

GTU Department of Computer Engineering

CSE 222/505 - Spring 2022

Homework 4 Report

Merve Horuz
1801042651

Q1 Pseudocode

- 1) if occurrence number equals 0 || index number equals big string length || index number + query string length > big string then returns -1
- 2) if occurrence number equals 1 && the string in this index is equals to query string then we find and returns index number.
- 3) if occurrence number greater than 1 && the string in this index is equals to query string then calling function itself by increasing index number and decreasing occurrence number.
- 4) else statement, calling function itself by increasing index number.

Q2 Pseudocode

- 1) Calling helper function to find index of first number.
- 2) Calling helper function to find index of second number.

Helper function:

- 1) Initialize start = 0 and end = size-1
- 2) Repeat until start >= end
- 3) Middle = (start+end)/2
- 4) If arr[middle] == num then return middle
- 5) If searched element bigger than all elements in the array, then returns returns the last index of the array
- 6) If searched element is not in the array and element > arr[middle] and element < arr[middle+1], then returns middle.
- 7) If the searched element is smaller than middle of the sub array, then search recursively by replacing end of the array with mid-1.
- 8) If the searched element is bigger than middle of the sub array, then search recursively by replacing start of the array with mid+1.
- 9) If it does not find returns -1.

Q3 Pseudocode

- 1) if index < length of array
 - 2) if element in the current index < number and sum < number and temporary index < length of array
 - 3) add to sum arr[index]
 - 4) if sum equals to number
 - 5) print the sub array
 - 6) calling method itself recursively by increasing temporary index by 1.
 - 7) calling method itself recursively by increasing index, setting sum to 0 and temporary index to index.

Q4

The output of the given recursive function is multiplication of given two integer numbers.

Algorithm works that:

---integer1 = 37, integer2 = 46;

- 1) max(number_of_digits(integer1), number_of_digits(integer2))
n = 2
- 2) half = int(n/2)
half = 1
- 3) int1 = integer1 / 10^{half}, int2 = integer1 % 10^{half}
int3 = integer2 / 10^{half}, int4 = integer2 % 10^{half}
int1 = 3, int2 = 7, int3 = 4, int4 = 6
- 4) sub0 = foo (int2, int4) → return integer1 * integer2 = 6*7 = 42
- 5) sub1 = foo ((int2 + int1), (int4 + int3))
---integer1 = 10, integer2 = 10;
 - 1) n = 2, half = 1, int1 = 1, int2 = 0, int3 = 1, int4 = 0
 - 2) sub0 = 0, sub1 = 1, sub2 = 1
 - 3) (sub2*10^(2*half))+((sub1-sub2-sub0)*10^(half))+(sub0)
= 100

$$6) \text{ sub2} = \text{int1} * \text{int3} = 12$$

$$7) (\text{sub2} * 10^{(2 * \text{half})}) + ((\text{sub1} - \text{sub2} - \text{sub0}) * 10^{(\text{half})}) + (\text{sub0}) \\ = 1702$$

$$\text{proof : } 37 * 46 = 1702$$

Time complexity analyzes

Q1)

```
if(i == 0 || index == bigString.length() ||
index+queryString.length() > bigString.length()) → O(1)
    return -1;
```

```
if(i == 1 && bigString.substring(index,
index+queryString.length()).equals(queryString)) → O(n)
    return index;
```

```
else if(i > 1 && bigString.substring(index,
index+queryString.length()).equals(queryString)) → O(n)
    return findIthOccurrence(i-1, index+=1, queryString, → O(n)
bigString);
```

```
else
    return findIthOccurrence(i, index+=1, queryString,
bigString); → O(n)
```

- complexity of substring and equals method is O(n)
- recursive O(n)
- so worst case complexity is O(n²)
- best case complexity is O(1) if execute first if statement.

