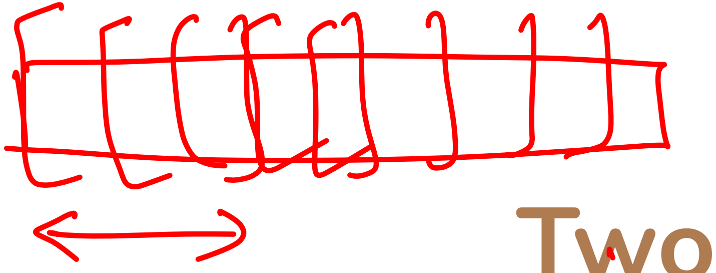
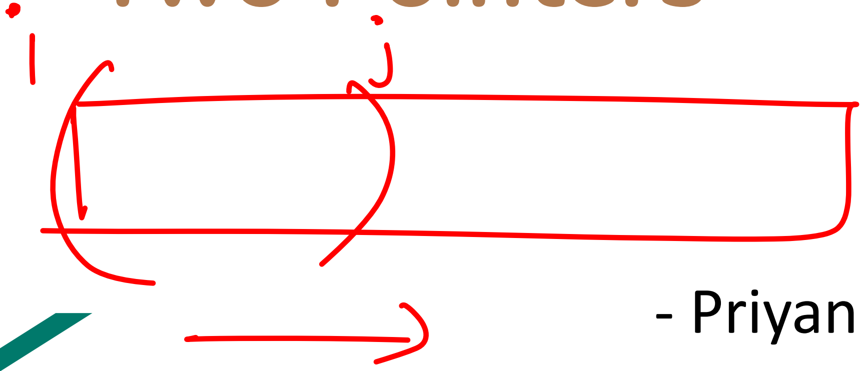
# Two Pointers

Sliding windows

variable size sliding window





- Priyansh Agarwal

# Binary search on Answer

O( log(search space) . Predicate function)

$\simeq$ O(logn) . O(n)

2 pointers → O(n)

# Two Pointers

- Widely used in Competitive Programming ✓
- Optimization Technique
- Most Two Pointer problems can be solved using Binary Search
- Useful for a lot of array based problems
- Super useful for interviews too

$O(n\log n)$

↓

$O(n)$

Given $\boxed{\text{2 sorted}}$ arrays, merge them into one single array keeping the elements sorted

