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

Muhammed Sinan Pehlivanoğlu
1901042664

1. SYSTEM REQUIREMENTS

Operating System need to jdk and jre for start the program.
Stack capacity should not be less.
You need to enough space to store data's of program according to the how many you have objects.

2.PROBLEM SOLUTION APPROACH

- *For 1.Problem*
  Best case scenerio is the if there is no substring in the main string return -1 and set the flag to the 1 that represent the error flag.

  Base case scenario is that if the there are substring as many as desired number then return the start index of the substring.

  Until the reach base case level, if the substring is found, then the new main string start from the end of the substring index and again start to find the substring in the new Main String.

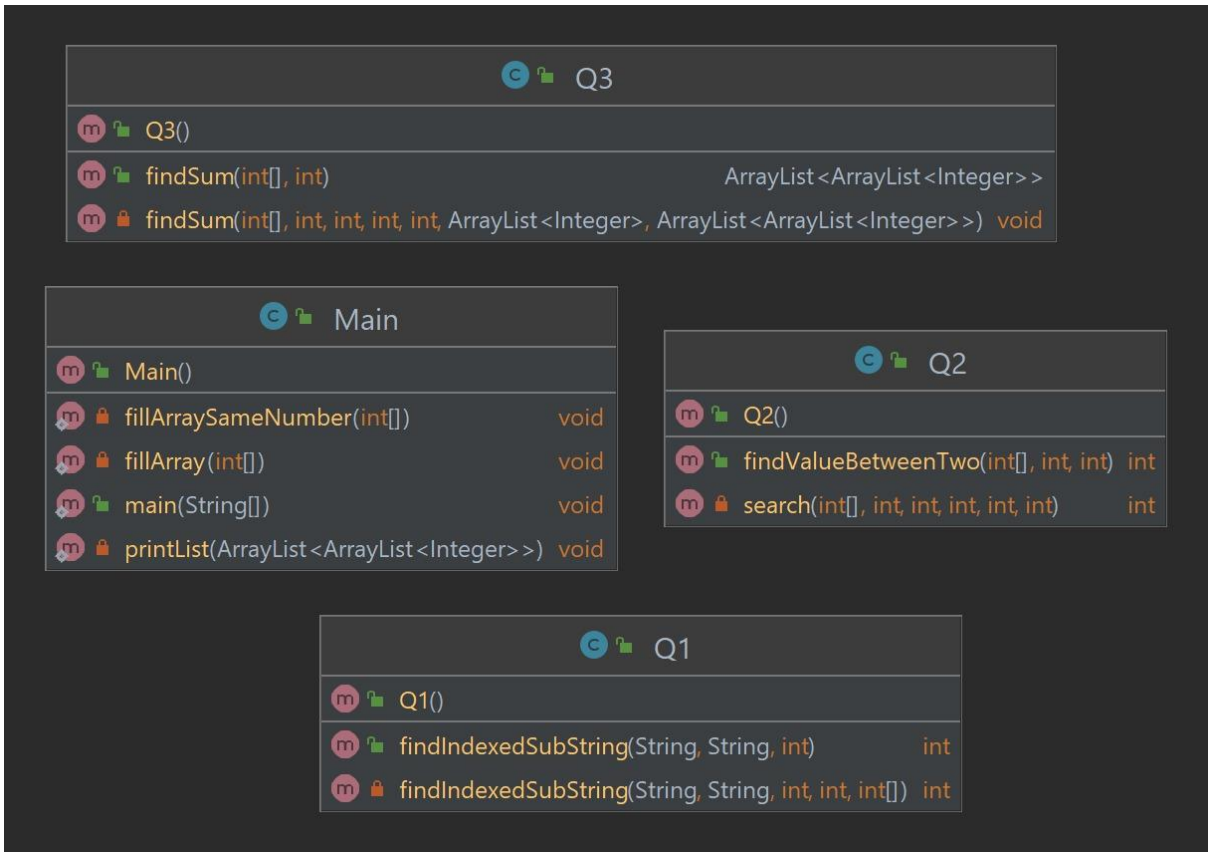
- *For 2.Problem The dividing method is used. The array is divided into the subarray until the only one element is left in the subarray.*

  Base case is that if the left index is larger than right index then return number of the element between desired numbers.

