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

BURAK KOCAUSTA
1901042605

1. SYSTEM REQUIREMENTS

There are 4 recursive implementations. Each one has its own class, and they must be called from their class.

```
/**
 *
 * Searches a string inside greater String, returns its index recursively.
 * @return -1 if String cannot be found, otherwise returns its index.
 * @param mainString String that will be searched inside.
 * @param queryString String that will be searched.
 * @param occurrence used to differ wanted String with its occurring time.
 */
public static int returnQueryStringIndex ( String mainString, String queryString, int occurrence )
```

returnQueryStringIndex method requires two string and occurrence number.

```
QueryStringRecursive.returnQueryStringIndex("appleelmaappleelmaee", "ee", 2)
```

It must be called from QueryStringRecursive class.

```
/**
 *
 * Method finds total number of integers, which uses binary search, between two given borders. Array must be ordered.
 * @param arr integer array which must be ordered.
 * @param below indicates lower limit.
 * @param above indicates upper limit.
 * @return int the total number of integers between given numbers.
 */
public static int findNumBetweensRecursively ( int[] arr, int below, int above ) {
```

findNumBetweensRecursively requires an integer array which is sorted. Also it requires boundary integer values.

```
FindNumBetweensRecursive.findNumBetweensRecursively( arr1, 2, 6 );
```

It must be called from FindNumBetweensRecursive class.

```
/**
 *
 * Method prints contiguous sub arrays of given sum inside given array.
 * @param arr integer array to search.
 * @param sum integer sum value.
 */
public static void findContSubArrRecursively ( int[] arr, int sum ) {
```

findContSubArrayRecursively requires an integer array and integer sum value.

```
FindContiguousSubArrRecursive.findContSubArrRecursively( arr1, 6 );
```

It must be called from FindContiguousSubArrRecursive class.

```java
/**
 *
 * Method prints different configurations of given array.
 * @param arr an integer array.
 */
public static void calcPossibleConfigArrRecursively ( int[] arr ) {
```

calcPossibleConfigArrRecursively requires an integer array.

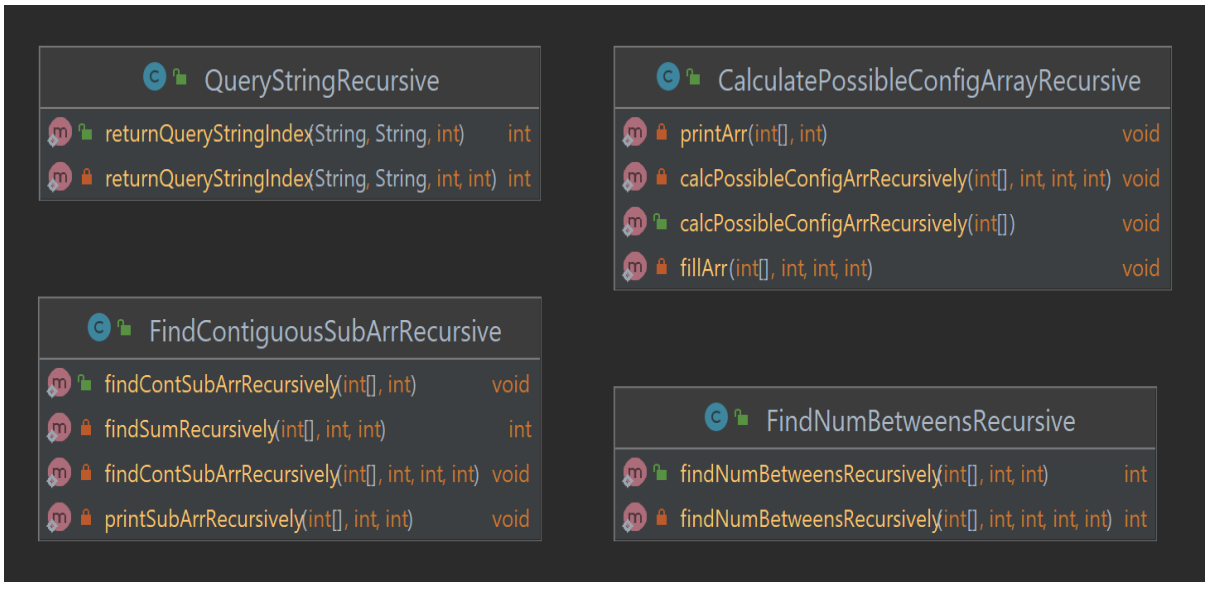
```java
CalculatePossibleConfigArrayRecursive.calcPossibleConfigArrRecursively(arr);
```

It must be called from CalculatePossibleConfigArrayRecursive class.

```java
import RecursionGTU.*;
```

All these implementations are inside RecursionGTU class, therefore this import statement is required.

