I have completed the coding without any syntax errors. It is a good opportunity to perform the testing.

I immediately noticed that I had surplus logic in my design.
Since the coding is very intricate, I am creating test cases which do not align with the cases provided in the challenge first.

TEST CASE 1: str = 1234689 FAIL

Keeping it extremely simple

```
Welcome to Online IDE!! Happy Coding :)
This is the block size: 1
There are minimum of two blocks of size: 1
This is the block size: 1
1234689 does not consist of ascending number in any block size(n)
1234689 does not consist of ascending number in any block size(n)
```

It can be seen that my code is a long way off the expected outcome.
But it has at least acknowledged the block size (line 1) and that there are minimum two blocks (line 2)
Although the last two statements are true, there is no evidence that it has explored any other block sizes.

TEST CASE 2: str = 123456789 FAIL

```
Welcome to Online IDE!! Happy Coding :)
This is the block size: 1
There are minimum of two blocks of size: 1
This is the block size: 1
123456789 does not consist of ascending number in any block size(n)
123456789 does not consist of ascending number in any block size(n)
```

We can clearly see that code is still nowhere near expectation since it has hit the most outer return statement which is false....

It will be a long hard look at the flow of my code.
I suspect it will not be clear cut.

TEST CASE 1a: str = 1234689 FAIL

I have enhanced my code. It can be seen now again that it returns to block size.
I also can see that it has correctly performed recursion and comparing the blocks.

```
Welcome to Online IDE!! Happy Coding :)
This is the block size: 1
There are minimum of two blocks of size: 1
This is block: 9
This is block before: 8
Performing recursion, examining block (8) with block before it(6)
This is the block size: 1
1234689 does not consist of ascending number in any block size(n)
1234689 does not consist of ascending number in any block size(n)
```

But it has not attempted the recursion in which it restores the String...
So I will once again examine the code further.
I will also remove the return statement

TEST CASE 2a: str = 123456789 FAIL

Even for a legitimate case in which it acknowledges the blocks, it still fails to proceed beyond one iteration, even though it returns back to top of recursive call as expected. So this clearly tells me that the screen output with following values might not be consistent with the actual logic. I will try to follow this code back into the recursive call.

```
Welcome to Online IDE!! Happy Coding :)
This is the block size: 1
There are minimum of two blocks of size: 1
This is block: 9
This is block before: 8
Performing recursion, examining block (8) with block before it(7)
This is the block size: 1
123456789 does not consist of ascending number in any block size(n)
123456789 does not consist of ascending number in any block size(n)
```

I can clearly see all the below statements are correct:

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

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

```
System.out.println("Performing recursion, examining block " + "("+blockBefore+")" +" with block before it" + "("+blockBeforeBlock+")");
```

But such is the intensity of the logic, I performed following in the recursive call

```
ascending(str.substring((str.length()-(3*n)),(str.length()-(2*n))));
```

But clearly that has affected the str..
The only intervention required is to truncate by existing block (width=n)

It has been updated as follows:

```
ascending(str.substring(0,(str.length()-n)));
```

TEST CASE 3: str = 1234689 FAIL

But I am getting much closer to the expected outcome

Welcome to Online IDE!! Happy Coding :)

Number to be examined: 123456789

This is the block size: 1

There are minimum of two blocks of size: 1

This is block: 9

This is block before: 8

Performing recursion, examining block (8) with block before it(7)

This is the block size: 1

There are minimum of two blocks of size: 1

This is block: 8

This is block before: 7

Performing recursion, examining block (7) with block before it(6)

This is the block size: 1

There are minimum of two blocks of size: 1

This is block: 7

This is block before: 6

Performing recursion, examining block (6) with block before it(5)

This is the block size: 1

There are minimum of two blocks of size: 1

This is block: 6

This is block before: 5

Performing recursion, examining block (5) with block before it(4)

This is the block size: 1

There are minimum of two blocks of size: 1

This is block: 5

This is block before: 4

Performing recursion, examining block (4) with block before it(3)

This is the block size: 1

There are minimum of two blocks of size: 1

This is block: 4

This is block before: 3

Performing recursion, examining block (3) with block before it(2)

This is the block size: 1

There are minimum of two blocks of size: 1

This is block: 3

This is block before: 2

Performing recursion, examining block (2) with block before it(1)

This is the block size: 1

There are minimum of two blocks of size: 1

This is block: 2

This is block before: 1

123456789 does not consist of ascending number in any block size(n)

123456789 does not consist of ascending number in any block size(n)

123456789 does not consist of ascending number in any block size(n)

