GIT Department of Computer Engineering
CSE 222/505 - Spring 2022
Homework #4 Report

Mert Gürşimşir
1901042646

PART 1

QUESTION 1

Time complexity:

- Best case:

  $T(n) = \theta(1)$ {At least 1 string is empty}

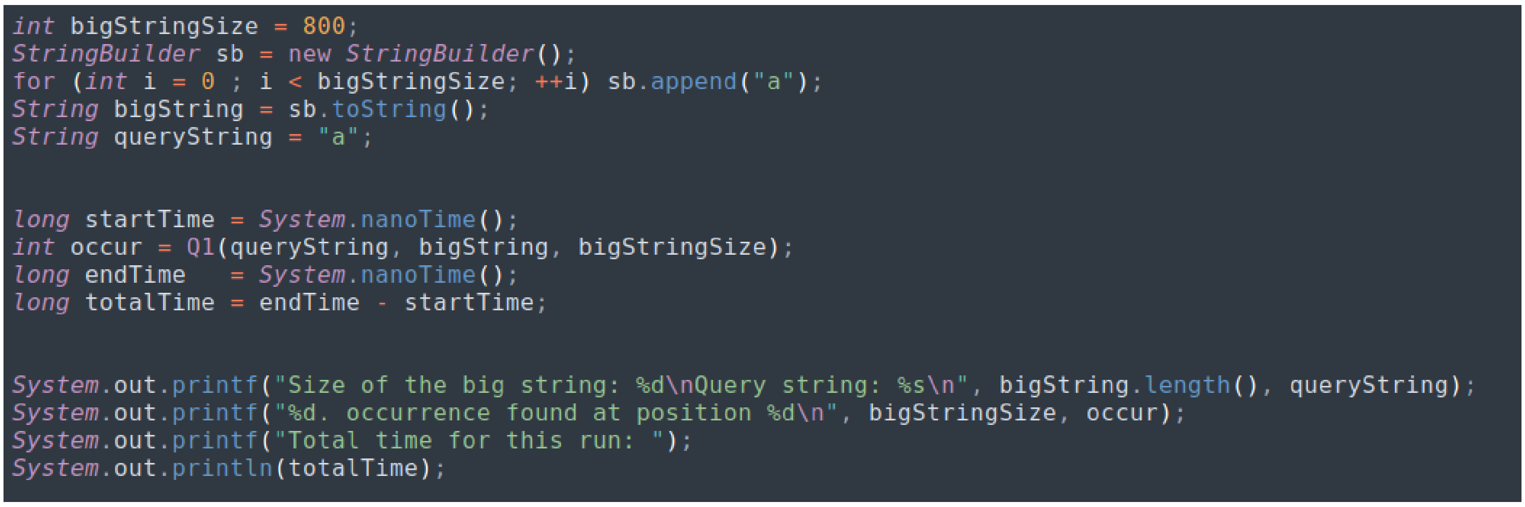
- Worst case:

$$T(n) = T(n-1) + \theta(1)$$
$$T(n) = T(n-2) + \theta(1) + \theta(1)$$
$$T(n) = T(n-3) + \theta(1) + \theta(1) + \theta(1)$$
$$\dots$$
$$T(n) = T(n-k) + k\theta(1)$$
$$k = n:$$
$$T(n) = T(0) + n\theta(1) \qquad (T(0) = \theta(1))$$
$$T(n) = \theta(n)$$

Overall : T(n) = O(n)

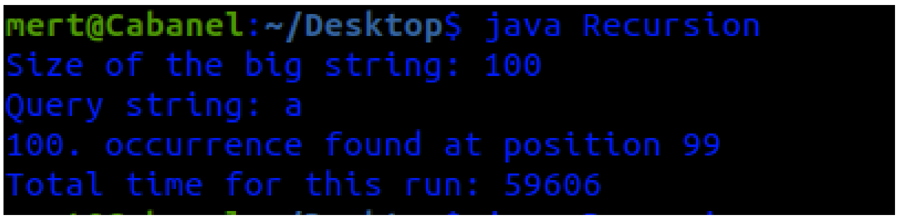
n is the size of the big string. The code looks to the big string and compare each character one by one. So running time is linear.

Test:

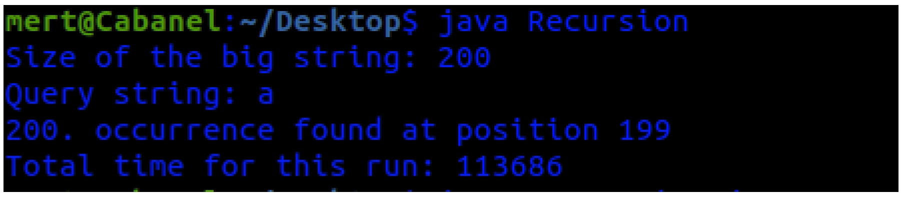


```java
int bigStringSize = 800;
StringBuilder sb = new StringBuilder();
for (int i = 0 ; i < bigStringSize; ++i) sb.append("a");
String bigString = sb.toString();
String queryString = "a";


long startTime = System.nanoTime();
int occur = Q1(queryString, bigString, bigStringSize);
long endTime   = System.nanoTime();
long totalTime = endTime - startTime;


System.out.printf("Size of the big string: %d\nQuery string: %s\n", bigString.length(), queryString);
System.out.printf("%d. occurrence found at position %d\n", bigStringSize, occur);
System.out.printf("Total time for this run: ");
System.out.println(totalTime);
```

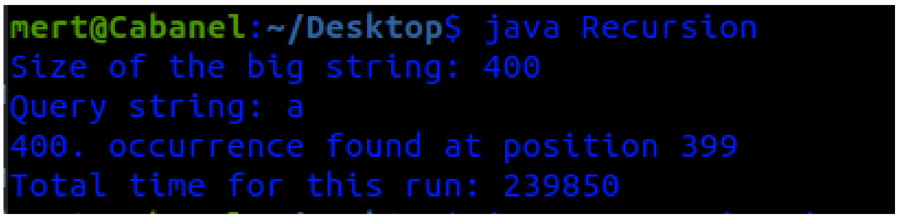
I am creating big string which is “aaaa…” (with size 800 as an example above).
Then I am looking for “a” which is query string for the size of the big string’s occurrence (for worst case).



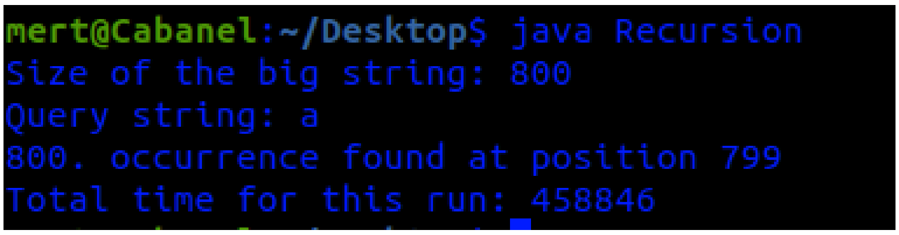
```
mert@Cabanel:~/Desktop$ java Recursion
Size of the big string: 100
Query string: a
100. occurrence found at position 99
Total time for this run: 59606
```



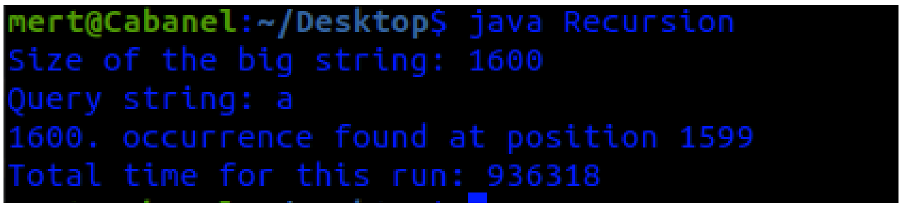
```
mert@Cabanel:~/Desktop$ java Recursion
Size of the big string: 200
Query string: a
200. occurrence found at position 199
Total time for this run: 113686
```



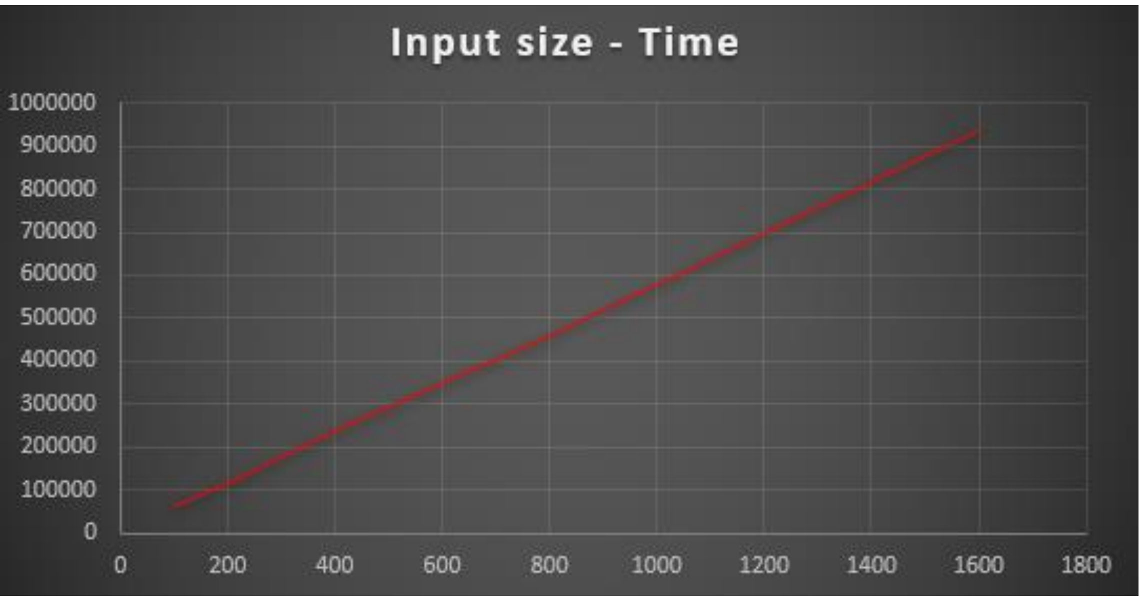
```
mert@Cabanel:~/Desktop$ java Recursion
Size of the big string: 400
Query string: a
400. occurrence found at position 399
Total time for this run: 239850
```



```
mert@Cabanel:~/Desktop$ java Recursion
Size of the big string: 800
Query string: a
800. occurrence found at position 799
Total time for this run: 458846
```



```
mert@Cabanel:~/Desktop$ java Recursion
Size of the big string: 1600
Query string: a
1600. occurrence found at position 1599
Total time for this run: 936318
```