Induction method for Q1

base case : `index == 0` and `bigString.length == 0`

$O(n^2) \rightarrow O(0^2) \rightarrow 0$ constant time

inductive step: `bigString.length == k`

$O(3) = 9$ times

$O(5) = 25$ times

$O(7) = 49$ times

$O(k^2) = O(n^2)$

Induction method for Q2

base case : `index == 0` and `arr.length == 0`

$O(n) \rightarrow O(0) \rightarrow 0$ constant time

inductive step: `arr.length == k`

$O(3) = 3$ times

$O(5) = 5$ times

$O(7) = 7$ times

$O(k) = O(n)$

Q2)

```
int i, items=0;    → O(1)
if (arr == null ) return -1;    → O(1)
int index1 = helper(arr,0,arr.length,num1);    → O(logn)
int index2 = helper(arr,0,arr.length,num2);    → O(logn)

if(index2 > index1) {    → O(1)
    for (i = index1 + 1; i < index2; i++) items++;    → O(n)
    if(arr[index2] < num2) items++;    → O(1)
}
else {
    for (i = index2 + 1; i < index1; i++) items++;    → O(n)
    if(arr[index1] < num1) items++;    → O(1)
}
return items;    → O(1)
```

- Binary search algorithm $O(\log n)$
- for loop takes $O(n)$ time
- so total complexity time is $O(n)$

Q3)

```
if(index < arr.length) {    → O(1)
```

```
    if (arr[index] < num && sum < num && index_temp <
arr.length) {    → O(1)
```

```
        sum += arr[index_temp];    → O(1)
```

```
    if (sum == num) {    → O(1)
```

```
        System.out.print("[");    → O(1)
```

```
        for (int i = index_temp ; sum > 0 && i>-1; i--) {    → O(n)
```

```
            System.out.print(arr[i]);    → O(1)
```

```
            if(sum - arr[i] > 0)    → O(1)
```

```
                System.out.print(",");    → O(1)
```

```
                sum -= arr[i];    → O(1)
```

```
            }
```

```
        System.out.println("]");    → O(1)
```

```
    }
```

```
    isSumOfElementsEqualToNum(arr, num, index, sum,
index_temp += 1);    → O(n)
```

```
    }
```

```
    else
```

```
        isSumOfElementsEqualToNum(arr, num, index += 1, sum =
0, index_temp = index);    → O(n)
```

```
    }
```

- for loop O(n)

- two recursive call O(n)

- so total complexity time is $O(n^3)$ in worst case.

Q4)

if (integer1 < 10) or (integer2 < 10) → O(1)

return integer1 * integer2 → O(1)

n = max(number_of_digits(integer1), number_of_digits(integer2))

→ O(n)

half = int(n/2) → O(1)

int1, int2 = split_integer (integer1, half) → O(1)

int3, int4 = split_integer (integer2, half) → O(1)

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)

→ O(1)

Time complexity is $O(3^n)$, or exponential, since each function call calls itself three times unless it has been recursed n times.

Problem solution approach

For q1, started with checking error conditions. If error exist, i.e ith occurrence is zero, then each recursive call increase index by one and checks query string exist in big string with substring() method. If exist in the big string decreases i variable by one, if I equals to 1 and checks query string exist in big string with substring() then returns index number.

For q2, called the recursive method for find the index of numbers. Recursive method uses binary search algorithm. In method, if the number exist in the array return the index; if the number does not exist in the array then algorithm search for left bound and right bound then if it finds the bounds returns left bound; if the number larger than all of the elements in the array then returns last index number of the array. Then my caller method calculates number of items between these two index and returns it.

For q3, by starting with first index, elements of array added to sum. If sum is equal to number than printed the arr[temp_index] by decreasing temp_index and sum. Shifted index one by one to right and this algorithm applied recursively.

For q4, given two integer to method and applied each step to these two integer. Then be sensible of that this algorithm is a MULTIPLICATION. Sir I would like to highlighted that I had a dream about this algorithm. In my dream, it was the algorithm of building the pyramids :))

For q5, each index set as cursor. In each index, all combinations of a certain block length are suppressed, and then the number of blocks is increased by one, and the same algorithm is applied recursively.

Q6 is not exactly completed.

System requirements

Functional requirements:

1) public int findIthOccurrence(int i, int index, String queryString, String bigString) : This method takes ith occurrence, I must be greater than zero; queryString and bigString different from null.

2) public int findItems(int []arr,int num1,int num2) : arr different from null. It takes two positive numbers to find items between these two numbers.

3) public void isSumOfElementsEqualToNum(int [] arr, int num, int index, int sum, int index_temp) : It takes array that is not null, num that equal to sum, index, sum, index_temp that starts from zero.

5) public void possibleColoredBlocks(int [] emptyArr, int index, int blockLength, int blockNum) : It takes array with filled with zero, index starts with zero, blockLength starts with 3, blockNum starts with 0.