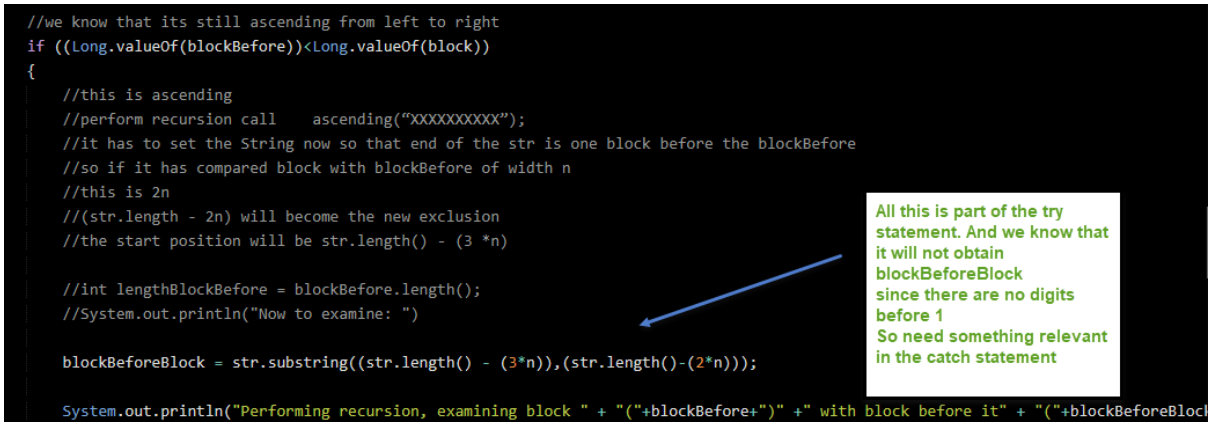
123456789 does not consist of ascending number in any block size(n)

123456789 does not consist of ascending number in any block size(n)

123456789 does not consist of ascending number in any block size(n)

123456789 does not consist of ascending number in any block size(n)

** Process exited - Return Code: 0 **



So I think I should be fairly close to the finish.

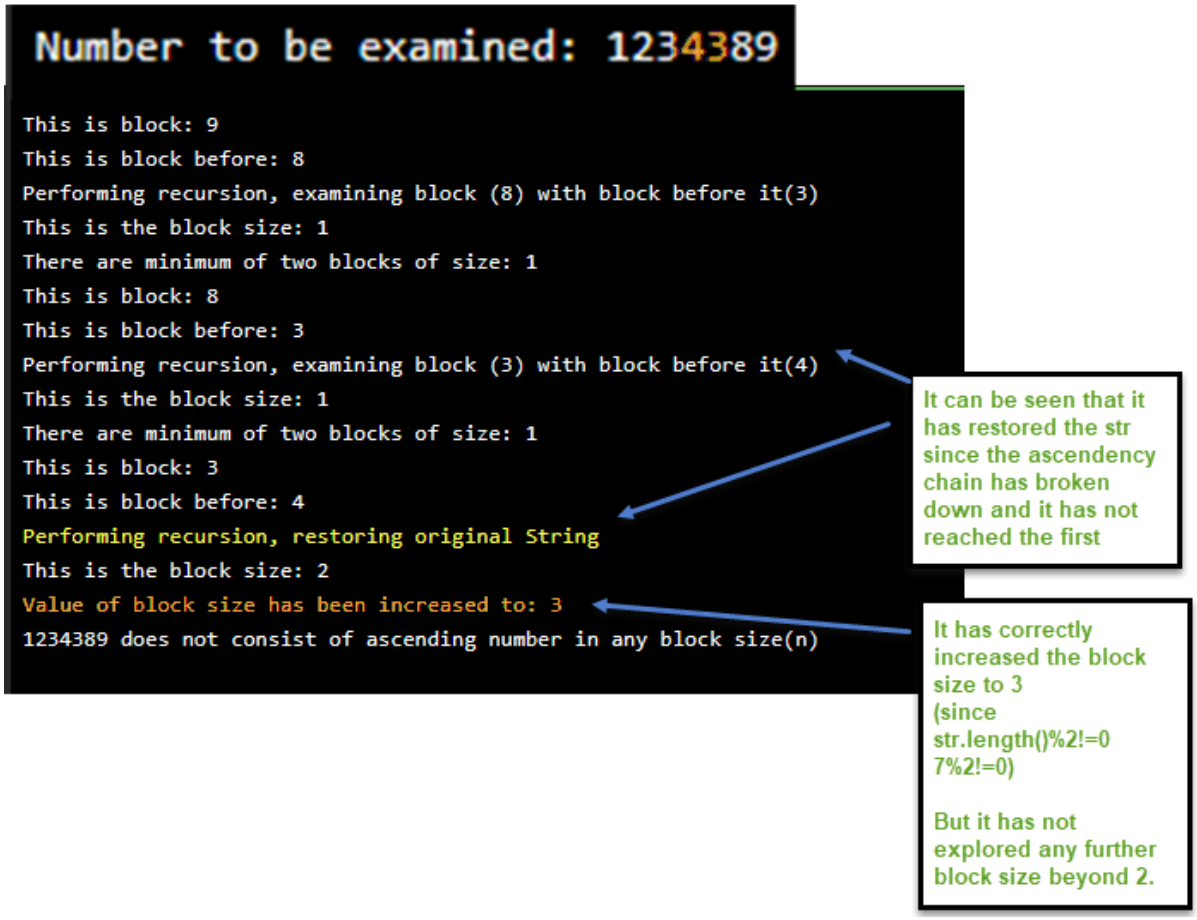
TEST CASE 4: str = 1234689 PASS

```
1 are digit(s) of first block n of 1234689
1234689 consists of ascending numbers of group size: 1
```

