I have now managed to progress the code further.
It was an absolute headache to try to understand the code again.
I had to get it out of my mental process that it was not entering the big else statement due to entire blocks in sequence descending. It was here in order to configure bigger block to start the test process again..

So I managed to get the code up to this point.....
(see code Test case v1).

```
//final static String str = "123456789";   //whole block ascending                PASS
//final static String str = "123124125";  //each 3 block ascending and digit in each block ascending     PASS
 //final static String str = "123412161211";    // each 3 block ascending ONLY   (215 not ascending)     PASS


 //final static String str = "98765431";   //whole block descending     PASS
 //final static String str =  "543542541";   //each 3 block descending and digit in each block descending    PASS

 final static String str = "542541516";    // each 3 block descending ONLY   (516 not descending)    FAIL
```

MY FIRST THOUGHTS ARE ON HOW THE FOLLOWING (UNDERLINED GREEN) CAN PASS

```
//final static String str = "123456789";   //whole block ascending                PASS
//final static String str = "123124125";  //each 3 block ascending and digit in each block ascending     PASS
//final static String str = "123124215";    // each 3 block ascending ONLY   (215 not ascending)     PASS


 //final static String str = "98765431";   //whole block descending     PASS
 //final static String str =  "543542541";   //each 3 block descending and digit in each block descending    PASS

 final static String str = "542541516";    // each 3 block descending ONLY   (516 not descending)    FAIL
```

So, I realise straight away that my main code is based on few basic principle changes as follows to facilitate ascending and descending.
Some of the major changes were in this section:

```
try
{
    blockBefore = str.substring((str.length() - (2*n)),(str.length()-n));


if ((Long.valueOf(blockBefore))<Long.valueOf(block) && recursionCall==0)
{
    initialState="ascending";
}

temp = blockBefore;
blockBefore = block;
block=temp;

if ((Long.valueOf(blockBefore))<Long.valueOf(block) && recursionCall==0)
{
    initialState="descending";
}
```

Need to remember we have recursionCall variable since this area of interest is ONLY applicable at first digit lookup in neighbouring blocks

This code was here since we can not flick the equality symbol via a variable. So ONLY way to perform comparison from other perspective is to flick the block values

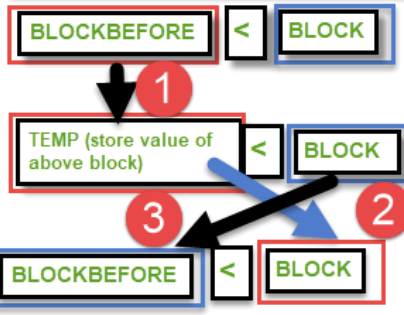

FIGURE 1

FIGURE 2

```
if (initialState=="ascending")
{
    System.out.println("RESTORE STATE FOR ASCENDING BLOCK SIZES");
    block = str.substring(str.length()-n, str.length());
    blockBefore = str.substring((str.length() - (2*n)),(str.length()-n));
}
```

```
if (initialState=="ascending")
{
    System.out.println("RESTORE STATE FOR ASCENDING BLOCK SIZES");
    block = str.substring(str.length()-n, str.length());
    blockBefore = str.substring((str.length() - (2*n)),(str.length()-n));
}
```

Now we need to remember if the initialState was ascending, it needs to go back to correct state.

We only performed the variable flip to identify if the block was ascending or descending and obtain correct initialState. This initialState is only used for the following purpose below (to ascertain the order in which numbers are stored.

SO JUST TO RE-ITERATE WHY WE RESET THE VARIABLES TO ORIGINAL: IN SIMPLE TERMS, IT IS SO THAT THE CURRENT BLOCK REMAINS ON THE RIGHT HAND SIDE (SINCE IT DID NOT EVALUATE TO **FIGURE 2**

BUT **FIGURE 1**:

```
    if (initialState=="descending")
{
    if (ascNums.length()==0)
    {
        sj.add(blockBeforeBlock);
        sj.add(block);
        sj.add(blockBefore);

    }
    else
    {
        System.out.println("1WHEN-------
        System.out.println(sequence);
        sj.add(blockBeforeBlock);
    }
}
```

```
if (initialState=="ascending")
{

if (ascNums.length()==0)
{
    sj.add(blockBeforeBlock);
    sj.add(blockBefore);
    sj.add(block);
}
else
{
    System.out.println("2WHEN---------
    System.out.println(sequence);
    sj.add(blockBeforeBlock);
}
```