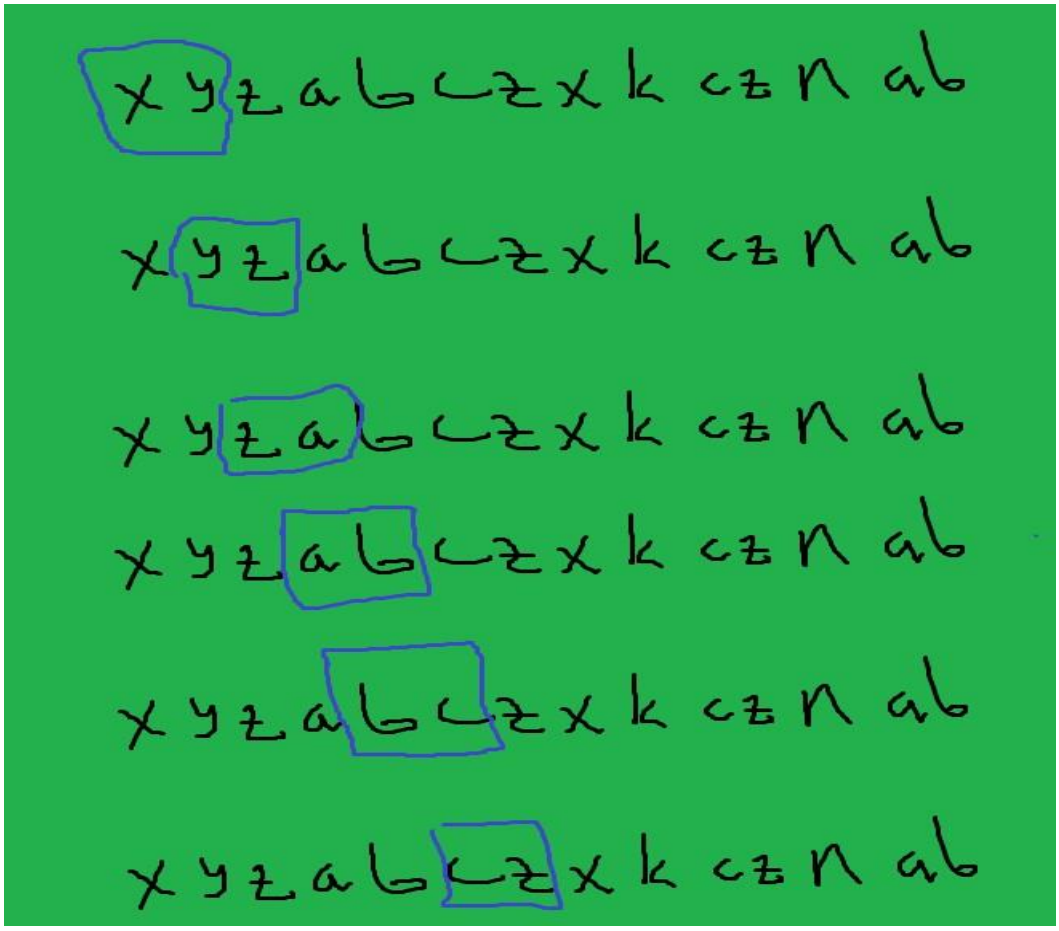
2. CLASS DIAGRAM



3. PROBLEM SOLUTION APPROACH

Q1

For the finding query string problem, I thought an algorithm which takes two String from user and an occurrence value. 3 parameter is enough for the public call. For the recursive part, we know that String class has method length(), so there is no need to hold the size of the strings. String class also has a method called substring(). I decided to check every inner string which are consecutive, till string ends or occurrence is reached. After finding the value return the current index. For this implementation it requires one more parameter which is pos(position). This parameter enables us to search the array. So I add it to wrapper method's parameters.



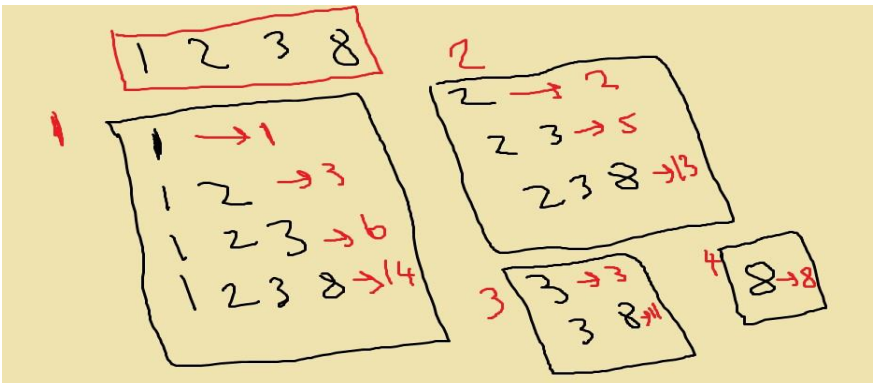
Algorithm searches the array like this for "two-character" length query String till occurrence is reached or outer string is finished.

Q2

For finding numbers between inside given array method. It is suggested inside homework document that we can use binary search approach. I thought the standard binary search implementation and modified it properly according to the given problem. Algorithm needs 3 parameters from user, which are integer array, which must be sorted, and boundary values. Boundary values are included during counting. For the wrapper method, I added 2 more parameter which are first and last index values. Algorithm's base case is same with binary search operation. If first index is greater than last index return. For the recursive cases, if given condition checks, search both sides of the array, left and right. If middle value is greater than our upper bound go left, otherwise go right. That last two condition is like if we found last value on left side of array, it goes one more and returns. Algorithm does not check the values which are beyond our boundaries, it makes the counting operation efficient. But in the worst-case algorithm definitely searches the whole array.