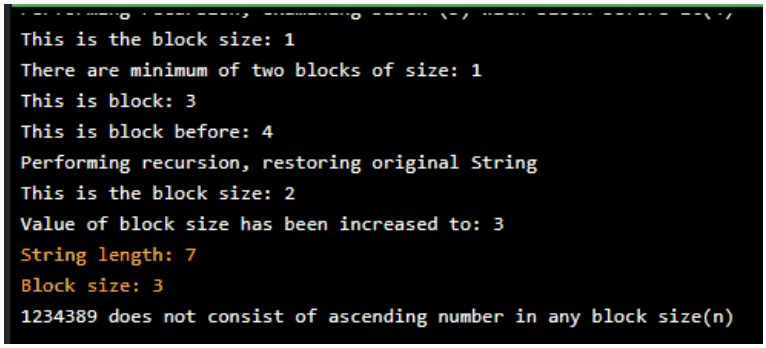
TEST CASE 4a: str = 123456789 PASS

```
1 are digit(s) of first block n of 123456789
123456789 consists of ascending numbers of group size: 1
```

TEST CASE 4b: str = 1234389 (examining a case where there is not ascendency) FAIL



We can see the problem arises here, since it will not enter the else statement.
It can be looked at from two perspectives:
Firstly, it is an associated else with the if



So even if the block size was a factor (in this instance it is not), the else would not execute.

Also, more importantly, it can be seen that block size has increased only up to 3.

Resolution:
if the length of n is even or odd number, we expect only to perform block sizes up to n/2
This will ensure there are two blocks

if the length of n is an odd number, can perform up to n+1/2

However this is the first time so far that iteration seems to be logic going forward. In real world, I would expect to use a do while loop or similar to increment n until the above two conditions are met for odd and even numbers.

Only other solution is to call recursive loop again until it satisfies above.

BEFORE:

```
if(!(str.length()%n==0))
{

    //It appears that first area to check would be to perform  Str.length()%n ==0  within
    //ascending("XXXXXXXXXX")
    //This will ensure that the divided block is viable.
    //Otherwise perform  n=n+1   and   ascending("XXXXXXXXXX")

    n++;
    System.out.println("Value of block size has been increased to: " + n);

    str=strBackup;

    System.out.println("String length: " + str.length());
    System.out.println("Block size: " + n);

    ascending(str);
```

AFTER:

```
    System.out.println("String length: " + str.length());
    System.out.println("Block size: " + n);

    if (str.length()%2==0)
    {
        if (n<(str.length()/2))
        {
            ascending(str);
        }
    }
    else
    {
        if (n<((str.length()+1)/2))
        {
            ascending(str);
        }
    }
}
```

TEST CASE 4b: str = 1234389 (examining a case where there is not ascendency) = PASS

N.B This only passes because there are no factors in 7 except 1 and 7 since it is a prime number.

If the length of String was 15, there would be an issue since n would have increased from 1,2,3 but it would have not entered the else loop since the incrementation is occurring in the if loop.

So my immediate priority is to now address the else loop.

```
        }
    }
}

else  //str.length()%n===0  // there are discrete number of blocks based on block size n
{

    //it can only perform operation if there are minimum two blocks in the str
```

I can see straight away that for an unknown reason I chose to execute the if with a not (!)
I rarely undertake this approach.

```
if(!(str.length()%n==0))
{
```

So this clearly points me towards making ALL contents in **if => else** statement....

