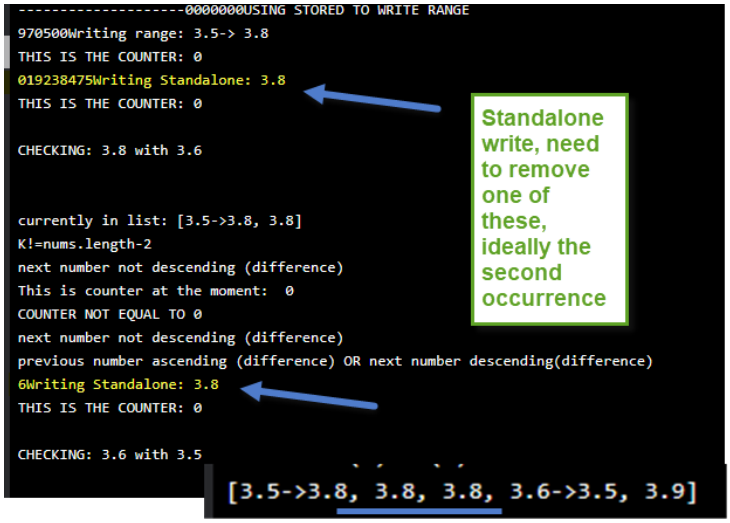
It looks to be resolved.

I suspect it will also work for ascending loop with repeat standalone.

I will quickly run a test.

TEST CASE:

```
3.5f,3.6f,3.7f,3.8f,3.8f,3.6f,3.5f,3.9f
```



I introduced following code

```
1013                                    else
1014                                    {
1015                                        if(!isStandaloneProcessed)
1016                                        {
1017                                            start = String.valueOf(nums[k]);
1018                                            sm.add(start);
1019                                            System.out.println("6Writing Standalone: " + start);
1020                                            completeTicker(start, start,k,lengthNums);
1021                                            isFirstOccurenceAscendingChain=false;
1022                                            isFirstOccurenceAscendingChainNoTransition=false;
1023                                        }
1024                                        isStandaloneProcessed=false;
```

```
1164                if (!isPrevNumAscending)
1165                {   //new code
1166                    start = String.valueOf(nums[k]);
1167
1168                    if (potentialfurtherAscendingBeyondThisStart=="" && potentialfurtherAscendingBeyondThisEnd=="")
1169                    {
1170                        sm.add(start);
1171                        System.out.println("019238475Writing Standalone: " + start);
1172                        isStandaloneProcessed=true;
1173                        completeTicker(start, start,k,lengthNums);
1174                    }
```

4 of 4

```
[3.5->3.8, 3.8, 3.8->3.6, 3.6->3.5, 3.9]
```

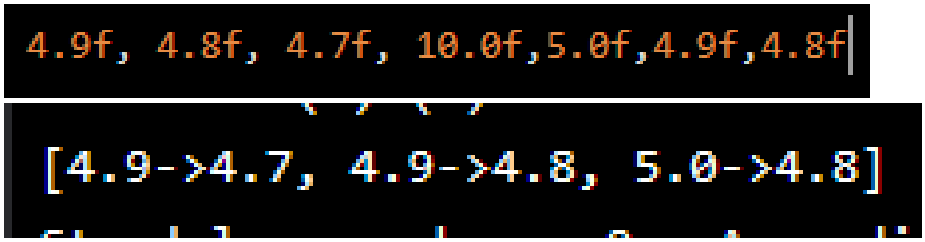
I will need to run through all my test cases again unfortunately…

