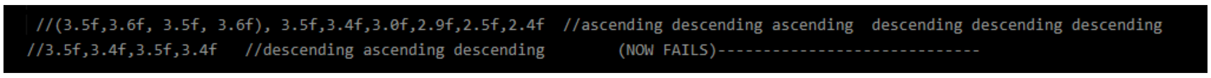
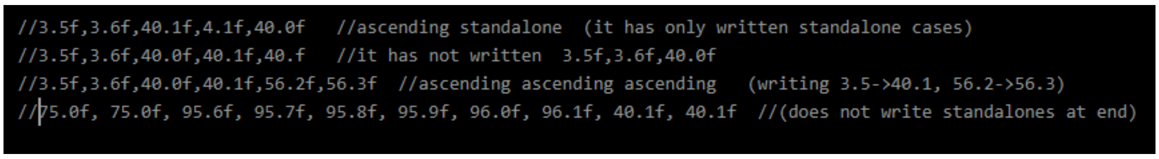
It now fails in these test cases on top, the reason is straight forward..
It happens in the section when it performs descending, ascending,descending
or ascending, descending, ascending... this is linked to my change above and I will try to factor more logic in it.
It is an area of code which heavily gets entered into.



```
 //(3.5f,3.6f, 3.5f, 3.6f), 3.5f,3.4f,3.0f,2.9f,2.5f,2.4f  //ascending descending ascending  descending descending descending
//3.5f,3.4f,3.5f,3.4f   //descending ascending descending        (NOW FAILS)----------------------------
```

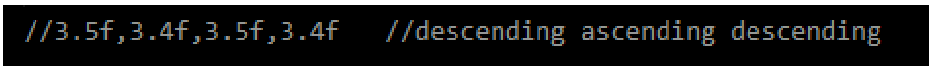
And the existing test cases still fail:



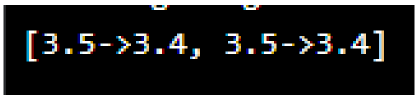
```
//3.5f,3.6f,40.1f,4.1f,40.0f   //ascending standalone  (it has only written standalone cases)
//3.5f,3.6f,40.0f,40.1f,40.f   //it has not written  3.5f,3.6f,40.0f
//3.5f,3.6f,40.0f,40.1f,56.2f,56.3f  //ascending ascending ascending   (writing 3.5->40.1, 56.2->56.3)
//75.0f, 75.0f, 95.6f, 95.7f, 95.8f, 95.9f, 96.0f, 96.1f, 40.1f, 40.1f  //(does not write standalones at end)
```

I feel I am fairly close to final outcome...

<u>TEST CASE:</u>

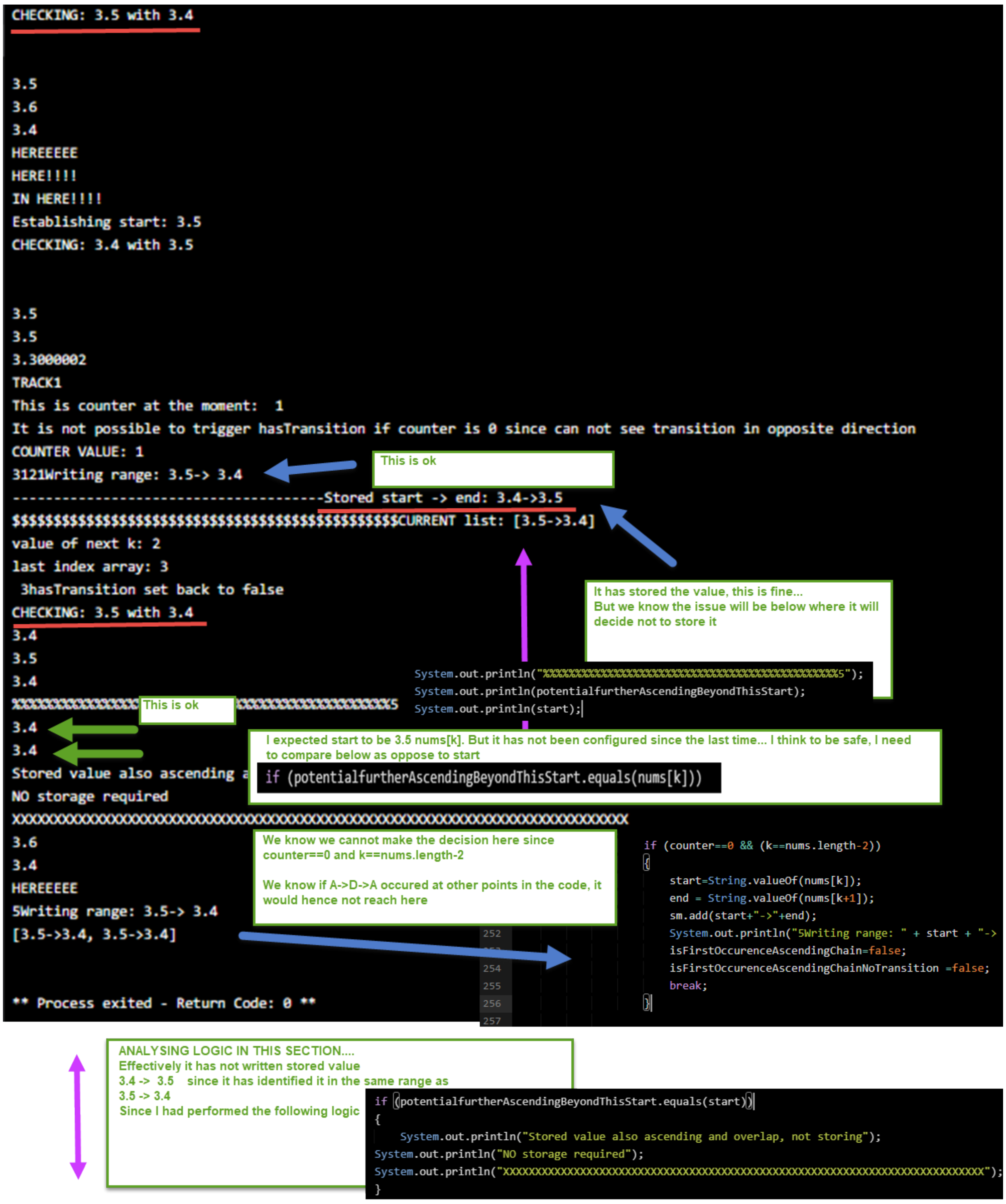


```
//3.5f,3.4f,3.5f,3.4f   //descending ascending descending
```