So, this effectively means that contents in **else** statement should become if statement.
And hence the boolean logic should trigger true

```
if((str.length()%n==0))
{

    //it can only perform operation if there are minimum two blocks in the str
    if (!(str.length()<(2*n)))
    {
        System.out.println("There are minimum of two blocks of size: " + n);
        //unsure if this can cause ArrayIndexOutOfBoundsException
        //or StringIndexOutOfBoundsException
        //most likely it can not unless there are not two instances of the the block size
```

I am now ready to perform Test cases given in the challenge.
Mentally it now appears I have identified all the areas during my testing phase.
I am surprised I failed to pick up on a few areas. But I expected this, hence I launched my testing at a suitable phase.

TEST CASE 5a: PASS

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

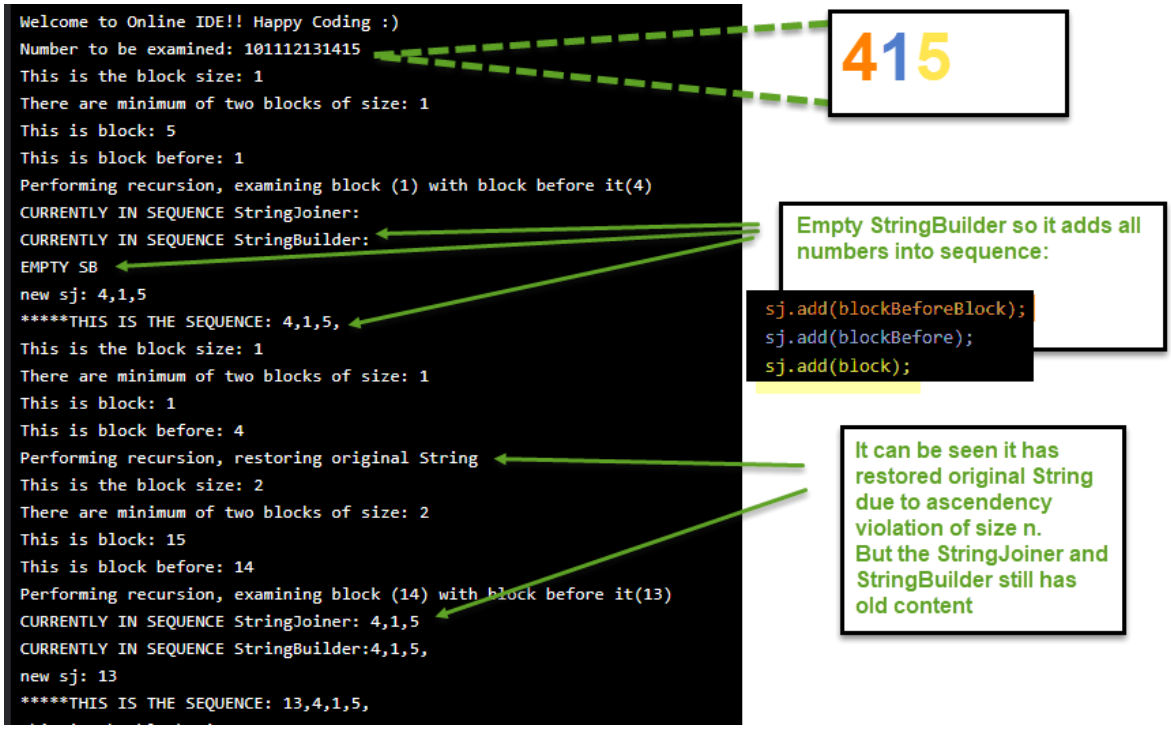
```
123 are digit(s) of first block n(3) of 123124125
123124125 consists of ascending numbers of group size(3): 123,124,125,
```

