

CSE 222

Md Sarwar Hossain
Id : 161044121

Ques 1:

Problem Requirement:

In this problem, We need to find the i^{th} occurrence of a sub-string in a string.

Problem Solution Approach:

To solve this problem using recursion, the base case that I selected is when the length of string is out of bound. If not from that point I called another recursive function which will find the exact match. If it's a exact match then I increase the counter value to see if it's the i^{th} occurrence. The helper function will take two string of same length. The base case is if the length is already 0.

```
public static boolean exactMatch(String text, String target)
{
    if (text.length() != target.length())
        return false;
    if (target.length() == 0)
        return true;

    if (text.charAt(0) == target.charAt(0))
        return exactMatch(text.substring(1), target.substring(1));
    return false;
}
```

Recurrence Function:

$$T(n) = \begin{matrix} T(0) & 1 \\ T(n-1) & +1 \end{matrix}$$

Best Case : $\Omega(1)$ when size doesn't match

Worst Case : $O(n)$ traverse till the end of the string

```
public static int findSubstring(String string,int index , String
                                target,int count,int ithIndex){
    if(index > (string.length()-target.length())) {
        return -1;
    }
    if(exactMatch(string.substring(index, index+target.length()),
        target)){
        count+=1;
        if(ithIndex == count) return index;
    }
    return findSubstring(string, index+1, target,count,ithIndex);
}
```

Recurrence Relation :

$$T(n) = \begin{matrix} T(1) & 1 \\ T(n-1) & + n \end{matrix}$$

Best Case: $\Omega(1)$ when the target string length is greater than the substring

Worst Case: $O(n^2)$

Test Case (1): Find 1st occurrence of sar in "mdsarwarhossain"

Expected Output: 2

Result : Pass

Test Case (2): Find 2nd occurrence of sar in "mdsarwarhossainsar"

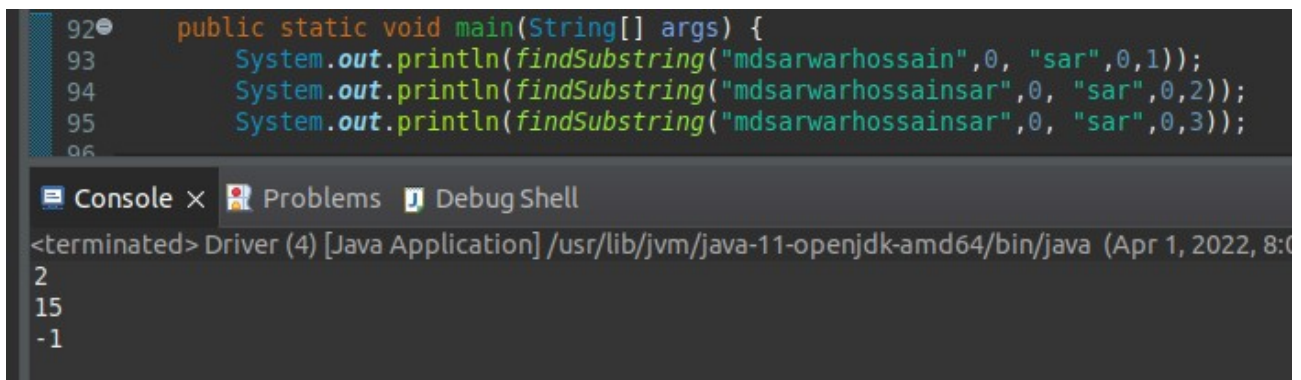
Expected Output : 15

Result : Pass

Test Case (3): Find 3rd occurrence of sar in "mdsarwarhossainsar"

Expected Output : -1

Result : Pass

A screenshot of an IDE window. The top pane shows Java code with line numbers 92 to 96. The code defines a main method that calls findSubstring three times with different strings and indices. The bottom pane shows the console output with three lines: 2, 15, and -1.

```
92 public static void main(String[] args) {  
93     System.out.println(findSubstring("mdsarwarhossain",0, "sar",0,1));  
94     System.out.println(findSubstring("mdsarwarhossainsar",0, "sar",0,2));  
95     System.out.println(findSubstring("mdsarwarhossainsar",0, "sar",0,3));  
96 }
```

Console x Problems Debug Shell

<terminated> Driver (4) [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (Apr 1, 2022, 8:00)
2
15
-1

Ques 2:

Problem Definition:

We need to find number of items in the array between two given Integer value. The array is sorted. We need to define a recursive algorithm to solve this problem.