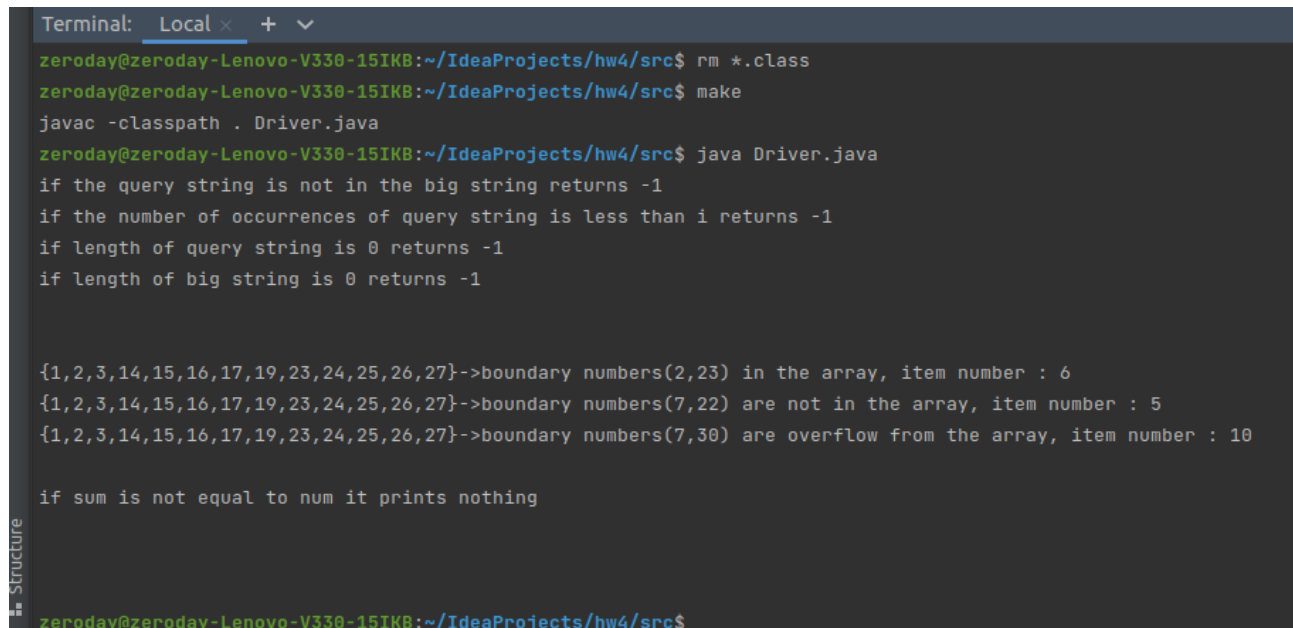
Non-functional requirements:

openjdk 17.0.1 2021-10-19

OpenJDK Runtime Environment (build 17.0.1+12-Ubuntu-120.04)

OpenJDK 64-Bit Server VM (build 17.0.1+12-Ubuntu-120.04, mixed mode, sharing)

Test cases



```
Terminal: Local x + v
zeroday@zeroday-Lenovo-V330-15IKB:~/IdeaProjects/hw4/src$ rm *.class
zeroday@zeroday-Lenovo-V330-15IKB:~/IdeaProjects/hw4/src$ make
javac -classpath . Driver.java
zeroday@zeroday-Lenovo-V330-15IKB:~/IdeaProjects/hw4/src$ java Driver.java
if the query string is not in the big string returns -1
if the number of occurrences of query string is less than i returns -1
if length of query string is 0 returns -1
if length of big string is 0 returns -1

{1,2,3,14,15,16,17,19,23,24,25,26,27}->boundary numbers(2,23) in the array, item number : 6
{1,2,3,14,15,16,17,19,23,24,25,26,27}->boundary numbers(7,22) are not in the array, item number : 5
{1,2,3,14,15,16,17,19,23,24,25,26,27}->boundary numbers(7,30) are overflow from the array, item number : 10

if sum is not equal to num it prints nothing
zeroday@zeroday-Lenovo-V330-15IKB:~/IdeaProjects/hw4/src$
```