TEST CASE 5b: FAIL

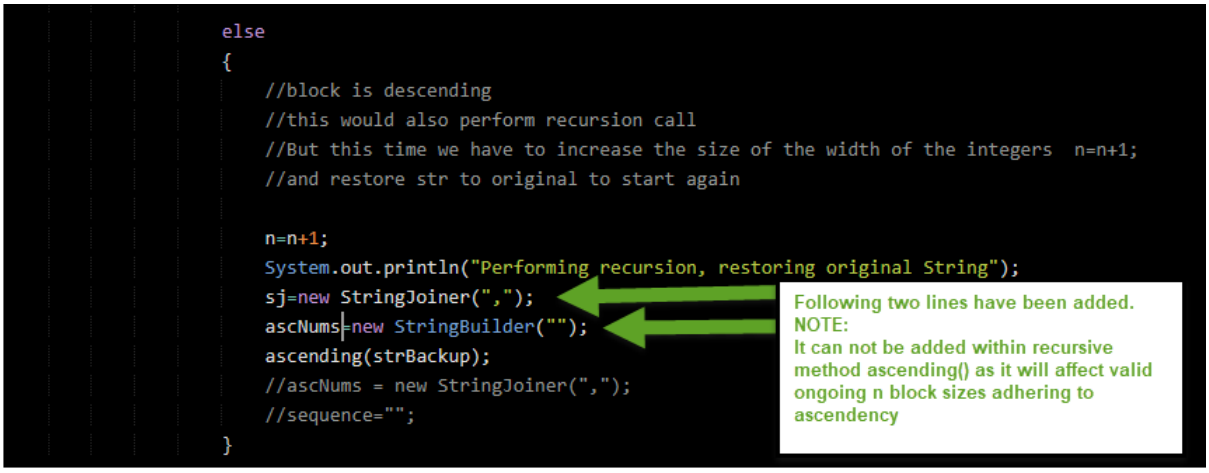
```
ascending("101112131415") → true
// Contains a set of consecutive ascending numbers
// if grouped into 2's : 10, 11, 12, 13, 14, 15
```

```
10 are digit(s) of first block n(2) of 101112131415
101112131415 consists of ascending numbers of group size(2): 10,11,12,13,4,1,5,
```

I am just so surprised how 5a has passed and 5b has not, so I am having to investigate the intermediate StringJoiner and StringBuilder!!



So, I will try to clear the contents.



TEST CASE 5bv2: PASS

Just to be sure no logic has had adverse effect, I retested TEST CASE 5a and it passed

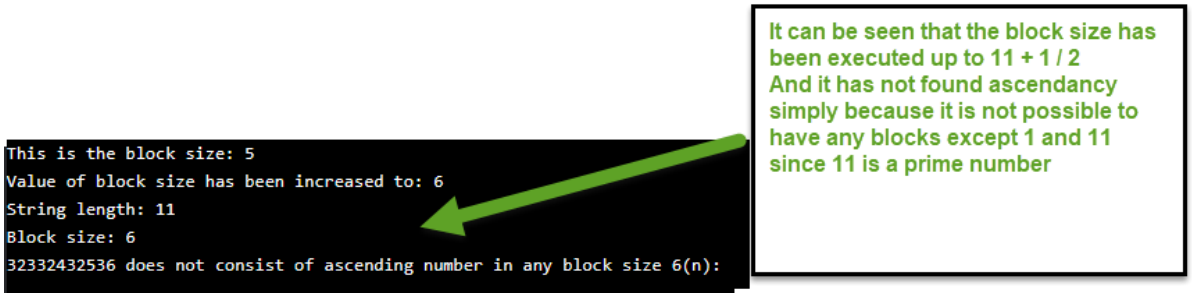
```
ascending("101112131415") → true
// Contains a set of consecutive ascending numbers
// if grouped into 2's : 10, 11, 12, 13, 14, 15
```

```
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,
```

TEST CASE 5c: I consider this the first real test to process a large count of block size and return no ascendency

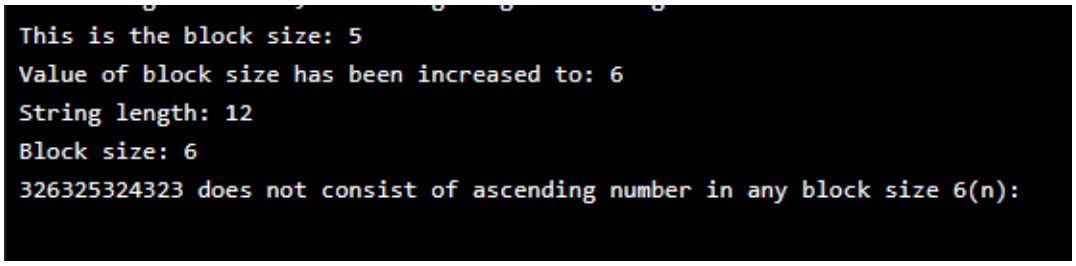
PASS

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



TEST CASE 5d: PASS

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



TEST CASE 5e: PASS

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

```
*****THIS IS THE SEQUENCE: 66,66,67,
This is the block size: 2
There are minimum of two blocks of size: 2
This is block: 66
This is block before: 66
Performing recursion, restoring original String
This is the block size: 3
There are minimum of two blocks of size: 3
This is block: 667
This is block before: 666
666 are digit(s) of first block n(3) of 666667
666667 consists of ascending numbers of group size(3):
```