CSE 222/CSE505 SPRING 2022

HOMEWORK 4 REPORT

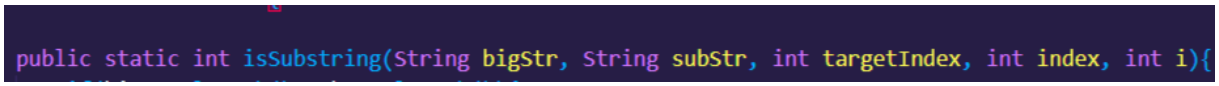
Burcu Sultan Orhan

1901042667

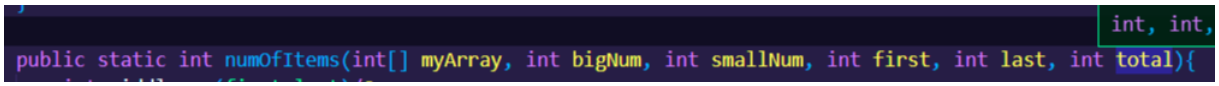
1- Detailed System Requirements

There needs to be several methods to perform all the required missions.

This function checks whether a string consists of other string index times. If substring occures index times, function returns i, otherwise it returns -1

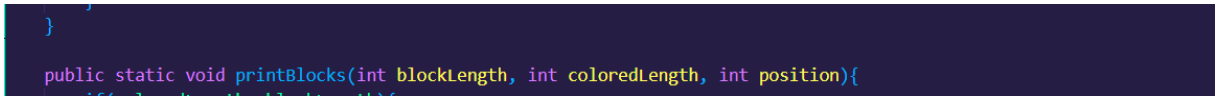


This function finds the number of items in the array between two given integer values. By using binary search-type algorithm

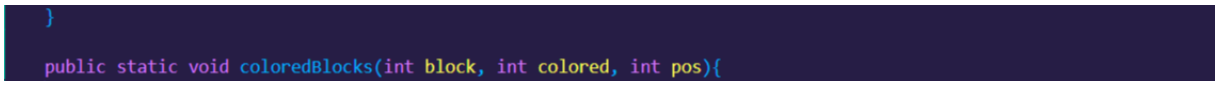


This function finds contiguous subarray/s that the sum of its/theirs items is equal to a given integer value. Prints the subarrays

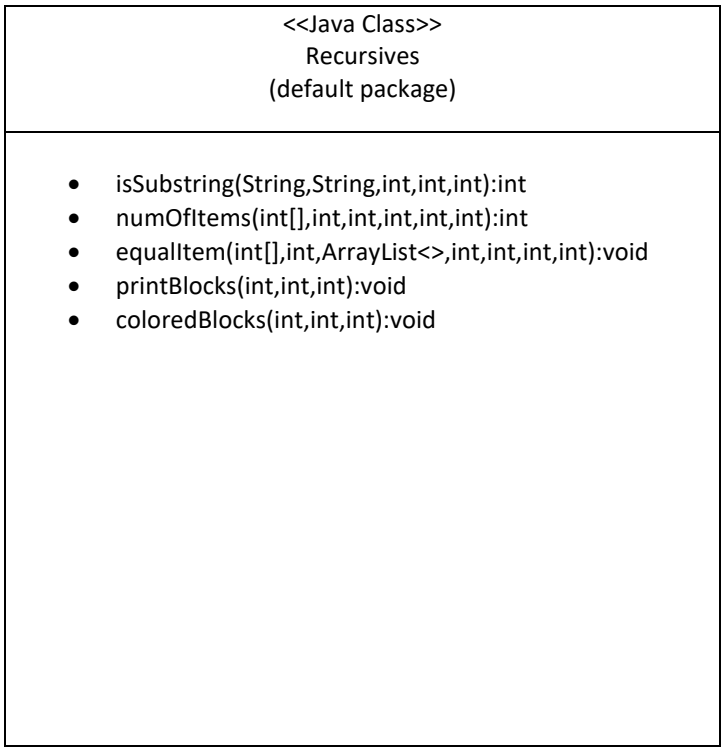
This function calculates all the possible configurations to fill an array with specific length with colored-blocks with length at least 3



And this is a helper function to print blocks with the previous function



2- Class Diagrams



3- Problem Solving Approach

-For the first algorithm, I checked if the small string is equal to bigger string's part which starts from the beginning and ends at small string's length. If they are equal, index increments and if index is equal to the targeted index, function returns the index. Else, functions calls itself but this time without the first char of the bigger string.

