GTU Department of Computer Engineering

CSE 222 / 505 – Spring 2022

Homework 4

Ahmet USLUOGLU

1801042602

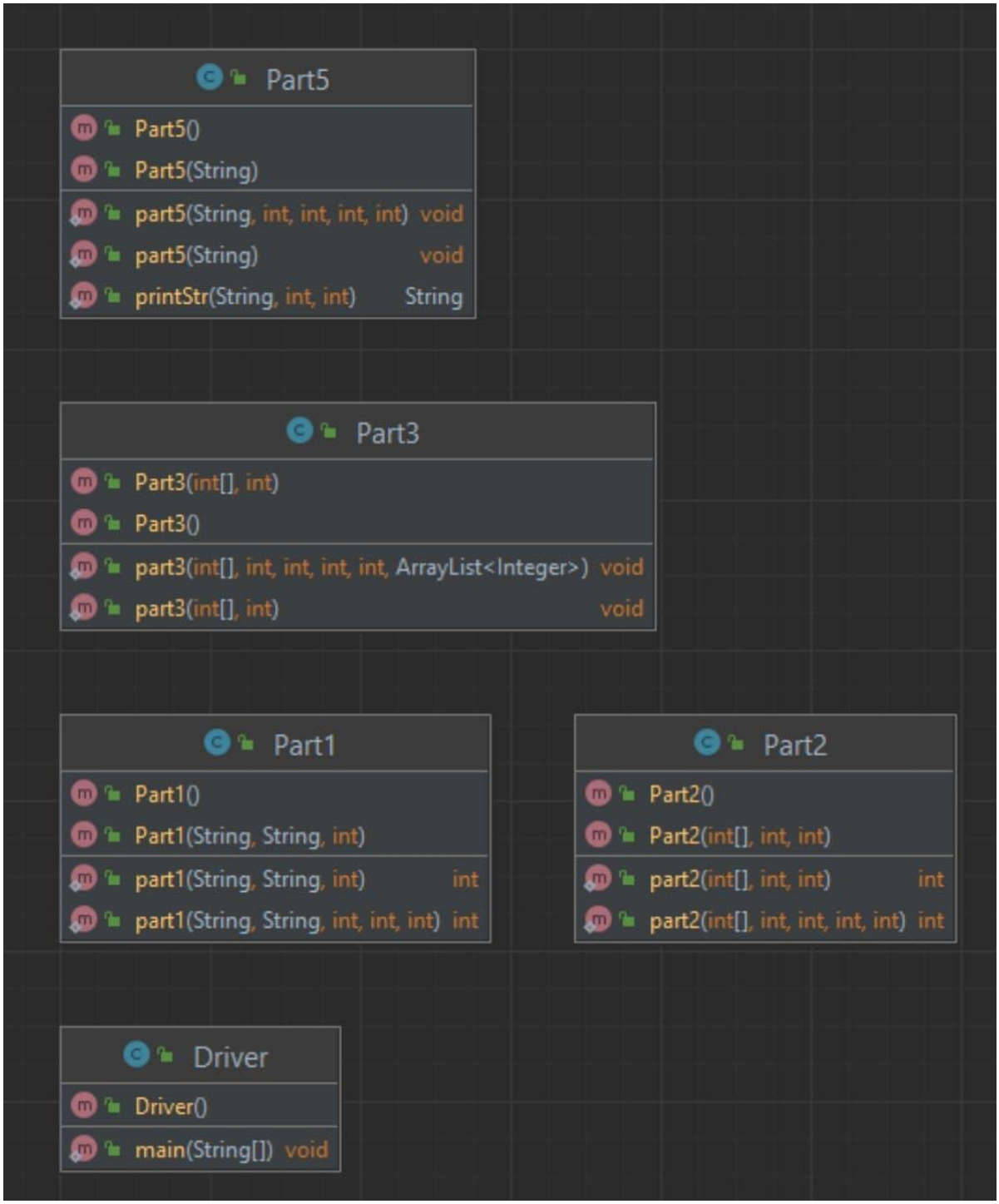
1 – System Requirement

Operating System must have JDK (Java Development Kit) 11 and JRE (Java Runtime Environment) 11 or higher.