Running commands

-make

-java Driver.java

if you would like to run one more time remove .class file

-rm *.class

Running results

```
zeroday@zeroday-Lenovo-V330-15IKB:~/IdeaProjects/hw4/src$ make
javac -classpath . Driver.java
zeroday@zeroday-Lenovo-V330-15IKB:~/IdeaProjects/hw4/src$ java Driver.java
query string: batuhan
big string: dgdshmervejdfmervedjflsmervedkjfsfldkfayse

if the query string is not in the big string returns -1

query string: merve
big string: dgdshmervejdfmervedjflsmervedkjfsfldkfayse

if the number of occurrences of query string is less than i returns -1

if length of query string is 0 returns -1

if length of big string is 0 returns -1

query string: batuhan
big string: dgdshmervejdfmervedbatuhanjflsmervedkjfsfldkfayse

index in normal condition 19

{1,2,3,14,15,16,17,19,23,24,25,26,27}->boundary numbers(2,23) in the array, item number : 6
{1,2,3,14,15,16,17,19,23,24,25,26,27}->boundary numbers(7,22) are not in the array, item number : 5
{1,2,3,14,15,16,17,19,23,24,25,26,27}->boundary numbers(7,30) are overflow from the array, item number : 10
sub array -> {1,2,3,14,7,4,16,5,3,5,22,26,27}

if sum is not equal to num it prints nothing

contiguous sub array under normal condition
[4,7,14]
[5,16,4]

multiplication : 21700

possible colored blocks
[1,1,1,0,0,0,0]
[1,1,1,0,1,1,1]
```

```

if length of big string is 0 returns -1

query string: batuhan
big string: dgdshmervejdfmervedbatuhanjflsmervedkjfsfldkfayse

index in normal condition 19

{1,2,3,14,15,16,17,19,23,24,25,26,27}->boundary numbers(2,23) in the array, item number : 6
{1,2,3,14,15,16,17,19,23,24,25,26,27}->boundary numbers(7,22) are not in the array, item number : 5
{1,2,3,14,15,16,17,19,23,24,25,26,27}->boundary numbers(7,30) are overflow from the array, item number : 10
sub array -> {1,2,3,14,7,4,16,5,3,5,22,26,27}

if sum is not equal to num it prints nothing

contiguous sub array under normal condition
[4,7,14]
[5,16,4]

multiplication : 21700

possible colored blocks
[1,1,1,0,0,0,0]
[1,1,1,0,1,1,1]
[0,1,1,1,0,0,0]
[0,0,1,1,1,0,0]
[0,0,0,1,1,1,0]
[0,0,0,0,1,1,1]
[1,1,1,1,0,0,0]
[0,1,1,1,1,0,0]
[0,0,1,1,1,1,0]
[0,0,0,1,1,1,1]
[1,1,1,1,1,0,0]
[0,1,1,1,1,1,0]
[0,0,1,1,1,1,1]
[1,1,1,1,1,1,0]
[0,1,1,1,1,1,1]
[1,1,1,1,1,1,1]
zeroday@zeroday-Lenovo-V330-15IKB:~/IdeaProjects/hw4/src$

```

Class diagram

SnakeCombinationsInMatrix	
f 🔒	allCombinations ArrayList<ArrayList<Integer>>
m 📁	SnakeCombinationsInMatrix()
m 📁	printMatrix(ArrayList<Integer>, int) void
m 📁	generateRandomIndex(int, ArrayList<Integer>) int
m 📁	generateRandomIndex(int, int) int
m 📁	snakeCombinationsNxNMatrix(ArrayList<Integer>, int, int, int, int) void

Q4	
m 📁	Q4()
m 🔒	split_integer(int, int) int
m 📁	foo(int, int) int
m 🔒	split_integer2(int, int) int
m 🔒	number_of_digits(int) int

ColoredBlocks	
m 📁	ColoredBlocks()
m 📁	possibleColoredBlocks(int[], int, int, int) void
m 🔒	helper(int[], int, int, int) void

FindItemsBetweenNumbers	
m 📁	FindItemsBetweenNumbers()
m 🔒	helper(int[], int, int, int) int
m 📁	findItems(int[], int, int) int

ContiguousSubarrays	
m 📁	ContiguousSubarrays()
m 📁	isSumOfElementsEqualToNum(int[], int, int, int, int) void

lthOccurrenceSubstring	
m 📁	lthOccurrenceSubstring()
m 📁	findlthOccurrence(int, int, String, String) int

Driver	
m 📁	Driver()
m 📁	main(String[]) void