-For the second algorithm, I used binary search to a point, and after that, I deleted the middle item in the array if it's between given integer values and incremented the index. After every deletion, function calls itself.

-For the third algorithm, I used arraylist to store subarray/s. I used recursion to access all the elements of the given array. At each opeation, function adds that current element of the array to the previous ones, if the summation equals to the item, subarray gets printed and function calls itself to check other elements. If the summation is smaller than the item, current element gets added to the subarray and function calls itself. If the summation is bigger than the item, first element in the subarray gets deleted and function calls itself.

-For the fifth algorithm, starting from length 3, I checked every starting position of the colored block's length and if it's not bigger than the total length, function called the helper printing function. If it's bigger than the total length, function sets position to 1 and increments the colored block's length, and calls itself.

4- Test Cases

```java
int i;
i = Recursives.isSubstring("lalalalalala", "la", 3, 0, 1);
System.out.println(i);
i = Recursives.isSubstring("hello world", "hi", 2, 0, 1);
System.out.println(i);
i = Recursives.isSubstring("wowowowo", "wo", 8, 0, 1);
System.out.println(i);


int[] myArr1 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14};
int x;
x = Recursives.numOfItems(myArr1, 9, 2, 0, 13, 0);
System.out.println(x);
x = Recursives.numOfItems(myArr1, 70, 60, 0, 13, 0);
System.out.println(x);
x = Recursives.numOfItems(myArr1, 13, 6, 0, 13, 0);
System.out.println(x);


int[] myArray = {1, 2, 3, 4, 5, 6, 7};
ArrayList<Integer> subArray = new ArrayList<>();
Recursives.equalItem(myArray, 20, subArray, 0, 0, 0, 7);
ArrayList<Integer> subArray2 = new ArrayList<>();
Recursives.equalItem(myArray, 6, subArray2, 0, 0, 0, 7);
ArrayList<Integer> subArray3 = new ArrayList<>();
Recursives.equalItem(myArray, 39, subArray3, 0, 0, 0, 7);


Recursives.printBlocks(6, 3, 1);
```

5- Running Command and Results

Q1:

```
xe' '-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:63058' '--enable-preview' '-XX:+Sho
:\Users\burcu\Desktop\bso_hw4\hw4\bin' 'Driver'
5
-1
-1
```