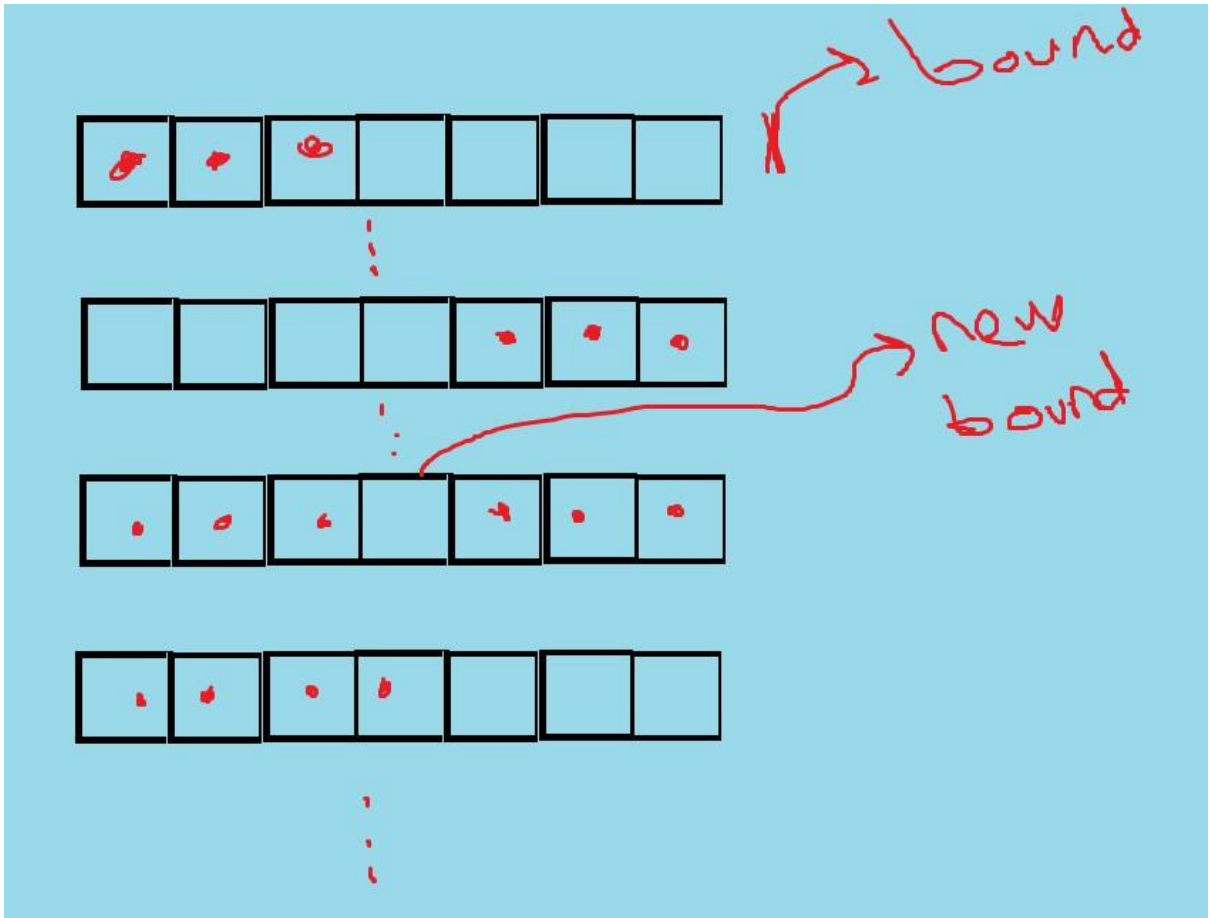
Q3

In finding contiguous sub array problem, I thought that whole integer array can be searched from new starts indexes till the end. Point is, incrementing the start index, till it is greater than array's length. So, the base case is checking starting position and array's length. In each inner recursive call start index is not incremented, end index will be incremented. Therefore, in the first call of the wrapper method first and last indexes are 0. In the public call, 2 parameters are enough which are integer array and sum value, but I needed 2 more parameter for the recursive algorithm which are starting position and ending position. In the wrapper method, I added another helper methods for calculating the sum, and printing the array. Check the sum for each step.



Q5

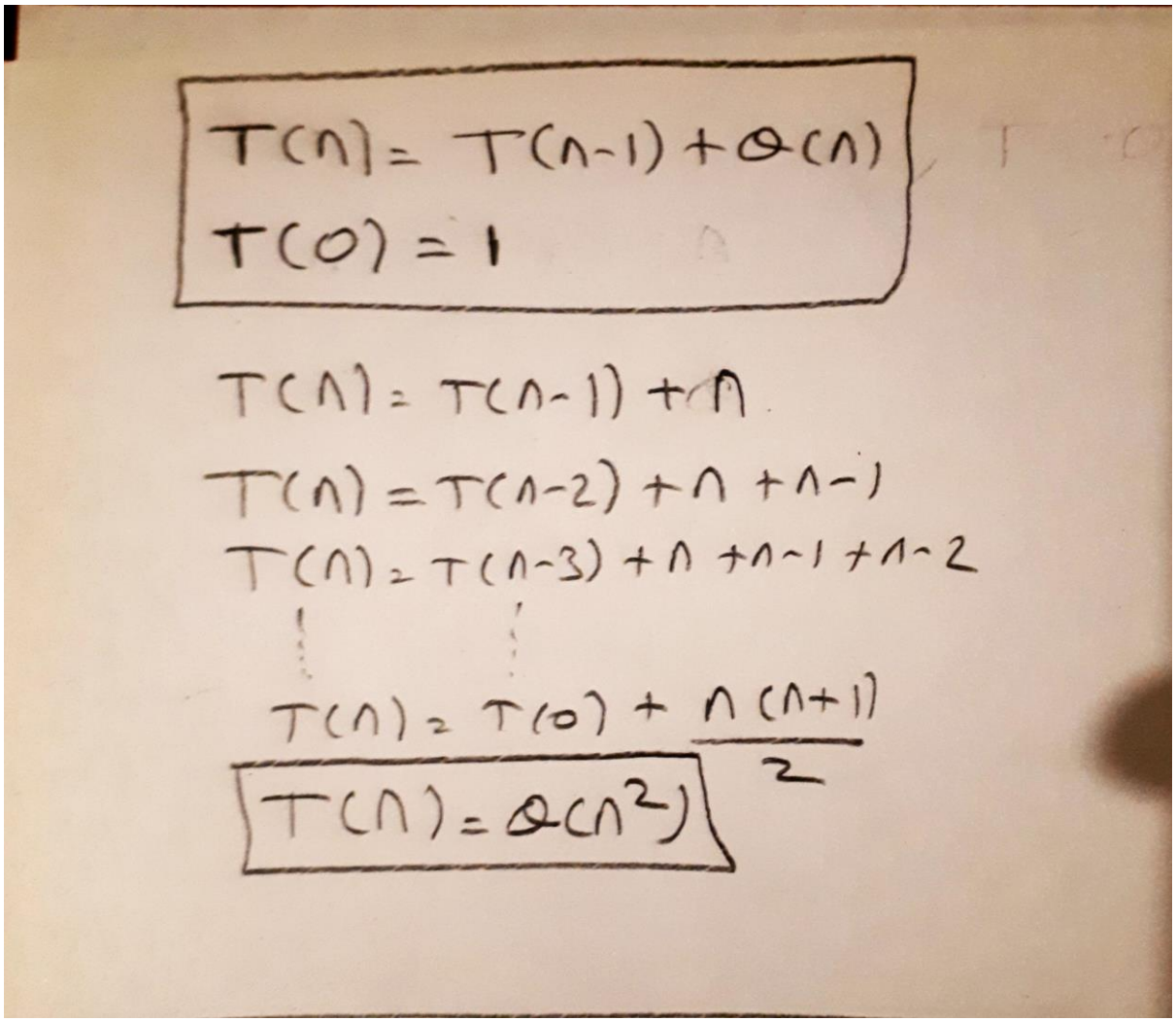
For the possible array configuration problem, first I thought that what to do with these configurations. Inside homework document, It is ambiguous, and I decided to print the configurations as indicated. For the base case, if block length is equal to array's length, return. After deciding the base case, firstly I printed the blocks, but only one in each step with incrementing. Inside the wrapper method, I added only position, bound and block length parameter. Block length starts with 3. Problem becomes harder when there could be more blocks than one. I decided to make bound smaller and smaller till it is full. To be more clear, It can be thought that I decremented the array size and set the position to 0. Therefore, It fills the array like there is one block, but only in permitted parts. I needed 2 more helper recursive method which does printing and filling the array with wanted value.