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

Without even looking at this too deeply, we can see it has decided to drop the
3.4 -> 3.5
I am sure it has entered in the section of code written



```
CHECKING: 3.5 with 3.4


3.5
3.6
3.4
HEREEEEE
HERE!!!!
IN HERE!!!!
Establishing start: 3.5
CHECKING: 3.4 with 3.5


3.5
3.5
3.3000002
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: 3.5-> 3.4
-------------------------------------Stored start -> end: 3.4->3.5
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$CURRENT list: [3.5->3.4]
value of next k: 2
last index array: 3
 3hasTransition set back to false
CHECKING: 3.5 with 3.4
3.4
3.5
3.4
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%5
3.4
3.4
Stored value also ascending a
NO storage required
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
3.6
3.4
HEREEEEE
5Writing range: 3.5-> 3.4
[3.5->3.4, 3.5->3.4]


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

This is ok

It has stored the value, this is fine...
But we know the issue will be below where it will decide not to store it

```
System.out.println("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%5");
System.out.println(potentialfurtherAscendingBeyondThisStart);
System.out.println(start);
```

This is ok

I expected start to be 3.5 nums[k]. But it has not been configured since the last time... I think to be safe, I need to compare below as oppose to start

```
if (potentialfurtherAscendingBeyondThisStart.equals(nums[k]))
```

We know we cannot make the decision here since counter==0 and k==nums.length-2

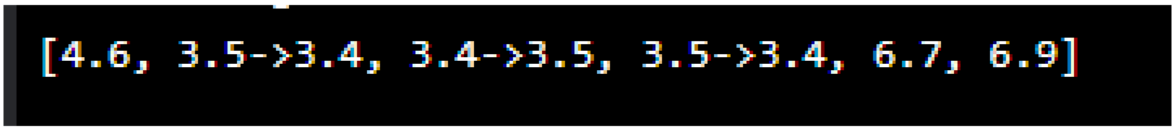
We know if A->D->A occured at other points in the code, it would hence not reach here

```
if (counter==0 && (k==nums.length-2))
{
    start=String.valueOf(nums[k]);
    end = String.valueOf(nums[k+1]);
    sm.add(start+"->"+end);
    System.out.println("5Writing range: " + start + "->
    isFirstOccurenceAscendingChain=false;
    isFirstOccurenceAscendingChainNoTransition =false;
    break;
}
```

ANALYSING LOGIC IN THIS SECTION....
Effectively it has not written stored value
3.4 -> 3.5 since it has identified it in the same range as
3.5 -> 3.4
Since I had performed the following logic

```
if ((potentialfurtherAscendingBeyondThisStart.equals(start)))
{
    System.out.println("Stored value also ascending and overlap, not storing");
System.out.println("NO storage required");
System.out.println("XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX");
}
```

I now get the correct result:



```
[4.6, 3.5->3.4, 3.4->3.5, 3.5->3.4, 6.7, 6.9]
```

I will now try the other failed test case above:

TEST CASE:

It seems fine now

```
3.5f,3.6f, 3.5f, 3.6f, 3.5f,3.4f,3.0f,2.9f,2.5f,2.4f
```

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

I will just check all my test cases again, I do not think it will fix other failing cases, but I want to be sure no others have failed..
In real world, I would also need to try this against the ChatGPT data again..
But since these small deflections do not occur in the data, there is no need

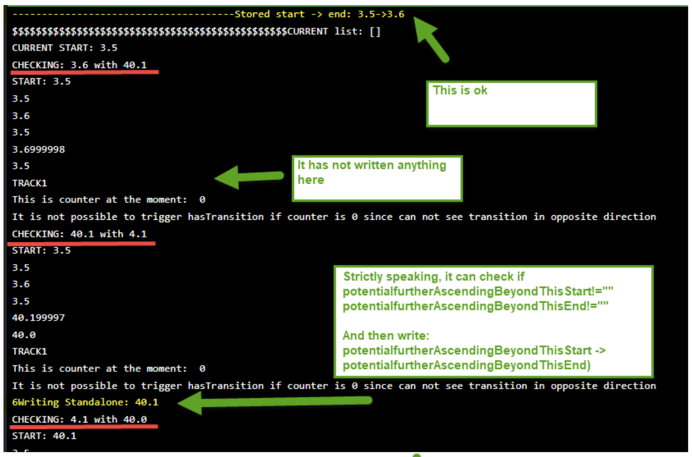
I have effectively now gone back to a test case which I tried to fix in my previous long documentation, but since it had adverse effect, I rolled back a lot.
I have to take a fresh approach

TEST CASE:

```
3.5f,3.6f,40.1f,4.1f,40.0f    //ascending standalone  (it has only written standalone cases)
```

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

Logic suggests it has to make a decision- before it writes the first standalone..
Like always, I will follow the logic



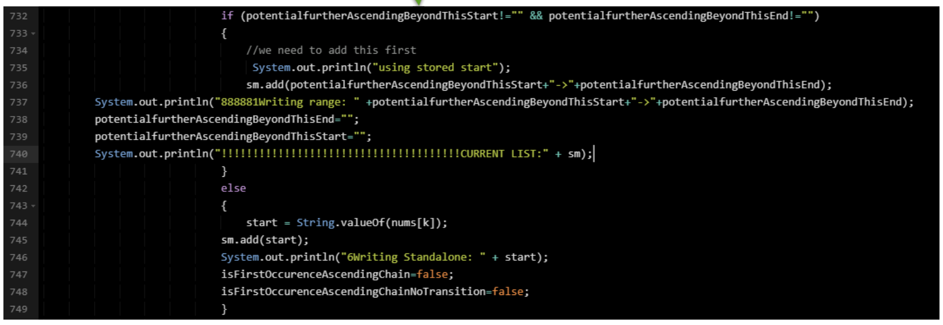
```
-------------------------------------Stored start -> end: 3.5->3.6
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$CURRENT list: []
CURRENT START: 3.5
CHECKING: 3.6 with 40.1
START: 3.5
3.5
3.6
3.5
3.6999998
3.5
TRACK1
This is counter at the moment:  0
It is not possible to trigger hasTransition if counter is 0 since can not see transition in opposite direction
CHECKING: 40.1 with 4.1
START: 3.5
3.5
3.6
3.5
40.199997
40.0
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
6Writing Standalone: 40.1
CHECKING: 4.1 with 40.0
START: 40.1
```

This is ok

It has not written anything here

Strictly speaking, it can check if
potentialfurtherAscendingBeyondThisStart!=""
potentialfurtherAscendingBeyondThisEnd!=""

And then write:
potentialfurtherAscendingBeyondThisStart ->
potentialfurtherAscendingBeyondThisEnd)



