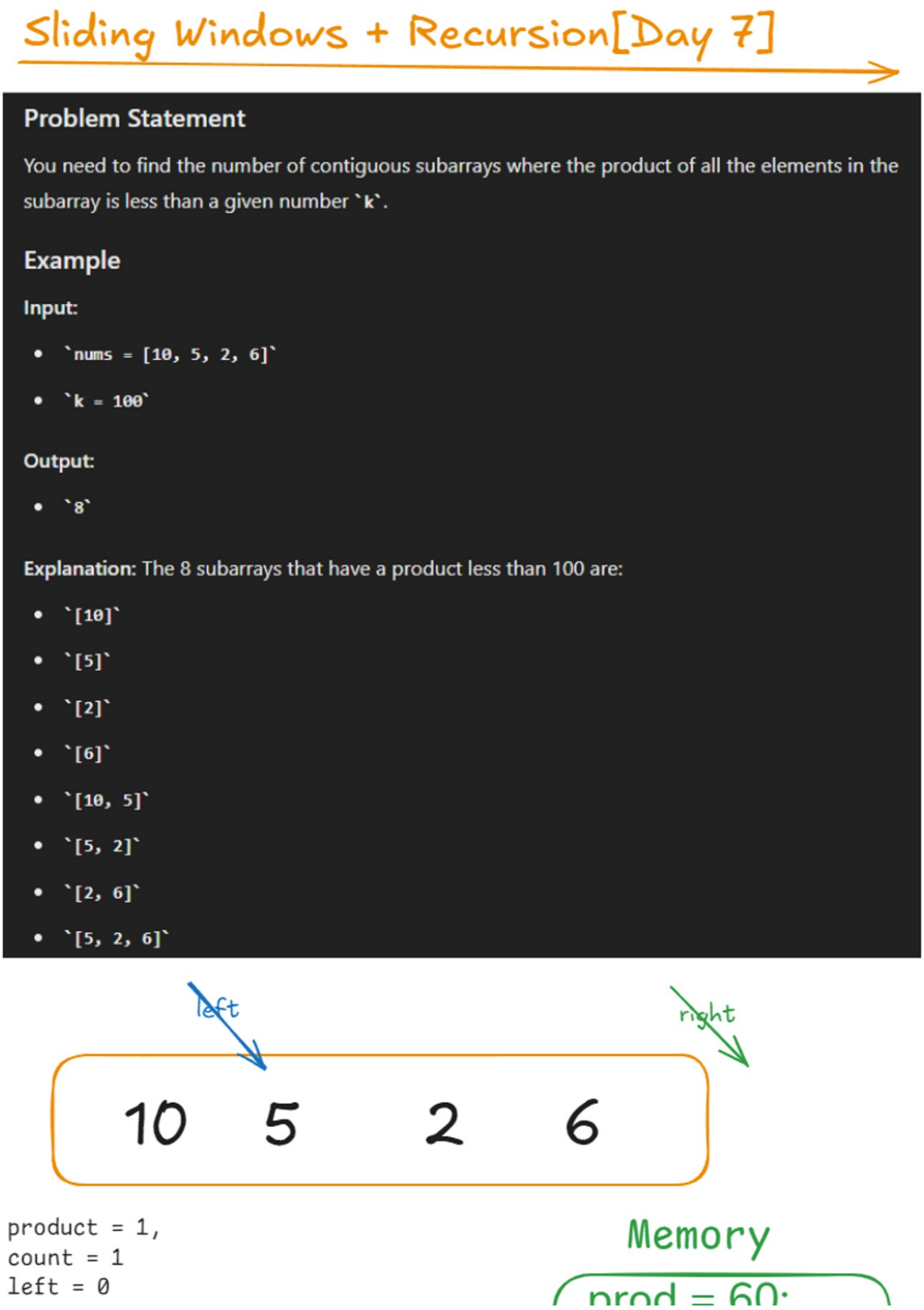
Sliding Windows + Recursion [Day 7]

09 September 2024 17:19



```
count = 1
left = 0

Run a loop - right 0 to n

    prod *= arr[right];
    while( prod >=k):
        //keep dividing it by arr[left];
        prod / = arr[left]
        left++;
    )
    count += right - left +1;
```

Memory

prod = 60;
left = 1;
right =4;
count = 5;
K = 100;

right = 0 -> [10]
count = 1 + (1-0) + 1 = 3; right = 1 -> [10] , [10,5] [5]
count = 3 + (2-1) + 1 = 5 right = 2 -> [10] , [10,5] [5] , [5,2] [2]
count = 5 + (3-1) + 1 = 8 right = 3 -> [10] , [10,5] [5] , [5,2] [2],[5,2,6] [2,6] [6]

```
product = 1,
count = 1
left = 0

Run a loop - right 0 to n

    prod *= arr[right];
    while( prod >=k):
        //keep dividing it by arr[left];
        prod / = arr[left]
        left++;
    )
    count += right - left +1;
```