4. Complexity Calculations and Induction

Q1

```
private static int returnQueryStringIndex ( String mainString, String queryString, int occurrence, int pos ) {
    if ( occurrence < 0 || mainString.equals("") || pos + queryString.length() > mainString.length()) → Θ(1)
        return -1;

    else if ( queryString.equals( mainString.substring( pos, pos + queryString.length())) ) { → O(n)
        occurrence--;

        if ( occurrence == 0 ) } → Θ(1)
            return pos;
        pos++;
        return returnQueryStringIndex( mainString, queryString, occurrence, pos ); → T(n-1)
    }

    else {
        pos++;
        return returnQueryStringIndex( mainString, queryString, occurrence, pos ); → T(n-1)
    }
```




```
public static int returnQueryStringIndex ( String mainString, String queryString, int occurrence ) {

    if ( queryString == null || mainString == null || queryString.length() > mainString.length() || occurrence <= 0 ) } → Θ(1)
        return -1;

    int retval = returnQueryStringIndex ( mainString, queryString, occurrence, 0 ); → Θ(n²)
    return retval;
}
```

So **T(n) = Θ(n²)**

Induction Proof of Question 1

In the base case, occurrence's positivity, and position + query strings length is checked. If one of these happens method returns -1. Other base case is when last occurrence is found and occurrence is equals to 0. That means Index is found so return the position. First mentioned base case always return -1, other one returns index value.

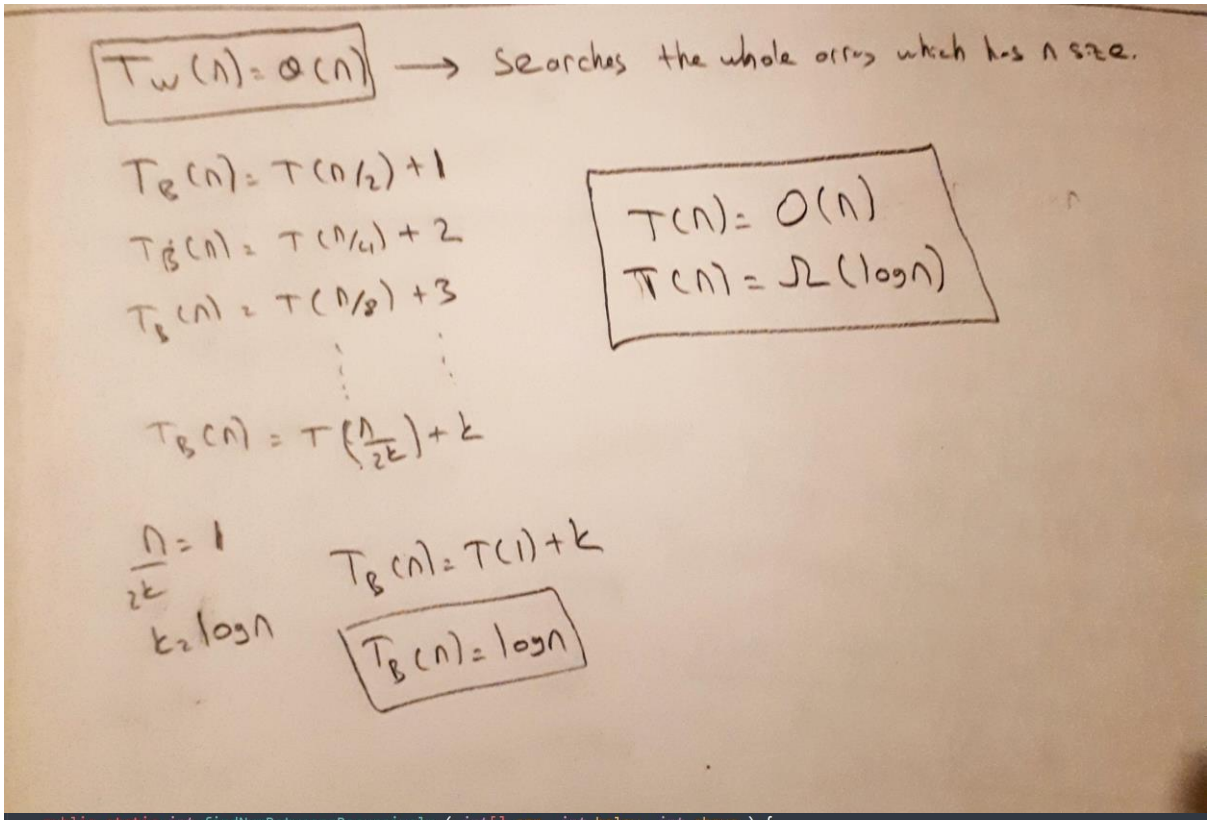
There are two recursive calls in first one occurrence is decremented position is incremented. That means there is progress towards the base case. One of them will be reached. Occurrence will be less or equal(for last base case) to 0 or position will greater than String length. For another recursive call, only position is incremented, that means in each recursive step there is a progress towards base case. At some time it will be greater than size and algorithm ends.

```
main string = nkaabfeageaabex
query string = aab
occurrence = 2
returns = 10
```

Smaller problem is solved correctly, so original problem is solved correctly. There are more test cases in test part.

