Q1: There are n children standing in a straight line and a robot which distributes candies among them.

n children → [ ]. (0 1 2 3 ------ (n-1))

Robot distributes a candy for every children in the given range, say [L -- R].
It repeats this process B no. of times.

Output: You have C queries (C is an array), return how many children have atleast C[i] candies.

Ex1:

n = 4.

$$B = \begin{bmatrix} 1, 2 \\ 4, 4 \\ 1, 3 \end{bmatrix}$$

	0	1	2	3
	0	0	0	0
-1 index →	1	1	0	0
-1 index →	1	1	0	1
-1 index →	2	2	1	1

C = [1, 2, 3]

how many children are having atleast 1 candy → 4.
how many children are having atleast 2 candy → 2
how many children are having atleast 3 candy → 0.

Output : [4 2 0].

Ex2:

n = 4

$$B = \begin{bmatrix} 3, 4 \\ 2, 3 \end{bmatrix}$$

	0	1	2	3
	0	0	0	0
-1 index →	0	0	1	1
-1 index →	0	1	2	1

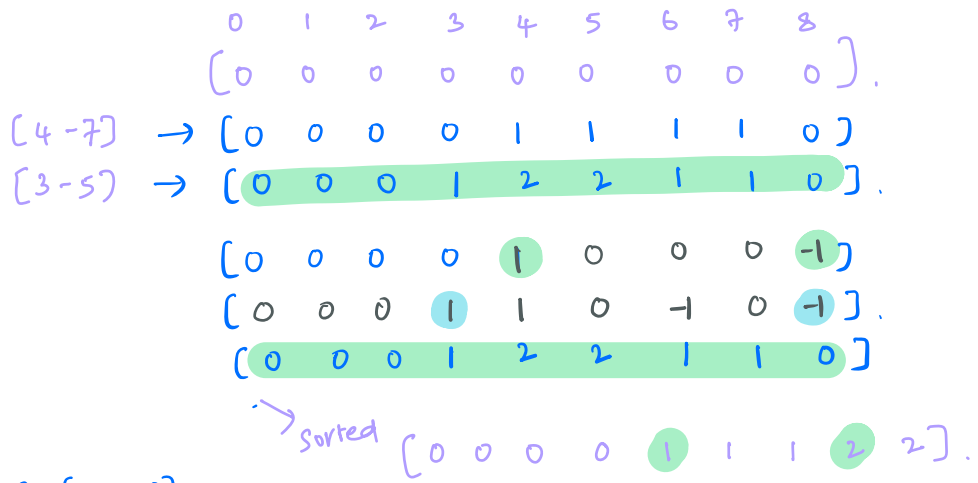
C = [2]

how many children are having atleast 2 candy → 1.

Idea 1:

* create an array, a → (0)*n.
* For every operation, iterate from [L-R], update 'a' by adding 1 to every a[i], i → [L→R].
* For every c[i], iterate over 'a' and check for entries ≥ c[i].

Idea 2: Prefix sum.



c = [1, 2, 3]

how many children are having atleast 1 candy → 5
how many children are having atleast 2 candy → 2
how many children are having atleast 3 candy → 0.

First occurence of c[i] → ?. in sorted array.

=> We can use **binary search**.

```
def distributeCandies(n, B, C)
{
    a = [0]*n, m = len(B).
    for i in range(0,m)
        l = B[i][0], r = B[i][1].
        a[l-1] = a[l-1]+1
        a[r]   = a[r]-1      <- check for index out of bounds.
    // pf on a, and sort it.
    ans = [0] * len(c).
    for i in range(0, len(c))
        count = firstOccurence(a, 0, n, c[i]) // binary search
        ans[i] = n - count
    return ans
}
```

O(m).

O(nlogn).

q*logn.

TC: O(m+nlogn + q×logn).

Q2: Given an array of integers A of size N, can contain [0, 1, 2 ---(N-1)] in any order.
You need to make max no. of chunks from this array

Conditions for chunking out the array.

(i) The array must be sorted when the individual chunks are sorted and concatenated.

① A = [1, 2, 3, 4, 0] , ans = 1.

Sort chunks → Concatenate

[1, 2, 3] [4, 0] → [1, 2, 3] [0, 4] → [1, 2, 3, 0, 4] ✗

[1, 2] [3, 4, 0] → [1, 2] [0, 3, 4] → [1, 2, 0, 3, 4] ✗

[1, 2, 3, 4, 0] → [0, 1, 2, 3, 4] → [0, 1, 2, 3, 4] ✓

② A = [2, 0, 1, 3] → ans = 2 chunks.

Sort chunks → Concatenate

[2, 0] [1] [3] → [0, 2] [1] [3] → [0, 2, 1, 3] ✗

[2] [0, 1] [3] → [2] [0, 1] [3] → [2, 0, 1, 3] ✗

[2] [0] [1] [3] → [2] [0] [1] [3] → [2, 0, 1, 3] ✗

[2, 0, 1] [3] → [0, 1, 2] [3] → [0, 1, 2, 3] ✓

③ A = [1, 2, 0, 3, 5, 4] , ans = 3.

0 1 2 3 4 5

[1] [ ] [ ] → [1

[1 2] [ ] [ ] → [1 2

+1 [1 2 0] → [0 1 2

+1 [3] → [0 1 2 3

+1 [5 4] → [0 1 2 3 4 5].

Hint: Smallest left chunk that you can have from [0 - - - i], contains all elements till I.

A = [1, 2, 0, | 3, | 5, 4] (indices 0 1 2 | 3 | 4 5)

till what I've → ; till what i should be having ← i

Range	I've		
[0 - 0]	I've 1 →	It should only contain till 0	✗
(0 - 1)	I've 2 →	It should only contain till 1.	✗
(0 - 2)	I've 2 →	It should only contain till 2.	✓
[0 - 3]	I've 3 →	till 3.	✓
[0 - 4]	I've 5 →	till 4.	✗
[0 - 5]	I've 5 →	till 5.	✓

Code:

```
def maxChunks(A, n)
{
    chunks = 0,  maxTillI = 0.

    for i in range (0,n)
        maxTillI = max(maxTillI, A[i])
        if(maxTillI == i)
            chunks = chunks + 1

    return chunks.
}.
```

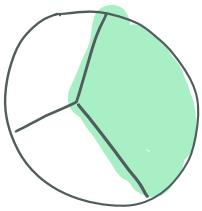
A = [1, 2, 0, 3, 5, 4] (indices 0 1 2 3 4 5)

chunk = 0,
maxTillI = 0.

i	maxTillI	chunk
0	1	0
1	2	0
2	2	1
3	3	2
4	5	2
5	5	3.

[0, 2, 1, 3, 5, 4].

[0] [2 1]



Q3: Given a string, re-arrange to get minimum period.

What is period of a string?

pqr pqr pqr. , period = 3.

abcd abcd. , period = 4.

aaaaaa , period = 1.

Ex 1: ab ac bc.