I am now finding that PASS

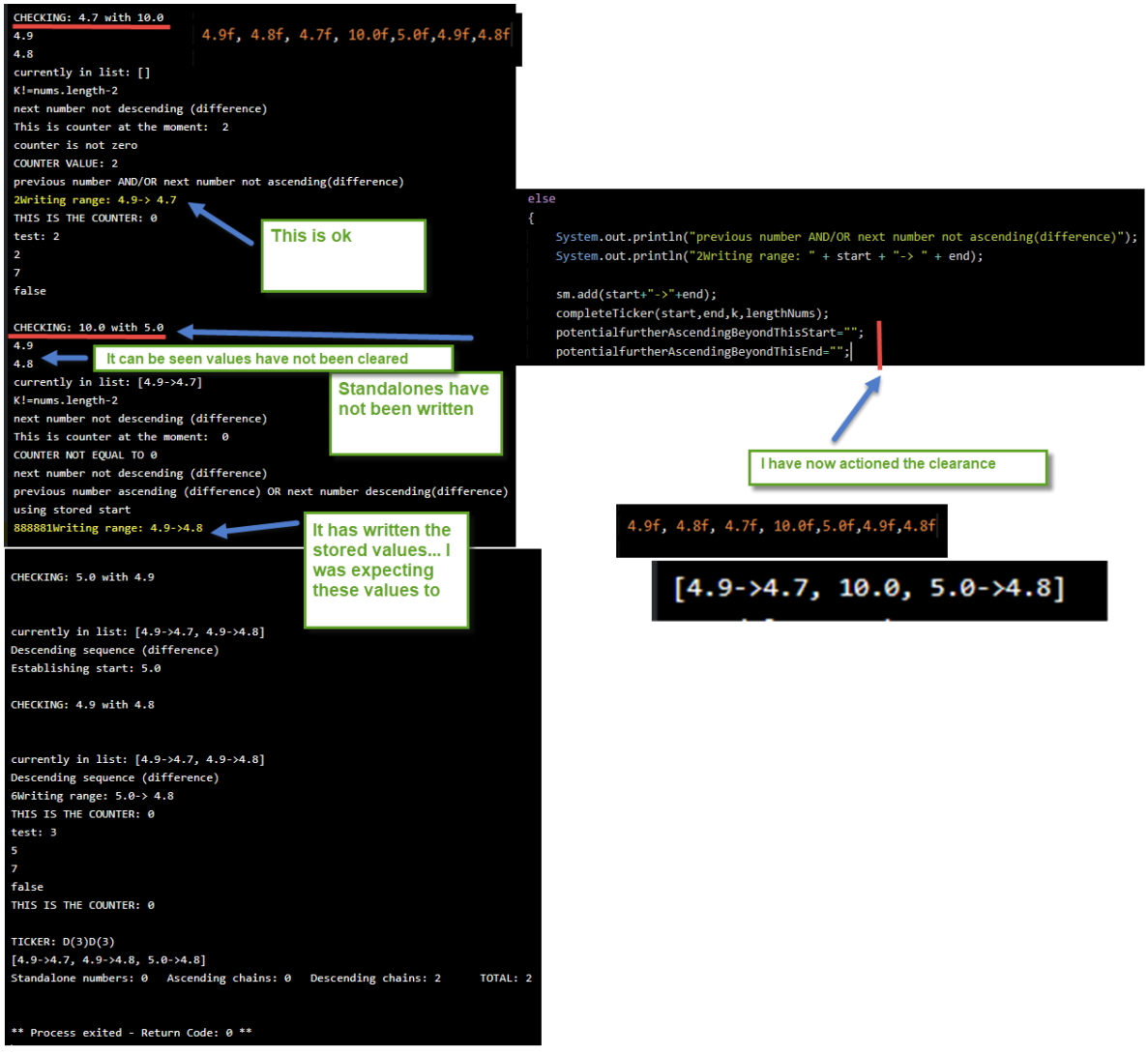
4.5f, 4.6f, 4.7f, 10.0f,5.0f,4.9f,4.8f //(no issues if 3 descending chains)

```
[4.5->4.7, 10.0, 5.0->4.8]
```

However the issue with descending going into standalone still persists:



I will examine the outputs as per usual:



I am just going to create another test case and extend the standalone numbers so that there is not a descending at the end...
And then I believe I can perform a full test.

TEST CASE:

```
4.9f, 4.8f, 4.7f, 10.0f,5.0f,74.9f,23.0f,4.8f
```

```
[4.9->4.7, 10.0, 5.0, 74.9, 23.0, 4.8]
```

TEST CASE:

I am however seeing this test case failure. This clearly suggests I need to understand the area surrounding ascending followed by multiple repeat standalone, then descending...