There should be enough space for storing data's.

2 – Class Diagrams

*Higher Resolution Version of the Class Diagram is in the files.



3. Problem - Solution Approach

Problem:

Q1. Write a recursive function to search a given string in another given bigger string. The function should return the index of the i'th occurrence of the query string and return -1 when the query string doesn't occur in the big string, or the number of occurrences is less than i.

Solution:

To be able to find the n'th occurrence of a substring in a string I have used String class indexOf method. indexOf method returns the index of the first occurrence of a sub string in the main string. After finding the first occurrence I called my function recursively and set the starting index the first occurrence's index + 1. At each iteration if indexOf method finds a value greater than -1 I increased the count. When count becomes equal to occurrence value function returns the last index it found.

Problem:

Q2. Suppose that you are given a sorted integer array. Suggest a recursive algorithm to find the number of items in the array between two given integer values.

Solution:

To find the number of items in between given integers I have used a similar algorithm to the binary search algorithm. If the middle element of the array is bigger than the upper border function recalls itself recursively to look for the left of the array which has smaller numbers. If the middle element of the array is smaller than the lower border function recalls itself recursively to look for the right of the array which has bigger numbers. If the number is in between borders function recursively calls itself to look for the right and left of

the array while adding 1 to return statement. Every time an element is in bounds it recalls the function. When the arrays left and right indexes are equal Function returns 1. After all recursive calls added to each other total is equal to number of items in between the borders. If there is no element in between borders function returns 0;

Problem:

Q3. Suppose that you are given an unsorted integer array. Propose a recursive solution to find contiguous subarray/s that the sum of its/theirs items is equal to a given integer value.

Solution:

When the program starts function checks if the 0^{th} index element is bigger than the target number. If it is smaller than the target number functions subtracts the number at the index from target number and keeps it as remain.

If the number at the index is bigger than the remained number function recursively calls itself and stars from the index + 1 that it's started at the beginning the functions reaches the end of the array it increases the next starting index by one and recalls itself to start again.

Problem:

Q5. Suppose that you are given a 1-D array of empty blocks of length L. Propose a recursive algorithm that calculates all the possible configurations to fill this array with colored-blocks with length at least 3 (i. e., the length of colored blocks can be from 3 to L). Note that there should be at least one empty block between each consecutive colored-block.

Solution:

Starts with 3 block size and moves one index at a time. To calculate all possibilities After every calculation recalls itself and sends the filled array from the index +1 that is filled.

4 – Test Cases

a-) Testing Question 1
b-) Testing Question 2
c-) Testing Question 3
d-) Testing Question 5

5 – Running Program and Results

a-) Testing Question 1

```
                Testing Part1

Long string is = aaaaaa
Short string is = aaa
4th occurence of short string is at index = 3

Long string is = ...aaa...aaa...aaa...
Short string is = aaa
4th occurence of short string is at index = -1
```