  İf the left and right size is equal to each other, this means that there are only 1 element is left in the divided subarray then compare this element with the desired numbers. If the condition is true, total number is increased.
- *For 3.Problem*
  Base case is that if the index is larger than the size of the array length -1, then return -1.

  There are two index holder. First index holder holds the start index of the array.
  Second index holder holds the contiguous elements index. It is increased by 1 during the query.
  İf the sum of the contiguous elements after the start index is less than or equal to desired sum, the contiguous elements are summed up. If the sum is equal to desired sum, the elements are added the list. Then start index is increased by 1 and second index holder is equalized the start index. The query is continued until the second index holder reach size of the array (Base case).

3.CLASS DIAGRAMS



4.TEST CASES

- Search a given string in another given bigger string. The function should return the index of the i_{th} occurrence of the query string .

- Find the number of items in the array between two given integer values.

- Find contiguous subarray/s that the sum of its/theirs items is equal to a given integer value.

5.RUNNING COMMANDS AND RESULTS

FIND Nth THE SUBSTRING 'S START INDEX

Q1 RESULT

```
1.PROBLEM
Main String : kabcdabcmabcdabceabc  Sub String : abcIndex : 2
RESULT 1.A  5

Main String : DATACSE222CSE222CSEDATACSE222DATA  Sub String : CSE  Index : 3
RESULT 1.B  16

Main String : DATACSE222CSE222CSEDATACSE222DATA  Sub String : CSE  Index : 7
RESULT 1.C  -1

Main String : DATACSE222CSE222CSEDATACSE222DATA  Sub String : cse  Index : 3
RESULT 1.D  -1
```

FIND THE NUMBER OF NUMBERS BETWEEN TWO NUMBER

Q2 RESULT AND ELAPSED TIME

```
2.PROBLEM
Fill arrays 100 , 1000, 10000 sorted numbers started with 1
Find number of numbers between 45 - 99 in arrays
Result : 55 Elapsed Time :16900
Result : 55 Elapsed Time :110700
Result : 55 Elapsed Time :167800
```

FİND CONTİGUOUS SUBARRAY/S THAT THE SUM OF İTS/THEİRS İTEMS İS EQUAL TO A GİVEN İNTEGER VALUE.