Q2

```
private static int findNumBetweensRecursively ( int[] arr, int below, int above, int first, int last ) {
    if ( first > last ) return 0;
    int retval = 0;
    int middle = ( first + last ) / 2;
    if ( arr[middle] <= above && arr[middle] >= below ) {
        retval = 1 + findNumBetweensRecursively( arr, below, above, first, middle - 1 )
        + findNumBetweensRecursively( arr, below, above, middle + 1, last );
    }
    else {
        if ( arr[middle] > above )  // go left
            retval = findNumBetweensRecursively( arr, below, above, first, middle - 1 );
        else                        // go right
            retval = findNumBetweensRecursively( arr, below, above, middle + 1, last );
    }

    return retval;
}
```




```
public static int findNumBetweensRecursively ( int[] arr, int below, int above ) {

    if ( arr == null )
        return -1;

    if ( above > below )
        return findNumBetweensRecursively( arr, above, below, 0, arr.length - 1 );
    return findNumBetweensRecursively( arr, below, above, 0, arr.length - 1 );
}
```

So, **T(n) = O(n)** or **T(n) = Ω(logn)**

Q3

```
private static int findSumRecursively ( int[] arr, int startPos, int endPos ) {
    if ( endPos < startPos )
        return 0;
    return arr[endPos] + findSumRecursively( arr, startPos, endPos - 1);
}
```

```
private static void printSubArrRecursively ( int[] arr, int startPos, int endPos ) {
    if ( startPos > endPos ) {
        System.out.print("\n");
        return;
    }
    System.out.print( arr[startPos] + " " );
    printSubArrRecursively( arr, startPos + 1, endPos );
}
```

```java
private static void findContSubArrRecursively ( int[] arr, int sum, int startPos, int endPos ) {

    if ( startPos >= arr.length ){
        System.out.println("----------------");
        return;
    }

    if ( endPos >= arr.length ) {
        ++startPos;
        findContSubArrRecursively ( arr, sum, startPos, startPos );
        return;
    }

    else if ( sum == findSumRecursively( arr, startPos, endPos ) ) {
        System.out.print("[");
        printSubArrRecursively( arr, startPos, endPos );
    }

    findContSubArrRecursively( arr, sum, startPos, endPos + 1 );
}
```

$\Theta(1)$

$\rightarrow T(n-1) + \Theta(n^2)$

$\rightarrow \Theta(n)$

$\rightarrow T(n-1) + \Theta(n)$

$$T(n) = T(n-1) + \Theta(n^2)$$
$$T(0) = 1$$

$$T(n) = T(n-1) + n^2$$
$$T(n) = T(n-2) + n^2 + (n-1)^2$$
$$T(n) = T(n-2) + n^2 + (n-1)^2 + (n-2)^2$$
$$T(n) = T(0) + \frac{n(n+1)(2n+1)}{6}$$
$$T(n) = \Theta(n^3)$$

```java
public static void findContSubArrRecursively ( int[] arr, int sum ) {

    if ( arr == null )
        System.out.println("Array is null!");

    else if ( arr.length == 0 )
        System.out.println( "Array is empty!" );

    else
        findContSubArrRecursively( arr, sum, 0, 0 );
}
```

$\Theta(1)$

$\Theta(1)$

$\rightarrow \Theta(n^3)$

So, **$T(n) = \Theta(n^3)$**

Induction Proof for Question 3

Algorithm's base case happens, when starting position is greater than array's length. When recursive call happens, ending position or starting position incremented. If started position is not incremented, ending position will incremented, therefore base case is enough for checking starting position is greater than array's length or not.

There are two recursive calls in this algorithm. First one occurs when ending position is greater or equal than array's length. In this case, it means that one of the sub array checking for one starting value is finished. Therefore, increment the starting position and set the ending position same value with starting position. Incrementing starting position means there is a progress towards base case. Other recursive call happens when starting and ending positions are not greater or equal to array's length. It only increments ending position. This also means there is a progress towards base case, because as stated above when ending position is equal or greater than array's length, starting position is incremented. Incrementing starting position means there is a progress towards base case. Therefore, for each step of the algorithm problem gets smaller and smaller eventually base case happens.

```
arr1 = 3 2 1 6 5 1

sum = 6
result
[3, 2, 1]
[6]
[5, 1]
---------------
```

Smaller problem is solved correctly, so original problem is solved correctly. There are more test cases in test part.

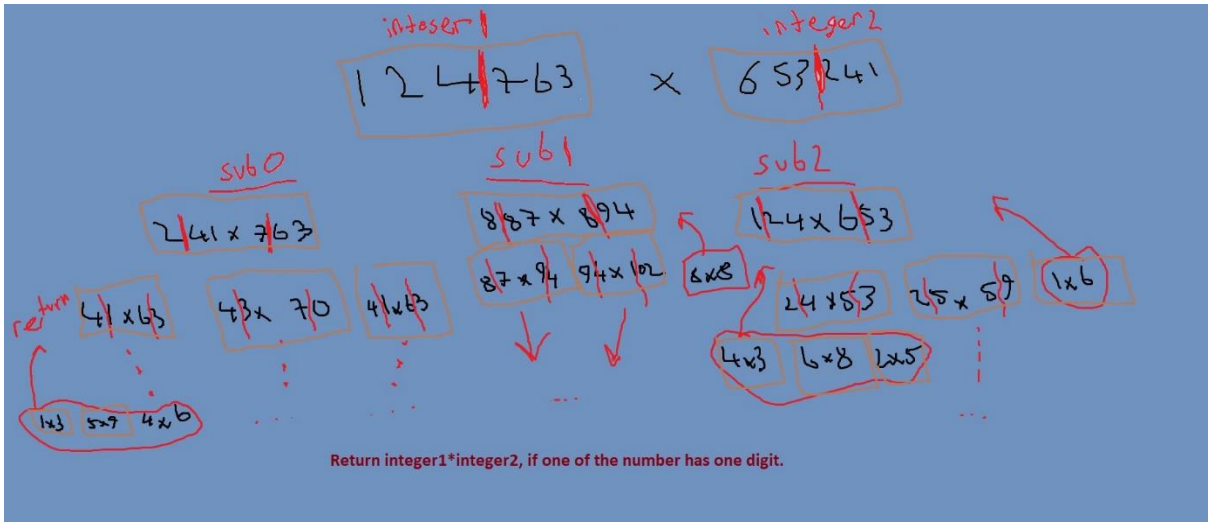
Q4

In this problem, it basically does multiplication of two integers. In the homework forum, it is said that it multiplies in binary format, but algorithm does not work correct for this problem. I'll explain why it is not correct for binary multiplication.