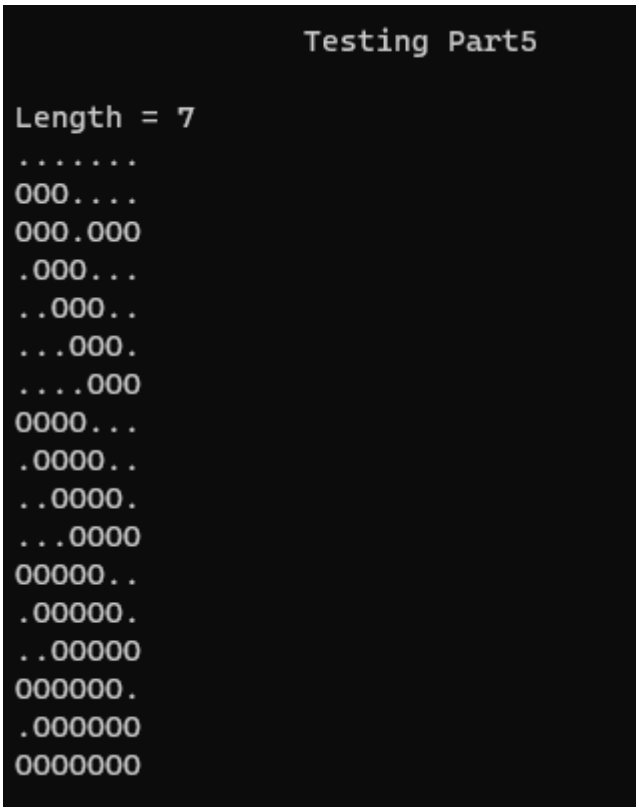
b-) Testing Question 2

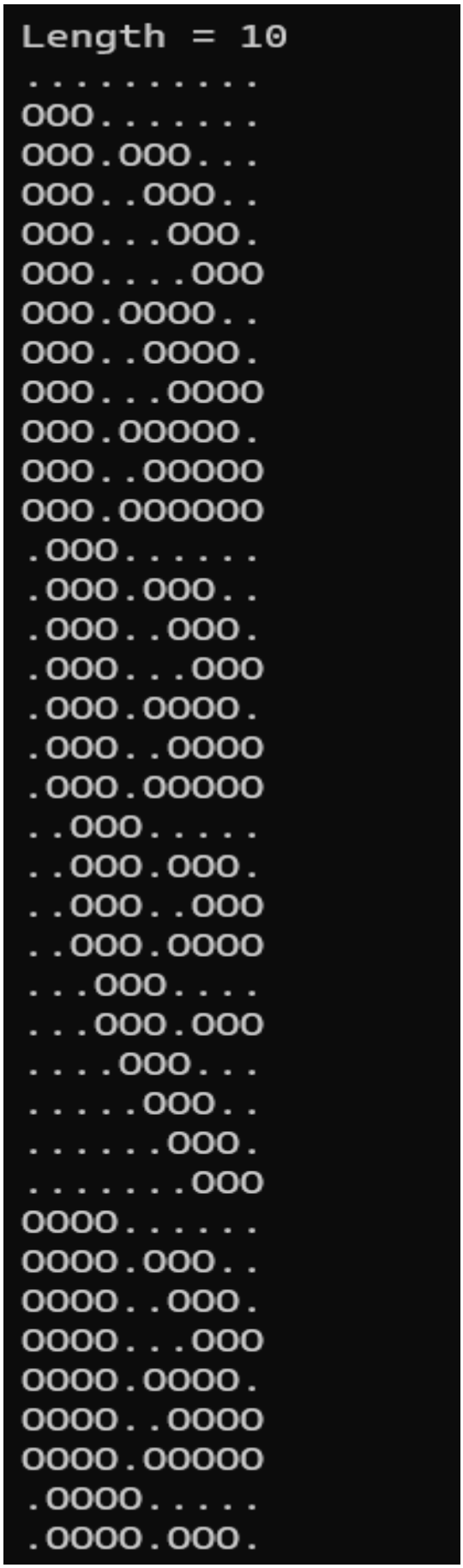
```
                Testing Part2

Array = {1,2,3,4,5,6,7,8,9,10,11,12,13}
Search between -5 - 145 in the Array
Number of items between -5 - 145 in the Array = 13

Array = {1,2,3,4,5,6,7,8,9,10,11,12,13}
Search between -5 - 145 in the Array
Number of items between 14 - 100 in the Array = 0

Array = {1,2,3,4,5,6,7,8,9,10,11,12,13}
Search between 4 - 10 in the Array
Number of items between 4 - 10 in the Array = 7
```

c-) Testing Question 3

```
                Testing Part3

Array = {5,2,3,8,6,4,2,7,9,1,3,5,4,2,4,9,1,3,4,2,6}
        Target is 28
Total is 28 = [5, 2, 3, 8, 6, 4]
Total is 28 = [6, 4, 2, 7, 9]
Total is 28 = [9, 1, 3, 5, 4, 2, 4]
Total is 28 = [1, 3, 5, 4, 2, 4, 9]
Total is 28 = [3, 5, 4, 2, 4, 9, 1]
Total is 28 = [5, 4, 2, 4, 9, 1, 3]


Array = {12,5,32,8,1,10,1,7,8,5,4,2,34,9,4,1,26,5,4}
        Target is 18
Total is 18 = [10, 1, 7]


Array = {1,1,1,1,1,-1,1,-1,1,0}
        Target is 5
Total is 5 = [1, 1, 1, 1, 1]
Total is 5 = [1, 1, 1, 1, 1, -1, 1]
Total is 5 = [1, 1, 1, 1, 1, -1, 1, -1, 1]
Total is 5 = [1, 1, 1, 1, 1, -1, 1, -1, 1, 0]
```