```
                    if (potentialfurtherAscendingBeyondThisStart!="" && potentialfurtherAscendingBeyondThisEnd!="")
                    {
                        //we need to add this first
                         System.out.println("using stored start");
                        sm.add(potentialfurtherAscendingBeyondThisStart+"->"+potentialfurtherAscendingBeyondThisEnd);
        System.out.println("888881Writing range: " +potentialfurtherAscendingBeyondThisStart+"->"+potentialfurtherAscendingBeyondThisEnd);
        potentialfurtherAscendingBeyondThisEnd="";
        potentialfurtherAscendingBeyondThisStart="";
        System.out.println("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!CURRENT LIST:" + sm);
                    }
                    else
                    {
                        start = String.valueOf(nums[k]);
                    sm.add(start);
                    System.out.println("6Writing Standalone: " + start);
                    isFirstOccurenceAscendingChain=false;
                    isFirstOccurenceAscendingChainNoTransition=false;
                    }
```

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

I will examine my other failed cases:

<u>TEST CASE: This looks similar to above but we can see ascending followed by ascending</u>

```
3.5f,3.6f,40.0f,40.1f,40.f    //it has not written  3.5f,3.6f,40.0f
```

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

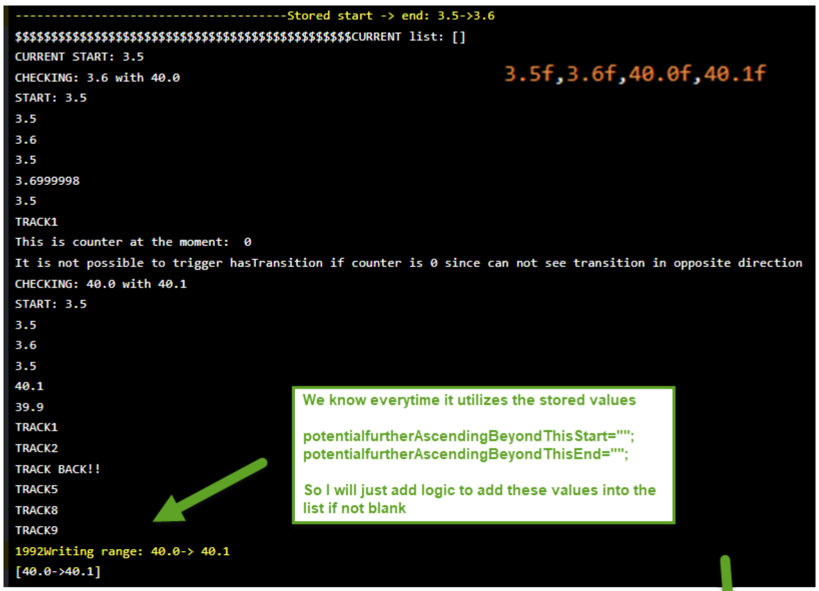
It is ascending followed by ascending
I feel as if I will hit an issue since it might need to look further ahead than nums[k+1] and I have not had to do this until now...

<u>TEST CASE:</u>
Also note even if I try there are issues...

```
3.5f,3.6f,40.0f,40.1f
```

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

So I will concentrate here....



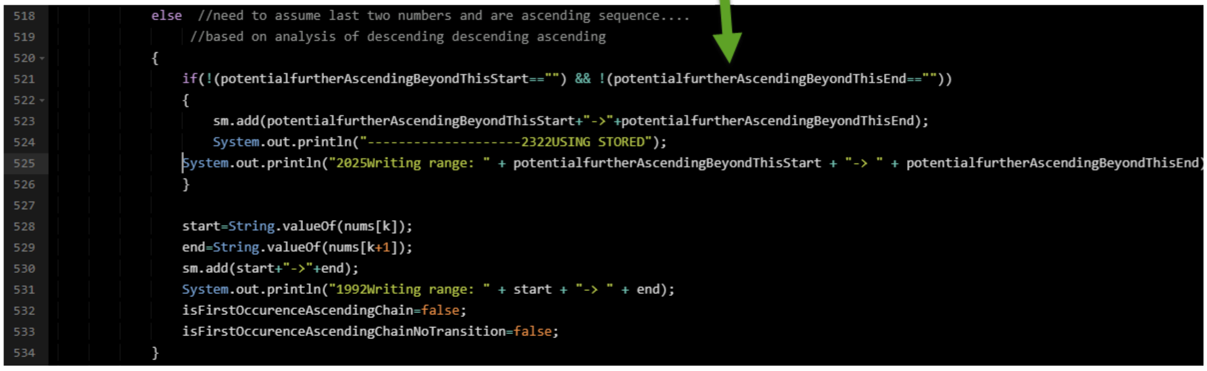
```
------------------------------------Stored start -> end: 3.5->3.6
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$CURRENT list: []
CURRENT START: 3.5
CHECKING: 3.6 with 40.0
START: 3.5
3.5
3.6
3.5
3.6999998
3.5
TRACK1
This is counter at the moment:  0
It is not possible to trigger hasTransition if counter is 0 since can not see transition in opposite direction
CHECKING: 40.0 with 40.1
START: 3.5
3.5
3.6
3.5
40.1
39.9
TRACK1
TRACK2
TRACK BACK!!
TRACK5
TRACK8
TRACK9
1992Writing range: 40.0-> 40.1
[40.0->40.1]
```

```
3.5f,3.6f,40.0f,40.1f
```

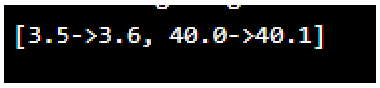
We know everytime it utilizes the stored values

potentialfurtherAscendingBeyondThisStart="";
potentialfurtherAscendingBeyondThisEnd="";

So I will just add logic to add these values into the list if not blank



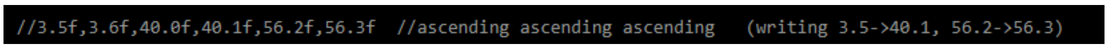
```java
        else  //need to assume last two numbers and are ascending sequence....
              //based on analysis of descending descending ascending
        {
            if(!(potentialfurtherAscendingBeyondThisStart=="") && !(potentialfurtherAscendingBeyondThisEnd==""))
            {
                sm.add(potentialfurtherAscendingBeyondThisStart+"->"+potentialfurtherAscendingBeyondThisEnd);
                System.out.println("-------------------2322USING STORED");
            System.out.println("2025Writing range: " + potentialfurtherAscendingBeyondThisStart + "-> " + potentialfurtherAscendingBeyondThisEnd)
            }

            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;
        }
```



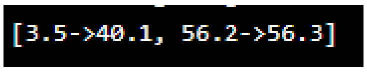
```
[3.5->3.6, 40.0->40.1]
```

So now I will run my other failed test cases

I can safely say it does not feel as if I will ruin any other logic, but I will run through my test cases again.
I am now exactly down to two failed cases...

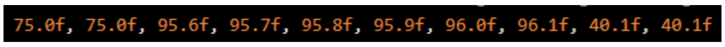


```
//3.5f,3.6f,40.0f,40.1f,56.2f,56.3f  //ascending ascending ascending   (writing 3.5->40.1, 56.2->56.3)
```

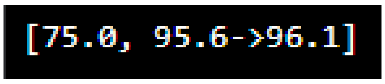