And also correct outputs to end user

```
System.out.println(strBackup + " consists of " + initialState + " numbers of group size" +"("+n+"): " + ascNums);
```

```
System.out.println(strBackup + " does not consist of " + initialState + " number in any block size " + n + "(n): " + ascNums);
```

And finally based on breaking this down, something triggered into my mindset.
The first time when it executed above code, recursionCall variable was set to 0 and increased in the following area of the code:

This is still in the try loop and it has established the existing block and the block before for the first time...

```
System.out.println("This is block: " + block);
System.out.println("This is block before: " + blockBefore);

if ((Long.valueOf(blockBefore))<Long.valueOf(block))
{
    recursionCall++;
```

So equally it would reach here again when the block size has increased.

So equally it would need to perform the following to establish the initialState again (as in the code in the first page since execution is effectively starting afresh).

But I realised it was not possible given that recursionCall variable and it has potentially been increased as part of the smaller block size.
So I knew at this point, I need to reset the variable back to 0 at point where it has ascertained that the block size is not compatible with ascending or descending.

```
else
{
    n++;
    System.out.println("Value of block size has been increased to: " + n);
    sequence="";
    ascNums=new StringBuilder("");
    sj=new StringJoiner(",");
    recursionCall=0;
```

And then it will continue as below if there are exact block sizes in the initial number provided...

```
if (str.length()%n==0)
{
    return ascending(str);
}
```

Or alternatively it has finished execution:

```
System.out.println(strBackup + " does not consist of " + initialState + " number in any block size " + n + "(n): " + ascNums);
System.exit(0);
return null;
```

So now, this is the first time I can run full documented test cases for all scenarios.
I will also include some longer length blocks just in case also:

TEST CASE 1: (as per challenge)

```
final static String str = "123124125";
```

```
123 are digit(s) of first block n(3) of 123124125
123124125 consists of ascending numbers of group size(3): 123,124,125,
```

```
ascending("123124125") → true
// Contains a set of consecutive ascending numbers
// if grouped into 3's : 123, 124, 125
```

TEST CASE 2: (as per challenge)

```
final static String str = "101112131415";
```

```
10 are digit(s) of first block n(2) of 101112131415
101112131415 consists of ascending numbers of group size(2): 10,11,12,13,14,15,
```

```
ascending("101112131415") → true
// Contains a set of consecutive ascending numbers
// if grouped into 2's : 10, 11, 12, 13, 14, 15
```

TEST CASE 3: (as per challenge)

```
final static String str = "32332432536";
```

```
This is block before: 5
Performing recursion, restoring original String(32332432536)
This is the block size: 2
Value of block size has been increased to: 3
String length: 11
Block size: 3
32332432536 does not consist of ascending number in any block size 3(n):
```

```
ascending("32332432536") → false
// Regardless of the grouping size, the numbers can't be consecutive.
```

TEST CASE 4: (as per challenge) = FAIL

```
final static String str = "326325324323";
```

```
324323 are digit(s) of first block n(6) of 326325324323
326325324323 consists of descending numbers of group size(6): 324323,326325
```

```
ascending("326325324323") → false
// Though the numbers (if grouped into 3's) are consecutive but descending.
```

I am just not able to understand at first instance why this issue has occurred.
It can be seen that the number I inputted is exactly the same as that in the challenge. And further more, it can be seen that the blocks are size 6 and it is not a descending sequence.

So at this point, I will try to create a block size of two and perform descending. I will ensure I also use four blocks.

TEST CASE 4a = FAIL

```
final static String str = "3231302928";
```

```
Block size: 4
3231302928 does not consist of ascending number in any block size 4(n):
```

So the issues are failing when performing:

descending numbers of block size two (in which it has reached block size 4 and terminated)
descending numbers of block size three (in which it has reached block size 6 and showed descendency but it has created totally incorrect blocks on screen which are not part of the linear numbers occurring!!!!)

For now, I will continue my tests since I am yet to reach the official descending testing.