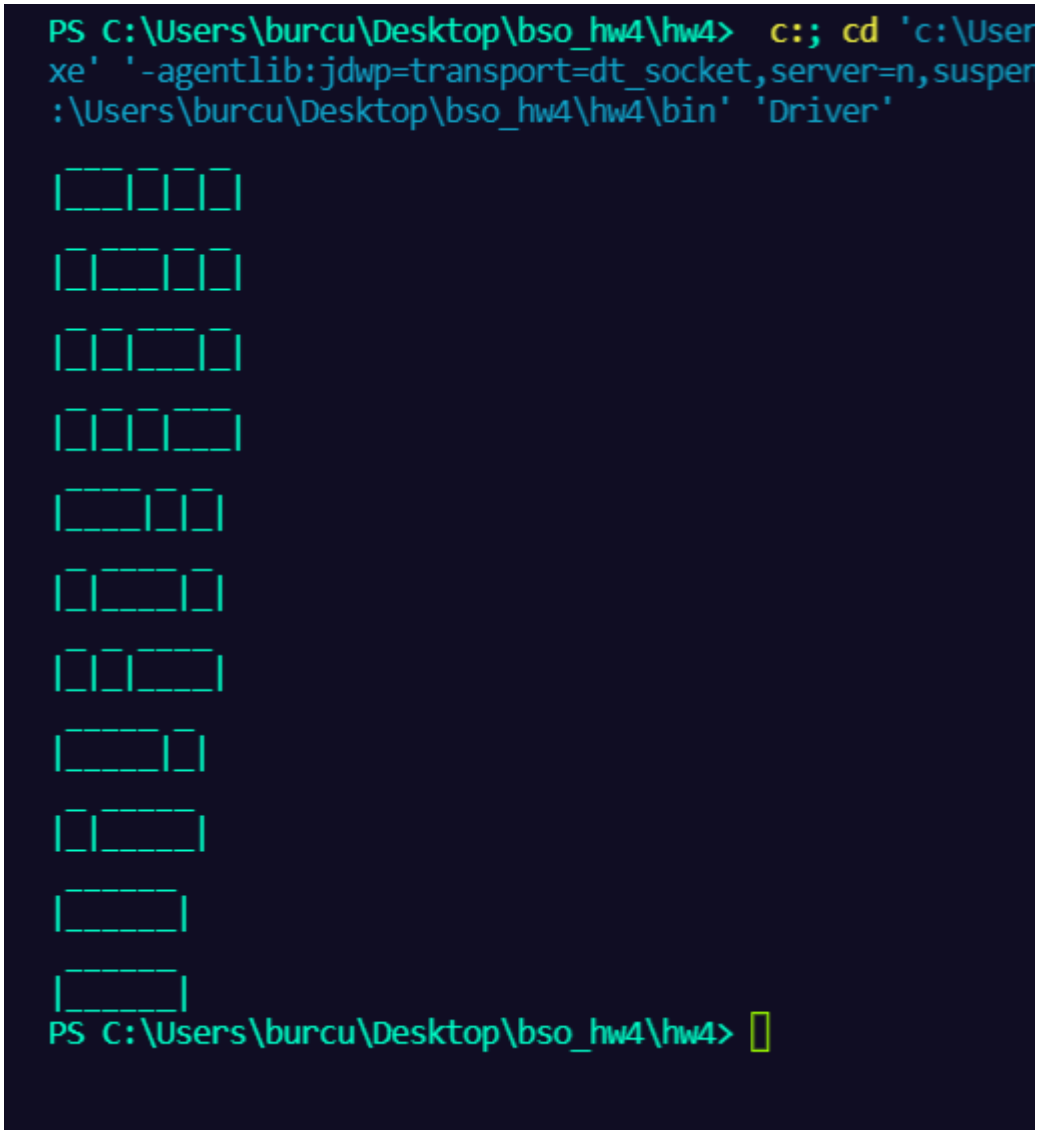
Q2:

```
PS C:\Users\burcu\Desktop\bso_hw4\hw4>  c:; cd 'c:\Users\burcu\Desktop\bso_hw4
xe' '-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:5
:\Users\burcu\Desktop\bso_hw4\hw4\bin' 'Driver'
8
0
7
PS C:\Users\burcu\Desktop\bso_hw4\hw4>
```

Q3:

```
PS C:\Users\burcu\Desktop\bso_hw4\hw4>  c:; cd 'c:\Users\burcu\Desktop\bso_hw4\h
xe' '-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:531
:\Users\burcu\Desktop\bso_hw4\hw4\bin' 'Driver'
[2, 3, 4, 5, 6]
[1, 2, 3]
[6]
PS C:\Users\burcu\Desktop\bso_hw4\hw4>
```

Q5:


PS C:\Users\burcu\Desktop\bso_hw4\hw4> c:; cd 'c:\User
xe' '-agentlib:jdwp=transport=dt_socket,server=n,suspen
:\Users\burcu\Desktop\bso_hw4\hw4\bin' 'Driver'
PS C:\Users\burcu\Desktop\bso_hw4\hw4>

TIME COMPLEXITY ANALYSIS

Q1:

```java
public class Recursives {

    public static int isSubstring(String bigStr, String subStr, int targetIndex, int index, int i){
        if(bigStr.length()<subStr.length()){
            return -1;
        }
        else if(bigStr.substring(0, subStr.length()).equals(subStr)){
            index++;
            if(index == targetIndex){
                return i;
            }
        }
        return isSubstring(bigStr.substring(1), subStr, targetIndex, index, i+1);
    }
```

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

-Is the algorithm correct for the base case?

Yes, if index gets incremented enough to match targetIndex, is successful. Or if the bigger string's length hit smaller's string length, is unsuccesful.

-Will the algorithm terminate?

Yes, since bigger string's length gets reduced each call, it will eventually be shorter than substring's length.