Memory

prod = 8;
left = 4;
right =5;
count = 9;
K = 50;

right = 0 [5] : [5]
right [1] - [7] : count = 1 + (1-0) +1
3 : [5][7][5,7]
right[2] - [3] : count = 3 + (1) + 1 = 5
[5][7][5,7][7,3][3]
right[3] - [10] : count = 5+ (1) + 1 = 7
[5][7][5,7][7,3][3][3,10][10]
right[4] - [4] : count = 7+ (1) + 1 = 9
[5][7][5,7][7,3][3][3,10][10][10.4][4]
right[5] - [2] : count = 9+ (1) + 1 =11
[5][7][5,7][7,3][3][3,10][10][10.4][4]
[4,2] [2]

```
import java.util.Scanner;
public class SubarrayProductLessThanK {

    public static int numSubarrayProductLessThanK(int[] nums , int k){
        if(k <= 1) return 0;
        int product = 1;
```

```
    public static int numSubarrayProductLessThanK(int[] nums , int k){
        if(k <= 1) return 0;
        int product = 1;
        int count = 0;
        int left = 0;
        for(int right = 0 ; right < arr.length; right++){
            product *= arr[right];
            while(product >=k){
                product /= arr[left];
                left++;
            }
            count += right - left + 1;
        }
        return count;
    }
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();

        int[] arr = new int[n];
        System.out.println("Enter the elements of an array");
        for(int i =0; i<n; i++){
            arr[i] = scn.nextInt();
        }

        System.out.println("Enter the Value of K :");
        int k = scn.nextInt();

        int result = numSubarrayProductLessThanK(arr , k);
        System.out.println(result);
    }
}
```

Boats To Save People.

Problem Statement

You are given an array `people` where `people[i]` is the weight of the `i`-th person, and an integer `limit` that represents the maximum weight that a boat can carry. Each boat can carry at most two people at the same time, provided the sum of their weights is less than or equal to `limit`.

Your task is to find the minimum number of boats required to save everyone.

Example

Input:

Input:

- `people = [3, 5, 3, 4]`
- `limit = 5`

Output:

- `4`

Input:

- `people = [1, 2, 2, 3]`
- `limit = 3`

Output:

- `3`

Input:

- `people = [5, 1, 4, 2]`
- `limit = 6`

Output:

- `2`

Input:

- `people = [5, 3, 4, 4]`
- `limit = 8`

Output:

- `2`

Input:

- `people = [2, 2, 2, 2]`
- `limit = 3`

Output:

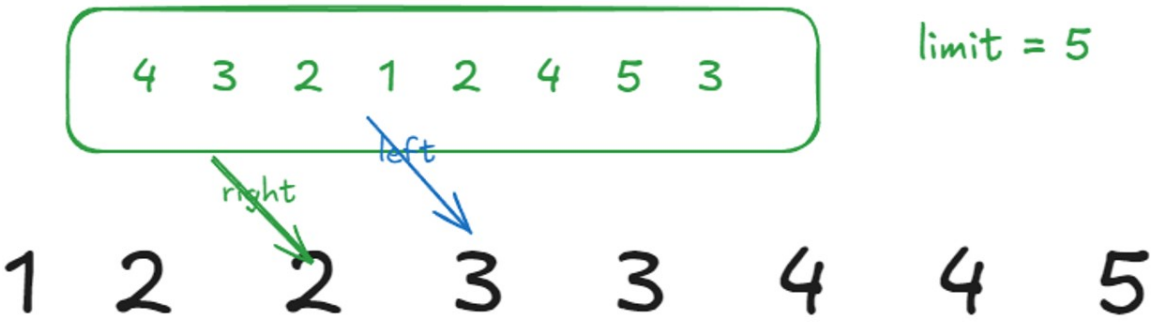
- `4`

limit = 3

[3] , [1,2] [2] using right and left pointer

```java
public static int CountOfBoat(int[] people , int limi
    Arrays.sort(people);
    int left = 0;
    int right = people.length - 1;
    int boats = 0;
    while(left <= right){
        if(people[left] + people[right] <= limit){
            left++;
        }
        right--;
        boats++;
    }
    return boats;
}
```

```
    return boats;
}
```



boat = 5

```
import java.util.Arrays;
import java.util.Scanner;
public class BoatsToSavePeople {

    public static int CountOfBoat(int[] people , int limit){
        Arrays.sort(people);
        int left = 0;
        int right = people.length - 1;
        int boats = 0;
        while(left <= right){
            if(people[left] + people[right] <= limit){
                left++;
            }
            right--;
            boats++;
        }
        return boats;
    }
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        System.out.println("Enter the number of People :");
        int n = scn.nextInt();

        int[] people = new int[n];
        System.out.println("Enter the Weights of the people");

        for(int i =0 ; i<n ; i++){
            people[i] = scn.nextInt();
        }

        System.out.println("Enter the boat weight Limits : ");
```

```
        System.out.println("Enter the boat weight Limits : ");
        int limit  = scn.nextInt();

        int result = CountOfBoat(people , limit);
        System.out.println(result);
    }
}
```

Problem Statement

You need to find all the elements in an array that appear more than once. The array contains integers where each integer is between `1` and `n` (inclusive), where `n` is the size of the array.