Problem Solution Approach:

To solve this problem, I modified the binary search algorithm to achieve the goal. I put both the point in a modified binary function to get either the index or value closest to the index depending on upper limit or lower limit. If it's the lower limit then I returned the index or the index of the value just lower than the target value. Opposite in the case of upper limit. Then the rest of the calculation is as the number of element will be upperlimit-lowerlimit+1.

```
public static int upperIndex(int[] array,int searchItem,int low,int high ) {  
    if(low > high) {  
        return high;  
    }  
    int mid = (low + high) / 2;  
    if(searchItem < array[mid] ) {  
        return upperIndex(array, searchItem, low, mid-1);  
    }  
    return upperIndex(array, searchItem, mid+1, high);  
}
```

Time Complexity: $O(\log n)$ same as binary search

```

public static int lowerIndex(int[] array,int searchItem,int low,int high) {
    if(low > high) {
        return low;
    }
    int mid = (low + high) / 2;
    if(searchItem <= array[mid] ) {
        return lowerIndex(array, searchItem, low, mid-1);
    }
    return lowerIndex(array, searchItem, mid+1, high);
}

```

Time Complexity: $O(\log n)$ same as binary search

```

public static int numberOfItemsBetween(int x,int y,int [] array) {
    int lIndex = lowerIndex(array, x, 0, array.length-1);
    int uIndex = upperIndex(array, y, 0, array.length-1);
    return uIndex-lIndex+1;
}

```

Time Complexity : $O(\log n)$

Array: {-10,2,5,11,17,20,26};

Test Case: number of items between 11-20

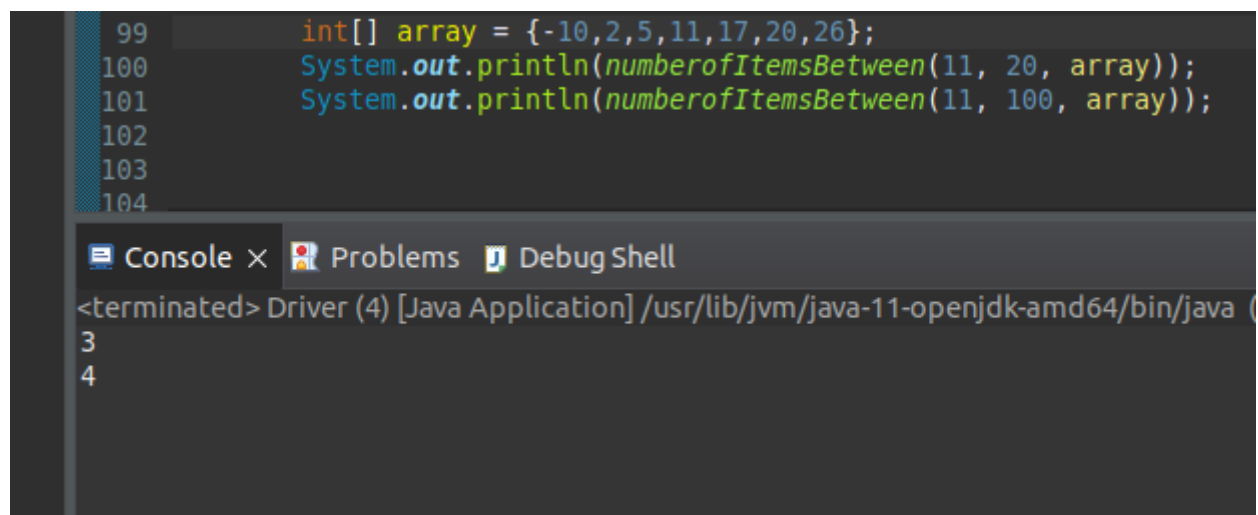
Expected Output : 3

Result : pass

Test Case: number of items between 11-100

Expected Output : 4

Result : pass



The screenshot shows an IDE with a Java file containing the following code:

```

99     int[] array = {-10,2,5,11,17,20,26};
100     System.out.println(numberOfItemsBetween(11, 20, array));
101     System.out.println(numberOfItemsBetween(11, 100, array));
102
103
104

```

Below the code editor, there is a console window with the following output:

```

<terminated> Driver (4) [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (
3
4

```

Ques 3:

Problem Definition :

In a unsorted array, the problem needs us to find the contiguous sub-array whose sum equal to a given integer.

Problem Solution Approach:

I used brute force technique to achieve the goal. As a base case I index is within the range of the array. If it's not then we are done with our solution. For each index I called a helper method which will sum up all the items starting from that index to check if the sum is equal to given sum. The helper recursive method also has the same base case. I used a arraylist of integer array to keep track of the sub-arrays

```
public static int isSumPresent(int[] array,int index,int sum,int targetSum) {
    if(index == array.length) {
        return -1;
    }
    sum+=array[index];
    if(sum > targetSum) {
        return -1;
    }else if(sum == targetSum) {
        return index;
    }
    return isSumPresent(array, index+1, sum,targetSum);
}
```

Recurrence Relation :

$T(n) = T(0) + 1$ when index is reached to zero same as $index == length$
 $T(n-1)+1$