Size	Time
100	59606
200	113686
400	239850
800	458846
1600	936318

(Times are in nanoseconds)

QUESTION 2

Time complexity:

- Best case:

  $T(n) = \theta(1)$ {Array is empty}

- Worst case:

  $T(n) = 2T(n/2) + \theta(1)$

  $T(n) = 2(2T(n/4) + \theta(1)) + \theta(1) = 2^2T(n/4) + 3\theta(1)$

  $T(n) = 2^2[2T(n/8) + \theta(1)] + 3\theta(1) = 2^3T(n/8) + 7\theta(1)$

  $T(n) = 2^3[2T(n/16) + \theta(1)] + 7\theta(1) = 2^4T(n/2^4) + 2^3\theta(1) + 2^2\theta(1) + 2^1\theta(1) + 2^0\theta(1)$

  $T(n) = 2^5T(n/2^5) + 2^4\theta(1) + 2^3\theta(1) + 2^2\theta(1) + 2^1\theta(1) + 2^0\theta(1)$

  …

  $T(n) = 2^kT(n/2^k) + 2^{k-1}\theta(1) + 2^{k-2}\theta(1) + \ldots + 2^0\theta(1)$

  $2^k = n$, $k = log_2 n$:

  $T(n) = nT(1) + (2^k - 1)\theta(1) = nT(1) + (n - 1)\theta(1)$ $(T(1) = \theta(1))$

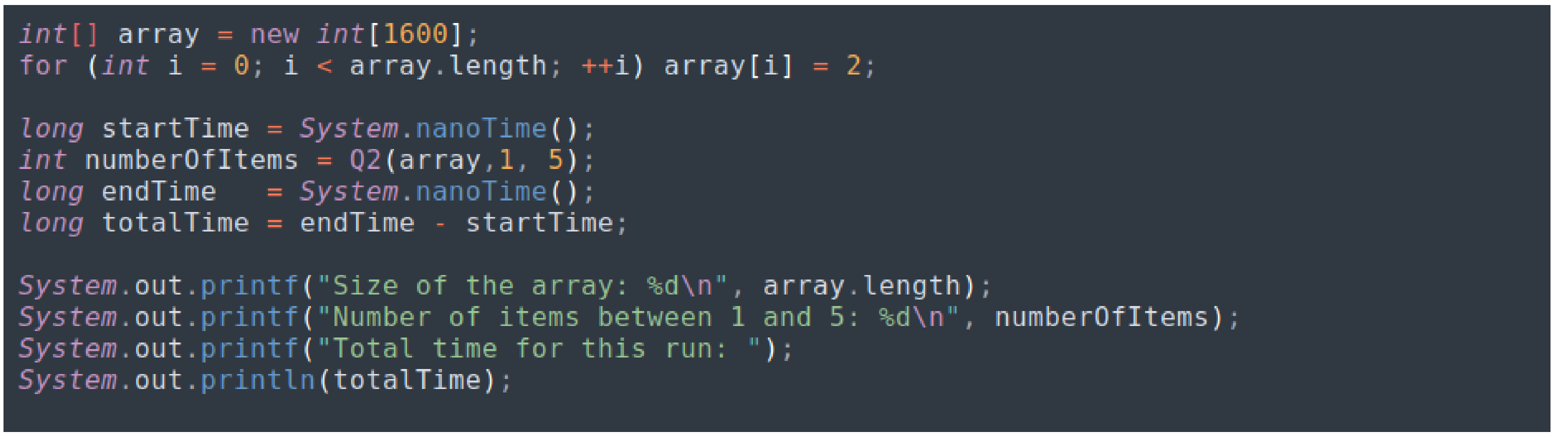
  $T(n) = n + n - 1 = 2n - 1$

  $T(n) = \theta(n)$

Overall : T(n) = O(n)

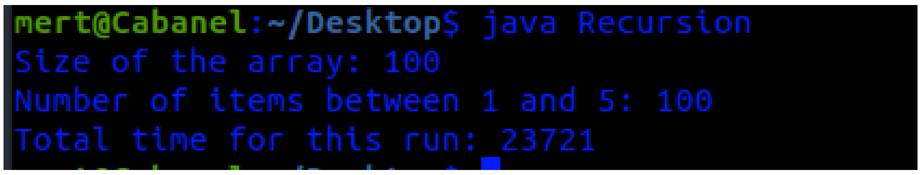
n is the size of the array. Firstly, code looks middle element then according to that it looks right or left part of the array. If middle element of the array is between the values that given as parameter, then code returns 1 + elements that are between the values in left subarray + elements that are between the values in right subarray. This creates linear time.

Test:

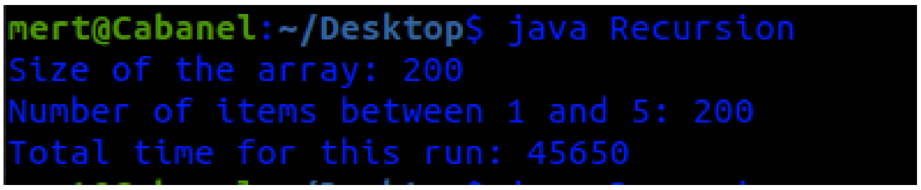


```java
int[] array = new int[1600];
for (int i = 0; i < array.length; ++i) array[i] = 2;

long startTime = System.nanoTime();
int numberOfItems = Q2(array,1, 5);
long endTime   = System.nanoTime();
long totalTime = endTime - startTime;

System.out.printf("Size of the array: %d\n", array.length);
System.out.printf("Number of items between 1 and 5: %d\n", numberOfItems);
System.out.printf("Total time for this run: ");
System.out.println(totalTime);
```

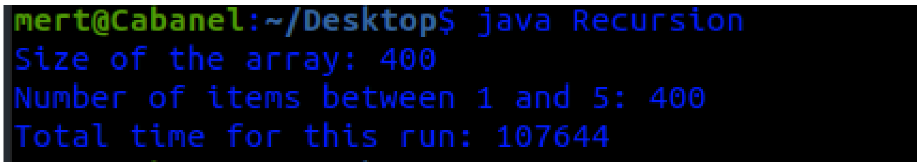
I am creating an array (with size 1600 as example above) whose values are all 2. Then I am looking for values between 1 and 5 (all elements). So this should create worst case.



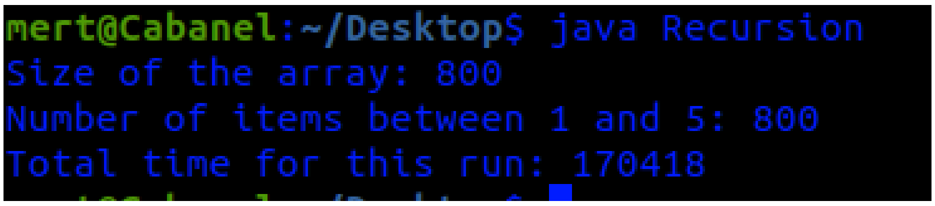
```
mert@Cabanel:~/Desktop$ java Recursion
Size of the array: 100
Number of items between 1 and 5: 100
Total time for this run: 23721
```



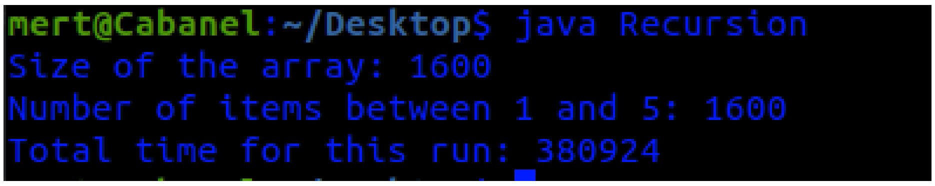
```
mert@Cabanel:~/Desktop$ java Recursion
Size of the array: 200
Number of items between 1 and 5: 200
Total time for this run: 45650
```



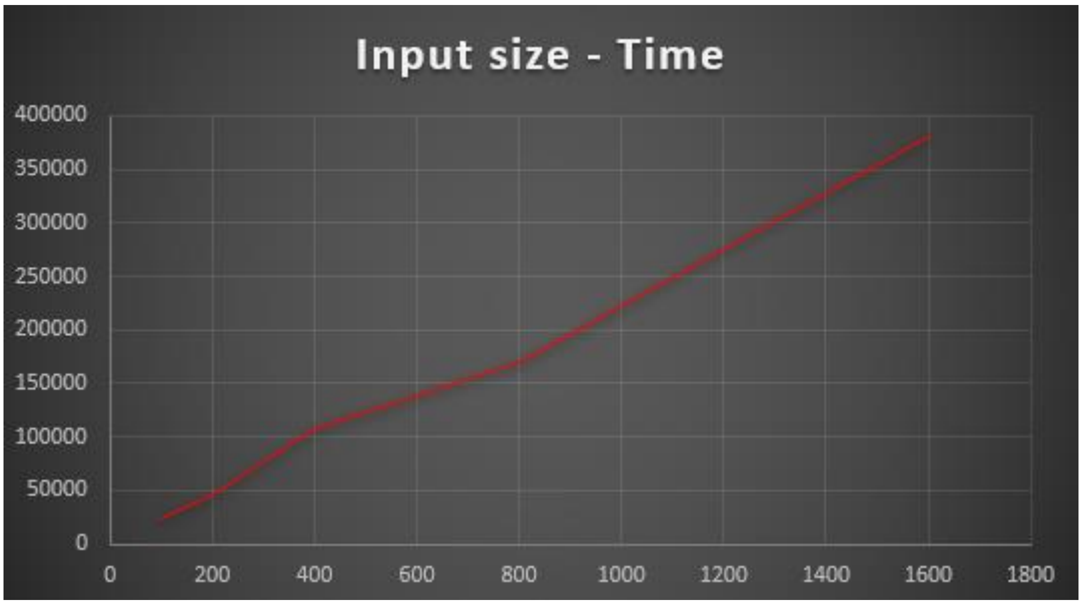
```
mert@Cabanel:~/Desktop$ java Recursion
Size of the array: 400
Number of items between 1 and 5: 400
Total time for this run: 107644
```



```
mert@Cabanel:~/Desktop$ java Recursion
Size of the array: 800
Number of items between 1 and 5: 800
Total time for this run: 170418
```



```
mert@Cabanel:~/Desktop$ java Recursion
Size of the array: 1600
Number of items between 1 and 5: 1600
Total time for this run: 380924
```



Size	Time
100	23721
200	45650
400	107644
800	170418
1600	380924

(Times are in nanoseconds)

QUESTION 3

Time complexity:

- Best case:

  $T(n) = \theta(1)$ {Array is empty}

- Worst case:

  $T(n) = T(n-1) + \theta(n)$

  $T(n) = T(n-2) + \theta(n-1) + \theta(n)$

  $T(n) = T(n-3) + \theta(n-2) + \theta(n-1) + \theta(n)$

  …

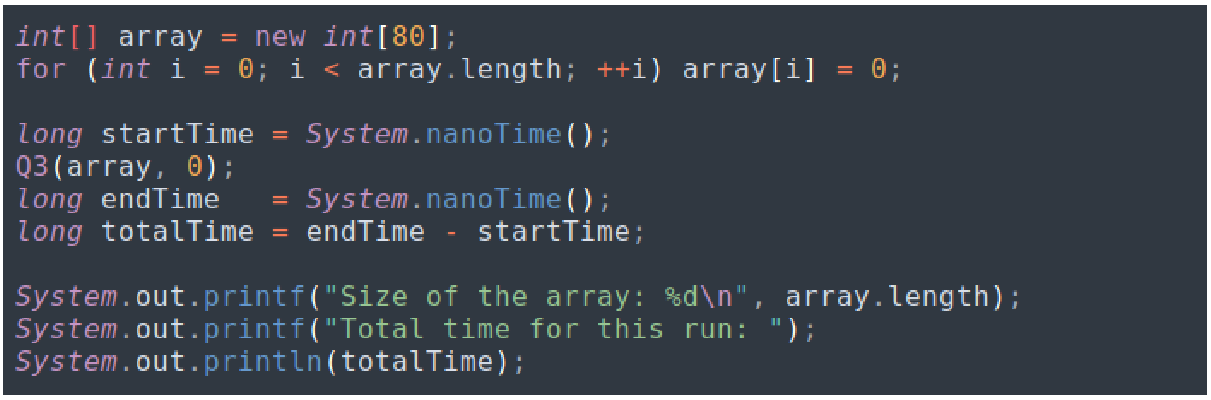
  $T(n) = T(0) + \theta(1) + \theta(2) + \ldots + \theta(n-1) + \theta(n)$ $\quad (T(0) = \theta(1))$

  $T(n) = \theta(n^2)$

Overall : T(n) = O(n^2)

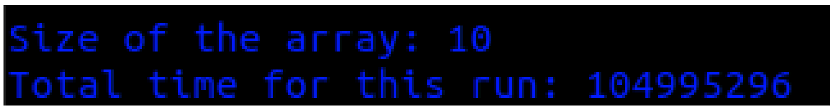
n is the size of the array. Firstly, code will look if any subarray that starts with first element creates the wanted sum. Then it looks if any subarray that starts with second element creates the wanted sum, etc. So overall running time is quadratic.

Test:

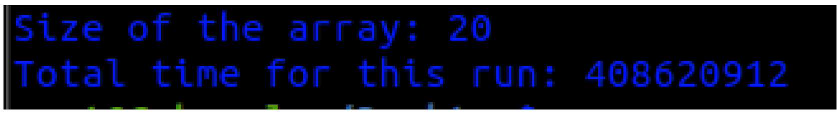


```java
int[] array = new int[80];
for (int i = 0; i < array.length; ++i) array[i] = 0;

long startTime = System.nanoTime();
Q3(array, 0);
long endTime   = System.nanoTime();
long totalTime = endTime - startTime;

System.out.printf("Size of the array: %d\n", array.length);
System.out.printf("Total time for this run: ");
System.out.println(totalTime);
```

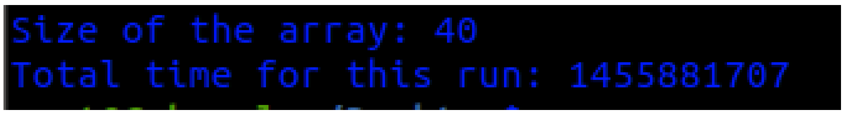
I am creating an array with every element is 0 (because if we look for the sum 0, every subarray provides the desired subarray for the worst case) with size 80 as example above.



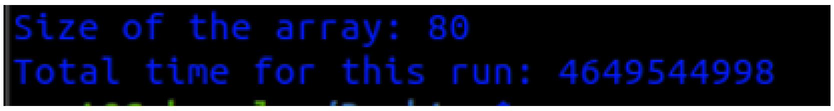
```
Size of the array: 10
Total time for this run: 104995296
```



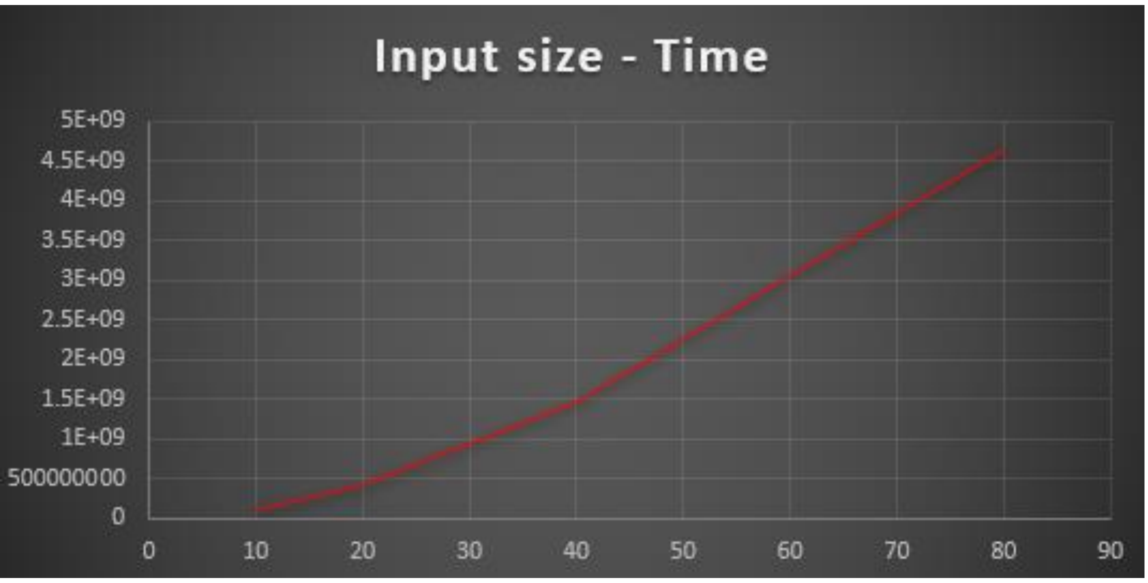
```
Size of the array: 20
Total time for this run: 408620912
```



```
Size of the array: 40
Total time for this run: 1455881707
```



```
Size of the array: 80
Total time for this run: 4649544998
```



Size	Time
10	84743391
20	408620912
40	1455881707
80	4649544998

(Times are in nanoseconds)

QUESTION 4

Output of this method is multiplication of 2 integers.

Firstly, our base case is one of the integers being less than 10.

Then we find maximum number of digits between 2 integer parameters.

"half" is the half of the maximum number of digits. We use this half value while we are splitting 2 integer parameters.

The integer which has the maximum number of digits is being split from the middle. Other integer is also being split according to the half value.

After these splitting process; int1 and int3 become left part of the integers, int2 and int4 become right part of the integers.

Now we can multiply right parts of the integers (sub0), and left part of the integers (sub1). We assumed our code is working well for multiplication.

Also, we add right and left parts of the 2 split integers individually and then multiply these 2 values (sub1).

return part is similar to grid method but not exactly. We have multiplied left parts of the integers and we should multiply it with 10^{2*half} because "half" is the half of the maximum number of digits, and we are working on decimal numbers.

We also multiply the sum of right and left parts of each two integers. Then we add this to result after we multiplied it with 10^{half} because we are working on decimal numbers. Left part of the result is completed one paragraph above, and right part of the array will be completed one paragraph below. In this paragraph, we have completed middle of the result.

Then we can easily add result of multiplication of the right parts of the integers.

Time complexity:
M(n) : number of multiplication operations

- M(n) = 1 if one of the integers < 10
- M(n) = 3M(n-1) + 1 if both of the integers > 10
    - M(n) = 3M(n-1) + 1
    - M(n) = 3[3M(n-2) + 1] + 1
    - M(n) = 3^2[3M(n-3) + 1] + 3 + 1 = 3^3T(n-3) + 3^2 + 3^1 + 3^0
    - …
    - M(n) = 3^k(M(n-k)) + ($3^k - 1$)/2
    - M(n) = 3^n

T(n) = O(3^n) as well because running time depends on the basic operation.

QUESTION 5

Other than recursive calls, method needs linear time to print the 1D array.
For each block, we may have 4 recursive calls at most.
So we can say the time complexity of this method is 4n x n = $4n^2$.
Overall: T(n) = θ(n^2)

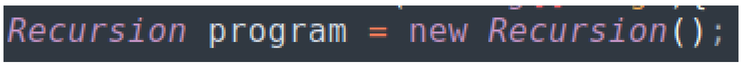
PART 2

1. SYSTEM REQUIREMENTS

This is a program for these 4:

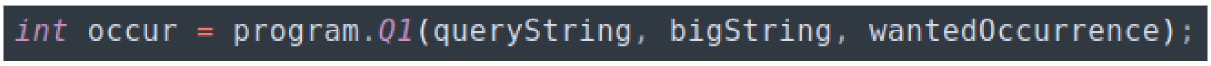
1. searching ith occurrence of the query string in the big string
2. finding the number of items in the sorted array between 2 given integers
3. finding contiguous subarray/s that the sum of theirs items is equal to given integer
4. calculating all the possible configurations to fill the 1D array with colored-blocks with length at least 3

Firstly, an object of the Recursion class must be created to use the recursive methods:



```
Recursion program = new Recursion();
```

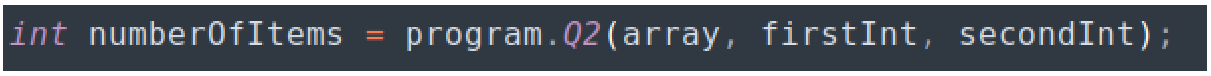
To run the first program, there must be a query string, a big string, and the occurrence number respectively.



```
int occur = program.Q1(queryString, bigString, wantedOccurrence);
```

Then occur will have the wantedOccurenceth number if it exists, else it will have -1.

To run the second program, there must be a sorted array, and 2 integers that find the elements in the array in between them.



```
int numberOfItems = program.Q2(array, firstInt, secondInt);
```

Then numberOfItems will have the number of items in the array between 2 given integer values.

