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

Atacan Başaran
200104004008

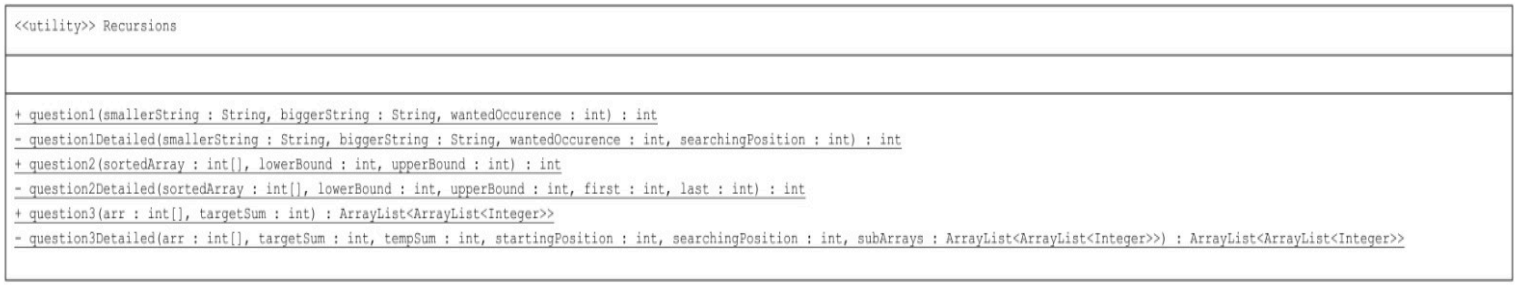
1. SYSTEM REQUIREMENTS

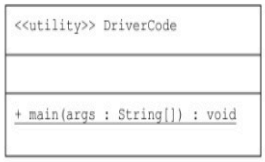
Question 1 -> Recursion method should find small string indice in bigger string according the occurrence

Question 2 -> Recursion method works as a binary search but it should not search an item,it should search for a range.Also after found the element it should look leftside and rightside
because there can be more element which in range

Question 3 -> Recursion method should traverse array and add them if totalsum equals to target that means it is a subarray.After found it,method should look until array ends.

2. CLASS DIAGRAM





3. PROBLEM SOLUTION APPROACH

Question 1 -> I searched the small string in big string using indexof(String,int) which in String class.I searched by scrolling through the big string(searchingPosition) until the occurrence reached the 0 and when I found it I returned the index of it.

Question 2 -> I checked with binarysearch if the array contains elements in the range. If I find it, I also searched the left and right of the position of that element because there may be more elements in that range

Question 3-> I aimed to return the start and end positions of the subarrays. I create tempSum and add the elements to my tempSum by shifting the end position in the array.If it meets the subarray conditions, I add it in 2D Arraylist, if not, I try to find new subarrays by shifting the start position to the right.

Question 4 -> In time complexity analysis

4. TEST CASES

Testing each possibility of Question1 with 1 big string and 2 different small string

```
String bigString = "GTUdenemeGTUdenemeGTUdenemeGTUdeneme";
String smallString = "GTU";
String smallString2 = "deneme";
System.out.println("Big String:  " + bigString);
System.out.println("Small string1:  " + smallString);
System.out.println("Small string2:  " + smallString2);
System.out.println("Search for smallstring1 in bigstring(occurrence 3):  "+Recursions.question1(smallString, bigString,3));
System.out.println("Search for smallstring1 in bigstring(occurrence 50):  "+Recursions.question1(smallString, bigString,50));
System.out.println("Search for Atacan in bigstring(occurrence 1):  "+ Recursions.question1("Atacan", bigString,1));
System.out.println("Search for smallstring2 in bigstring(occurrence 1):  "+Recursions.question1(smallString2, bigString,1));
System.out.println("Search for smallstring2 in bigstring(occurrence -5):  "+Recursions.question1(smallString2, bigString,-5) + "\n\n\n");
```

Testing each possibility of Question2 with an array

```
int[] myarray = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 };
int[] myarray2 = {};

System.out.print("myarray for Question2: ");
for (int i = 0; i < myarray.length; i++)
    System.out.printf("%d ", myarray[i]);

System.out.println("\nQuestion 1 for myarray(Bound 4,10):" + Recursions.question2(myarray, 4, 10));
System.out.println("Question 1 for myarray(Bound 15,25):" + Recursions.question2(myarray, 15, 25));
System.out.println("Question 1 for myarray(Bound -5,0):" + Recursions.question2(myarray, -5, 0));
System.out.println("Question 1 for myarray(Bound 25,50):" + Recursions.question2(myarray, 25, 50));
System.out.println("Question 1 for empty array:" + Recursions.question2(myarray2, 0, 10)+ "\n\n\n");
```

Testing each possibility of Question2 with an array

```
int[] myarray3 = {1,5,2,9,3,4,3,2,5};
int[] myarray4 = {};

System.out.print("myarray for Question3: ");
for (int i = 0; i < myarray3.length; i++)
    System.out.printf("%d ", myarray3[i]);

System.out.println("\nQuestion 3 for myarray(Target 7):  "+ Recursions.question3(myarray3,7));
System.out.println("Question 3 for myarray(Target 0):  "+ Recursions.question3(myarray3,0));
System.out.println("Question 3 for myarray(Target 5):  "+ Recursions.question3(myarray3,5));
System.out.println("Question 3 for empty array(Target 5):  "+ Recursions.question3(myarray4,5));
```

5. RUNNING AND RESULTS

Q1)

```
Big String:  GTUdenemeGTUdenemeGTUdenemeGTUdeneme
Small string1:  GTU
Small string2:  deneme
Search for smallstring1 in bigstring(occurrence 3):  18
Search for smallstring1 in bigstring(occurrence 50):  -1
Search for Atacan in bigstring(occurrence 1):  -1
Search for smallstring2 in bigstring(occurrence 1):  3
Search for smallstring2 in bigstring(occurrence -5):  -1
```