So now, I am going to try ascending and configure 323, 324,325,326 Similar numbers to above, but changed the direction of the numbers.

TEST CASE 4b = PASS

```
final static String str = "323324325326";
```

```
323 are digit(s) of first block n(3) of 323324325326
323324325326 consists of ascending numbers of group size(3): 323,324,325,326,
```

And now I will try ascending but 4 x block size(2)

TEST CASE 4c = PASS

```
final static String str = "32333435";
```

```
32 are digit(s) of first block n(2) of 32333435
32333435 consists of ascending numbers of group size(2): 32,33,34,35,
```

TEST CASE 5: (as per challenge)

```
final static String str = "666667";
```

```
ascending("666667") → true
// Consecutive numbers: 666 and 667.
```

TEST CASE 6:

```
final static String str = "123456789";   //whole block ascending
```

```
1 are digit(s) of first block n(1) of 123456789
123456789 consists of ascending numbers of group size(1): 1,2,3,4,5,6,7,8,9,
```

TEST CASE 7:

```
final static String str = "123124125";  //each 3 block ascending and digit in each block ascending
```

```
123 are digit(s) of first block n(3) of 123124125
123124125 consists of ascending numbers of group size(3): 123,124,125,
```

TEST CASE 8:

```
final static String str = "123124215";    // each 3 block ascending ONLY   (215 not ascending)
```

```
123 are digit(s) of first block n(3) of 123124215
123124215 consists of ascending numbers of group size(3): 123,124,215,
```

TEST CASE 9:

```
final static String str = "123212421543452";    // each 5 block ascending ONLY
```

```
12321 are digit(s) of first block n(5) of 123212421543452
123212421543452 consists of ascending numbers of group size(5): 12321,24215,43452,
```

TEST CASE 10:

```
final static String str = "123452467956789";
```

```
12345 are digit(s) of first block n(5) of 123452467956789
123452467956789 consists of ascending numbers of group size(5): 12345,24679,56789,
```

I am now officially performing my descending numbers.
I will run the basic ones first.

TEST CASE 11:

```
final static String str = "98765431";   //whole block descending
```

```
8 are digit(s) of first block n(1) of 98765431
98765431 consists of descending numbers of group size(1): 9,8,7,6,5,4,3,1,
```

TEST CASE 12:

```
final static String str =  "543542541";   //each 3 block descending and digit in each block descending
```

```
542 are digit(s) of first block n(3) of 543542541
543542541 consists of descending numbers of group size(3): 543,542,541,
```

TEST CASE 13:

```
final static String str = "542541526";    // each 3 block descending ONLY
542541526 consists of descending numbers of group size(3): 542,541,526,
```

This is same as Test Case 4 but it uses only less block and passes. Perhaps it is a good chance to increase the block size here and see the outcome…

TEST CASE 13a: Increasing numbers blocks(4) of size 3 = FAIL

```
final static String str =  "543542541536";
541536 are digit(s) of first block n(6) of 543542541536
543542541536 consists of descending numbers of group size(6): 541536,543542
```

This fails in exactly the same way as Test case 4

But what if I now make sure that each of the four blocks also has digits that are descending

TEST CASE 13b: Leaving number blocks 4 of size 3 , setting 536 =>532 = PASS

```
final static String str =  "543542541532
542 are digit(s) of first block n(3) of 543542541532
543542541532 consists of descending numbers of group size(3): 543,542,541,532
```

So this issue is ONLY occurring when the individual block does not consist of descending numbers. And it causes a scramble for an odd reason…

We can just quickly try ascending sequence with 4 blocks of size 3

TEST CASE 13c: Leaving number blocks 4 of size 3 , ascending but digits of last block not ascending = FAIL

```
final static String str =  "123134167198";   //each 3 block ascending and digits in 198 not ascending
123134 are digit(s) of first block n(6) of 123134167198
123134167198 consists of ascending numbers of group size(6): 123134,167198
```

I will now quickly change the 198 => 189

TEST CASE 13d: Leaving number blocks 4 of size 3 , ascending all digits in each block

```
final static String str =  "123134167189";   //each 3 block ascending and digits each block ascending
```

```
123 are digit(s) of first block n(3) of 123134167189
123134167189 consists of ascending numbers of group size(3): 123,134,167,189,
```

So it is the same for descending and ascending, if one block has digits in incorrect order it affects the outcome.
So only area for me to test is if its only related to the last block or irrelevant.