To run the third program, there must be an array, and sum value.

```
program.Q3(arrayUnsorted, sum);
```

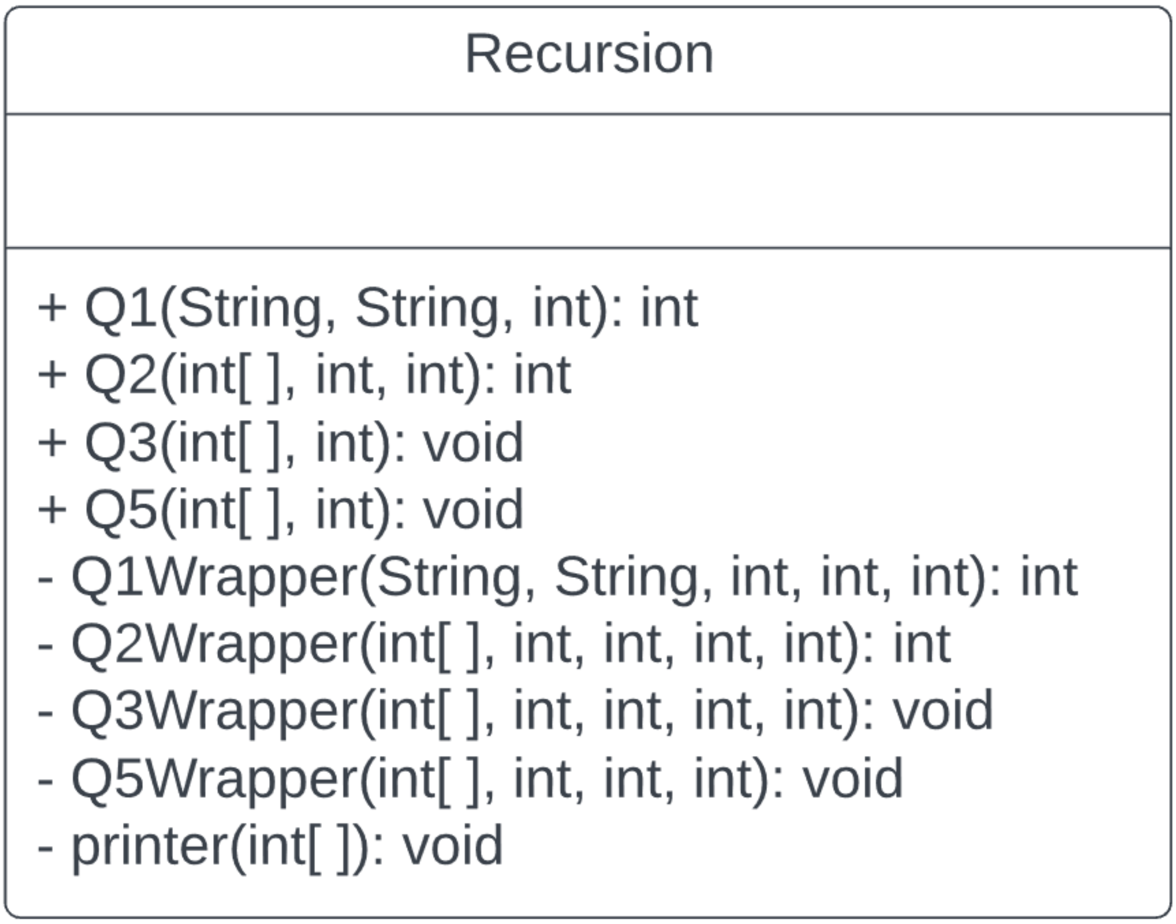
Then this will find and print contiguous subarray/s that the sum of theirs items is equal to given integer (sum).

To run the fourth program, there must be a 1D array, and length of that 1D array.

```
program.Q5(arrayBlocks, L);
```

Then this will print all the possible configurations to fill this array with colored-blocks with length at least 3. Colored-blocks are represented with 'X'.

2. CLASS DIAGRAM



```
Recursion
---------------------------------------------
---------------------------------------------
+ Q1(String, String, int): int
+ Q2(int[ ], int, int): int
+ Q3(int[ ], int): void
+ Q5(int[ ], int): void
- Q1Wrapper(String, String, int, int, int): int
- Q2Wrapper(int[ ], int, int, int, int): int
- Q3Wrapper(int[ ], int, int, int, int): void
- Q5Wrapper(int[ ], int, int, int): void
- printer(int[ ]): void
```

3. PROBLEM SOLUTION APPROACH

For the first question, I have to look ith occurrence of the query string in big string. So firstly, I have decided to look at the big string's characters one by one each in recursive call. Then I have compared every these characters with the query strings character starting from query string's first character. If there is a match, I continued to look other remaining characters to identify the exact match.

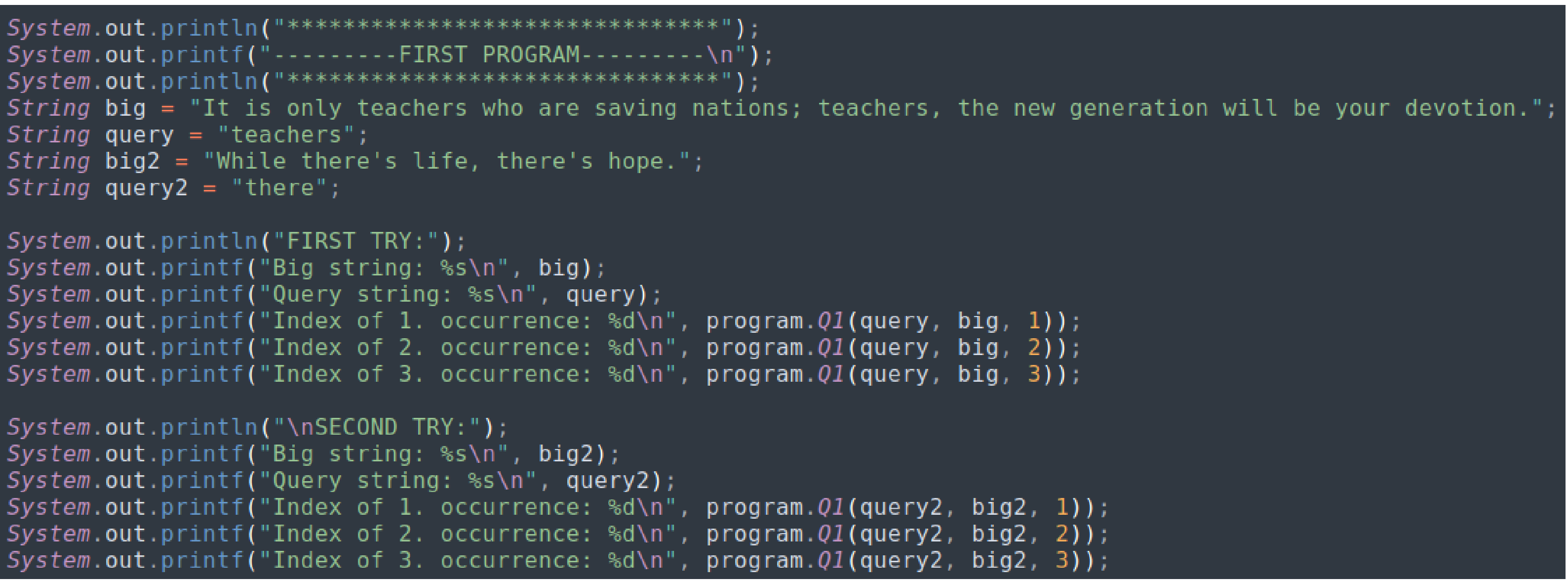
For the second question, I have to search number of items in the sorted array between two given integer values. So firstly, I have looked at the middle element. If it is bigger than the bigger one of the given integers, I looked at the left part of the array; otherwise I looked right part of the array. Then I compared the middle value with the given integer values. If it is between the 2 integers, I have returned 1 + elements that are between the values in left subarray + elements that are between the values in right subarray with 2 recursive calls. That solved my problem.

For the third question, I have to search contiguous subarray/s that the sum of its/theirs items is equal to a given integer value. This was hard for me because after I finished my first version, I have noticed the existence of negative values. So I couldn't just remove first element and look remaining array if the first element is bigger than the given integer. So firstly, code will look if any subarray that starts with first element creates the wanted sum. Then it looks if any subarray that starts with second element creates the wanted sum, and it goes on like this.

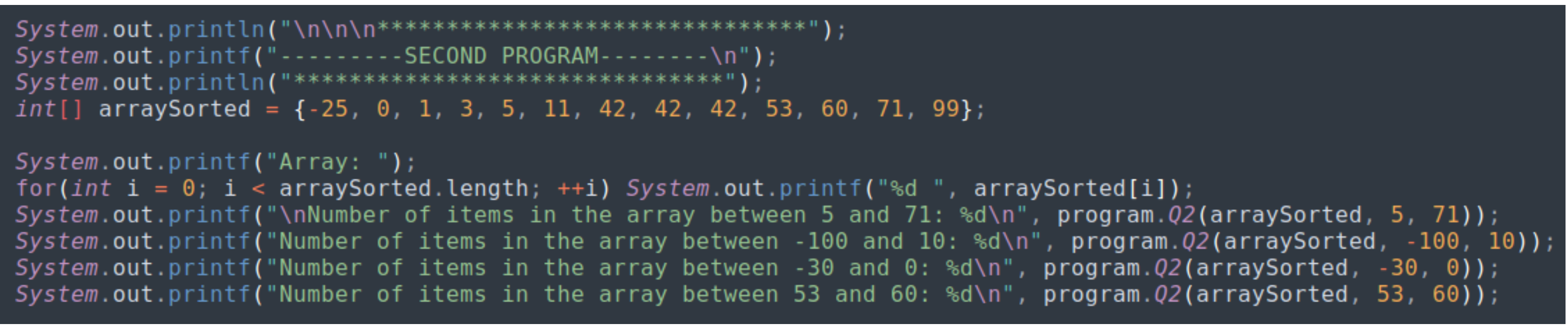
For the fourth question, I have to explain the given method. Firstly, I have tracked the method on paper. In that way I have found an integer at the end and noticed that it is the multiplication of the given 2 numbers. Then I try to understand each part of the method in detail.

For the fifth question, I have to print all the configurations to fill the 1D array with colored-blocks with length at least 3. Firstly, I have created my base cases. That makes my life quite easy. After that I need to design an algorithm. At my first try, my code works only for up to size 7 arrays. That was not wanted. So I improved my algorithm with 3 more recursive calls to try to fill the remaining array with same length blocks, 1 less length blocks, and 1 more length blocks . That solved my problem.