$n \geq m$

$O(n)$

| 1 | 4 | 5 | 9 |
|---|---|---|---|

$O(m)$

| 2 | 3 | 6 | 10 |
|---|---|---|---|

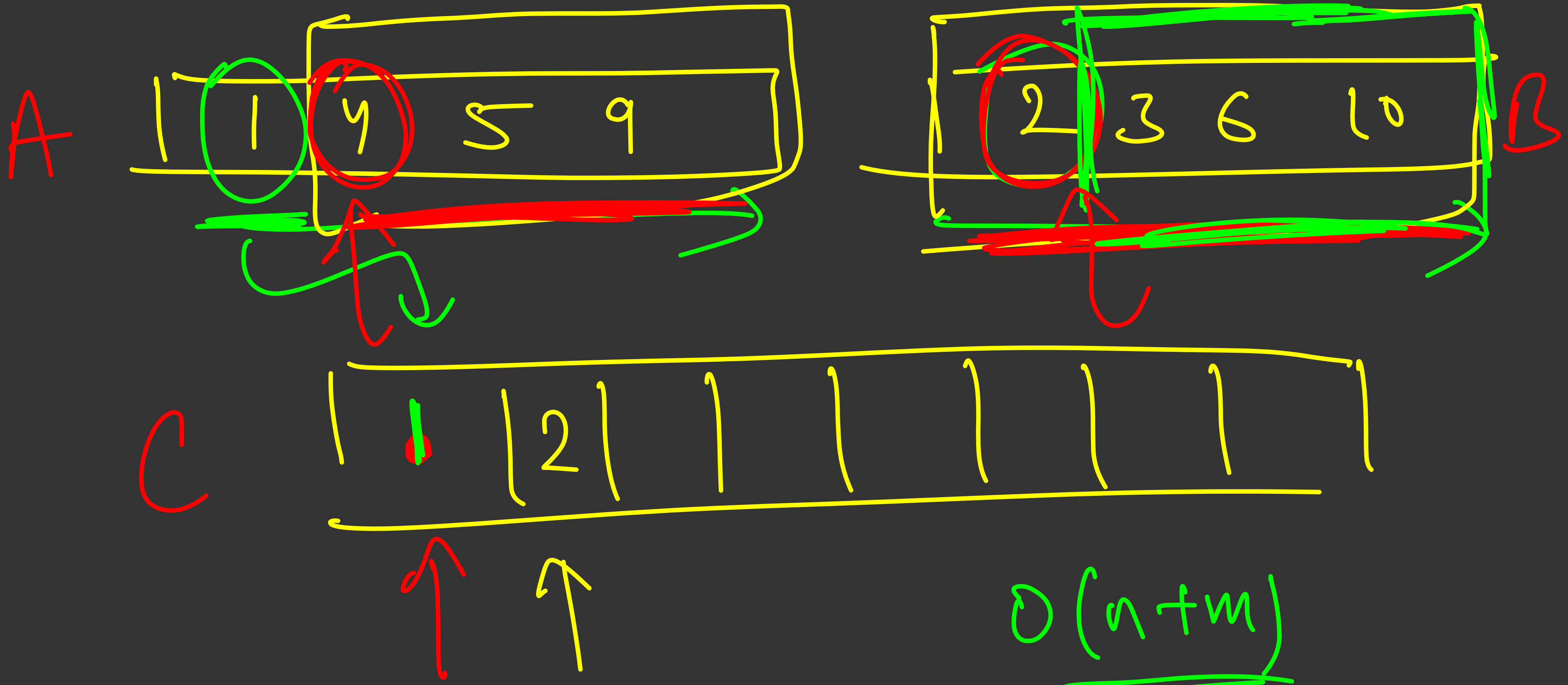
$\boxed{n\log n}$

| 1 | 2 | 3 | 4 | 5 | 6 | 9 | 10 |
|---|---|---|---|---|---|---|---|

First Approach: Add all elements in an array and sort it

Second Approach: Use 2 pointers

$O((n+m) \cdot \log(n+m))$



$O(n+m)$

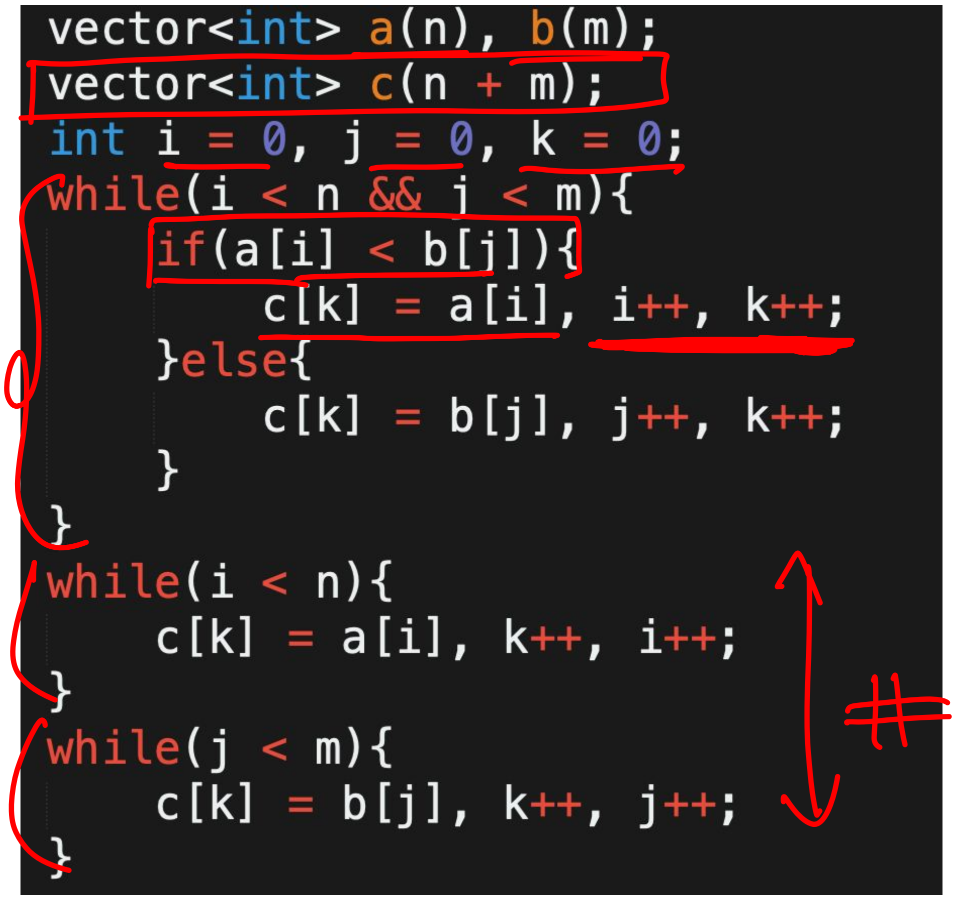
C is the smallest element among both the lists