TEST CASE 13e: Leaving number blocks 4 of size 3 , ascending but digits of penultimate block not ascending = FAIL

```
final static String str =  "123134176189";
```

```
123 are digit(s) of first block n(3) of 123134176189
123134176189 consists of ascending numbers of group size(3): 123,134,176,189,
```

I will quickly try this similar to TEST CASE 13a , but this time the penultimate block will have numbers that are not descending

TEST CASE 13f: Similar to 13a, but this time the penultimate digits in block will not be descending = PASS

```
final static String str =  "543542539532";
```

```
542 are digit(s) of first block n(3) of 543542539532
543542539532 consists of descending numbers of group size(3): 543,542,539,532,
```

TEST CASE 14:

```
final static String str = "434522421512321";    // each 5 block descending  ONLY
```

```
24215 are digit(s) of first block n(5) of 434522421512321
434522421512321 consists of descending numbers of group size(5): 43452,24215,12321,
```

TEST CASE 15:

```
final static String str = "987348765276543";
```

```
87652 are digit(s) of first block n(5) of 987348765276543
987348765276543 consists of descending numbers of group size(5): 98734,87652,76543,
```

ANALYSIS

SO AFTER ALL THE TESTING, THIS IS STATE OF PLAY:

For for descending and ascending ascending sequences, if the last block (most right hand side) has digits in incorrect order it affects the outcome.

This has been shown for numerous block sizes

So, there is something relating to operations undertaken in which it has examined the first block (as in that appearing in the inputted number), which is having an adverse effect as such:

```
final static String str =  "8769";
```

```
There are minimum of two blocks of size: 2
This is block: 69
This is block before: 87
Performing recursion, restoring original String(8769)
This is the block size: 3
Value of block size has been increased to: 4
String length: 4
Block size: 4
This is the block size: 4
8769 does not consist of ascending number in any block size 4(n):
```

It has jumped a block size

TEST CASE 16: I will do my investigations based on this failure and the screen outputs:

```
final static String str =  "235461";
```

```
Welcome to Online IDE!! Happy Coding :)
Number to be examined: 235461
This is the block size: 1
There are minimum of two blocks of size: 1
**** IN TRY STATEMENT ****
DESCENDING BLOCK
This is block: 6
This is block before: 1
Performing recursion, examining block (1) with block before it(4)
*****THIS IS THE SEQUENCE: 4,6,1,
This is the block size: 1
There are minimum of two blocks of size: 1
**** IN TRY STATEMENT ****
This is block: 4
This is block before: 6
Performing recursion, restoring original String(235461)
This is the block size: 2
There are minimum of two blocks of size: 2
**** IN TRY STATEMENT ****
This is block: 54
This is block before: 61
Performing recursion, restoring original String(235461)
This is the block size: 3
There are minimum of two blocks of size: 3
**** IN TRY STATEMENT ****
This is block: 235
This is block before: 461
Performing recursion, restoring original String(235461)
This is the block size: 4
Value of block size has been increased to: 5
String length: 6
Block size: 5
235461 does not consist of descending number in any block size 5(n):
```

We know it will always enter try statement. I wanted to be sure

It has identified a descending block , so it has switched the variables about blockBefore = 1
block = 6
It would now compare

```
if ((Long.valueOf(blockBefore))<Long.valueOf(block))
```

This is try if statement, so it performs recursion
But it is now comparing 1 with 4...
I think it should be comparing 6 with 4.
So my instinct tells me that

examining block(1) should be the value in
This is block:

So I have introduced this code: It has no effect on the recursion signature, so I will perform execution again

```
if ((Long.valueOf(blockBefore))<Long.valueOf(block))
{
    recursionCall++;

    String blockBeforeBlock = str.substring((str.length() - (3*n)),(str.length()-(2*n)));

    if (initialState=="descending")
    {
        blockBefore=block;
    }
```