Q2:

```java
public static int numOfItems(int[] myArray, int bigNum, int smallNum, int first, int last, int total){
    int middle = (first+last)/2;

    if(myArray[middle]>bigNum){
        return numOfItems(myArray, bigNum, smallNum, first, middle, total);
    }
    if(myArray[middle]<smallNum){
        if(last<=first+1){
            return total;
        }
        else{
            return numOfItems(myArray, bigNum, smallNum, middle, last, total);
        }
    }
    else if(myArray[middle] >= smallNum && myArray[middle] <= bigNum){
        for(int i=middle; i<last; i++){
            myArray[i] = myArray[i+1];
        }
        total++;
        last--;
        return numOfItems(myArray, bigNum, smallNum, first, last, total);
    }
    else{
        return total;
    }
}
```

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

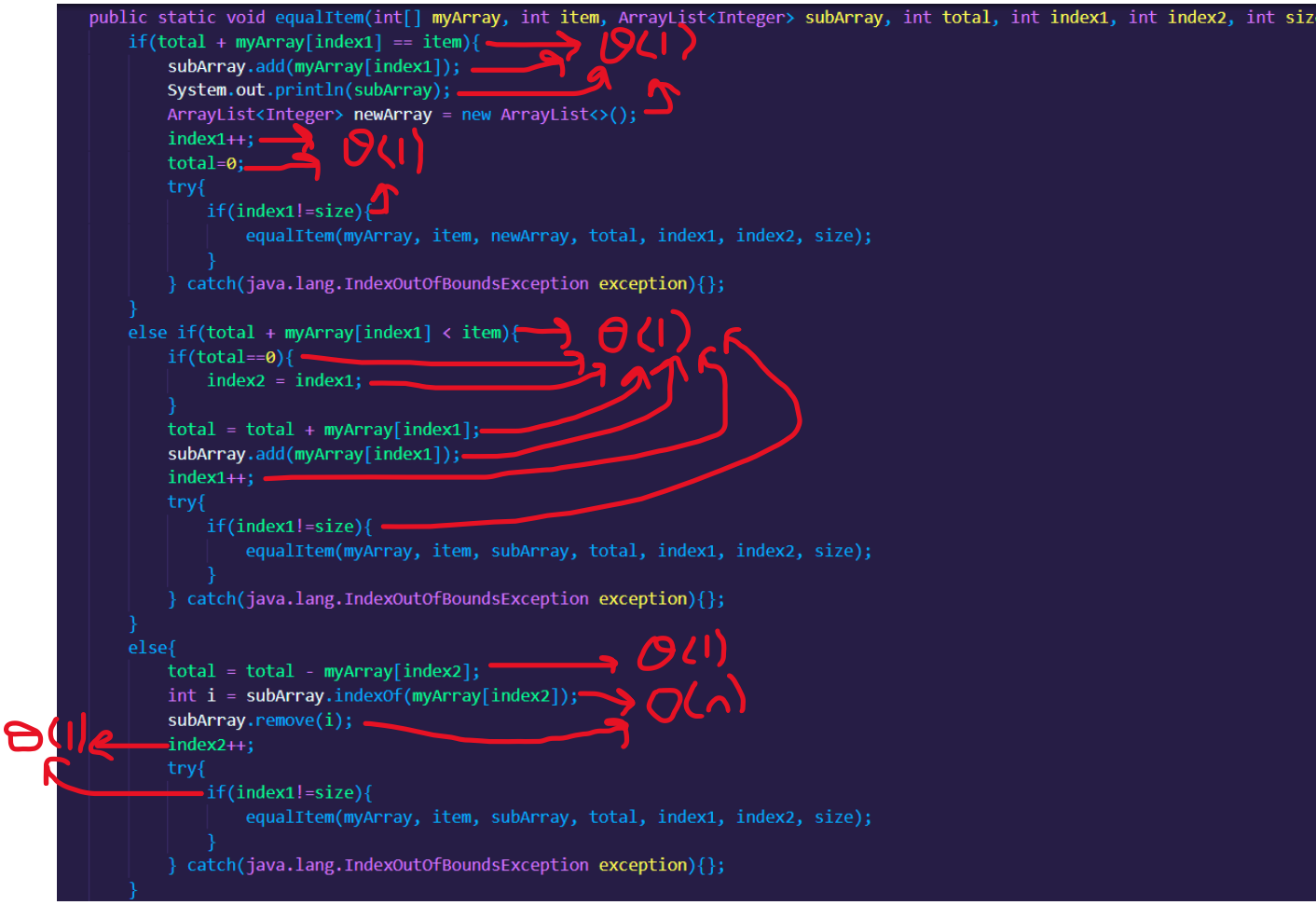
-Is the algorithm correct for the base case?

Yes, if there's no element left which is between the two given integers, then our search is succesful.

-Will the algorithm terminate?

Since the algorithm always reduces the array to search, it will eventually hit base case

Q3:



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

-Is the algorithm correct for the base case?