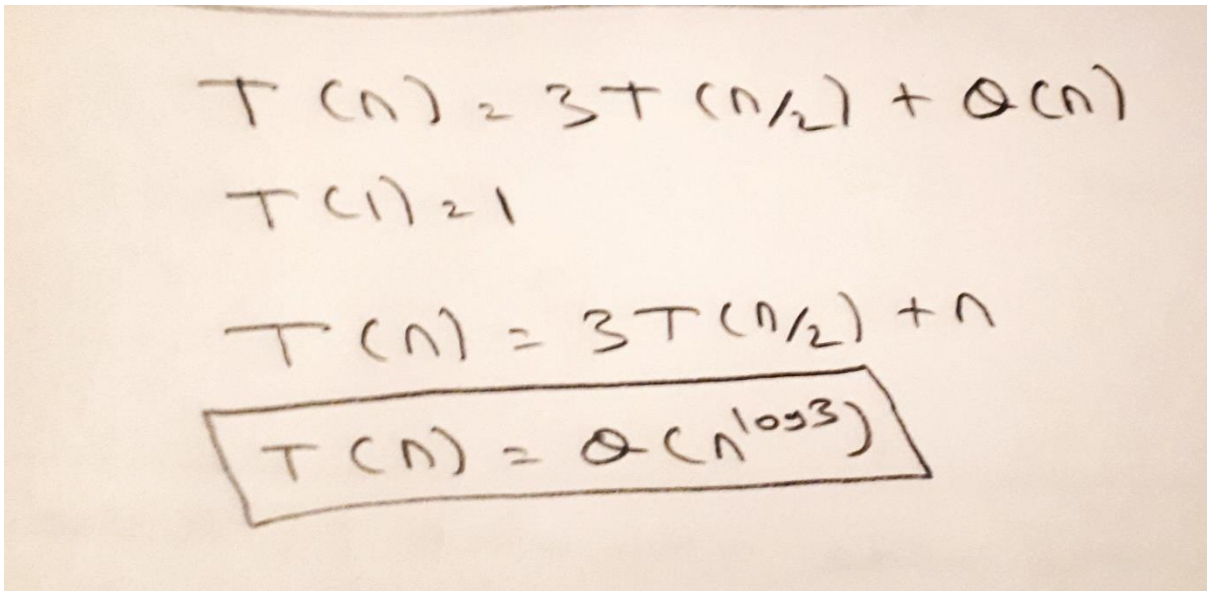
Firstly, problem size is greatest given integer's number of digits. If one of them is less than 10, it returns the multiplication of both. Algorithm basically divides the problem to subproblems. After getting the n value, it enables us to calculate the same number with (first n/2 digit)*10^{half} + rest. split_integer function divides the number like this. After dividing two integers like that we have 4 different integers. For sub0 the rest part of two numbers multiplied with this algorithm. In sub1, addition of each number's two part is multiplied. sub2 is equals to first half of two numbers multiplication. sub2 gives us the first half of the result. sub0 and sub1 gives us the second half of the result. Because of that when result is calculated, sub2 is multiplied with 10^2*half and other part is multiplied with 10^half because it is the other half. sub0 is just the rest part. (sub1-sub2-sub0) means first half of the integer1 multiplied with second half of integer2 plus first half of integer2 multiplied with second half of integer1.

Reason for why this is decimal multiplication, not binary multiplication. Because for the addition parts. It makes decimal addition not binary addition. Probably languages + operator makes default decimal addition. It can be converted to binary addition with making split_integer functions base number 2, and inside the return part change 10s with 2. That is not enough because of the reason stated above. Making a class like BinaryNumber and using its addition method will solve the problem, and algorithm will be converted to binary multiplication.

A demonstration of how the algorithm works is indicated below. This shows how the problem is divided into sub problems and solved. (demonstration is on the other page)




```
def foo (integer1, integer2)

    if (integer1 < 10) or (integer2 < 10)
        return integer1 * integer2

    n = max(number_of_digits(integer1), number_of_digits(integer2))
    half = int(n/2)

    int1, int2 = split_integer(integer1, half)
    int3, int4 = split_integer(integer2, half)

    sub0 = foo (int2, int4)
    sub1 = foo ((int2 + int1), (int4 + int3))
    sub2 = foo (int1, int3)

    return (sub2*10^(2*half))+((sub1-sub2-sub0)*10^(half))+(sub0)
```



T(n) = Θ(n^{log3}) (n is number of digits for greater integer)