Q3 RESULT

ELAPSED TIMES

- **SIZE 30 = 79700 ns**
- **SIZE 60 = 146000 ns**
- **SIZE 120 = 438900 ns**

```
3.PROBLEM
Result of 30 size array filled with 1 (Find the value 10)
0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9 10
2 3 4 5 6 7 8 9 10 11
3 4 5 6 7 8 9 10 11 12
4 5 6 7 8 9 10 11 12 13
5 6 7 8 9 10 11 12 13 14
6 7 8 9 10 11 12 13 14 15
7 8 9 10 11 12 13 14 15 16
8 9 10 11 12 13 14 15 16 17
9 10 11 12 13 14 15 16 17 18
10 11 12 13 14 15 16 17 18 19
11 12 13 14 15 16 17 18 19 20
12 13 14 15 16 17 18 19 20 21
13 14 15 16 17 18 19 20 21 22
14 15 16 17 18 19 20 21 22 23
15 16 17 18 19 20 21 22 23 24
16 17 18 19 20 21 22 23 24 25
17 18 19 20 21 22 23 24 25 26
18 19 20 21 22 23 24 25 26 27
19 20 21 22 23 24 25 26 27 28
20 21 22 23 24 25 26 27 28 29
Elapsed time is 79700
```

```
Result of 60 size array filled with 1(Find the 10)
0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9 10
2 3 4 5 6 7 8 9 10 11
3 4 5 6 7 8 9 10 11 12
4 5 6 7 8 9 10 11 12 13
5 6 7 8 9 10 11 12 13 14
6 7 8 9 10 11 12 13 14 15
7 8 9 10 11 12 13 14 15 16
8 9 10 11 12 13 14 15 16 17
9 10 11 12 13 14 15 16 17 18
10 11 12 13 14 15 16 17 18 19
11 12 13 14 15 16 17 18 19 20
12 13 14 15 16 17 18 19 20 21
13 14 15 16 17 18 19 20 21 22
14 15 16 17 18 19 20 21 22 23
15 16 17 18 19 20 21 22 23 24
16 17 18 19 20 21 22 23 24 25
17 18 19 20 21 22 23 24 25 26
18 19 20 21 22 23 24 25 26 27
19 20 21 22 23 24 25 26 27 28
20 21 22 23 24 25 26 27 28 29
21 22 23 24 25 26 27 28 29 30
22 23 24 25 26 27 28 29 30 31
23 24 25 26 27 28 29 30 31 32
24 25 26 27 28 29 30 31 32 33
25 26 27 28 29 30 31 32 33 34
26 27 28 29 30 31 32 33 34 35
27 28 29 30 31 32 33 34 35 36
28 29 30 31 32 33 34 35 36 37
29 30 31 32 33 34 35 36 37 38
30 31 32 33 34 35 36 37 38 39
31 32 33 34 35 36 37 38 39 40
32 33 34 35 36 37 38 39 40 41
33 34 35 36 37 38 39 40 41 42
34 35 36 37 38 39 40 41 42 43
35 36 37 38 39 40 41 42 43 44
36 37 38 39 40 41 42 43 44 45
37 38 39 40 41 42 43 44 45 46
38 39 40 41 42 43 44 45 46 47
39 40 41 42 43 44 45 46 47 48
40 41 42 43 44 45 46 47 48 49
41 42 43 44 45 46 47 48 49 50
42 43 44 45 46 47 48 49 50 51
43 44 45 46 47 48 49 50 51 52
44 45 46 47 48 49 50 51 52 53
45 46 47 48 49 50 51 52 53 54
46 47 48 49 50 51 52 53 54 55
47 48 49 50 51 52 53 54 55 56
48 49 50 51 52 53 54 55 56 57
49 50 51 52 53 54 55 56 57 58
50 51 52 53 54 55 56 57 58 59
Elapsed time is 146000
```

```
Result of 120 size array filled with 1 (Find the 10)
0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9 10
2 3 4 5 6 7 8 9 10 11
3 4 5 6 7 8 9 10 11 12
4 5 6 7 8 9 10 11 12 13
5 6 7 8 9 10 11 12 13 14
6 7 8 9 10 11 12 13 14 15
7 8 9 10 11 12 13 14 15 16
8 9 10 11 12 13 14 15 16 17
9 10 11 12 13 14 15 16 17 18
10 11 12 13 14 15 16 17 18 19
11 12 13 14 15 16 17 18 19 20
12 13 14 15 16 17 18 19 20 21
13 14 15 16 17 18 19 20 21 22
14 15 16 17 18 19 20 21 22 23
15 16 17 18 19 20 21 22 23 24
16 17 18 19 20 21 22 23 24 25
17 18 19 20 21 22 23 24 25 26
18 19 20 21 22 23 24 25 26 27
19 20 21 22 23 24 25 26 27 28