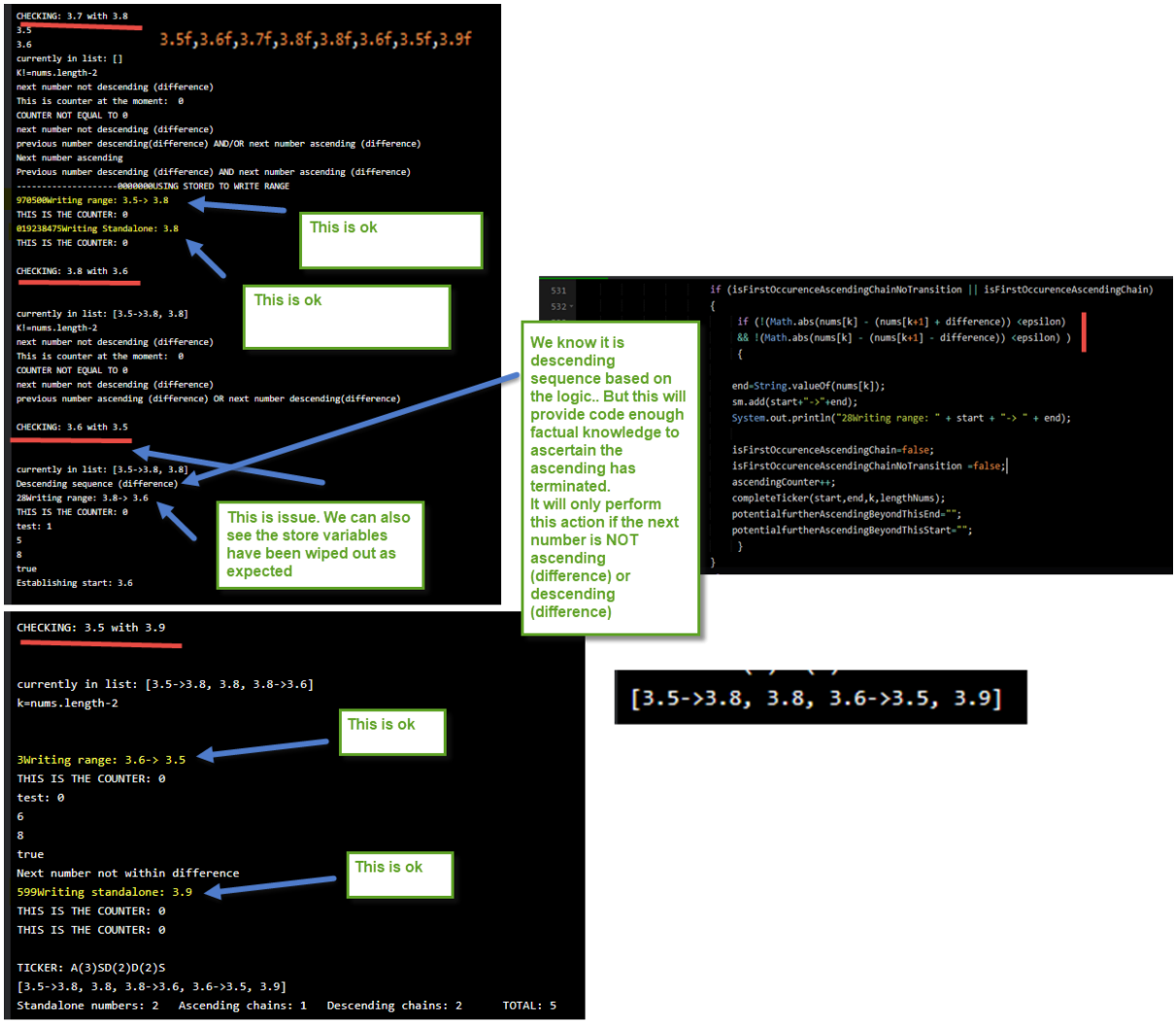
```
3.5f,3.6f,3.7f,3.8f,3.8f,3.6f,3.5f,3.9f
```

INCORRECT:

```
[3.5->3.8, 3.8, 3.8->3.6, 3.6->3.5, 3.9]
```

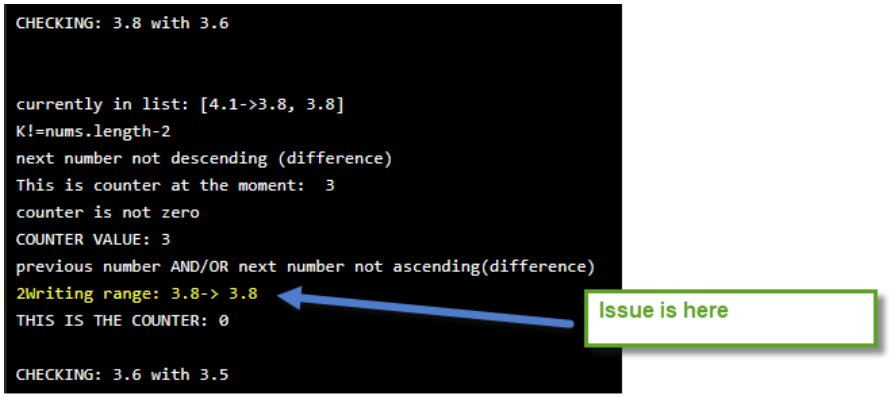
For me, it suggests that variables are not cleared for the store..