Q2)

```
myarray for Question2: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Question 1 for myarray(Bound 4,10):7
Question 1 for myarray(Bound 15,25):6
Question 1 for myarray(Bound -5,0):0
Question 1 for myarray(Bound 25,50):0
Question 1 for empty array:0
```

Q3)

```
myarray for Question3: 1 5 2 9 3 4 3 2 5
Question 3 for myarray(Target 7):  [[1, 2], [4, 5], [5, 6], [7, 8]]
Question 3 for myarray(Target 0):  []
Question 3 for myarray(Target 5):  [[1, 1], [6, 7], [8, 8]]
Question 3 for empty array(Target 5):  []
```

6. TIME COMPLEXITY ANALYSIS

Question 1

```
private static int question1Detailed(String smallerString, String biggerString, int wantedOccurence,
        int searchingPosition) {

    int index = biggerString.indexOf(smallerString, searchingPosition); // Finding first occurrence smallstring in biggerstring
    if (index == -1) // If there is not smallstring in biggerstring return abnormal value
        return -1;


    if (wantedOccurence == 0)   // Base case: If first occurrence is wantedOccurrence return  index of beginning smallstring in biggerstring
        return index;
    else
        return question1Detailed(smallerString, biggerString, wantedOccurence-1, index + smallerString.length());
        // If wantedOccurrence is not first occurrence, find the other occurrence until reach wantedOccurrence
}
```

T(n) = T(n-1) + O(l*m)
n = occurrence
O(l*m) comes from indeox(String,int) method
l = small string size
m = big string size

Mathematical Induction

n = 0 -> (True) if indexfof method founds small string return index otherwise return -1 (Base case)

Let assume n<= k is True

n = k+1 -> (True) Method found k. occurrence and start searching again after new searchingPosition(k. occurrence + length of small string) . We reach base case. if indexof method founds small string return index otherwise return -1

T(n) = O(n*l*m)

Question 2

```
private static int question2Detailed(int sortedArray[], int lowerBound, int upperBound, int first, int last) {
    if (first > last) // It is for unsuccessful search
        return 0;

    int middle = (first + last) / 2;

    if (sortedArray[middle] >= lowerBound && sortedArray[middle] <= upperBound) {
        // I include lower and upperbound to my output counter
        int found = 1;
        found += question2Detailed(sortedArray, lowerBound, upperBound, first, middle - 1);
        // It found the element inside the bounds but need to look leftside and rightside for all elements which inside the bounds
        found += question2Detailed(sortedArray, lowerBound, upperBound, middle + 1, last);

        return found;
    }

    else if (sortedArray[middle] < lowerBound)
        return question2Detailed(sortedArray, lowerBound, upperBound, middle + 1, last);
        // Element  smaller than we are looking for so search in rightside

    else
        return question2Detailed(sortedArray, lowerBound, upperBound, first, middle - 1);
        // Element bigger than we are looking for so search in rightside

}
```

T(n) = T(n/2) + θ(1)
T(0) = 1
n = number of array elements

T(n) = T(n/2^2) +2* θ(1)
T(n) = T(n/2^k) + 2^k * θ(1)
T(0) = 1 , k= logn
T(n) = O(logn)

Question 3

```
private static ArrayList<ArrayList<Integer>> question3Detailed(int arr[], int targetSum, int tempSum,
        int startingPosition,
        int searchingPosition,
        ArrayList<ArrayList<Integer>> subArrays) {

    if (searchingPosition >= arr.length) // If searching position equals array size it means finding subarray is finished
        return subArrays;

    tempSum += arr[searchingPosition];
    if (tempSum == targetSum) { // It means it found a subarray add it into subArrays and continue to finding new subarray
        ArrayList<Integer> subArraysElement = new ArrayList<>();
        subArraysElement.add(startingPosition);
        subArraysElement.add(searchingPosition);
        subArrays.add(subArraysElement);
        question3Detailed(arr, targetSum, 0, startingPosition + 1, startingPosition + 1, subArrays);

    } else if (tempSum < targetSum) // If it could not find subarray yet continue to finding subarray
        question3Detailed(arr, targetSum, tempSum, startingPosition, searchingPosition + 1, subArrays);

    else if (tempSum > targetSum) // It means it could not be a subarray because exceeded the sum,find a new subarray
        question3Detailed(arr, targetSum, 0, startingPosition + 1, startingPosition + 1, subArrays);

    return subArrays;
}
```

T(n) = T(n-1) + O(n)
T(0) = 1
n = number of array elements

T(n) = T(n-2) + 2*O(n)
T(n) = T(n-k) +k*O(n)
T(0) = 1, k=n
T(n) = O(n)*n + 1
T(n) = O(n^2)

Question 4

foo method splits the integers on half and calling byself 3 times until splits to 1 digit and returning a value which represent with formula

$T(n) = 3*T(n/2) + \theta(n)$
$\theta(n)$ for split_integer
n = digit number of element
$T(n) = 3^k * T(n/2^k) + 3^k * \theta(n)$
$T(1) = 1$, k = logn (I couldn't type logn in base 2 in pdf editor but logn means in base 2 logn in below)

$T(n) = n^{log3} + n^{log3} * \theta(n)$
$T(n) = \theta(n^2)$

Mathematical Induction

n= 1 -> (True) return integer1*integer2 (base case)

Let assume n<=k is true

n = k+1 -> (True) Program did all n<=k foo method(each n calls 3 foo) in n=k+1 program doing extra mathematical equation (constant time) so n=k+1 is true