THIS IS THE NEW STATE OF PLAY:

```
Welcome to Online IDE!! Happy Coding :)
Number to be examined: 235461
This is the block size: 1
There are minimum of two blocks of size: 1
**** IN TRY STATEMENT ****
DESCENDING BLOCK
This is block: 6
This is block before: 1
INITAL STATE WAS DECENDING
Performing recursion, examining block before current (6) with block before it(4)
*****THIS IS THE SEQUENCE: 4,6,6,
This is the block size: 1
There are minimum of two blocks of size: 1
**** IN TRY STATEMENT ****
This is block: 4
This is block before: 6
Performing recursion, restoring original String(235461)
This is the block size: 2
There are minimum of two blocks of size: 2
**** IN TRY STATEMENT ****
This is block: 54
This is block before: 61
Performing recursion, restoring original String(235461)
This is the block size: 3
There are minimum of two blocks of size: 3
**** IN TRY STATEMENT ****
This is block: 235
This is block before: 461
Performing recursion, restoring original String(235461)
This is the block size: 4
Value of block size has been increased to: 5
String length: 6
Block size: 5
235461 does not consist of descending number in any block size 5(n):
```

This is now corrected

It has entered try and clearly skipped lots of logic since recursionCall is now greater than 0....

I see an issue here... Now in any sense of execution, we expect and also see that blocks were calculated like this when he had both ascending and descending

```
block = str.substring(str.length()-n, str.length());
blockBefore = str.substring((str.length() - (2*n)),(str.length()-n));
```

But looking at the code I see this. I understand I only set this up so that I can ascertain if its going to be a descending or ascending test. And as it can be seen in code, there is absolutely no conditional loop around it.
And since initialState=="descending" , it will not restore the correct sequence back... So as a result blockBefore (4=>6) and block (6=>4) which is

```
temp = blockBefore;
blockBefore = block;
block=temp;

if ((Long.valueOf(blockBefore))<Long.valueOf(block) && recursionCall==0)
{
    System.out.println("DESCENDING BLOCK");
    initialState="descending";
}

if (initialState=="ascending")
{
    block = str.substring(str.length()-n, str.length());
    blockBefore = str.substring((str.length() - (2*n)),(str.length()-n));
}
```

So I have introduced this code, I am really hoping this is going to be the breakthrough

```
if (recursionCall==0)
{
temp = blockBefore;
blockBefore = block;
block=temp;
}
```

I can see straight away that my results are looking much closer to expectations:

```
There are minimum of two blocks of size: 2
**** IN TRY STATEMENT ****
This is block: 61
This is block before: 54
INITAL STATE WAS DECENDING
Performing recursion, examining block before current (61) with block before it(23)
*****THIS IS THE SEQUENCE: 23,61,61,
This is the block size: 2
There are minimum of two blocks of size: 2
**** IN TRY STATEMENT ****
This is block: 54
This is block before: 23
**** IN CATCH STATEMENT ****
23 are digit(s) of first block n(2) of 235461
235461 consists of descending numbers of group size(2): 23,61,61,
```

It can be seen immediately that

This statement is still incorrect

```
Welcome to Online IDE!! Happy Coding :)
Number to be examined: 235461
This is the block size: 1
There are minimum of two blocks of size: 1
**** IN TRY STATEMENT ****
```

```
There are minimum of two blocks of size: 2
**** IN TRY STATEMENT ****
This is block: 61
This is block before: 54
INITAL STATE WAS DECENDING
Performing recursion, examining block before current (61) with block before it(23)
*****THIS IS THE SEQUENCE: 23,61,61,
This is the block size: 2
There are minimum of two blocks of size: 2
**** IN TRY STATEMENT ****
This is block: 54
This is block before: 23
**** IN CATCH STATEMENT ****
23 are digit(s) of first block n(2) of 235461
235461 consists of descending numbers of group size(2): 23,61,61,
```

It can be seen immediately that it is processing the block and block before it correctly...