d-) Testing Question 5


Testing Part5
Length = 7
....
000....
000.000
.000...
..000..
...000.
....000
0000...
.0000..
..0000.
...0000
00000..
.00000.
..00000
000000.
.000000
0000000


Length = 10
.....
OOO.....
OOO.OOO...
OOO..OOO..
OOO...OOO.
OOO....OOO
OOO.OOOO..
OOO..OOOO.
OOO...OOOO
OOO.OOOOO.
OOO..OOOOO
OOO.OOOOOO
.OOO.....
.OOO.OOO..
.OOO..OOO.
.OOO...OOO
.OOO.OOOO.
.OOO..OOOO
.OOO.OOOOO
..OOO.....
..OOO.OOO.
..OOO..OOO
..OOO.OOOO
...OOO....
...OOO.OOO
....OOO...
.....OOO..
......OOO.
.......OOO
OOOO.....
OOOO.OOO..
OOOO..OOO.
OOOO...OOO
OOOO.OOOO.
OOOO..OOOO
OOOO.OOOOO
.OOOO.....
.OOOO.OOO.
.OOOO..OOO
.OOOO.OOOO
..OOOO....
..OOOO.OOO
...OOOO...
....OOOO..
.....OOOO.
......OOOO
OOOOO.....
OOOOO.OOO.
OOOOO..OOO
OOOOO.OOOO
.OOOOO....
.OOOOO.OOO
..OOOOO...
...OOOOO..
....OOOOO.
.....OOOOO
OOOOOO....
OOOOOO.OOO
.OOOOOO...
..OOOOOO..
...OOOOOO.
....OOOOOO
OOOOOOO...
.OOOOOOO..
..OOOOOOO.
...OOOOOOO
OOOOOOOO..
.OOOOOOOO.
..OOOOOOOO
OOOOOOOOO.
.OOOOOOOOO
OOOOOOOOOO

6 – Calculate Time complexity

Time Complexity for Question 1

```java
public static int part1(String longStr, String shortStr, int occurence, int search, int count)
{
    if(longStr.length() == 0) return -1;
    if(shortStr.length() == 0) return -1;
    if(occurence <= 0) return -1;

    search = longStr.indexOf(shortStr, search);

    if(search >= 0) count++;
    else return -1;
    if(count == occurence) return search;
    if(search + shortStr.length() == longStr.length()) return -1;

    return part1(longStr, shortStr, occurence, ++search, count);
}
```

Best case = O(n). If the first occurrence asked and it is in the first index

Worst Case = O(n^2). If occurrence is more than the actual repetition

T(n) = O(n^3)

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

T(n) = T(n-1) + O(n^2)

T(n) = T(n-1) + O(n-1^2) + O(n^2)

T(n) = T(n-1) + O(n-2^2) + O(n-1^2) + O(n^2)

*

*

T(N) = N * (N-1)*(2N-1)/6

Time Complexity for Question 2

```java
public static int part2(int[] arr, int num1, int num2 ,int min, int max)
{
    if(min == max)
    {
        if(min >= arr.length) return 0;
        if(arr[min] >= num1 && arr[max] <= num2) return 1;
        else return 0;
    }
    else if(max > min)
    {
        int mid = min + (max - min) / 2;
        if(arr[mid] >= num1 && arr[mid] <= num2)
        {
            return 1 + part2(arr, num1, num2, mid + 1, max) + part2(arr, num1, num2, min, mid - 1);
        }
        else if(arr[mid] > num2) return part2(arr, num1, num2, min, mid - 1);
        else if(arr[mid] < num1) return part2(arr, num1, num2, mid + 1, max);
    }
    return 0;
}

```

Best case = O(logn). If there is no element between bounds

Worst Case = O(n). If all elements are between bounds

T(n) = O(n)

T(n) = 1 n = 1

T(n) = 2T(n/2)+1 n > 1

for n = 1 ➔T(n) = 1.