Example

Input:

- `nums = [4, 3, 2, 7, 8, 2, 1, 3]`

Output:

- `[2, 3]`

Explanation:

- The numbers `2` and `3` appear more than once in the array.

-4	-3	-2	-7	8	2	-1	-3
0	1	2	3	4	5	6	7

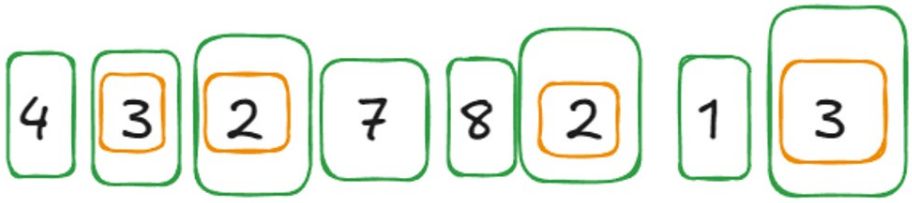
```
index = abs(arr[i]) - 1
marked on that index -ve to it.
& if its already negative
add into list with index + 1;
```

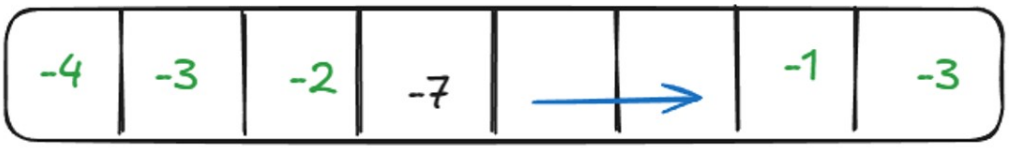
```
      list.add(2 ,3);
```

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
public class FindAllDuplicates {

   public static List<Integer> findDuplicates(int[] arr){
```

```
public class FindAllDuplicates {

    public static List<Integer> findDuplicates(int[] arr){
        List<Integer> duplicates = new ArrayList<>();

        for(int i =0; i<arr.length; i++){
            int index = Math.abs(arr[i]) - 1;

            if(arr[index] < 0){
                duplicates.add(index + 1);
            }

            arr[index] = - arr[index];
        }
        return duplicates;
    }
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        System.out.println("Enter the size of the array:");
        int n = scn.nextInt();

        int[] arr = new int[n];
        System.out.println("Enter the elements of an array");
        for(int i =0; i<n; i++){
            arr[i] = scn.nextInt();
        }
        List<Integer> result = findDuplicates(arr);
        System.out.println("Duplicates " + result);
    }
}
```

```
Enter the size of the array:
8
Enter the elements of an array
4
3
2
7
8
2
1
3
Duplicates [2, 3]
```





duplicate

```
List<Integer> duplicates = new ArrayList<>();

for(int i =0; i<arr.length; i++){
    int index = Math.abs(arr[i]) - 1;