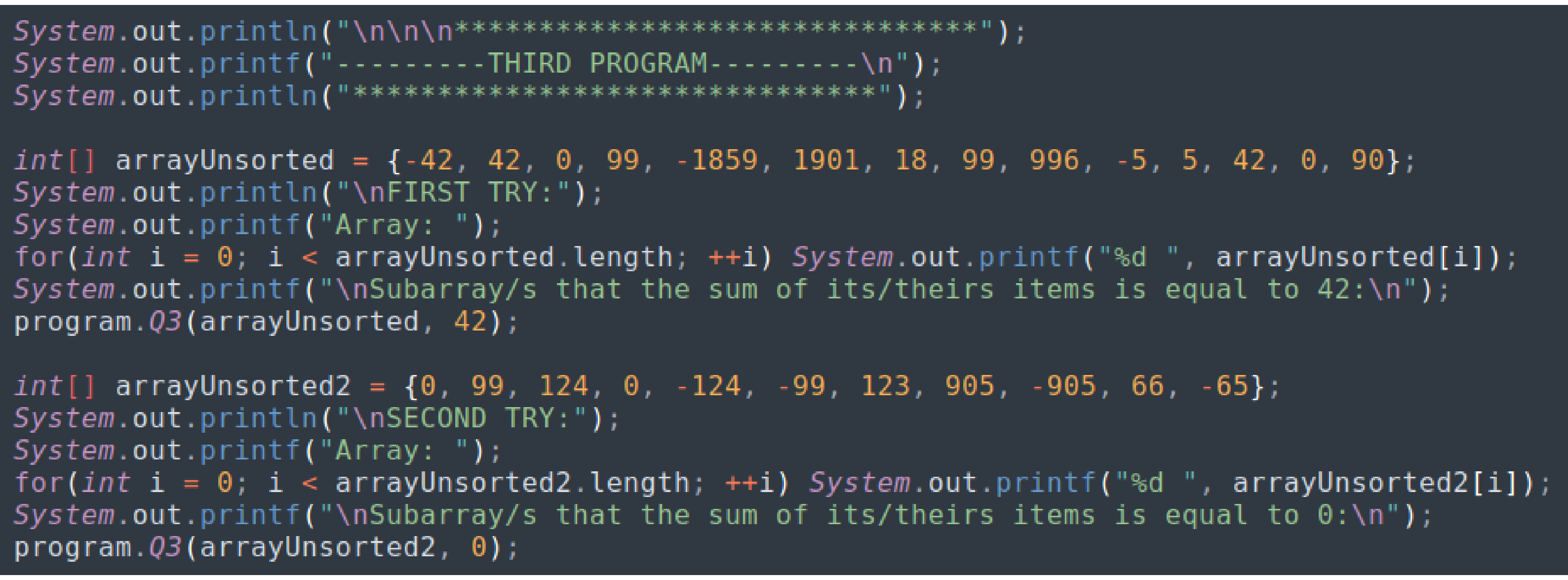
4. TEST CASES



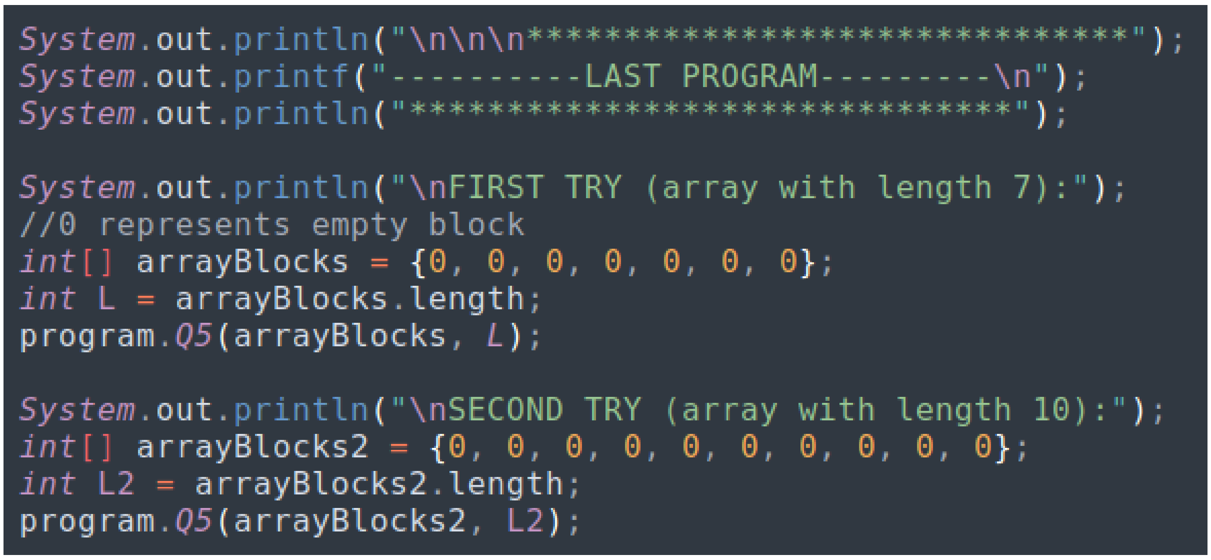
```java
System.out.println("********************************");
System.out.printf("---------FIRST PROGRAM--------\n");
System.out.println("********************************");
String big = "It is only teachers who are saving nations; teachers, the new generation will be your devotion.";
String query = "teachers";
String big2 = "While there's life, there's hope.";
String query2 = "there";

System.out.println("FIRST TRY:");
System.out.printf("Big string: %s\n", big);
System.out.printf("Query string: %s\n", query);
System.out.printf("Index of 1. occurrence: %d\n", program.Q1(query, big, 1));
System.out.printf("Index of 2. occurrence: %d\n", program.Q1(query, big, 2));
System.out.printf("Index of 3. occurrence: %d\n", program.Q1(query, big, 3));

System.out.println("\nSECOND TRY:");
System.out.printf("Big string: %s\n", big2);
System.out.printf("Query string: %s\n", query2);
System.out.printf("Index of 1. occurrence: %d\n", program.Q1(query2, big2, 1));
System.out.printf("Index of 2. occurrence: %d\n", program.Q1(query2, big2, 2));
System.out.printf("Index of 3. occurrence: %d\n", program.Q1(query2, big2, 3));
```



```java
System.out.println("\n\n\n********************************");
System.out.printf("---------SECOND PROGRAM--------\n");
System.out.println("********************************");
int[] arraySorted = {-25, 0, 1, 3, 5, 11, 42, 42, 42, 53, 60, 71, 99};

System.out.printf("Array: ");
for(int i = 0; i < arraySorted.length; ++i) System.out.printf("%d ", arraySorted[i]);
System.out.printf("\nNumber of items in the array between 5 and 71: %d\n", program.Q2(arraySorted, 5, 71));
System.out.printf("Number of items in the array between -100 and 10: %d\n", program.Q2(arraySorted, -100, 10));
System.out.printf("Number of items in the array between -30 and 0: %d\n", program.Q2(arraySorted, -30, 0));
System.out.printf("Number of items in the array between 53 and 60: %d\n", program.Q2(arraySorted, 53, 60));
```

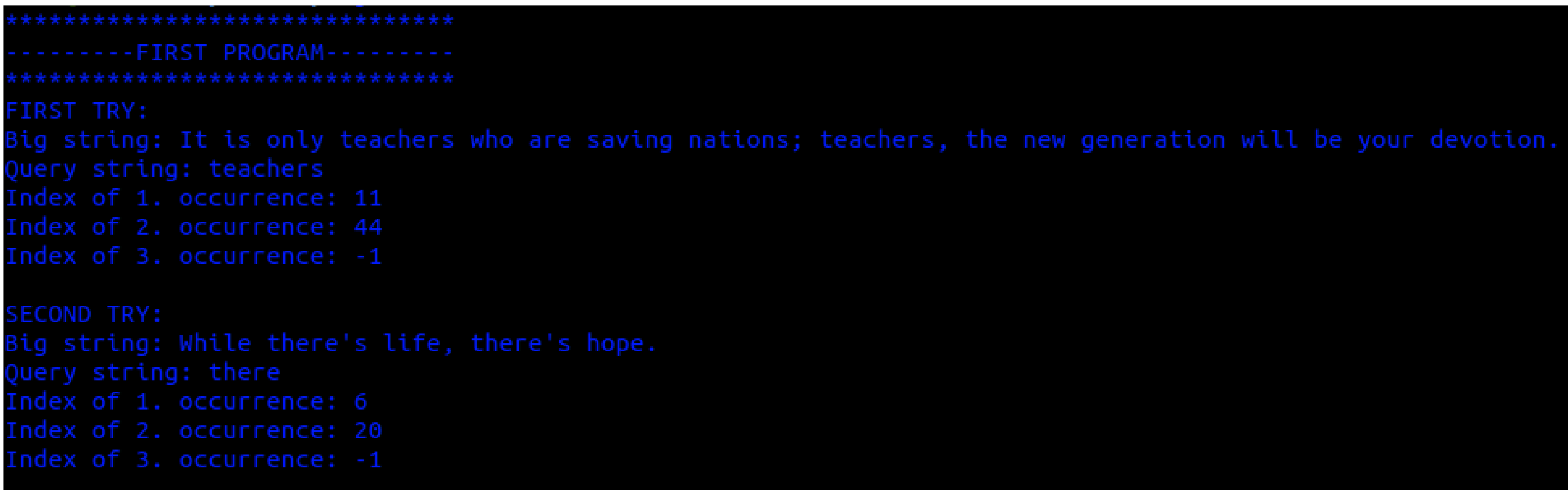


```java
System.out.println("\n\n\n********************************");
System.out.printf("---------THIRD PROGRAM---------\n");
System.out.println("********************************");

int[] arrayUnsorted = {-42, 42, 0, 99, -1859, 1901, 18, 99, 996, -5, 5, 42, 0, 90};
System.out.println("\nFIRST TRY:");
System.out.printf("Array: ");
for(int i = 0; i < arrayUnsorted.length; ++i) System.out.printf("%d ", arrayUnsorted[i]);
System.out.printf("\nSubarray/s that the sum of its/theirs items is equal to 42:\n");
program.Q3(arrayUnsorted, 42);

int[] arrayUnsorted2 = {0, 99, 124, 0, -124, -99, 123, 905, -905, 66, -65};
System.out.println("\nSECOND TRY:");
System.out.printf("Array: ");
for(int i = 0; i < arrayUnsorted2.length; ++i) System.out.printf("%d ", arrayUnsorted2[i]);
System.out.printf("\nSubarray/s that the sum of its/theirs items is equal to 0:\n");
program.Q3(arrayUnsorted2, 0);
```