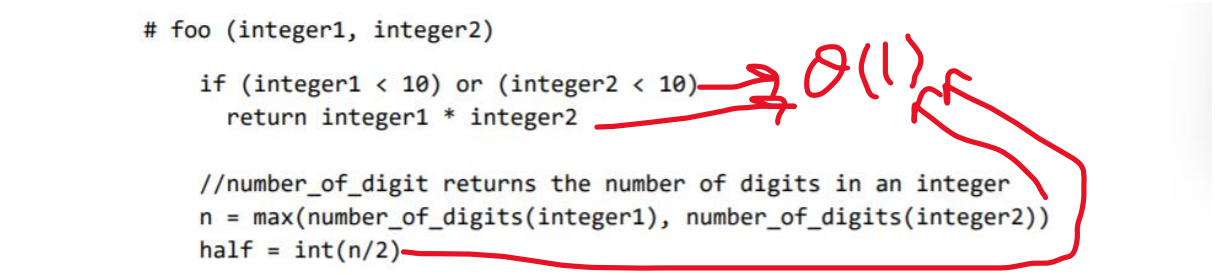
This algorithm's base case is when all the elements are searched.

-Will the algorithm terminate?

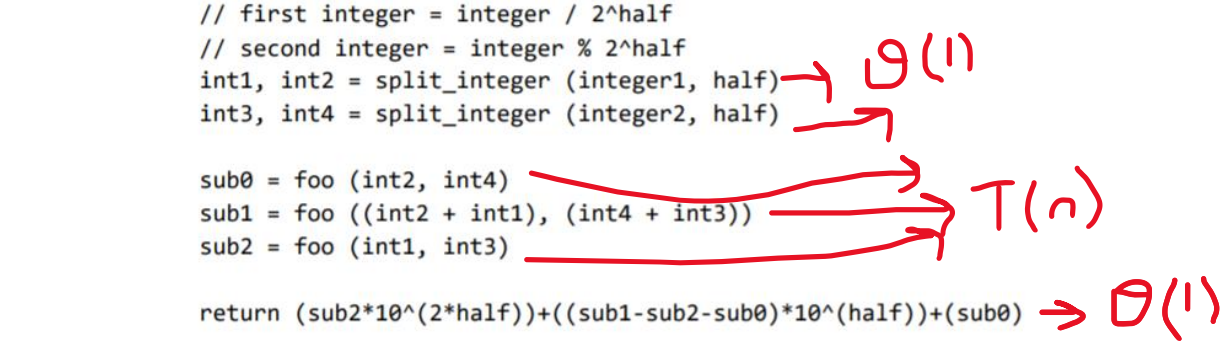
Since index1 gets incremented with each call, it will eventually hit size

Q4:

```
# foo (integer1, integer2)

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

    //number_of_digit returns the number of digits in an integer
    n = max(number_of_digits(integer1), number_of_digits(integer2))
    half = int(n/2)

    // split_integer splits the integer into returns two integers
    //               from the digit at position half. i.e.,
```




```
    // first integer = integer / 2^half
    // second integer = integer % 2^half
    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)
```



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

This function basically implements mathematical multiplication. It splits two given integers until they hit smaller than 10, and implements this with 3 different step value generators. Since this function works on 1-digit integer level, step values are dismissed until function hit return step. In return step, values of the three recursive calls get multiplied by 10 required times, and finally summed.

Q5:

```java
public static void printBlocks(int blockLength, int coloredLength, int position){
    if(coloredLength==blockLength){
        coloredBlocks(blockLength, coloredLength, position);
    }
    if(coloredLength+position>blockLength+1){
        return;
    }
    else if(coloredLength+position==blockLength+1){
        coloredBlocks(blockLength, coloredLength, position);
        printBlocks(blockLength, coloredLength+1, 1);
    }
    else{
        coloredBlocks(blockLength, coloredLength, position);
        printBlocks(blockLength, coloredLength, position+1);
    }
}

public static void coloredBlocks(int block, int colored, int pos){
    for(int i=1; i<=block; i++){
        if(!(i>pos && i<colored+pos)){
            System.out.print(" _");
        }
        else{
            System.out.print("_");
        }
    }
    System.out.print("\n");
    for(int i=1; i<=block; i++){
        if(!(i>pos && i<colored+pos)){
            System.out.print("|_");
        }
        else{
            System.out.print("_");
        }
    }
    System.out.println("|");
}
```

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