```
[3.5->40.1, 56.2->56.3]
```

I think it is easier to attempt and resolve this one first

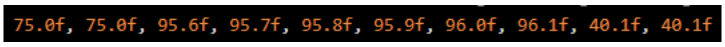


```
75.0f, 75.0f, 95.6f, 95.7f, 95.8f, 95.9f, 96.0f, 96.1f, 40.1f, 40.1f
```

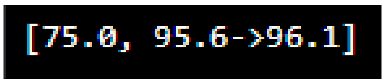


```
[75.0, 95.6->96.1]
```

<u>TEST CASE:</u>

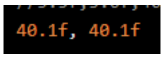


```
75.0f, 75.0f, 95.6f, 95.7f, 95.8f, 95.9f, 96.0f, 96.1f, 40.1f, 40.1f
```



```
[75.0, 95.6->96.1]
```

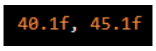
Problem occurs even if I perform:



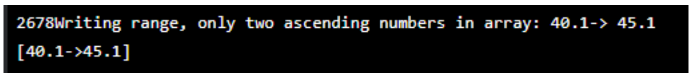
```
40.1f, 40.1f
```

This tells me it is related to having identical numbers

Firstly I am going to resolve this issue as I envisage a quick fix...
<u>TEST CASE:</u>

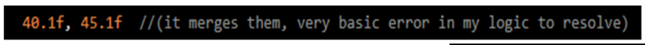


```
40.1f, 45.1f
```

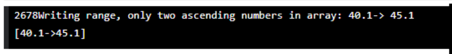


```
2678Writing range, only two ascending numbers in array: 40.1-> 45.1
[40.1->45.1]
```

So I can see I have flawed logic in my code, and hopefully I can resolve this readily.
I will add this to my failed cases

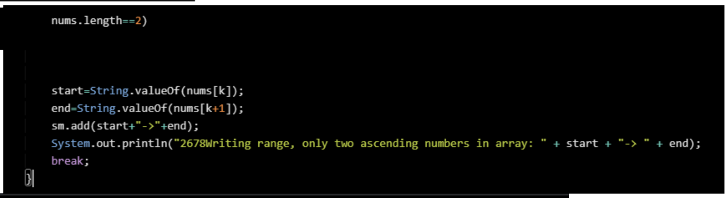


```
40.1f, 45.1f  //(it merges them, very basic error in my logic to resolve)
```

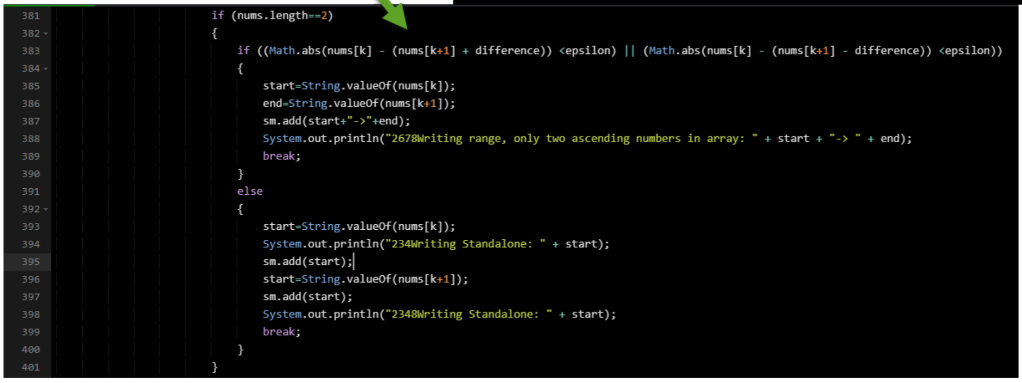


```
2678Writing range, only two ascending numbers in array: 40.1-> 45.1
[40.1->45.1]
```

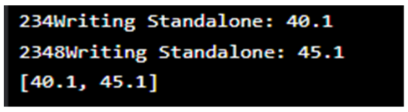
This was a total poor assumption which I implemented quickly. I have adjusted it to the below