5. TEST CASES

```
public static void testQueryStringRecursive( ) {

    System.out.println("\n\n___Testing QueryStringRecursive___");

    System.out.print( "\nmain string = appleelmaappleelmaee\n" + "query string = " + "ee\n" +
                      "occurrence = 2\n"+ "returns = " );
    System.out.println( QueryStringRecursive.returnQueryStringIndex("appleelmaappleelmaee", "ee", 2) );

    System.out.print( "\nmain string = aaeexxyyaaoo\n" + "query string = aa\n" +
                      "occurrence = 1\n"+ "returns = " );
    System.out.println( QueryStringRecursive.returnQueryStringIndex("aaeexxyyaaoo", "aa", 1) );

    System.out.print( "\nmain string = abc\n" + "query string = abc\n" +
                      "occurrence = 1\n"+ "returns = " );
    System.out.println( QueryStringRecursive.returnQueryStringIndex("abc", "abc", 1) );

    System.out.print( "\nmain string = nkaabfeageaabex\n" + "query string = " + "aab\n" +
                      "occurrence = 2\n"+ "returns = " );
    System.out.println( QueryStringRecursive.returnQueryStringIndex("nkaabfeageaabex", "aab", 2) );

    System.out.print( "\nmain string = abcxyzneabxynekmxabnenkenemm\n" + "query string = " + "ne\n" +
                      "occurrence = 3\n"+ "returns = " );
    System.out.println( QueryStringRecursive.returnQueryStringIndex("abcxyzneabxynekmxabnenkenemm", "ne", 3) );

    System.out.print( "\nmain string = efaeeaffefa\n" + "query string = " + "xk\n" +
                      "occurrence = 2\n"+ "returns = " );
    System.out.println( QueryStringRecursive.returnQueryStringIndex("efaeeaffefa", "xk" , 2) );

    System.out.print( "\nmain string = efaeeaffexk\n" + "query string = " + "xk\n" +
                      "occurrence = 1\n"+ "returns = " );
    System.out.println( QueryStringRecursive.returnQueryStringIndex("efaeeaffexk", "xk", 1) );
```

```
    System.out.print( "\nmain string = xyz\n" + "query string = " + "apple\n" +
                      "occurrence = 1\n"+ "returns = " );
    System.out.println( QueryStringRecursive.returnQueryStringIndex("xyz", "apple", 1) );

    System.out.print( "\nmain string = xxxxxx\n" + "query string = " + "xx\n" +
                      "occurrence = 3\n"+ "returns = " );
    System.out.println( QueryStringRecursive.returnQueryStringIndex("xxxx", "xx", 3) );
```

```
public static void testFindNumBetweensRecursive ( ) {

    System.out.println("\n\n___Testing FindNumBetweensRecursive___");

    int[] arr1 = { 1,2,3,4,5,6,7,8,9,10 };

    System.out.print("arr1 = ");
    for ( int i = 0; i < arr1.length; ++i ) System.out.print( arr1[i] + " " );

    int retVal = FindNumBetweensRecursive.findNumBetweensRecursively( arr1, 2, 6 );
    System.out.print( "\n\nbetween = 2 and 6" + "\nreturned value = " + retVal + "\n");

    retVal = FindNumBetweensRecursive.findNumBetweensRecursively( arr1, 1, 1 );
    System.out.print( "\nbetween = 1 and 1" + "\nreturned value = " + retVal + "\n");

    retVal = FindNumBetweensRecursive.findNumBetweensRecursively( arr1, -2, 3 );
    System.out.print( "\nbetween = -2 and 3" + "\nreturned value = " + retVal + "\n");

    retVal = FindNumBetweensRecursive.findNumBetweensRecursively( arr1, 3, 15 );
    System.out.print( "\nbetween = 3 and 15" + "\nreturned value = " + retVal + "\n");

    retVal = FindNumBetweensRecursive.findNumBetweensRecursively( arr1, 13, 15 );
    System.out.print( "\nbetween = 13 and 15" + "\nreturned value = " + retVal + "\n");

    retVal = FindNumBetweensRecursive.findNumBetweensRecursively( arr1, 7, 10 );
    System.out.print( "\nbetween = 7 and 10" + "\nreturned value = " + retVal + "\n");

    retVal = FindNumBetweensRecursive.findNumBetweensRecursively( arr1, 9, 3 );
    System.out.print( "\nbetween = 9 and 3(algorithm handles if inputs are opposite)" + "\nreturned value = " + retVal + "\n");
```

```
    retVal = FindNumBetweensRecursive.findNumBetweensRecursively( null, 9, 3 );
    System.out.print( "\nif array is null" + "\nreturned value = " + retVal + "\n");
    System.out.println("----------------------------");

    int[] arr2 = { -51,-40,-31,-27,-23,-16,-10,-8,-5,-3,0,5,17,21,26,33,42 };
    System.out.print("arr2 = ");
    for ( int i = 0; i < arr2.length; ++i ) System.out.print( arr2[i] + " " );

    retVal = FindNumBetweensRecursive.findNumBetweensRecursively( arr2, -42, -30 );
    System.out.print( "\n\nbetween = -42 and -30" + "\nreturned value = " + retVal + "\n");

    retVal = FindNumBetweensRecursive.findNumBetweensRecursively( arr2, -10, 15 );
    System.out.print( "\nbetween = -10 and 15" + "\nreturned value = " + retVal + "\n");

    retVal = FindNumBetweensRecursive.findNumBetweensRecursively( arr2, 22, 25 );
    System.out.print( "\nbetween = 22 and 25" + "\nreturned value = " + retVal + "\n");

    retVal = FindNumBetweensRecursive.findNumBetweensRecursively( arr2, -22, -22 );
    System.out.print( "\nbetween = -22 and -22" + "\nreturned value = " + retVal + "\n");

    retVal = FindNumBetweensRecursive.findNumBetweensRecursively( arr2, -50, 30 );
    System.out.print( "\nbetween = -50 and 30" + "\nreturned value = " + retVal + "\n");
}
```

```
    public static void testFindContiguousSubArrRecursive ( ) {
        System.out.println("\n\n___Testing FindContiguousSubArrRecursive___");

        int[] arr1 = { 3,2,1,6,5,1 };
        System.out.print("\narr1 = ");
        for ( int i = 0; i < arr1.length; ++i ) System.out.print(arr1[i] + " ");
        System.out.println("\n\nsum = 6\nresult");
        FindContiguousSubArrRecursive.findContSubArrRecursively( arr1, 6 );

        System.out.println("\n\nsum = -5\nresult");
        FindContiguousSubArrRecursive.findContSubArrRecursively( arr1, -5 );

        System.out.println("\nsum = 5\nresult");
        FindContiguousSubArrRecursive.findContSubArrRecursively( arr1, 5 );

        System.out.println("\nsending null array\nresult");
        FindContiguousSubArrRecursive.findContSubArrRecursively( null, 5 );

        int[] arr2 = { -7,10,3,0,8,-5,-4,7,5,-2,5,1,1,1,2 };
        System.out.print("\narr2 = ");
        for ( int i = 0; i < arr2.length; ++i ) System.out.print(arr2[i] + " ");
        System.out.println("\n\nsum = 3\nresult");
        FindContiguousSubArrRecursive.findContSubArrRecursively( arr2, 3 );

        System.out.println("\nsum = 5\nresult");
        FindContiguousSubArrRecursive.findContSubArrRecursively( arr2, 5 );
    }
```

```
public static void testCalculatePossibleConfigArrayRecursive ( ) {
    System.out.println("\n\n___Testing CalculatePossibleConfigArrayRecursive___");
    for( int i = 3; i <= 9; ++i ) {
        System.out.print("\nFor array size " + i + " result is\n\n" );
        int[] arr = new int[i];
        CalculatePossibleConfigArrayRecursive.calcPossibleConfigArrRecursively(arr);
        System.out.print("-----------\n");
    }
}
```