# Solution using 2 pointers

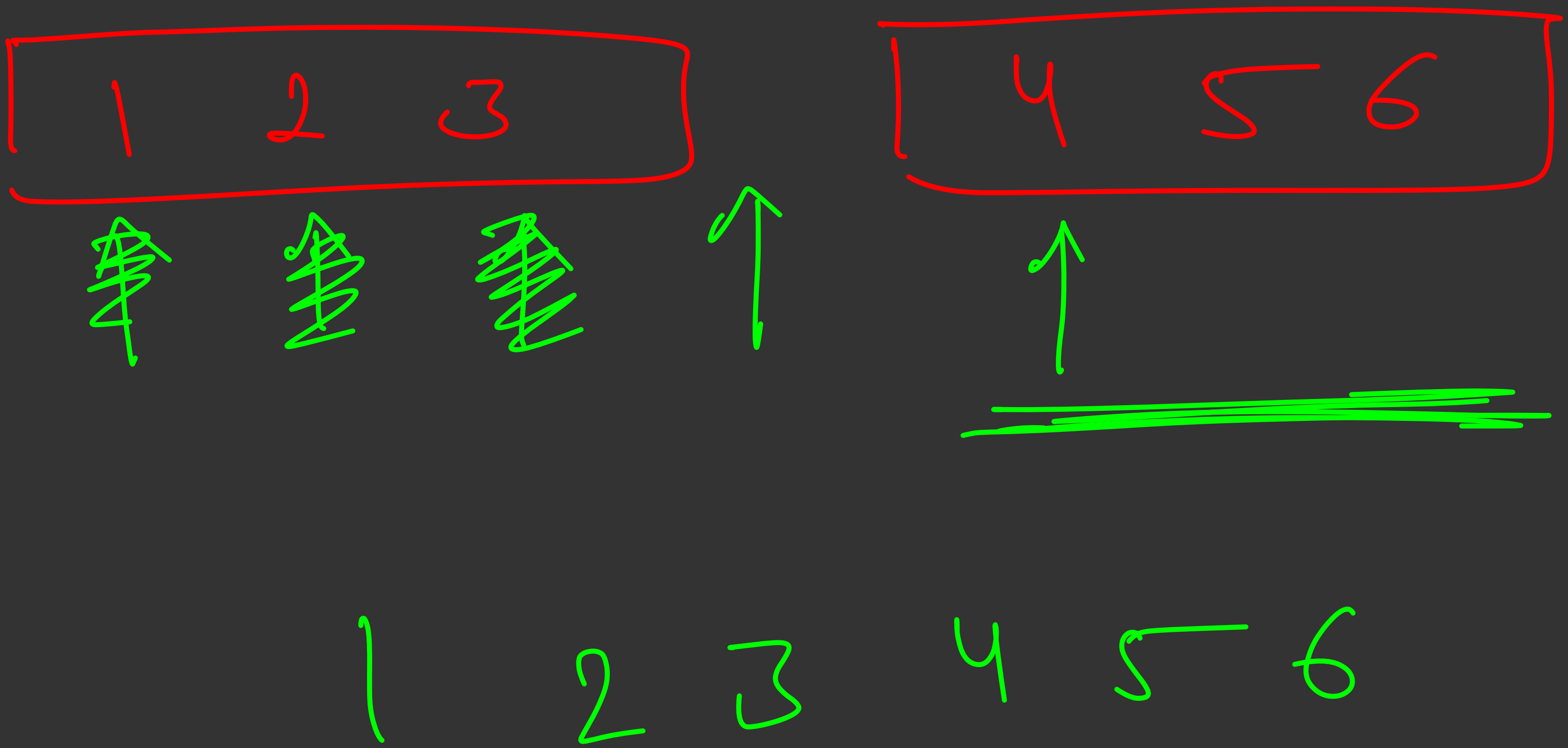
Maintain 2 Pointers, i and j both starting from the left ends of the arrays

