**Range Query Sum :**

You re given an array of size N with values 0 and number of queries Q.Each Query consists of a 1 D array containing [L , R] .

Increase the value inside the range by 1 .

Step 1 : Get the input from the user .

Step 2 : get the no of query from the user.

Step 3 : For each query run a for loop from L to R and increment the value.

**Time complexity : O (N \* Q)**

But the brute force approach takes a lot of time. We need to optimized it .

So we can go for range update + prefix sum.

For any query [L , R] update the array as follows :

* Nums[L] += 1;
* Nums[R + 1 ] -= 1;

Let the array be **[ 0 , 0 ,0 ,0 , 0, 0 , 0]**

**l = 2 & r = 5**

**[0 , 0 , 1 , 0 , 0 , 0 , -1 , 0]**

Now , when calculating the prefix sum the array will be **[0 , 0 , 1 ,1 ,1 ,1 ,0 , 0]**

**L = 3 & r = 6**

**Original = [0 , 0 , 1 , 1 , 0 , 0 , -1 , -1]**

**Prefix = [0 , 0 , 1 , 2 , 2 , 2 , 1 , 0]**

**Time complexity : O ( N + Q )**

N for calculating the prefix sum & Q for Updating the array.

class Solution {

    public int[] rangeQuery(int [] nums , int [][] query){

        int n = nums.length;

        int [] prefix = new int[n];

        for(int [] i : query){

            int l = i[0];

            int r = i[1];

            nums[l] += 1;

            if(r + 1 < n){

                nums[r + 1] -= 1;

            }

        }

        prefix[0] = nums[0];

        for(int i = 1 ; i < n ; i++){

            prefix[i] = prefix[i - 1] + nums[i];

        }

        return prefix;

    }

}

Real Qn : <https://www.desiqna.in/13650/google-girl-hackathon-coding-questions-solutions-2023-kumar>

* You re given an array of size n and an interger k .
* You can update a number in the array possibly only one time in the range [-k , k]
* Return the maximum number of elements the can have equal values .

Example : [7 , 9 , 11] k = 2

Output : 3 🡪 [9 , 9 , 9 ] [+2 , +0 , - 2]

Example : [1 , 8 , 10] k = 5

Output : 2 🡪 [1 , 10 , 10] or [1 , 8 , 8 ]

Observation :

* For each value in the array , the possibility can be [ nums[i] – k , nums[i] + k ]
* For example nums[i] = 5 & k = 2
* Possiblitiles are [5 -2 , 5-1 , 5- 0 , 5 + 1, 5 + 2] 🡪 [3 , 4 , 5 , 6 , 7]
* So the index is between 3 to 7 .Now you can use the range Query Logic.

Sample Test Case :

Nums = [5 , 8 , 10] k = 2

[ 0 , 0 , 0 , 0 , 0 , 0 , 0, 0, 0, 0, 0 , 0 ]

**Nums[0] = 5**

**L = 3 & R = 8**

**[0 , 0 , 0 , 1 , 0 , 0 , 0 ,0 , -1 , 0 , 0 , 0 ]’**

**Nums[1] = 8**

**L = 6 & R = 11**

**[0 , 0 , 0 , 1 , 0 , 0 , 1 , 0 , -1 , 0 , 0 , -1 , 0]**

**Nums[2] = 10**

**L = 8 & R = 13**

**[0 , 0 , 0 , 1 , 0 ,0 , 1 , 0 , 1 , 0 , 0 , -1 , 0 ] 13 is out of bound**

Calculate the prefix sum of the array

[0 , 0 , 0 , 1 , 1, 1, 2 , 2 , 2 , 2, 2, 2 , 2]

So , the maximum value is 2.

    class Solution {

         public static  int maxLengthOfSubsequence(int [] nums , int k){

            int n = nums.length;

            int max = Integer.MIN\_VALUE;

            int [] prefix = new int[15];

            int [] dummy = new int[15];

            for(int i = 0 ; i < n ; i++){

                int l = nums[i] - k ;

                int r = nums[i] + k ;

                dummy[l] += 1;

                if(r + 1 < 15){

                    dummy[r + 1] -= 1;

                }

            }

            System.out.println(Arrays.toString(dummy));

            prefix[0] = dummy[0];

            for(int i = 1 ; i < 15 ; i++){

                prefix[i] = prefix[i - 1] + dummy[i];

                max = Math.max(prefix[i] , max);

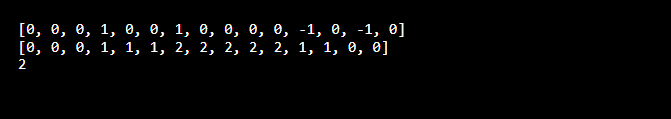
            }

            System.out.println(Arrays.toString(prefix));

            return max;

        }

    }



Time conplexcity : O (N + N) ~ O(N)