Best Case : $\Omega(1)$

Worst Case : $O(n)$

```
public static void contiguousSubarray(int[] array,int currentIndex,int
                                     sum,ArrayList<int[]> result) {
    if(currentIndex == array.length) {
        return;
    }
    if(array[currentIndex] <= sum) {
        int isPresent = isSumPresent(array, currentIndex, 0, sum);
        if(isPresent!=-1) {
            int [] subArray = Arrays.copyOfRange(array,
            currentIndex, isPresent +1);
            result.add(subArray);
        }
    }
    contiguousSubarray(array,currentIndex+1, sum,result);
}
```

$T(n) = T(0) + 1$ when index is reached to zero same as $index == length$
 $T(n-1)+ n$

Best Case : $\Omega(1)$

Worst Case : $O(n^2)$

Array : `int[] array = {-10,2,5,11,-10,17,26};`
Test Case: find contiguous array whose sum equal to 7
Result : pass

```
112     System.out.println("\tQuestion Three");
113     {
114         int[] array = {-10,2,5,11,-10,17,26};
115         ArrayList<int[]> result = new ArrayList<int[]>();
116         contiguousSubarray(array, 0, 7,result);
117         for(int i = 0; i < result.size(); i++) {
118             System.out.print("[ ");
119             for(int j = 0; j < result.get(i).length; j++) {
120                 System.out.print(result.get(i)[j]+" ");
121             }
122             System.out.println("]");
123         }
124     }
125 }
126
127
```

Console × Problems Debug Shell

<terminated> Driver (4) [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/jav
Question Three

[2 5]
[-10 17]

Ques 4:

```
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)
```

According to this function, The base condition is either of these number less than 10 then we return `num1 * num2`

We take the maximum of these number then produce 4 number from 2 number by splitting the number into half.

Then we call the function recursively with these new number.

Considering the maximum of these two number will give us the worst case complexity, in each recursive call we reduce the problem size by half.

The recurrence Relation from these function will be,
 $T(n) = 3T(n/2) + 1$ considering the other internal function takes constant amount of time. Lets say The base case $T(1) = 1$
 So the time complexity is $O(\log n)$

Ques 5:

Problem Definition :

In this problem, we are asked to find the number of combination in which a 1D array can be fill with a block of size ≥ 3 . There should be a gap between each consecutive block.

Problem Solution Approach:

To solve this problem, I used dynamic programming approach where the problem is divided into small part to solve the larger piece of the puzzle.

[illegible]

U P P E R P A R T							
L O W E R P A R T							

For a block of size 3, from the the array size , 5,6,7,8 we can see a patter. We can divide the pattern into two parts. First part where the patter is increase by 1 each time followed by the result of (n-1) block.

For the upper part the the value is $f(n-1)+1$. Now for the lower part, when the number of empty cell in the last cell of upper part $>$ block size then, we got another factorial. For the lower part we get $f(n-\text{blocksize})+1$.

If we combine them then,

```
int fact(int n, int blockSize)
    if (n <= blockSize) return 0;
return 1 + fact(n - 1, blockSize)

int foo(int n, int blockSize) {
    if (n < blockSize) return 0;
return 1 +foo(n-1,blockSize)+fact(n-blockSize,blockSize);
}
```

Foo function will give us the total number of possible combination. I used some other helper functions to print combination in a arraylist for visual representation.

Test Cases:

Case 1 : array length 7 , block size 3

Expected Result: 6

Output: 6

Result : pass

Case 2 : array length 8 , block size 3

Expected Result : 9

Output : 9

Result : pass

Case 3: array length 8, block size 4

Expected Result : 5

Output : 5

Result : pass

<terminated> Driver (4) [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (

Question Five

For block Size: 3 and Array Length: 7 total possible combination: 6

[0, 0, 0, 0, 1, 1, 1]

[0, 0, 0, 1, 1, 1, 0]

[0, 0, 1, 1, 1, 0, 0]

[0, 1, 1, 1, 0, 0, 0]

[1, 1, 1, 0, 0, 0, 0]

[1, 1, 1, 0, 1, 1, 1]

For block Size: 3 and Array Length: 8 total possible combination: 9

[0, 0, 0, 0, 0, 1, 1, 1]

[0, 0, 0, 0, 1, 1, 1, 0]

[0, 0, 0, 1, 1, 1, 0, 0]

[0, 0, 1, 1, 1, 0, 0, 0]

[0, 1, 1, 1, 0, 0, 0, 0]

[1, 1, 1, 0, 0, 0, 0, 0]

[1, 1, 1, 0, 0, 1, 1, 1]

[1, 1, 1, 0, 1, 1, 1, 0]

[0, 1, 1, 1, 0, 1, 1, 1]

For block Size: 4 and Array Length: 8 total possible combination: 5

[0, 0, 0, 0, 1, 1, 1, 1]

[0, 0, 0, 1, 1, 1, 1, 0]

[0, 0, 1, 1, 1, 1, 0, 0]

[0, 1, 1, 1, 1, 0, 0, 0]

[1, 1, 1, 1, 0, 0, 0, 0]