$c[k] = \min(a[i], b[j])$

Keep pushing the smaller of the 2 elements from the arrays into the output array



```
vector<int> a(n), b(m);
vector<int> c(n + m);
int i = 0, j = 0, k = 0;
while(i < n && j < m){
    if(a[i] < b[j]){
        c[k] = a[i], i++, k++;
    }else{
        c[k] = b[j], j++, k++;
    }
}
while(i < n){
    c[k] = a[i], k++, i++;
}
while(j < m){
    c[k] = b[j], k++, j++;
}
```


1 2 3
4 5 6
1 2 3 4 5 6

① first sort each of them then merge ✓✓

$$\boxed{n\log n + m\log m} + \boxed{n+m}$$

② first merge and then sort

$$\boxed{(n+m)} + \boxed{(n+m)\log(n+m)}$$

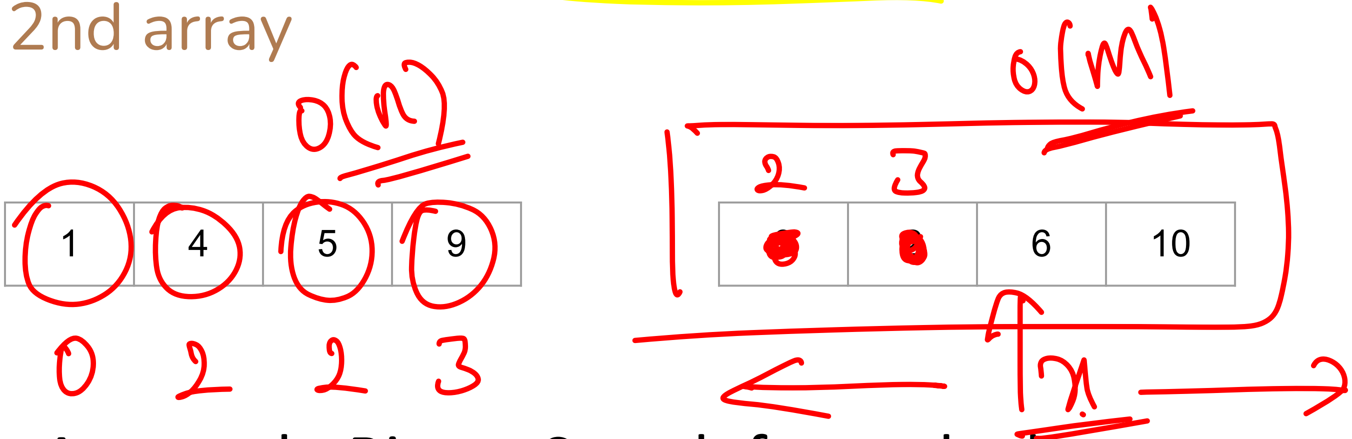
①

$$n\log n + m\log m$$

②

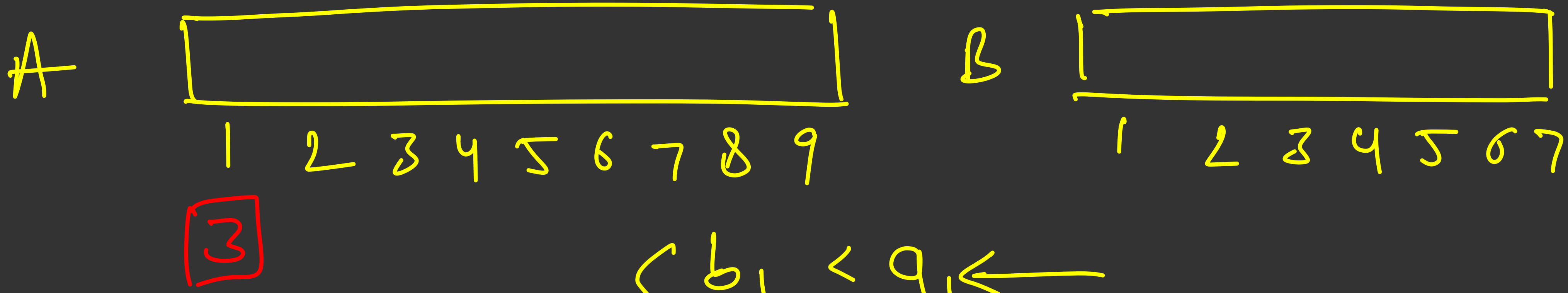
$$n\log(n+m) + m\log(n+m)$$

Given 2 sorted arrays, for each element in 1st array find number of elements smaller than that in the 2nd array



