Homework 4 Report

Ömer Kaan Uslu

1801042642

Q1

Problem Solution Approach(PSA):

Corresponding function holds CurrI to hold repeating number of query in string.

If currI reaches i which is entered by user, function returns index at the moment. If it is not reached, returns -1.

```java
public class Main{
    public static int searchString(String query,String field,int index,int i,int currI){
        System.out.println(query+"    "+field);
        if(query.length()>field.length()){
            return -1;
        }
        if(field.substring(0,query.length()).equals(query))
            currI+=1;
        if(i==currI)
            return index;
        return searchString(query,field.substring(1,field.length()),index+1,i,currI);
    }
    public static void main(String [] args){
        System.out.println(searchString("a","aaaaa",0,3,0));

    }
}
```

```
C:\Users\90555\Desktop\hw4\Q1>javac Main.java

C:\Users\90555\Desktop\hw4\Q1>java Main.java
a    aaaaa
a    aaaa
a    aaa
2

C:\Users\90555\Desktop\hw4\Q1>
```

```java
public static void main(String [] args){
    System.out.println(searchString("abc","abcdabcd",0,3,0));

}
```

```
C:\Users\90555\
abc    abcdabcd
abc    bcdabcd
abc    cdabcd
abc    dabcd
abc    abcd
abc    bcd
abc    cd
-1
```

Q2

Problem Solution Approach(PSA):

Binary search algorithm finds lower bound firstly, upper bound secondly. And returns the difference.

```
public static int findNum(int[] arr,int first, int last,int targetLower,int targetUpper,boolean check){
    if(check){
        if(first>last)
            return -1;
        else{
            int middle=(first+last)/2;
                    if(arr[middle]==targetLower){
                        return findNum(arr,0,arr.length-1,targetLower,targetUpper,false)-middle;
                    }
                    else if(arr[middle]>targetLower){
                        return findNum(arr,first,middle-1,targetLower,targetUpper,check);
                    }
                    else if(arr[middle]<targetLower){
                        return findNum(arr,middle+1,last,targetLower,targetUpper,check);
                    }
            }
    }
    else{
        if(first>last)
            return -1;
        else{
            int middle=(first+last)/2;
                    if(arr[middle]==targetUpper){
                        return middle;
                    }
                    else if(arr[middle]>targetUpper){
                        return findNum(arr,first,middle-1,targetLower,targetUpper,check);
                    }
                    else if(arr[middle]<targetUpper){
                        return findNum(arr,middle+1,last,targetLower,targetUpper,check);
                    }
            }
    }
    return 0;
}
```

```
C:\Users\90555\Desktop\hw4\Q2>javac Main.java

C:\Users\90555\Desktop\hw4\Q2>java Main.java
4

C:\Users\90555\Desktop\hw4\Q2>
```

Q3

Problem Solution Approach(PSA):

Algorithm basically hold i and j which are indexes, and increments j for every occur.

If j reaches length, i increments and j initializes by i. In every occur, function checks if the target equals sum. If equal, it prints the corresponding elements.

```
public class Main{
    public static void findNum(int[] arr,int i, int j,int target){
        if(i==arr.length)
            return;
        else if(j==arr.length)
            findNum(arr,i+1,i+1,target);
        else{
            int sum=0;
            for(int k=i;k<=j;k++){
                sum+=arr[k];
            }
            if(sum==target){
                for(int k=i;k<=j;k++){
                    System.out.print(arr[k]+"  ");
                }
                System.out.println();
            }
            findNum(arr,i,j+1,target);
        }
    }
    public static void main(String [] args){
        int[] arr={0,1,2,3,4,1,5,0,6,-1};

        findNum(arr,0,0,5);

    }
}
```

```
C:\Users\90555\Desktop\hw4\Q3>java Main.java
2  3
4  1
5
5  0
0  6  -1
6  -1

C:\Users\90555\Desktop\hw4\Q3>
```

Q5

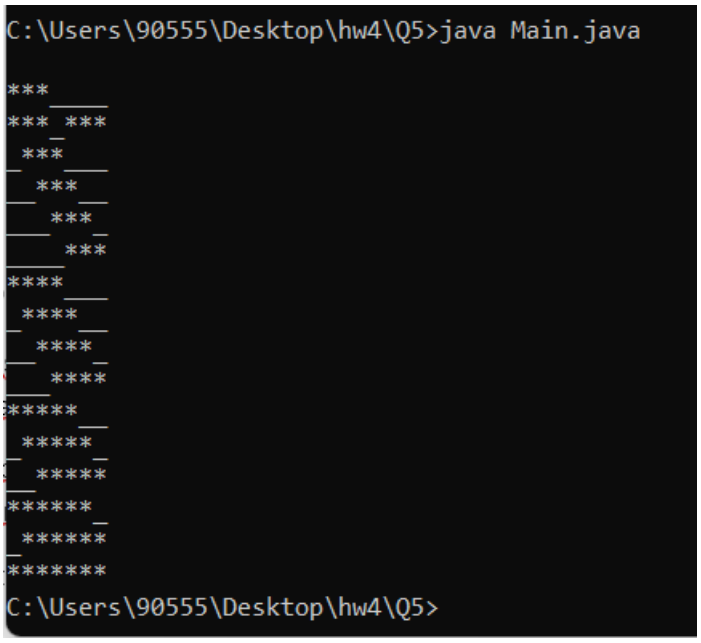
Problem Solution Approach(PSA):

Function starts with length which is determined by 3. Let's say array has length with 7. First 3 element of array is filled. And it prints. There is breakdown point.