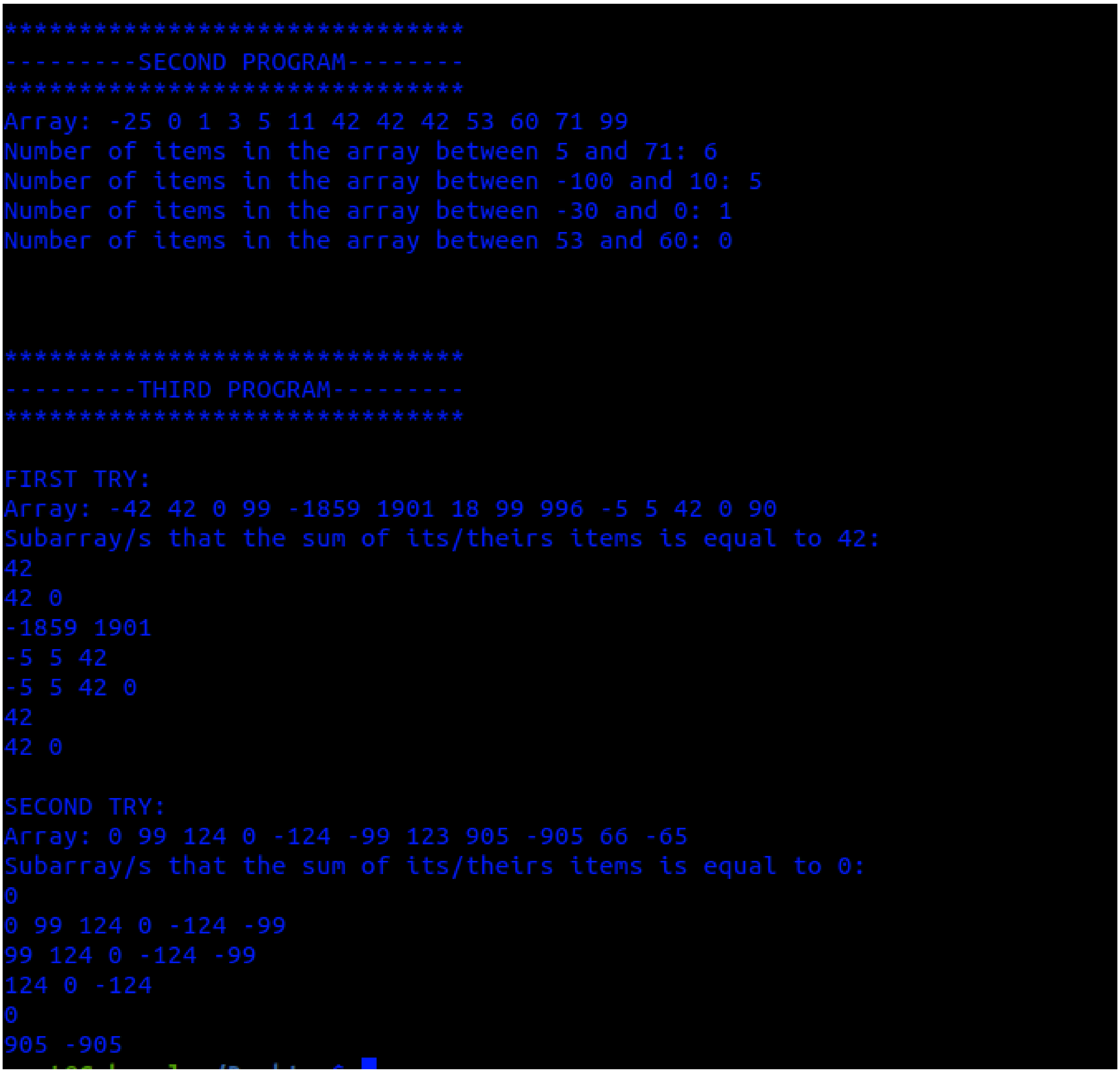


```java
System.out.println("\n\n\n********************************");
System.out.printf("----------LAST PROGRAM---------\n");
System.out.println("********************************");

System.out.println("\nFIRST TRY (array with length 7):");
//0 represents empty block
int[] arrayBlocks = {0, 0, 0, 0, 0, 0, 0};
int L = arrayBlocks.length;
program.Q5(arrayBlocks, L);

System.out.println("\nSECOND TRY (array with length 10):");
int[] arrayBlocks2 = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int L2 = arrayBlocks2.length;
program.Q5(arrayBlocks2, L2);
```

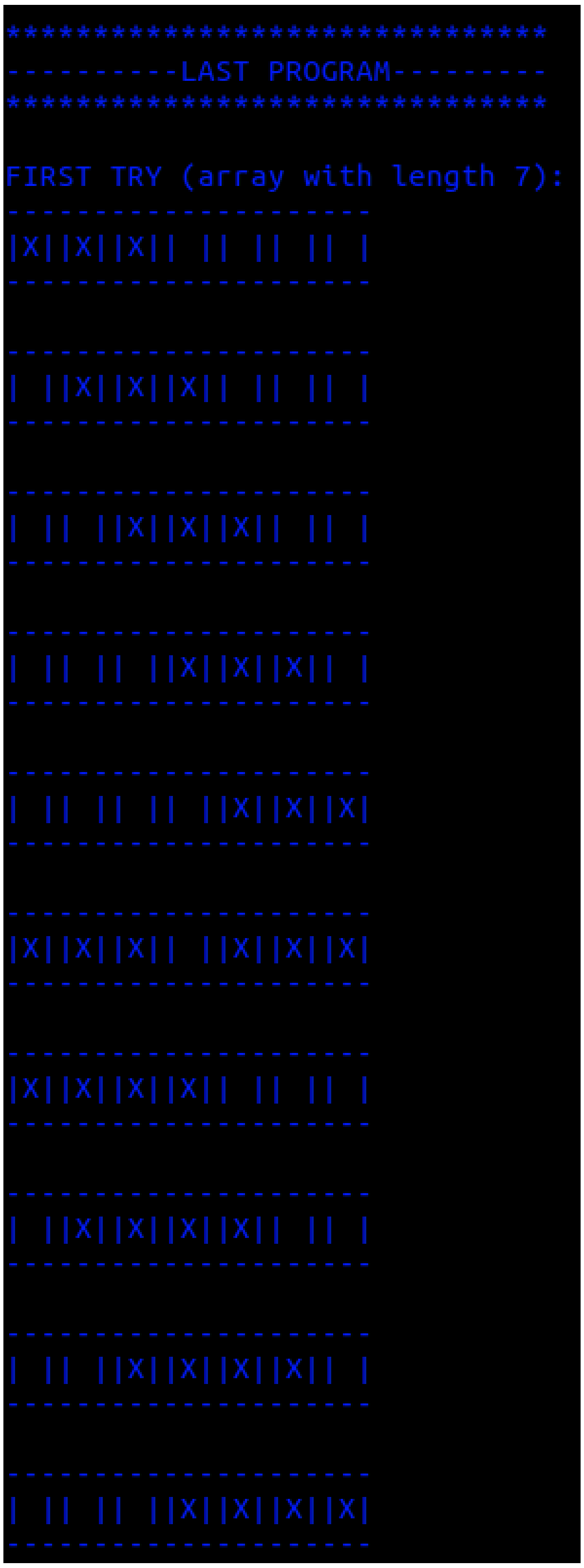
5. RUNNING AND RESULTS



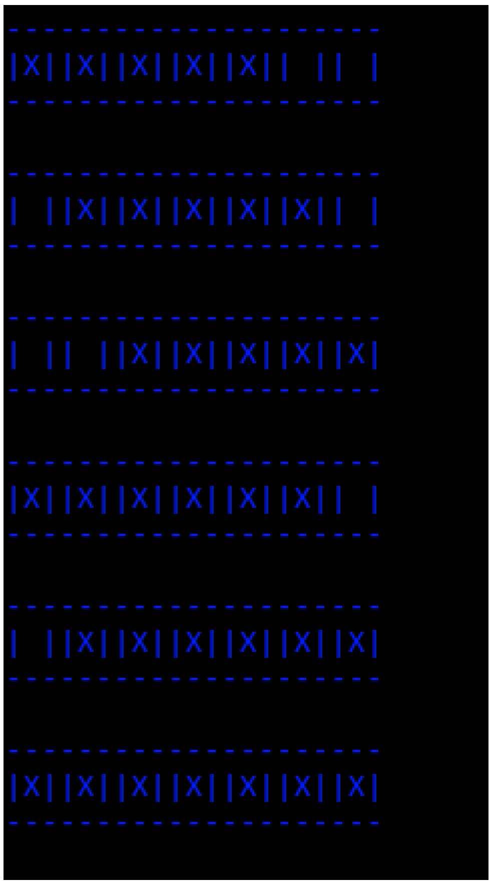
```
*******************************
---------FIRST PROGRAM---------
*******************************
FIRST TRY:
Big string: It is only teachers who are saving nations; teachers, the new generation will be your devotion.
Query string: teachers
Index of 1. occurrence: 11
Index of 2. occurrence: 44
Index of 3. occurrence: -1

SECOND TRY:
Big string: While there's life, there's hope.
Query string: there
Index of 1. occurrence: 6
Index of 2. occurrence: 20
Index of 3. occurrence: -1
```