PROOF BY INDUCTION METHOD

Suppose n = 2^k is true T(2^k) = 2^k

For n = (2^k+1) T(2^k+1) = 2^k+1 must be true

T(2^k+1) = 2T((2^k+1) /2)+1

T(2^k+1) = 2T(2^k) == 2^k+1 means it is true.

Time Complexity for Question 3

```java
public static void part3(int[] arr, int num, int remain, int start1, int start2, ArrayList<Integer> sub)
{
    if(arr.length == 0) return;
    if(start2 == arr.length) return;

    if(start1 == arr.length)
    {
        sub.clear();
        start2++;
        start1 = start2;
        remain = num;
    }
    else
    {
        if(arr[start1] == remain)
        {

            sub.add(arr[start1]);
            System.out.print("Total is " + num + " = ");
            System.out.println(sub);
            if(start1 < arr.length - 1 && 0 - arr[start1+1] >=0) part3(arr, num, remain- arr[start1], ++start1, start2, sub);
            sub.clear();
            start2++;
            start1 = start2;
            remain = num;
        }
        else if(arr[start1] < remain)
        {
            remain = remain - arr[start1];
            sub.add(arr[start1]);
            start1++;
        }
        else if(arr[start1] > remain)
        {
            sub.clear();
            start2++;
            start1 = start2;
            remain = num;
        }
    }
    part3(arr, num, remain, start1, start2, sub);
}

```

Best case = O(2^n). At all conditions function iterates over all elements.

Worst Case = O(2^n)

T(n) = Tetha(2^n)

T(n) = 1 n = 1

T(n) = 2T(n -1)+k n > 1 = 2(2T(n -2)+k)+k n > 1 ....

= 2^r *T(n-r) + k(2^r - 1) for r = n

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

Time Complexity for Question 4

```
# 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)
```

This function multiplies 2 long integers by using Karatsuba fast multiplication Algorithm.

If Either of the numbers less than 2 digits it returns their multiplication.

Divides the given numbers to 2 different parts.

First part is the division from 2 ^ half of the max digit

Second part is modulo by 2^half of max digit

Then recursively calls 3 times .

After each recursion finishes returns the values calculated by Karatsuba formula with return statement.

T(n) = Tetha(n^1.58)

At each recursive call instructions divided by 2 == T(n/2).

Return statement is called n time O(1) = θ(N).

$T(N) = 3T(N/2) + \theta(N)$. $N>=2$

$T(N) = 1$ $N<2$

$T(N) = aT(N/b) + f(n)$.

$f(N) = O(N^k*((logN))^p)$

$f(N) = O(N)$.

İf the $log_b(a) > k$ ➔ $T(N) = O(N^{(log_b(a))}$

$T(N) = O(N^{(log_2(3)})$

$=log_2(3) \cong 1.59$

Time Complexity for Question 5

```java
public static void part5(String str, int L, int start1, int lenx, int initialStart)
{
    if(str.length()<3) return;
    if(L == lenx)
    {
        printStr(str, initialStart, lenx);
        return;
    }
    if(start1 + lenx < L)
    {
        part5(printStr(str, start1, lenx),L, start1 + lenx + 1, 3 , start1 + lenx + 1);
        start1++;
    }
    else if(start1 + lenx == L)
    {
        printStr(str, start1, lenx);
        start1 = initialStart;
        lenx++;
    }
    else
    {
        start1 = initialStart;
        lenx++;
    }
    part5(str, L, start1, lenx, initialStart);
}

```

```java
public static String printStr(String str, int start1, int lenx)
{
    StringBuilder sb = new StringBuilder(str);
    if(start1 + lenx <= str.length())
    {
        for(int i = start1; i < start1 + lenx; i++)
        {
            sb.setCharAt(i, '0');
        }
        System.out.println(sb);
    }

    return new String(sb);
}

```

Best case = O(1).

Worst Case = O(n^3)

T(n) = O(n^3)