```
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);
break;
}
```



```
if (nums.length==2)
{
    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("2678Writing range, only two ascending numbers in array: " + start + "-> " + end);
        break;
    }
    else
    {
        start=String.valueOf(nums[k]);
        System.out.println("234Writing Standalone: " + start);
        sm.add(start);
        start=String.valueOf(nums[k+1]);
        sm.add(start);
        System.out.println("2348Writing Standalone: " + start);
        break;
    }
}
```



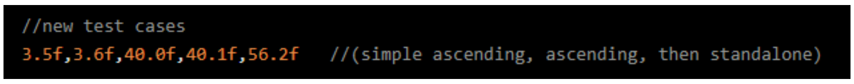
```
234Writing Standalone: 40.1
2348Writing Standalone: 45.1
[40.1, 45.1]
```

I have one more failed test case and then only ones left are the standalone repeat number issues...

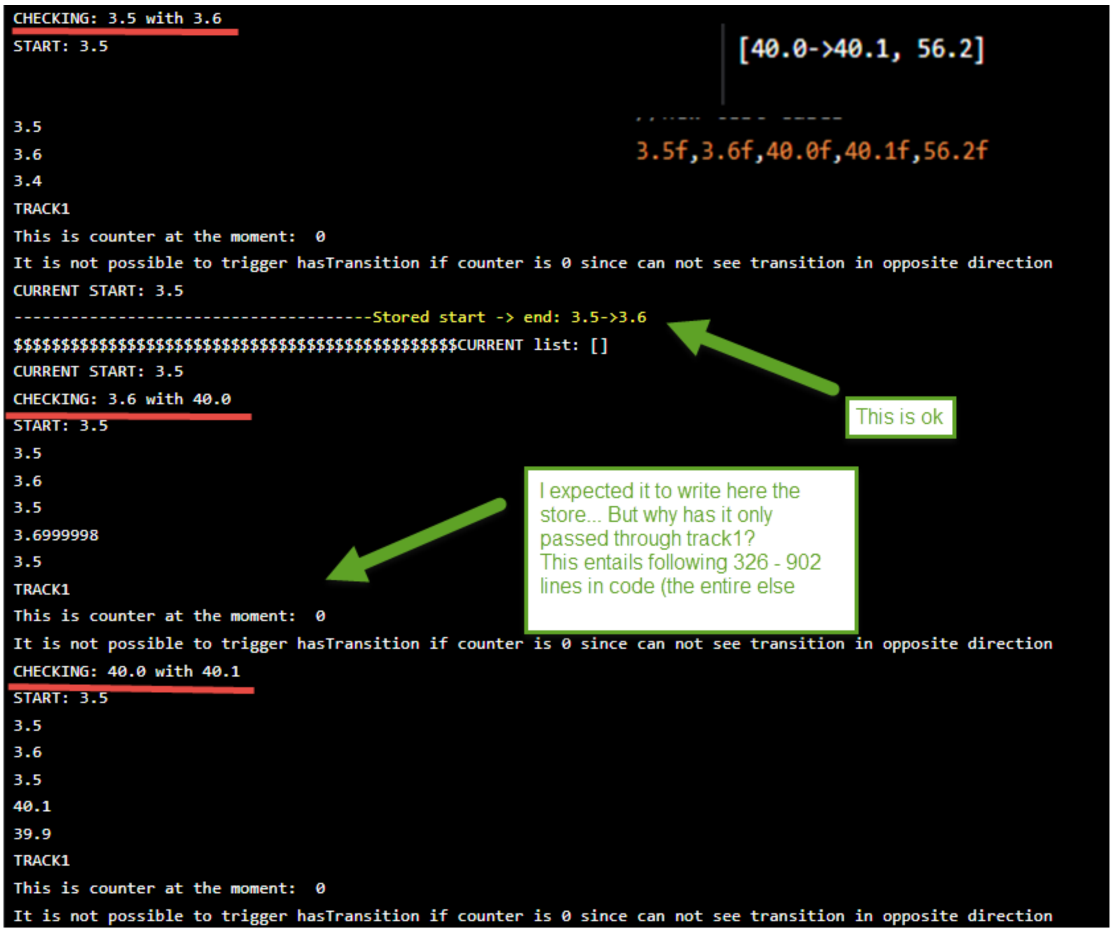
Unfortunately number test cases have built up, but there is a pattern for lots of them. If I run basic test case Ascend sequence and a standalone or ascending sequence with ascending sequence it fails, so my focus will start here

My instinct tells me any changes might now break the code.....

<u>TEST CASE:</u>



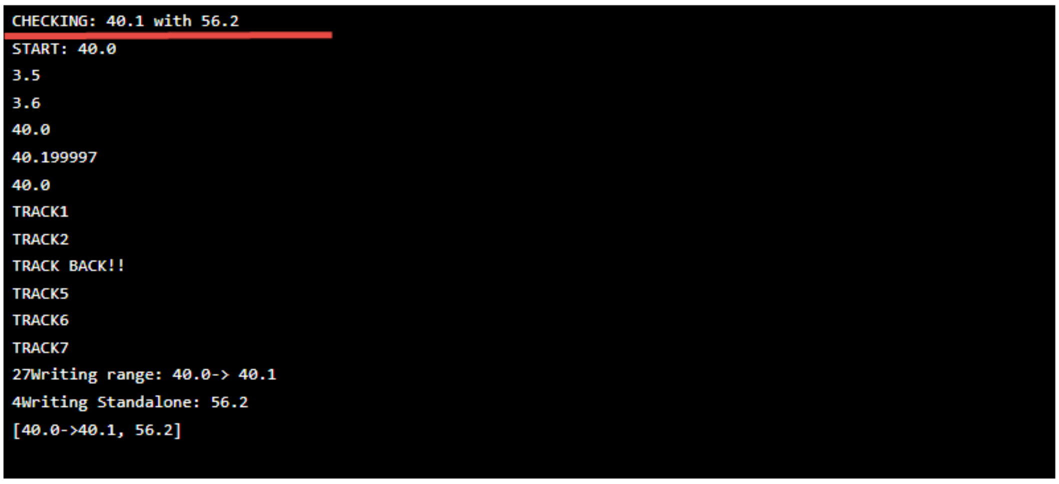
```
//new test cases
3.5f,3.6f,40.0f,40.1f,56.2f    //(simple ascending, ascending, then standalone)
```



```
CHECKING: 3.5 with 3.6
START: 3.5


3.5
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
CURRENT START: 3.5
--------------------------------------Stored start -> end: 3.5->3.6
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$CURRENT list: []
CURRENT START: 3.5
CHECKING: 3.6 with 40.0
START: 3.5
3.5
3.6
3.5
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
START: 3.5
3.5
3.6
3.5
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
```

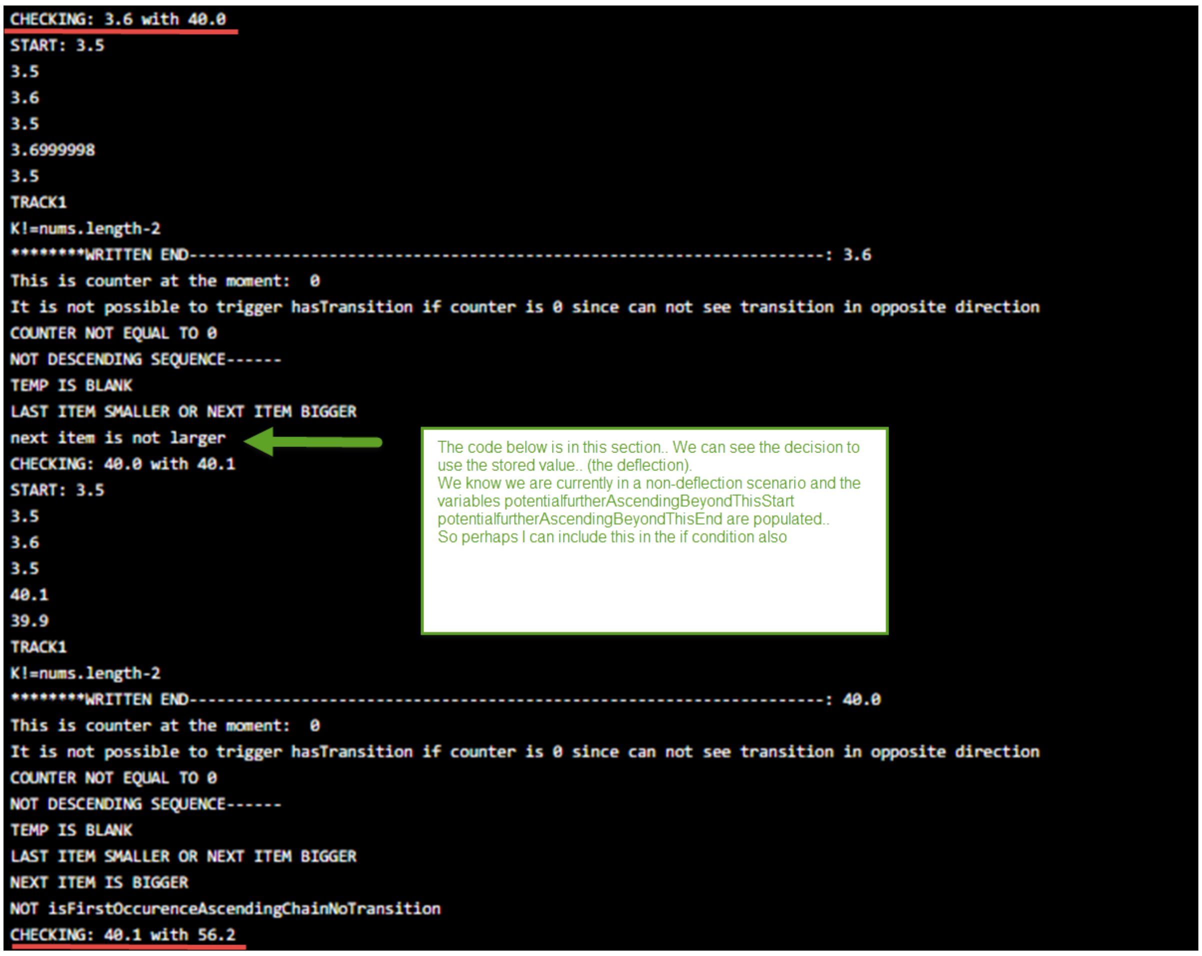
```
[40.0->40.1, 56.2]