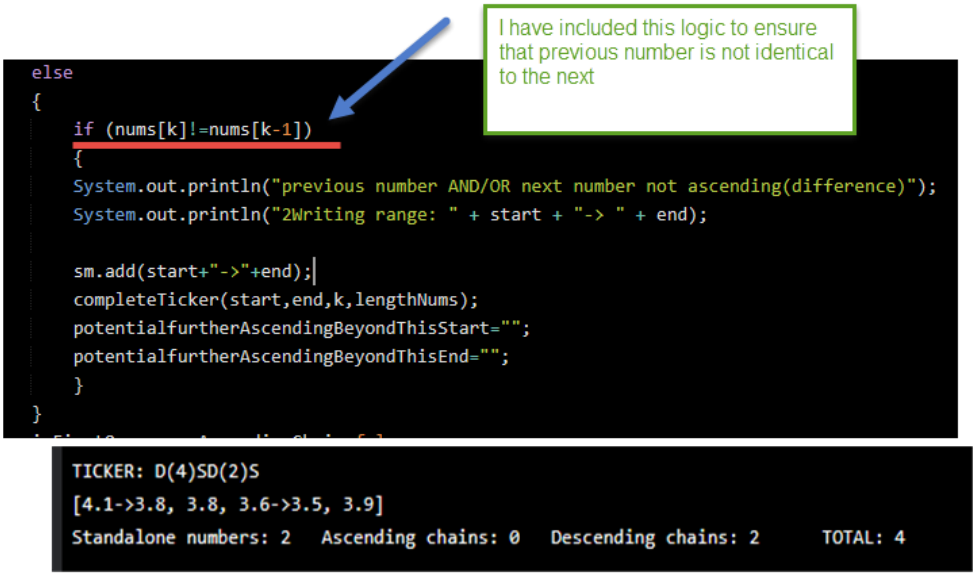
But I will take time as usual to investigate...



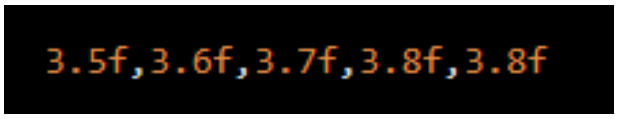
I am extremely desperate to bring this challenge to a close.
I will run through all the test cases again slowly...

This is an issue appearing in relation to having repeat standalone and its resolution...

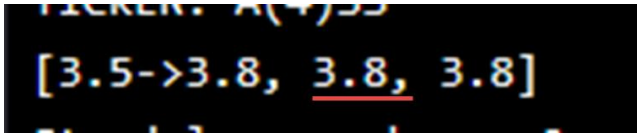




This is another scenario with issues:



Something suggests now since there are lots of standalone writes in the code, it might be a universal fix required... But for now, I will just focus on the area that causes isues...



```
1174                else
1175                {                                          3.5f,3.6f,3.7f,3.8f,3.8f
1176                    start = String.valueOf(nums[k]);
1177                    sm.add(start);
1178                    System.out.println("07774544Writing Standalone: " + start);
1179                    isStandaloneProcessed=true;
1180                    completeTicker(start, start,k,lengthNums);
1181                }
1182            }
1183            if (!isPrevNumAscending)
1184            {   //new code
1185                start = String.valueOf(nums[k]);
1186
1187                if (potentialfurtherAscendingBeyondThisStart=="" && potentialfurtherAscendingBeyondThisEnd=="")
1188                {
1189                    if (!isStandaloneProcessed)
1190                    {
1191                    sm.add(start);
1192                    System.out.println("019238475Writing Standalone: " + start);
1193                    isStandaloneProcessed=true;
1194                    completeTicker(start, start,k,lengthNums);
1195                    }
1196                    isStandaloneProcessed=false;
```

With this change, I need to be extremely careful there is no impact in other tests.....

```
[3.5->3.8, 3.8]
```

But we can see it is an inherent problem. I have included 2 x standalone 3.6 higher in the chain... So I need to accommodate for the repetition at every scenario...

```
3.5f,3.6f,3.6f,3.7f,3.8f,3.8f
```

```
1189                if (potentialfurtherAscendingBeyondThisStart=="" && potentialfurtherAscendingBeyondThisEnd=="")
1190                {
1191                    if (!isStandaloneProcessed)
1192                    {
1193                        //new code, keep track if too much logic
1194                        if (!(Math.abs(nums[k] - (nums[k+1] - difference)) <epsilon)
1195                        && !(Math.abs(nums[k] - (nums[k+1] + difference)) <epsilon)
1196                        && (nums[k]!=nums[k+1]))
1197                        {
1198                    sm.add(start);
1199                    System.out.println("019238475Writing Standalone: " + start);
1200                    isStandaloneProcessed=true;
1201                    completeTicker(start, start,k,lengthNums);
1202                        }
```

This willl ensure standalone if next number is not within sequence (descending or ascending) and also if not the same.....

```
[3.5->3.6, 3.6->3.8, 3.8]
```

But I do not think this is the solution that is required