First Approach: Binary Search for each elements

Second Approach: 2 pointers



$a_2$

$$\boxed{a_2 > a_1}$$

$$\left\{\begin{array}{l} b_1 < a_1 \leftarrow \\ b_2 < a_1 \\ b_3 < a_1 \end{array}\right.$$

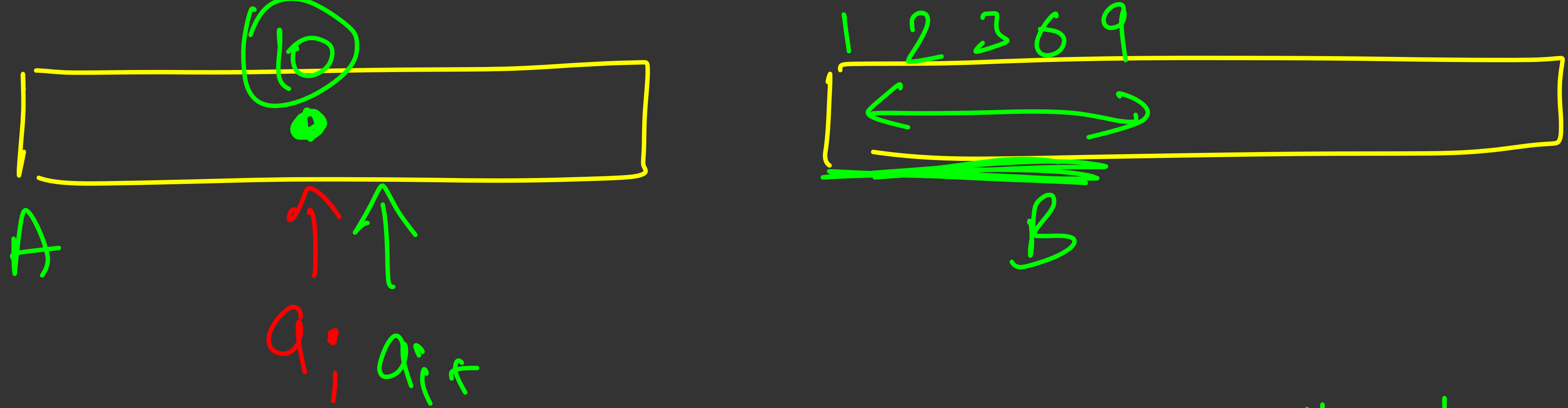
$b_4 > a_1$

if $\forall$ i from 1 to k

$$\boxed{b_i < a_j}$$

$\Rightarrow$ that $\forall$ i from 1 to k

$$\boxed{b_i < a_{j+1}}$$



5 elements are smaller than $a_i$

how many elements will be smaller than $a_{i+1}$

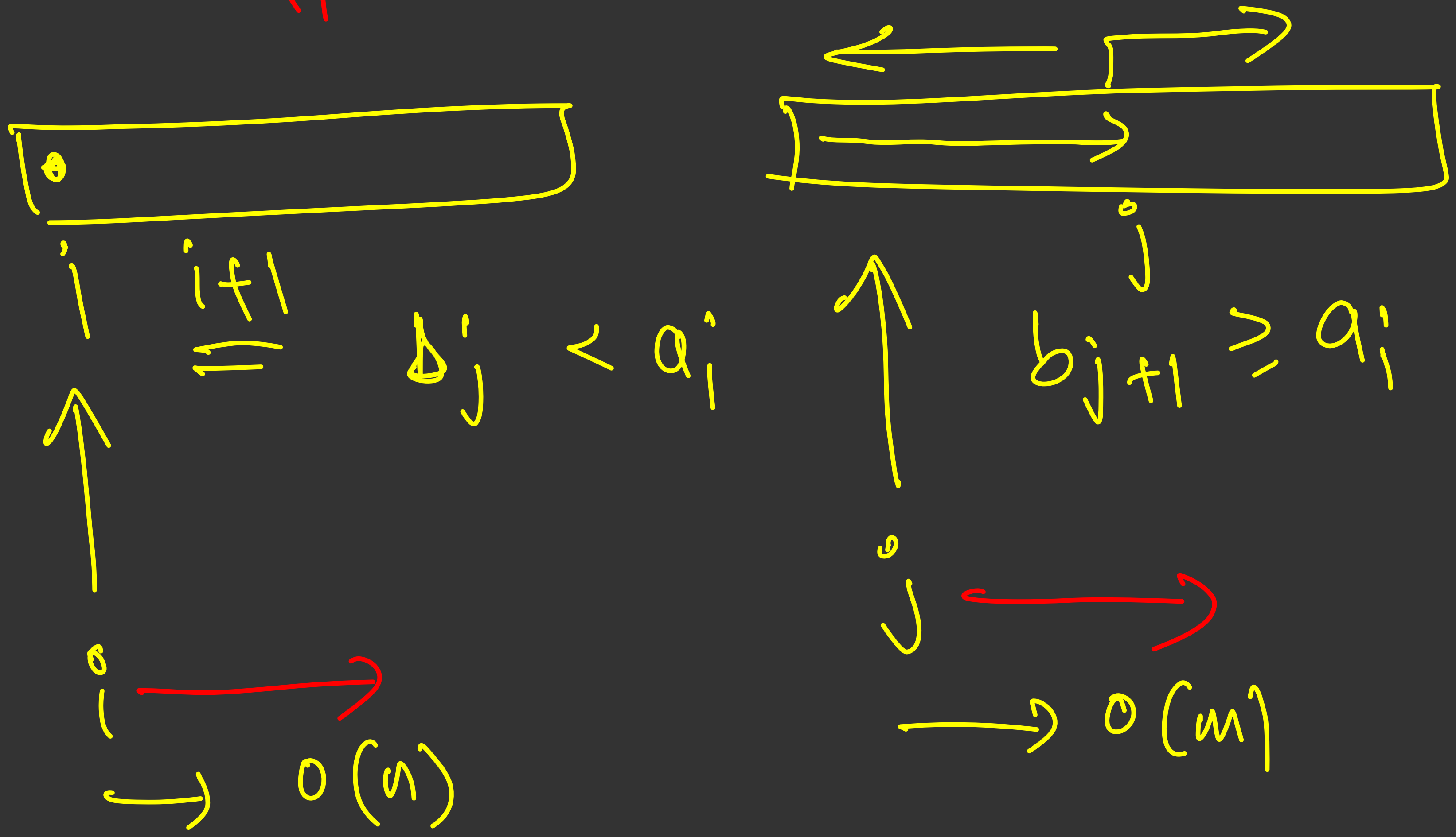
① $\geq 5$

② might be $< 5$

Going from left to right in A

the no. of element smaller than

$a_i$ never decreases