3.5f,3.6f,40.0f,40.1f,56.2f
```

This is ok
I expected it to write here the store... But why has it only passed through track1?
This entails following 326 - 902 lines in code (the entire else



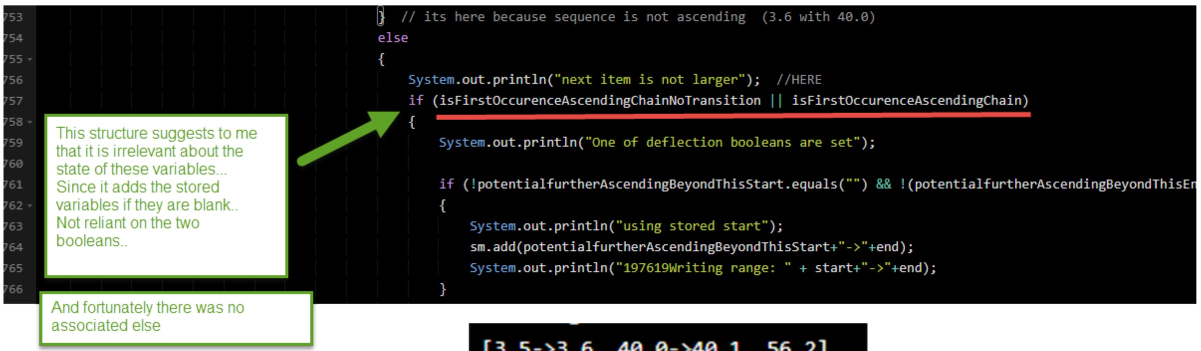
```
CHECKING: 40.1 with 56.2
START: 40.0
3.5
3.6
40.0
40.199997
40.0
TRACK1
TRACK2
TRACK BACK!!
TRACK5
TRACK6
TRACK7
27Writing range: 40.0-> 40.1
4Writing Standalone: 56.2
[40.0->40.1, 56.2]
```

Clearly this is way too little information to make any sort of judgement since we know we have to insert the stored value in the most suitable place..
I think it could have been completed anywhere in principle, but I have kept everything fairly tidy which has assisted reaching here..
So I have created screen outputs in each area I believe the code has traversed...



```
CHECKING: 3.6 with 40.0
START: 3.5
3.5
3.6
3.5
3.6999998
3.5
TRACK1
K!=nums.length-2
********WRITTEN END----------------------------------------------------------------------: 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
COUNTER NOT EQUAL TO 0
NOT DESCENDING SEQUENCE------
TEMP IS BLANK
LAST ITEM SMALLER OR NEXT ITEM BIGGER
next item is not larger
CHECKING: 40.0 with 40.1
START: 3.5
3.5
3.6
3.5
40.1
39.9
TRACK1
K!=nums.length-2
********WRITTEN END----------------------------------------------------------------------: 40.0
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
COUNTER NOT EQUAL TO 0
NOT DESCENDING SEQUENCE------
TEMP IS BLANK
LAST ITEM SMALLER OR NEXT ITEM BIGGER
NEXT ITEM IS BIGGER
NOT isFirstOccurenceAscendingChainNoTransition
CHECKING: 40.1 with 56.2
```

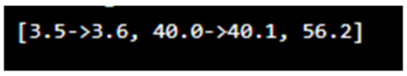
The code below is in this section.. We can see the decision to use the stored value.. (the deflection).
We know we are currently in a non-deflection scenario and the variables potentialfurtherAscendingBeyondThisStart potentialfurtherAscendingBeyondThisEnd are populated..
So perhaps I can include this in the if condition also



```
} // its here because sequence is not ascending  (3.6 with 40.0)
else
{
    System.out.println("next item is not larger");  //HERE
    if (isFirstOccurenceAscendingChainNoTransition || isFirstOccurenceAscendingChain)
    {
        System.out.println("One of deflection booleans are set");

        if (!potentialfurtherAscendingBeyondThisStart.equals("") && !(potentialfurtherAscendingBeyondThisEn
        {
            System.out.println("using stored start");
            sm.add(potentialfurtherAscendingBeyondThisStart+"->"+end);
            System.out.println("197619Writing range: " + start+"->"+end);
        }
```

This structure suggests to me that it is irrelevant about the state of these variables... Since it adds the stored variables if they are blank.. Not reliant on the two booleans..

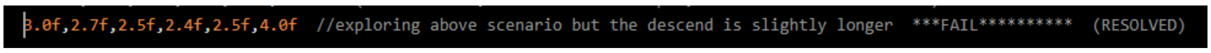
And fortunately there was no associated else