6. RUNNING AND RESULTS

```
___Testing QueryStringRecursive___

main string = appleelmaappleelmaee
query string = ee
occurrence = 2
returns = 13

main string = aaeexxyyaaoo
query string = aa
occurrence = 1
returns = 0

main string = abc
query string = abc
occurrence = 1
returns = 0

main string = nkaabfeageaabex
query string = aab
occurrence = 2
returns = 10

main string = abcxyzneabxynekmxabnenkenemm
query string = ne
occurrence = 3
returns = 19

main string = efaeeaffefa
query string = xk
occurrence = 2
returns = -1

main string = efaeeaffexk
query string = xk
occurrence = 1
returns = 9

main string = xyz
query string = apple
occurrence = 1
returns = -1

main string = xxxxxx
query string = xx
occurrence = 3
returns = 2
```

```
___Testing FindNumBetweensRecursive___
arr1 = 1 2 3 4 5 6 7 8 9 10

between = 2 and 6
returned value = 5

between = 1 and 1
returned value = 1

between = -2 and 3
returned value = 3

between = 3 and 15
returned value = 8

between = 13 and 15
returned value = 0

between = 7 and 10
returned value = 4

between = 9 and 3(algorithm handles if inputs are opposite)
returned value = 7

if array is null
returned value = -1
-----------------------------
arr2 = -51 -40 -31 -27 -23 -16 -10 -8 -5 -3 0 5 17 21 26 33 42

between = -42 and -30
returned value = 2

between = -10 and 15
returned value = 6

between = 22 and 25
returned value = 0

between = -22 and -22
returned value = 0

between = -50 and 30
returned value = 14
```

```
___Testing FindContiguousSubArrRecursive___

arr1 = 3 2 1 6 5 1

sum = 6
result
[3, 2, 1]
[6]
[5, 1]
----------------


sum = -5
result
----------------

sum = 5
result
[3, 2]
[5]
----------------

sending null array
result
Array is null!
```

```
arr2 = -7 10 3 0 8 -5 -4 7 5 -2 5 1 1 1 2

sum = 3
result
[-7, 10]
[3]
[3, 0]
[0, 8, -5]
[8, -5]
[-5, -4, 7, 5]
[-4, 7]
[5, -2]
[-2, 5]
[1, 1, 1]
[1, 2]
----------------

sum = 5
result
[-7, 10, 3, 0, 8, -5, -4]
[5]
[-2, 5, 1, 1]
[5]
[1, 1, 1, 2]
----------------
```

```
___Testing CalculatePossibleConfigArrayRecursive___

For array size 3 result is

|*||*||*|

-----------

For array size 4 result is

|*||*||*||_|

|_||*||*||*|

|*||*||*||*|

-----------

For array size 5 result is

|*||*||*||_||_|

|_||*||*||*||_|

|_||_||*||*||*|

|*||*||*||*||_|

|_||*||*||*||*|

|*||*||*||*||*|

-----------
```


For array size 6 result is
*		*		*		_		_		_
_		*		*		*		_		_
_		_		*		*		*		_
_		_		_		*		*		*
*		*		*		*		_		_
_		*		*		*		*		_
_		_		*		*		*		*
*		*		*		*		*		_
_		*		*		*		*		*
*		*		*		*		*		*

```
For array size 7 result is

|*||*||*||_||_||_||_|

|_||*||*||*||_||_||_|

|_||_||*||*||*||_||_|

|_||_||_||*||*||*||_|

|_||_||_||_||*||*||*|

|*||*||*||_||*||*||*|

|*||*||*||*||_||_||_|

|_||*||*||*||*||_||_|

|_||_||*||*||*||*||_|

|_||_||*||*||*||_||*|

|*||*||*||*||_||*||_|

|_||*||*||*||*||*||_|

|_||_||*||*||*||*||*|

|*||*||*||*||*||_||_|

|_||*||*||*||*||*||_|

|*||*||*||*||*||*||*|

-----------
```

```
For array size 8 result is

|*||*||*||_||_||_||_||_|

|_||*||*||*||_||_||_||_|

|_||_||*||*||*||_||_||_|

|_||_||_||*||*||*||_||_|

|_||_||_||_||*||*||*||_|

|_||_||_||_||_||*||*||*|

|*||*||*||_||_||*||*||*|

|_||*||*||*||_||*||*||*|

|*||*||*||*||_||_||_||_|

|_||*||*||*||*||_||_||_|

|_||_||*||*||*||*||_||_|

|_||_||_||*||*||*||*||_|

|_||_||_||_||*||*||*||*|

|*||*||*||*||*||_||_||_|

|_||*||*||*||*||*||_||_|

|_||_||*||*||*||*||*||_|

|_||_||_||*||*||*||*||*|

|*||*||*||*||*||*||_||_|

|_||*||*||*||*||*||*||_|

|_||_||*||*||*||*||*||*|

|*||*||*||*||*||*||*||_|

|_||*||*||*||*||*||*||*|

|*||*||*||*||*||*||*||*|

-----------
```

“make” command compiles, and “make run” command runs the program.