20 21 22 23 24 25 26 27 28 29
21 22 23 24 25 26 27 28 29 30
22 23 24 25 26 27 28 29 30 31
23 24 25 26 27 28 29 30 31 32
24 25 26 27 28 29 30 31 32 33
25 26 27 28 29 30 31 32 33 34
26 27 28 29 30 31 32 33 34 35
27 28 29 30 31 32 33 34 35 36
28 29 30 31 32 33 34 35 36 37
```

```
29 30 31 32 33 34 35 36 37 38
30 31 32 33 34 35 36 37 38 39
31 32 33 34 35 36 37 38 39 40
32 33 34 35 36 37 38 39 40 41
33 34 35 36 37 38 39 40 41 42
34 35 36 37 38 39 40 41 42 43
35 36 37 38 39 40 41 42 43 44
36 37 38 39 40 41 42 43 44 45
37 38 39 40 41 42 43 44 45 46
38 39 40 41 42 43 44 45 46 47
39 40 41 42 43 44 45 46 47 48
40 41 42 43 44 45 46 47 48 49
41 42 43 44 45 46 47 48 49 50
42 43 44 45 46 47 48 49 50 51
43 44 45 46 47 48 49 50 51 52
44 45 46 47 48 49 50 51 52 53
45 46 47 48 49 50 51 52 53 54
46 47 48 49 50 51 52 53 54 55
47 48 49 50 51 52 53 54 55 56
48 49 50 51 52 53 54 55 56 57
49 50 51 52 53 54 55 56 57 58
50 51 52 53 54 55 56 57 58 59
51 52 53 54 55 56 57 58 59 60
52 53 54 55 56 57 58 59 60 61
53 54 55 56 57 58 59 60 61 62
54 55 56 57 58 59 60 61 62 63
55 56 57 58 59 60 61 62 63 64
56 57 58 59 60 61 62 63 64 65
57 58 59 60 61 62 63 64 65 66
58 59 60 61 62 63 64 65 66 67
```

```
 59 60 61 62 63 64 65 66 67 68
 60 61 62 63 64 65 66 67 68 69
 61 62 63 64 65 66 67 68 69 70
 62 63 64 65 66 67 68 69 70 71
 63 64 65 66 67 68 69 70 71 72
 64 65 66 67 68 69 70 71 72 73
 65 66 67 68 69 70 71 72 73 74
 66 67 68 69 70 71 72 73 74 75
 67 68 69 70 71 72 73 74 75 76
 68 69 70 71 72 73 74 75 76 77
 69 70 71 72 73 74 75 76 77 78
 70 71 72 73 74 75 76 77 78 79
 71 72 73 74 75 76 77 78 79 80
 72 73 74 75 76 77 78 79 80 81
 73 74 75 76 77 78 79 80 81 82
 74 75 76 77 78 79 80 81 82 83
 75 76 77 78 79 80 81 82 83 84
 76 77 78 79 80 81 82 83 84 85
 77 78 79 80 81 82 83 84 85 86
 78 79 80 81 82 83 84 85 86 87
 79 80 81 82 83 84 85 86 87 88
 80 81 82 83 84 85 86 87 88 89
 81 82 83 84 85 86 87 88 89 90
 82 83 84 85 86 87 88 89 90 91
 83 84 85 86 87 88 89 90 91 92
 84 85 86 87 88 89 90 91 92 93
 85 86 87 88 89 90 91 92 93 94
 86 87 88 89 90 91 92 93 94 95
 87 88 89 90 91 92 93 94 95 96
 88 89 90 91 92 93 94 95 96 97
89 90 91 92 93 94 95 96 97 98
90 91 92 93 94 95 96 97 98 99
91 92 93 94 95 96 97 98 99 100
92 93 94 95 96 97 98 99 100 101
93 94 95 96 97 98 99 100 101 102
94 95 96 97 98 99 100 101 102 103
95 96 97 98 99 100 101 102 103 104
96 97 98 99 100 101 102 103 104 105
97 98 99 100 101 102 103 104 105 106
98 99 100 101 102 103 104 105 106 107
99 100 101 102 103 104 105 106 107 108
100 101 102 103 104 105 106 107 108 109
101 102 103 104 105 106 107 108 109 110
102 103 104 105 106 107 108 109 110 111
103 104 105 106 107 108 109 110 111 112
104 105 106 107 108 109 110 111 112 113
105 106 107 108 109 110 111 112 113 114
106 107 108 109 110 111 112 113 114 115
107 108 109 110 111 112 113 114 115 116
108 109 110 111 112 113 114 115 116 117
109 110 111 112 113 114 115 116 117 118
110 111 112 113 114 115 116 117 118 119
Elapsed time is 438900
PS D:\Java\HW4\src>
```