```
[3.5->3.6, 40.0->40.1, 56.2]
```

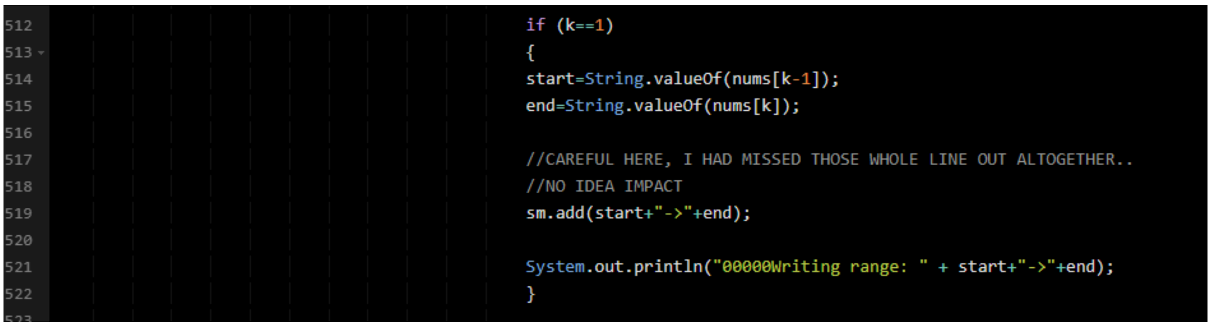
I believe these are very massive changes in my code..
I will run through all my tests now and also ChatGPT data...

I found that in test cases such as this.



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

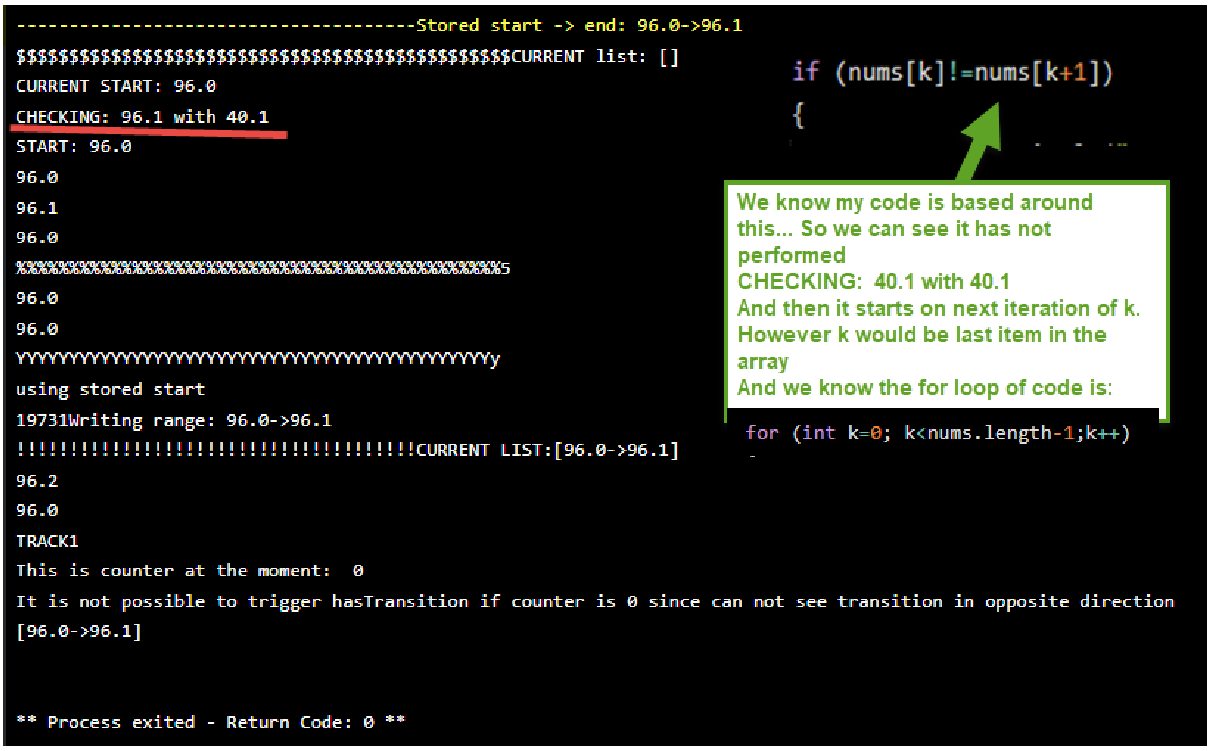
It was now performing 2.4->2.5 twice.
I realised whilst I was coding that already I had written to screen that I had written an item out to the screen, I had not physically done this...
So I inserted this code in...
But I found that it only had to perform this whilst k==1



```
                                    if (k==1)
                                    {
                                    start=String.valueOf(nums[k-1]);
                                    end=String.valueOf(nums[k]);

                                    //CAREFUL HERE, I HAD MISSED THOSE WHOLE LINE OUT ALTOGETHER..
                                    //NO IDEA IMPACT
                                    sm.add(start+"->"+end);

                                    System.out.println("00000Writing range: " + start+"->"+end);
                                    }
```

This fixed the issue that arose...

Also one other shortfall in ChatGPT data was not it did not have duplicate standalone numbers. We can see how it could cause issues...



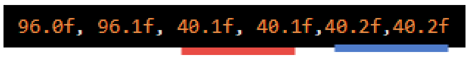
```
--------------------------------------Stored start -> end: 96.0->96.1
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$CURRENT list: []
CURRENT START: 96.0
CHECKING: 96.1 with 40.1
START: 96.0
96.0
96.1
96.0
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%5
96.0
96.0
YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYy
using stored start
19731Writing range: 96.0->96.1
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!CURRENT LIST:[96.0->96.1]
96.2
96.0
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
[96.0->96.1]


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

```
if (nums[k]!=nums[k+1])
{
```

We know my code is based around this... So we can see it has not performed
CHECKING: 40.1 with 40.1
And then it starts on next iteration of k.
However k would be last item in the array
And we know the for loop of code is:

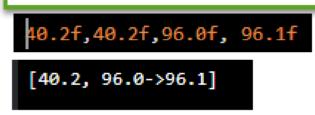
```
for (int k=0; k<nums.length-1;k++)
```

Outcome would be the same if there was another set of repeat standalones...



```
96.0f, 96.1f, 40.1f, 40.1f,40.2f,40.2f
```

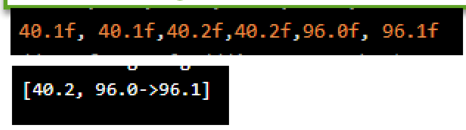
We would not have an issue here



```
40.2f,40.2f,96.0f, 96.1f
```

```
[40.2, 96.0->96.1]
```

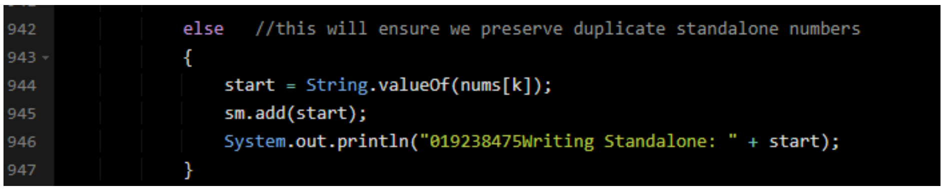
We would have issue here. And its ironic this is a very common type of data to arise in some datasets, but ChatGPT did not explore either. I will add this into my failed test cases



```
40.1f, 40.1f,40.2f,40.2f,96.0f, 96.1f
```