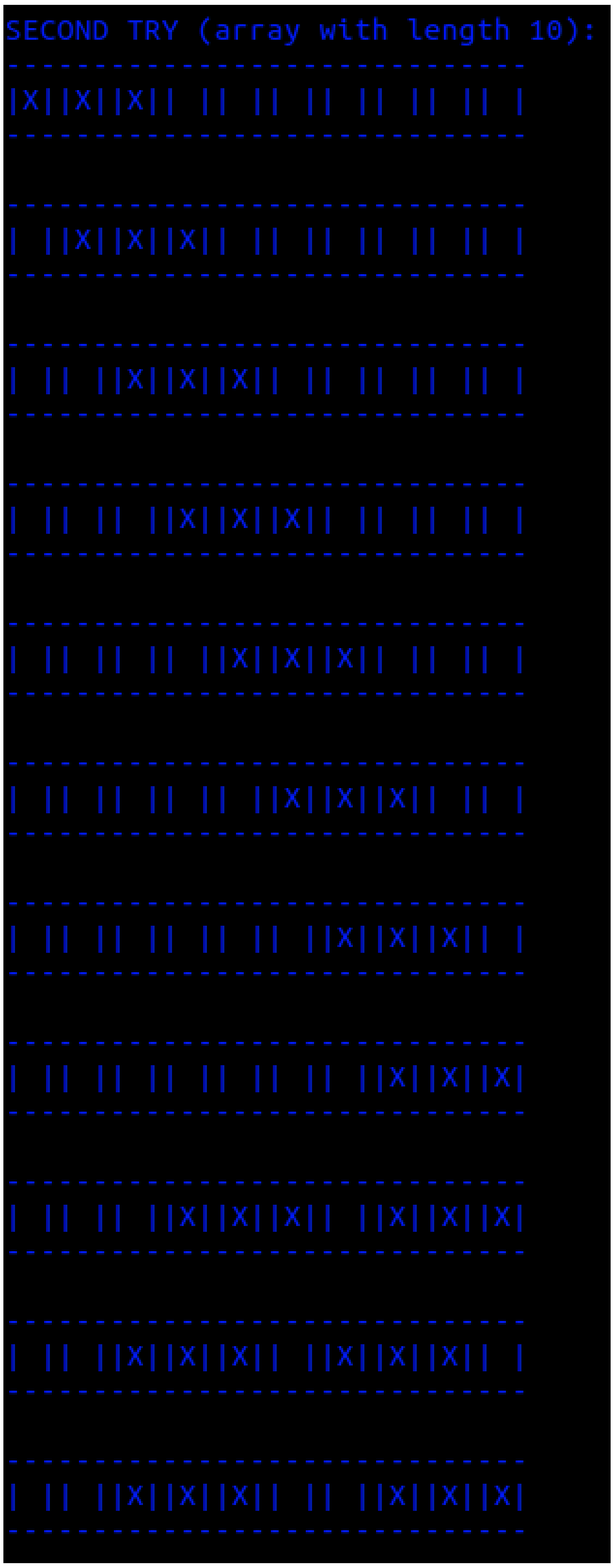
```
*******************************
---------SECOND PROGRAM--------
*******************************
Array: -25 0 1 3 5 11 42 42 42 53 60 71 99
Number of items in the array between 5 and 71: 6
Number of items in the array between -100 and 10: 5
Number of items in the array between -30 and 0: 1
Number of items in the array between 53 and 60: 0



*******************************
---------THIRD PROGRAM---------
*******************************

FIRST TRY:
Array: -42 42 0 99 -1859 1901 18 99 996 -5 5 42 0 90
Subarray/s that the sum of its/theirs items is equal to 42:
42
42 0
-1859 1901
-5 5 42
-5 5 42 0
42
42 0

SECOND TRY:
Array: 0 99 124 0 -124 -99 123 905 -905 66 -65
Subarray/s that the sum of its/theirs items is equal to 0:
0
0 99 124 0 -124 -99
99 124 0 -124 -99
124 0 -124
0
905 -905
```



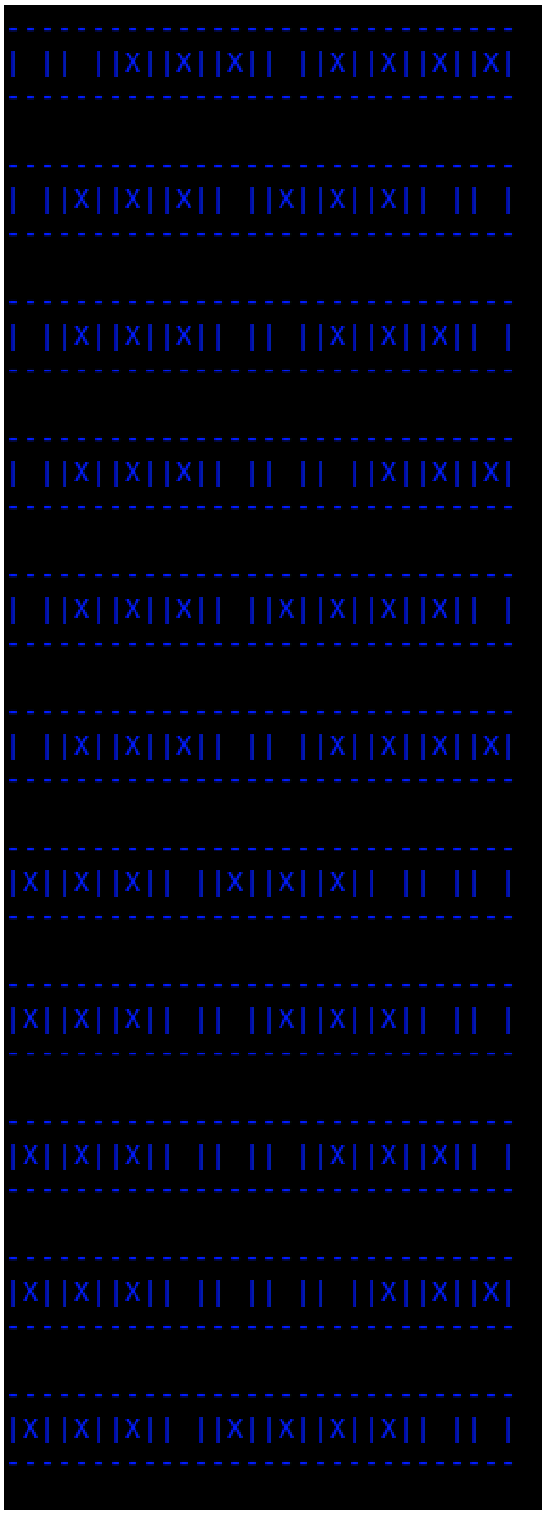
```
*******************************
----------LAST PROGRAM---------
*******************************

FIRST TRY (array with length 7):
---------------------
|X||X||X|| || || || |
---------------------

---------------------
| ||X||X||X|| || || |
---------------------

---------------------
| || ||X||X||X|| || |
---------------------

---------------------
| || || ||X||X||X|| |
---------------------

---------------------
| || || || ||X||X||X|
---------------------

---------------------
|X||X||X|| ||X||X||X|
---------------------

---------------------
|X||X||X||X|| || || |
---------------------

---------------------
| ||X||X||X||X|| || |
---------------------

---------------------
| || ||X||X||X||X|| |
---------------------

---------------------
| || || ||X||X||X||X|
---------------------
```



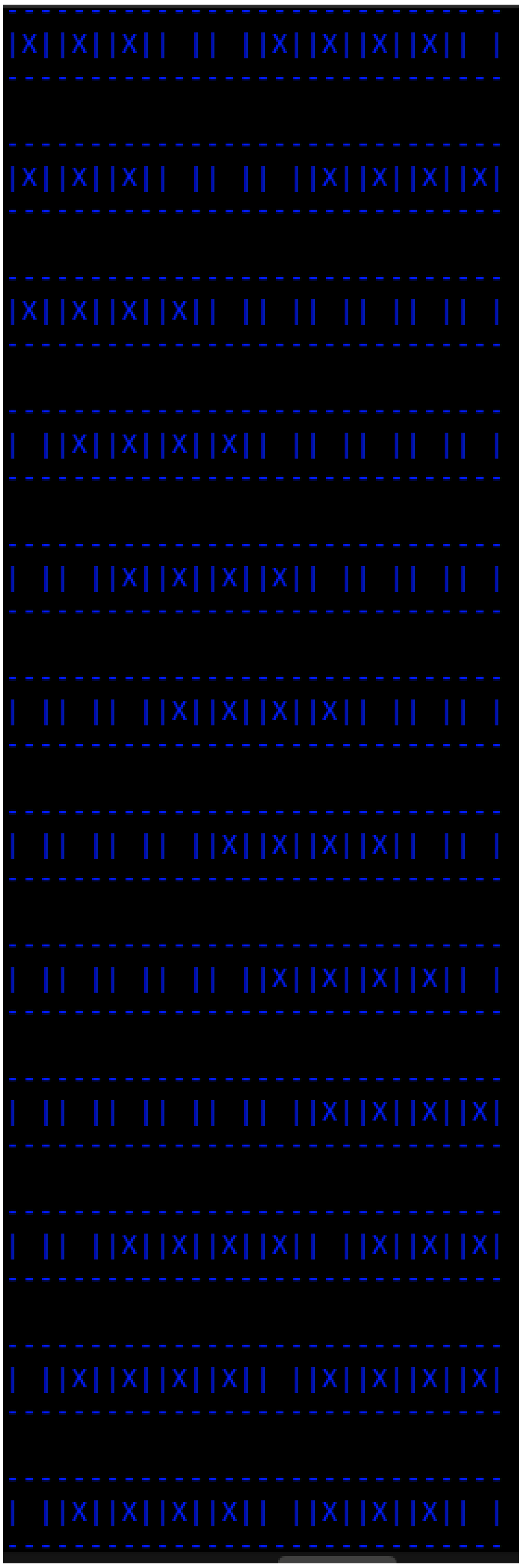
```
---------------------
|X||X||X||X||X|| || |
---------------------

---------------------
| ||X||X||X||X||X|| |
---------------------

---------------------
| || ||X||X||X||X||X|
---------------------

---------------------
|X||X||X||X||X||X|| |
---------------------

---------------------
| ||X||X||X||X||X||X|
---------------------

---------------------
|X||X||X||X||X||X||X|
---------------------
```