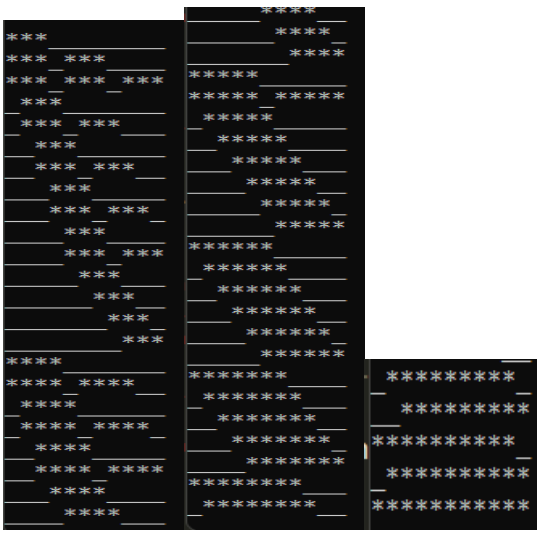
If array has enough space to fill rest of the array with 3 length blocks, it holds old array and rerun function with initilazion i of value i+length+1. If array has length with 15, it continues

Because there is a lot of space in order to continue the track.

Array with length 7:



Array with length 11:



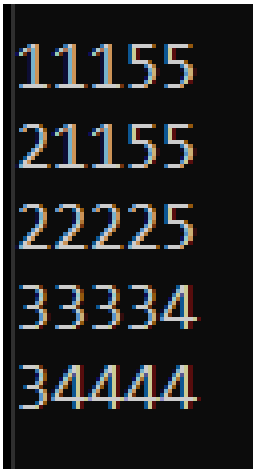
Q6

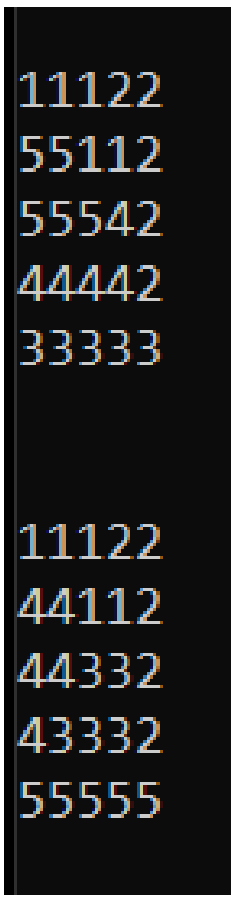
Problem Solution Approach(PSA):

Function adds corresponding element with number at the moment. And make four calls for every direction. If occuring this number in array reaches user input length of snake, it increments filling number by one. It means the snake is changed and other snake has to find a path to fill his body in array. If this number reaches upper limit, It means array is completed and checks if any 0 in array occurs. If does not, it prints the array.

Results may have same, it does not mean those are wrong. It means the head and tail of the snakes may have changed. It may cause printing same array with same paths.

Some of results(I can't put all of them because there is billlion of results):





Q1) Time complexity of substring method = $O(n)$

$T_{best} = O(1)$

$T_{worst} = O(n)$

$T(n) = T(n-1) + n$ $\qquad T(n-1) = T(n-2) + 1$

$T(n) = T(n-2) + n + n$

$T(n) = T(n-k) + \underbrace{n+n+n}_{k \text{ times}}$

$T(n) = 0 + n.n = n^2 = \boxed{O(n^2)}$

Q2) Binary search algorithm has $O(\log n)$ time complexity.

In function, there is 2 different binary search algorithm one after other.

$2.\log n = \boxed{O(\log n)}$

$T(n) = T(n/2) + 1$ $\qquad T(n/2) = T(n/4) + 1$

$T(n) = T(n/4) + \underbrace{1+1}_{\log n \text{ times}}$

$T(n) = \log n$

Q3) $T(n) = T(n-1) + n$

$T(n) = n^2$ $\qquad \boxed{O(n^2)}$

Scanned with CamScanner

Q4) Input = 10,10 10' Basic operation → 3 times

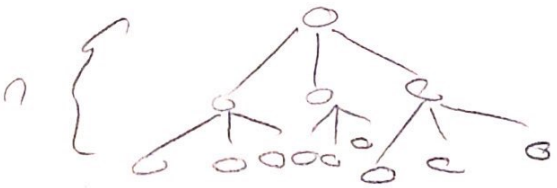
Input = 100,100 Basic operation → 15 times

Input = 1000,1000 " " → 31 times

Program takes partition and rest of dividing operation over 2^ half. Split integer method does that.

And it recurses over by 3 subtrees it.

It divides all nodes in recursion tree by 3



TimeComplexity = 3^n

= $O(3^n)$

Scanned with CamScanner