This sequence is still incorrect

Now it shows that it is performing a descending number check based on preliminary observation with 1 and 6

```
try
{

    System.out.println("**** IN TRY STATEMENT ****");
    blockBefore = str.substring((str.length() - (2*n)),(str.length()-n));

    if (recursionCall>0)
    {
        if (Long.valueOf(blockBefore)<Long.valueOf(block))
        {
            state="ASCENDING";
        }
        if (Long.valueOf(blockBefore)>Long.valueOf(block))
        {
            state="DESCENDING";
        }
    }
```

SO I INTRODUCED CODE ON THE RIGHT IMMEDIATELY SINCE WE KNOW THAT END RESULT IS NOT BASED ON THE INITIALSTATE, IT IS BASED ON THE ACTUAL RECURSION WHICH OCCURS AT EITHER NEW BLOCK SIZE OR PERFORMING AGAIN ON EXISTING BLOCK SIZE FOR REMAINING PART OF THE INITIAL NUMBER

I also feel there variables should be based on state. But I am not sure if it will impact outcome for other test cases

```
if (initialState=="ascending")
{
    if (state.equals("DESCENDING"))
```

```
if (initialState=="descending")
{
    if (state.equals("ASCENDING"))
```

And with an element of severe luck and persistence, I can see that it has infact given the correct output!!!

```
IN CATCH STATEMENT
23 are digit(s) of first block n(2) of 235461
235461 consists of ASCENDING numbers of group size(2): 23,54,61,
```

So now, I will run through all my test cases again including 13a, 13b.....
And I am hoping this is the final piece of the jigsaw... It has been a totally mentally exhausting challenge.

TEST CASE 1: (as per challenge) = FAIL (previously PASS)

```
final static String str = "123124125";
```

```
123 are digit(s) of first block n(3) of 123124125
123124125 consists of ASCENDING numbers of group size(3): ,
```

```
ascending("123124125") → true
// Contains a set of consecutive ascending numbers
// if grouped into 3's : 123, 124, 125
```

TEST CASE 2: (as per challenge) = PASS

```
final static String str = "101112131415";
```

```
10 are digit(s) of first block n(2) of 101112131415
101112131415 consists of ASCENDING numbers of group size(2): 10,11,12,13,14,15,
```

```
ascending("101112131415") → true
// Contains a set of consecutive ascending numbers
// if grouped into 2's : 10, 11, 12, 13, 14, 15
```

TEST CASE 3: (as per challenge) = PASS

```
final static String str = "32332432536";
```

```
String length: 11
Block size: 3
32332432536 does not consist of  number in any block size 3(n):
```

```
ascending("32332432536") → false
// Regardless of the grouping size, the numbers can't be consecutive.
```

TEST CASE 4: (as per challenge) = PASS (previously failed)

```
final static String str = "326325324323";
```

```
323 are digit(s) of first block n(3) of 323324325326
323324325326 consists of ASCENDING numbers of group size(3): 323,324,325,326,
```

```
ascending("326325324323") → false
// Though the numbers (if grouped into 3's) are consecutive but descending.
```

TEST CASE 4a = FAIL (previously failed)

```
final static String str = "3231302928";
```

```
Block size: 4
3231302928 does not consist of ascending number in any block size 4(n):
```

TEST CASE 4b = PASS

```
final static String str = "323324325326";
```

```
323 are digit(s) of first block n(3) of 323324325326
323324325326 consists of ASCENDING numbers of group size(3): 323,324,325,326,
```

TEST CASE 4c = PASS

```
final static String str = "32333435";
```

```
32 are digit(s) of first block n(2) of 32333435
32333435 consists of ascending numbers of group size(2): 32,33,34,35,
```

```
**** IN CATCH STATEMENT ****
32 are digit(s) of first block n(2) of 32333435
32333435 consists of ASCENDING numbers of group size(2): 32,33,34,35,
```

TEST CASE 5: (as per challenge) = PASS

```
final static String str = "666667";
```

```
ascending("666667") → true
// Consecutive numbers: 666 and 667.
```

```
666 are digit(s) of first block n(3) of 666667
666667 consists of ASCENDING numbers of group size(3): 666,667
```

TEST CASE 6: FAIL (previously PASS)

```
final static String str = "123456789";   //whole block ascending
```

```
1 are digit(s) of first block n(1) of 123456789
123456789 consists of ASCENDING numbers of group size(1): 1,2,3,4,5,6,,
```

TEST CASE 7: FAIL (previously PASS)

```
final static String str = "123124125";  //each 3 block ascending and digit in each block ascending
```

```
123 are digit(s) of first block n(3) of 123124125
123124125 consists of ascending numbers of group size(3): 123,124,125,
```

```
123 are digit(s) of first block n(3) of 123124125
123124125 consists of ASCENDING numbers of group size(3): ,
```

At this point, although some test cases have passed that previously failed...