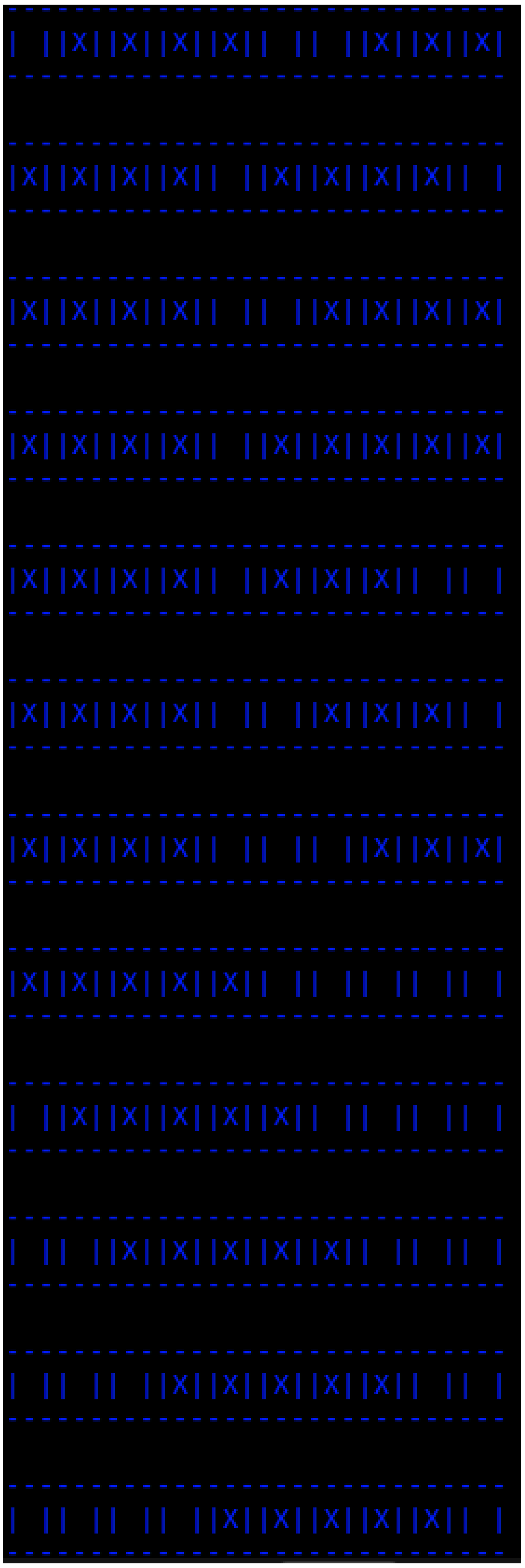
```
SECOND TRY (array with length 10):
------------------------------
|X||X||X|| || || || || || || |
------------------------------

------------------------------
| ||X||X||X|| || || || || || |
------------------------------

------------------------------
| || ||X||X||X|| || || || || |
------------------------------

------------------------------
| || || ||X||X||X|| || || || |
------------------------------

------------------------------
| || || || ||X||X||X|| || || |
------------------------------

------------------------------
| || || || || ||X||X||X|| || |
------------------------------

------------------------------
| || || || || || ||X||X||X|| |
------------------------------

------------------------------
| || || || || || || ||X||X||X|
------------------------------

------------------------------
| || || ||X||X||X|| ||X||X||X|
------------------------------

------------------------------
| || ||X||X||X|| ||X||X||X|| |
------------------------------

------------------------------
| || ||X||X||X|| || ||X||X||X|
------------------------------
```



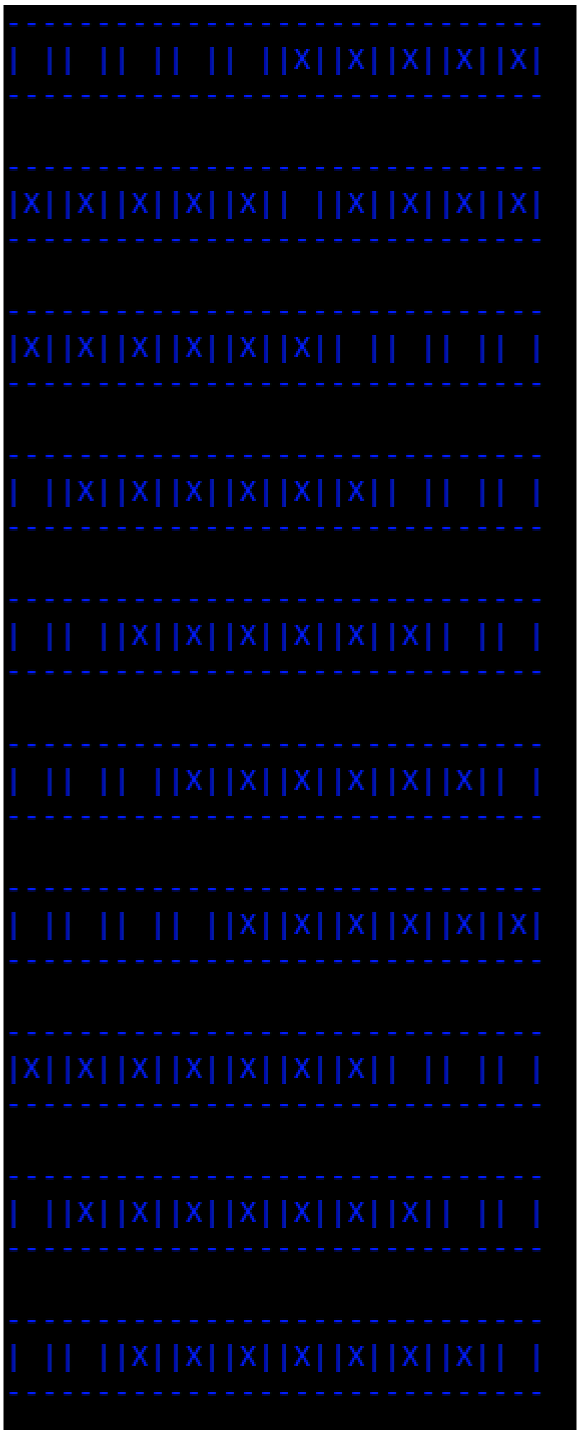
```
------------------------------
| || ||X||X||X|| ||X||X||X||X|
------------------------------

------------------------------
| ||X||X||X|| ||X||X||X|| || |
------------------------------

------------------------------
| ||X||X||X|| || ||X||X||X|| |
------------------------------

------------------------------
| ||X||X||X|| || || ||X||X||X|
------------------------------

------------------------------
| ||X||X||X|| ||X||X||X||X|| |
------------------------------

------------------------------
| ||X||X||X|| || ||X||X||X||X|
------------------------------

------------------------------
|X||X||X|| ||X||X||X|| || || |
------------------------------

------------------------------
|X||X||X|| || ||X||X||X|| || |
------------------------------

------------------------------
|X||X||X|| || || ||X||X||X|| |
------------------------------

------------------------------
|X||X||X|| || || || ||X||X||X|
------------------------------

------------------------------
|X||X||X|| ||X||X||X||X|| || |
------------------------------
```



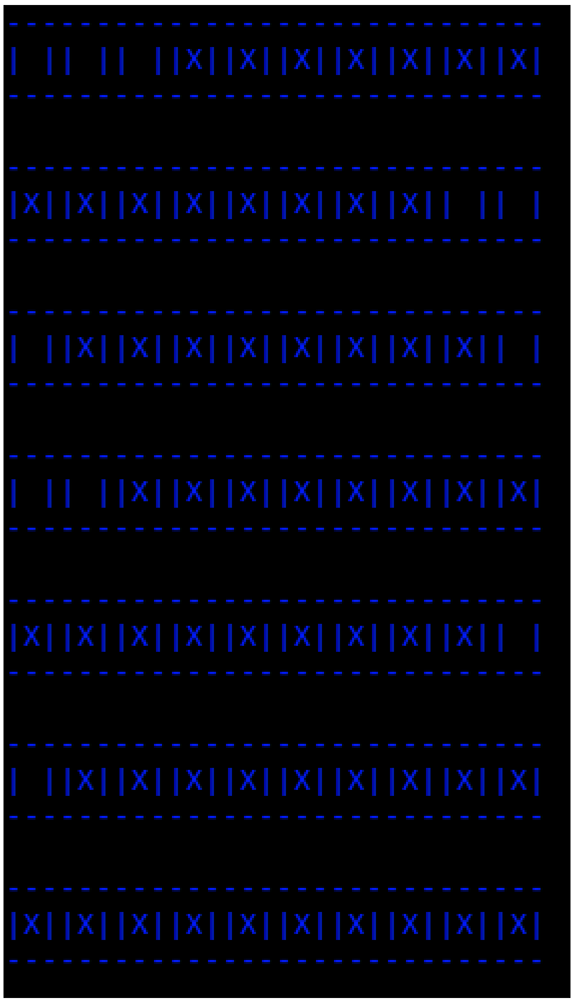
```
------------------------------
|X||X||X|| || ||X||X||X||X|| |
------------------------------

------------------------------
|X||X||X|| || || ||X||X||X||X|
------------------------------

------------------------------
|X||X||X||X|| || || || || || |
------------------------------

------------------------------
| ||X||X||X||X|| || || || || |
------------------------------

------------------------------
| || ||X||X||X||X|| || || || |
------------------------------

------------------------------
| || || ||X||X||X||X|| || || |
------------------------------

------------------------------
| || || || ||X||X||X||X|| || |
------------------------------

------------------------------
| || || || || ||X||X||X||X|| |
------------------------------

------------------------------
| || || || || || ||X||X||X||X|
------------------------------

------------------------------
| || ||X||X||X||X|| ||X||X||X|
------------------------------

------------------------------
| ||X||X||X||X|| ||X||X||X||X|
------------------------------

------------------------------
| ||X||X||X||X|| ||X||X||X|| |
------------------------------
```



```
------------------------------
| ||X||X||X||X|| || ||X||X||X|
------------------------------

------------------------------
|X||X||X||X|| ||X||X||X||X|| |
------------------------------

------------------------------
|X||X||X||X|| || ||X||X||X||X|
------------------------------

------------------------------
|X||X||X||X|| ||X||X||X||X||X|
------------------------------

------------------------------
|X||X||X||X|| ||X||X||X|| || |
------------------------------

------------------------------
|X||X||X||X|| || ||X||X||X|| |
------------------------------

------------------------------
|X||X||X||X|| || || ||X||X||X|
------------------------------

------------------------------
|X||X||X||X||X|| || || || || |
------------------------------

------------------------------
| ||X||X||X||X||X|| || || || |
------------------------------

------------------------------
| || ||X||X||X||X||X|| || || |
------------------------------

------------------------------
| || || ||X||X||X||X||X|| || |
------------------------------

------------------------------
| || || || ||X||X||X||X||X|| |
------------------------------
```



```
------------------------------
| || || || || ||X||X||X||X||X|
------------------------------

------------------------------
|X||X||X||X||X|| ||X||X||X||X|
------------------------------

------------------------------
|X||X||X||X||X||X|| || || || |
------------------------------

------------------------------
| ||X||X||X||X||X||X|| || || |
------------------------------

------------------------------
| || ||X||X||X||X||X||X|| || |
------------------------------

------------------------------
| || || ||X||X||X||X||X||X|| |
------------------------------

------------------------------
| || || || ||X||X||X||X||X||X|
------------------------------

------------------------------
|X||X||X||X||X||X||X|| || || |
------------------------------

------------------------------
| ||X||X||X||X||X||X||X|| || |
------------------------------

------------------------------
| || ||X||X||X||X||X||X||X|| |
------------------------------
```



```
------------------------------
| || || ||X||X||X||X||X||X||X|
------------------------------

------------------------------
|X||X||X||X||X||X||X||X|| || |
------------------------------

------------------------------
| ||X||X||X||X||X||X||X||X|| |
------------------------------

------------------------------
| || ||X||X||X||X||X||X||X||X|
------------------------------

------------------------------
|X||X||X||X||X||X||X||X||X|| |
------------------------------

------------------------------
| ||X||X||X||X||X||X||X||X||X|
------------------------------

------------------------------
|X||X||X||X||X||X||X||X||X||X|
------------------------------
```