6.TIME COMPLEXITY ANALYSIS

Q1:

```java
public class Q1 {

    public int findIndexedSubString(String mainString, String subString,int index){
        if(mainString == null || subString == null){
            throw new NullPointerException();
        }
        int[] flag = new int[1];
        return findIndexedSubString(mainString,subString, index: index-1, foundedIndex: 0,flag);
    }
```

```java
private int findIndexedSubString(String mainString, String subString,int index,int foundedIndex,int[]flag){

    int startindex = mainString.indexOf(subString);
    int endIndex = startindex + subString.length();
    if(startindex == -1){
        flag[0] = 1;
        return -1;
    }
    if(index == foundedIndex){
        return  startindex;
    }

    int result =  endIndex + findIndexedSubString(mainString.substring(endIndex), subString, index, ++foundedIndex,flag);

    if(flag[0] == 1) result =   -1;
    return result;


}
```

Base cases are if the start index == -1 and if the desired index == founded index. Tbase(N)→ O(N) . indexOf method takes O(N). Substring method takes O(N).

T(N) = T(N+1) + O(N).

SO The time complexity is T(N) = O(N^2)*M = O(N^2) . M = desired index.

Q2:

```
public class Q2 {
    public  int findValueBetweenTwo(int[]array,int start,int end){

        if(array == null){
            throw new NullPointerException();
        }
        else{
            return search(array, left: 0, right: array.length-1,start,end, number: 0);
        }
    }
```

```
private int search(int[] array,int left,int right,int start,int end,int number){
    if(left == right){
        if(array[left] >= start && array[left] <=end){
            ++number;
        }
    }
    if(left < right) {
         int mid = (right + left) / 2;
         number =search(array, left, mid, start, end,number);
         number =search(array,  left: mid + 1, right, start, end,number);
    }
    return  number;
}
```

BASE CASE IS THAT LEFT == RIGHT. → Θ(1) .

Searching begin with the left half until the number of the left half is 1. Then search right half until the number of right half's element reach 1.

Number of element is halved.

So T(N) = 2T(N/2) + Θ(1) N>1
T(N) = Θ(1) N =1

With using Tree method T(N) = N;

PROOF BY INDUCTION METHOD

- **For n = 1 , it is true. T(1) = 1.**
- **Assume that n = 2^k is true.**
  T(2^k) = 2^k .
- **So n = 2^(k+1) must be true.**

  T(2^(k+1)) = 2T(2^(k+1)/2) + 1; and 2T(2^(k + 1)/ 2) = 2T(2^k) = 2^k .

  Thus T(2^(k+1)) = 2^k+ 1. It is true.

Q3:

```java
public class Q3 {
    public ArrayList<ArrayList<Integer>> findSum(int [] array, int value){
        if(array == null){
            throw new NullPointerException();
        }
        ArrayList<Integer> list = new ArrayList();
        ArrayList<ArrayList<Integer>> totalList = new ArrayList<ArrayList<Integer>>();
        findSum(array,value, i: 0, j: 0, sum: 0,list,totalList);
        return totalList;
    }
```

```java
private void findSum(int[] array , int value, int i,int j, int sum, ArrayList<Integer> list,ArrayList<ArrayList<Integer>> totalList){
    if(j> array.length-1){
        return;
    }
    sum += array[j];
    if(sum > value){
        list.clear();
        j = i++;
        sum = 0;
    }
    else{
        list.add(j);
        if((j+1) < array.length-1){
            if(array[j+1] == 0){
                list.add(j+1);
            }
        }
    }
    if(sum == value){
        totalList.add((ArrayList<Integer>) list.clone());
        list.clear();
        j = i++;
        sum = 0;
    }
    findSum(array,value,i,++j,sum,list,totalList);
}
```

Base case are if the index is greater than array length -1 .

Tbase(N) = Θ(1) .

Clone method takes O(N).

Clean method takes O(N) .

Add method takes Θ (1).

If the all elements are greater than desired sum . Best case is Θ (1) + T(N+1) + Θ(1) = Θ(N).

If the all element's is less than desired sum . Worst case is O(N) + T(N+1) + Θ(1)= Θ(N^2).

So total time complexity is O(N^2).

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)
```

There are 3 recursive call and number of digits is halved each recursive call.

Return statment is θ(N).

SO T(N) = 3T(N/2) + θ(N) . N>=2

T(N) = 1 N <2

The master therom say that T(N) = aT(N/b) + f(n). İf a>= and b>1

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

$f(N) = O(N)$. *So* $k = 1$ *and* $p = 0$.

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

THUS T(N) $= O(N^{\log_2 3})$ *and* $\log_2 3 \cong 1.59$

PROOF BY INDUCTION

- **For N = 1 , T(N) = 1. IT is true.**
- **Suppose that N =** 2^k **. It is true. T(**2^k**) =** $2^{k^{1.59}}$
- **For N =** 2^{k+1} **must be true.**

T(2^{k+1}**) = 3T(**$2^{k+1}/2$**) +** 2^{k+1}
3T($2^{k+1}/2$**) = 3T(**2^k**)**
SO T(2^{k+1}**) = 3T(**2^k**) +** 2^{k+1}

T(2^k**) =** $2^{k^{1.59}}$

T(2^{k+1}**) =** $3*2^{k^{1.59}} + 2^{k+1} <= 2^{(k+1)*1.59}$

For k = 2 $3*2^{k^{1.59}} + 2^{k+1} \cong 27$
$2^{(k+1)*1.59} \cong 35$

27<= 35 . So it is true.