It can be seen the impact of `if (state.equals("ASCENDING"))
{` instead of initialState=”Ascending”~

I have investigated it as follows

```
There are minimum of two blocks of size: 3
**** IN TRY STATEMENT ****
ASCENDING BLOCK
This is block: 125
This is block before: 124
This is state:
This is initial state: ascending
```

it picks up state variable once recursionCall is greater than 0. This is not case when it starts first examination at certain block size

This variable is set when recursionCall is equal to 0

As a result it fails to execute this:

```
if (state.equals("ASCENDING"))
{
    if (ascNums.length()==0)
    {
        sj.add(blockBeforeBlock);
        sj.add(blockBefore);
        sj.add(block);
    }
```

Easiest fix would be to perform state=initialState when recursionCall=0

And then we know state will be independent once the recursionCall exceeds 0

```
System.out.println("Performing recursion, examin
sj=new StringJoiner(",");

if (recursionCall==0)
{
    state=initialState;
}
```

```
if (recursionCall>0)
{
    if (Long.valueOf(blockBefore)<Long.valueOf(block))
    {
        state="ASCENDING";
```

I will quickly revisit all my test cases above...

TEST CASE 7: Applied fix PASS

So I have now run the test again:

```
123 are digit(s) of first block n(3) of 123124125
123124125 consists of ASCENDING numbers of group size(3): 123,124,125,
```

TEST CASE 6: PASS

```
1 are digit(s) of first block n(1) of 123456789
123456789 consists of ASCENDING numbers of group size(1): 1,2,3,4,5,6,7,8,9,
```

TEST CASE 4a: FAIL

TEST CASE 1: PASS

I will now continue rest of the tests, so far there are two failing test cases

TEST CASE 8: PASS

```
final static String str = "123124215";    // each 3 block ascending ONLY   (215 not ascending)
```

```
123 are digit(s) of first block n(3) of 123124215
123124215 consists of ascending numbers of group size(3): 123,124,215,
```

TEST CASE 9: PASS (previously FAIL)

```
final static String str = "123212421543452";    // each 5 block ascending ONLY
```

```
123 are digit(s) of first block n(3) of 123212421543452
123212421543452 consists of ASCENDING numbers of group size(3): 123,212,421,543,452,
```

TEST CASE 10: PASS (previously FAIL)

```
final static String str = "123452467956789";
```

```
123 are digit(s) of first block n(3) of 123452467956789
123452467956789 consists of ASCENDING numbers of group size(3): 123,452,467,956,789,
```

TEST CASE 11: FAIL (PREVIOUSLY PASS)

```
final static String str = "98765431";   //whole block descending
```

```
8 are digit(s) of first block n(1) of 98765431
98765431 consists of descending numbers of group size(1): 9,8,7,6,5,4,3,1,
```

```
This is block: 9876
This is block before: 5431
This is state:
This is initial state: DESCENDING
**** IN CATCH STATEMENT ****
5431 are digit(s) of first block n(4) of 98765431
98765431 consists of  numbers of group size(4): 5431,9876
```

TEST CASE 12: FAIL (PREVIOUSLY PASS)

```
final static String str =  "543542541";   //each 3 block descending and digit in each block descending
```

```
**** IN CATCH STATEMENT ****
541532 are digit(s) of first block n(6) of 543542541532
543542541532 consists of  numbers of group size(6): 541532,543542
```

I think its quite important to understand the failed cases before proceeding much further.

TEST CASE 13: FAIL (PREVIOUSLY PASS)

```
final static String str = "542541526";    // each 3 block descending ONLY
```

UNFORTUNATELY IT WAS GETTING TOO OUT OF CONTROL, THE MOMENT I TRIED TO PATCH ONE AREA, IT CAUSED SOMETHING ELSE TO DYSFUNCTION.
SO I HAVE ROLLED MY CODE BACK.. IT WAS APPROXIMATELY BEFORE IMPLEMENTATIONS IN TEST CASE 16 OCCURRED.

I have rebuilt the code up again.
And unfortunately find my self in the situation where I have to run through all the tests..
This will be my final attempt...