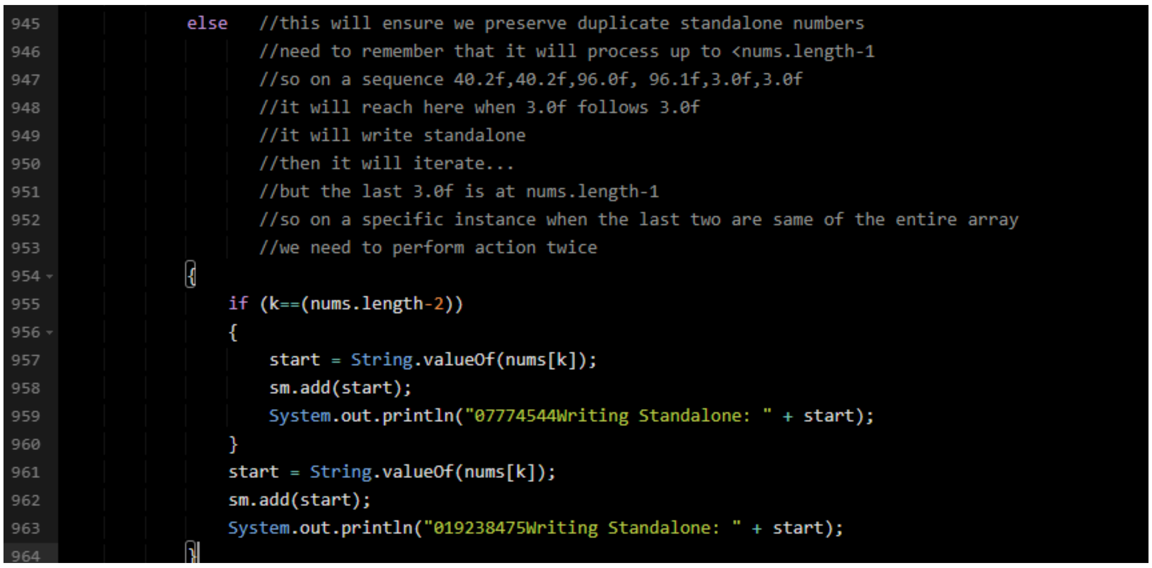
```
[40.2, 96.0->96.1]
```

We can see that we don't want it compare same number against each other.
But at the same time, we can not just forget it existed...
For instance something like this 40.3f,40.3f,54.5f
It would not be best if I miss the leading 40.3f
So I have created an else statement in my code to add this..



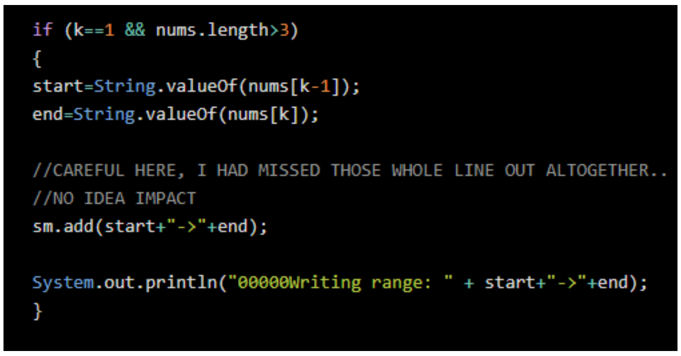
```java
else   //this will ensure we preserve duplicate standalone numbers
{
    start = String.valueOf(nums[k]);
    sm.add(start);
    System.out.println("019238475Writing Standalone: " + start);
}
```

Infact I had to enhance the loop even further as follows, otherwise it would miss the last number if it was same as penultimate



```java
else   //this will ensure we preserve duplicate standalone numbers
       //need to remember that it will process up to <nums.length-1
       //so on a sequence 40.2f,40.2f,96.0f, 96.1f,3.0f,3.0f
       //it will reach here when 3.0f follows 3.0f
       //it will write standalone
       //then it will iterate...
       //but the last 3.0f is at nums.length-1
       //so on a specific instance when the last two are same of the entire array
       //we need to perform action twice
{
    if (k==(nums.length-2))
    {
        start = String.valueOf(nums[k]);
        sm.add(start);
        System.out.println("07774544Writing Standalone: " + start);
    }
    start = String.valueOf(nums[k]);
    sm.add(start);
    System.out.println("019238475Writing Standalone: " + start);
}
```

I also found I had to lock this section of code to nums.length>=3, otherwise for cases such as `40.0f,40.1f,3.5f` it would add stored value 40.0->40.1 and also again as below…

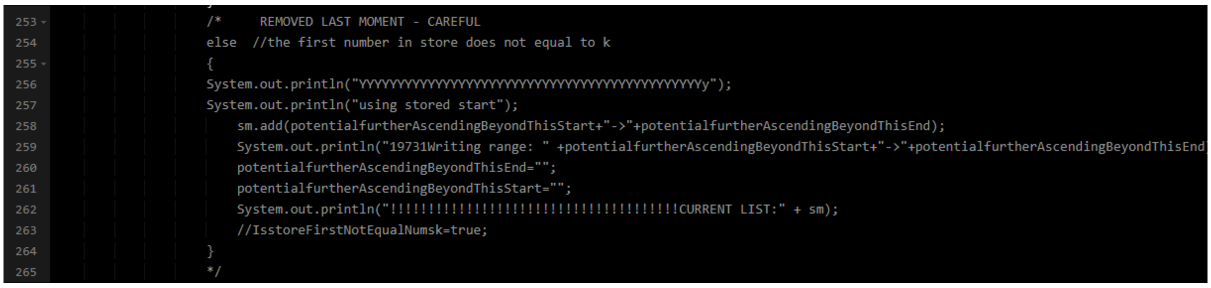


```java
if (k==1 && nums.length>3)
{
start=String.valueOf(nums[k-1]);
end=String.valueOf(nums[k]);

//CAREFUL HERE, I HAD MISSED THOSE WHOLE LINE OUT ALTOGETHER..
//NO IDEA IMPACT
sm.add(start+"->"+end);

System.out.println("00000Writing range: " + start+"->"+end);
}
```

But I am not entirely sure about this but it relates to having used the stored area already.

So I tested code with my ChatGPT data, I found that it failed in exact same locations as before:

I checked the duplicate information being written and it pointed me here.
My first instinct was to remove the content since I implemented it earlier.
It fixed my issues and it also fixed remaining issues in my code.



```
/*     REMOVED LAST MOMENT - CAREFUL
else  //the first number in store does not equal to k
{
System.out.println("YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYy");
System.out.println("using stored start");
    sm.add(potentialfurtherAscendingBeyondThisStart+"->"+potentialfurtherAscendingBeyondThisEnd);
    System.out.println("19731Writing range: " +potentialfurtherAscendingBeyondThisStart+"->"+potentialfurtherAscendingBeyondThisEnd
    potentialfurtherAscendingBeyondThisEnd="";
    potentialfurtherAscendingBeyondThisStart="";
    System.out.println("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!CURRENT LIST:" + sm);
    //IsstoreFirstNotEqualNumsk=true;
}
*/
```

So at the moment, it appears my code is fully functional with all my test cases and ChatGPT. There was extremely strong value in having ChatGPT data as my reference…