    if(arr[index] < 0){
        duplicates.add(index + 1);
    }

    arr[index] = - arr[index];
}
return duplicates;
```

Recursion :

Recursion is the technique of making a function call itself. A method in java that calls itself is called recursive method.

Recursion makes the code compact but it is complex to understand that code. All problems that are solved using recursion can also be solved using iterations.

Recursion is made for solving problems that can be broken down into smaller, repetitive problems.

down into smaller, repetitive problems.

Find out the sum of N Natural Numbers.

Faith

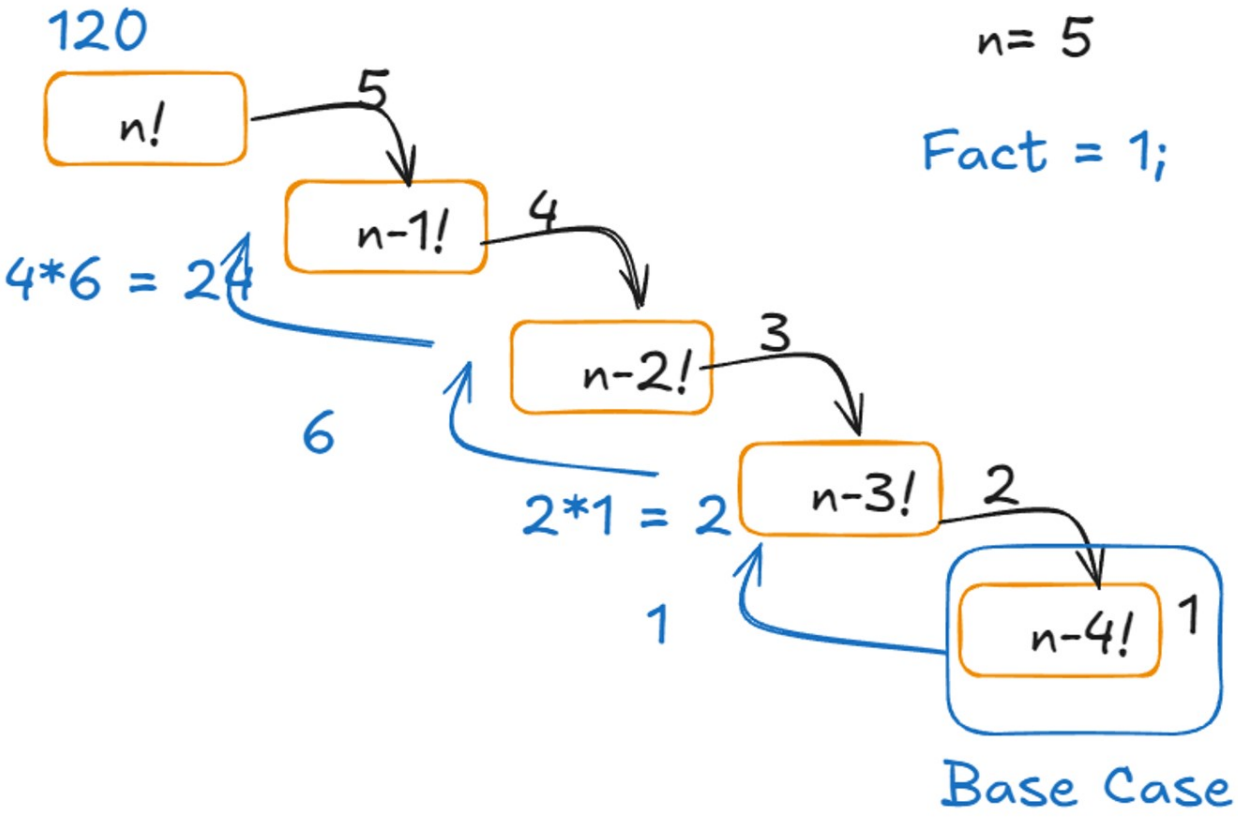
return 5 10+54

Expectation

4 , 3 , 2 , 1. = 10

Faith & Expectation.

5 , 4 , 3 , 2 , 1



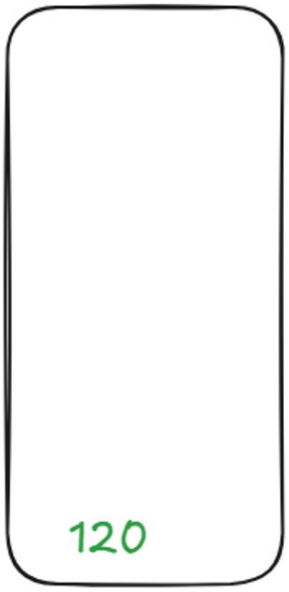
5! = 5 * 4![24] = 120
4! = 4 * 3![6] = 24
3! = 3 * 2![2] = 6
2! = 2 * 1 = 2
1! = 1 * 0!

```
if(fact == 0 || fact == 1){
    return 1;
}
```

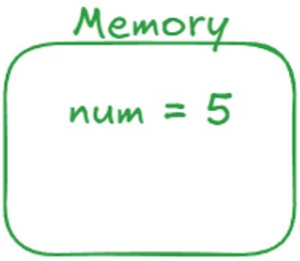
```
public static int factorial(int n){
    if(n == 0){
```

```java
public static int factorial(int n){
    if(n == 0){
        return 1;
    }else{
        return n * factorial(n - 1);
    }
}
```

```java
Scanner scn = new Scanner(System.in);
int num = scn.nextInt();
int fact = factorial(num);
System.out.println(fact);
```



call stack



Completed

Attendance Code: 525AF998

Print a the similar output using recursion.

```
5
4
3
2
1
```

```
1
2
3
4
5
```

```
5
4
3
2
1
1
2
3
4
5
```

```java
if(n > 0){
    System.out.println(n);
    printPattern(n-1);
}
```

```java
if(n > 0){

    printPattern(n-1);
    System.out.println(n);
}
```

```java
if(n > 0){

    System.out.println(n);
    printPattern(n-1);
    System.out.println(n);
}
```

Generate n-th Fibonacci Number using Recursion.

11 12 16 18 21 6 8 5