$\Rightarrow b_j < a_i$

$b_{j+1} \geq a_i$

i → O(n)

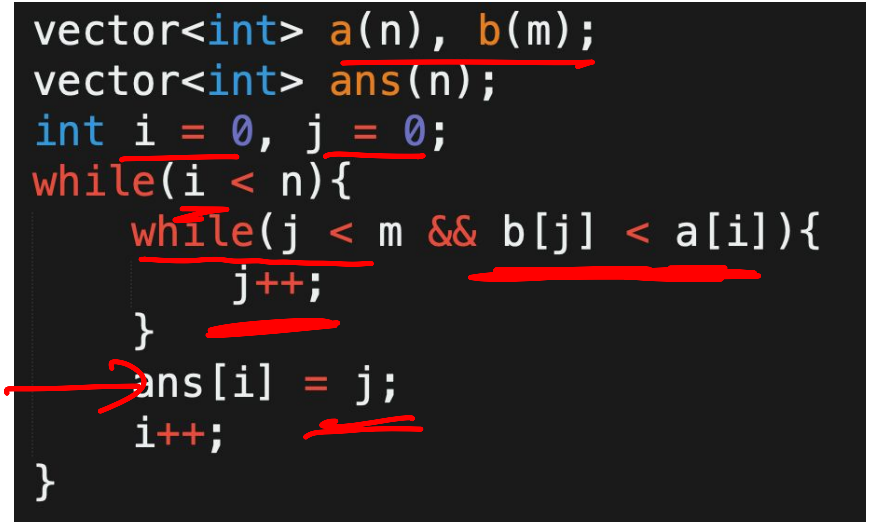
j → O(m)

# Solution using 2 pointers

If 5 elements are smaller than a[i], how many elements will be lesser than a[i + 1]?

Clearly, we should check for elements bigger than first 5 elements now as a[i + 1] >= a[i]

Having 2 pointers and both only move right. Time complexity?



```
vector<int> a(n), b(m);
vector<int> ans(n);
int i = 0, j = 0;
while(i < n){
    while(j < m && b[j] < a[i]){
        j++;
    }
    ans[i] = j;
    i++;
}
```

O(n+m)

10 20 30 40

1 1 1 1 1 1 1 1 | $10^9$ $10^9$

$10^6$ integers

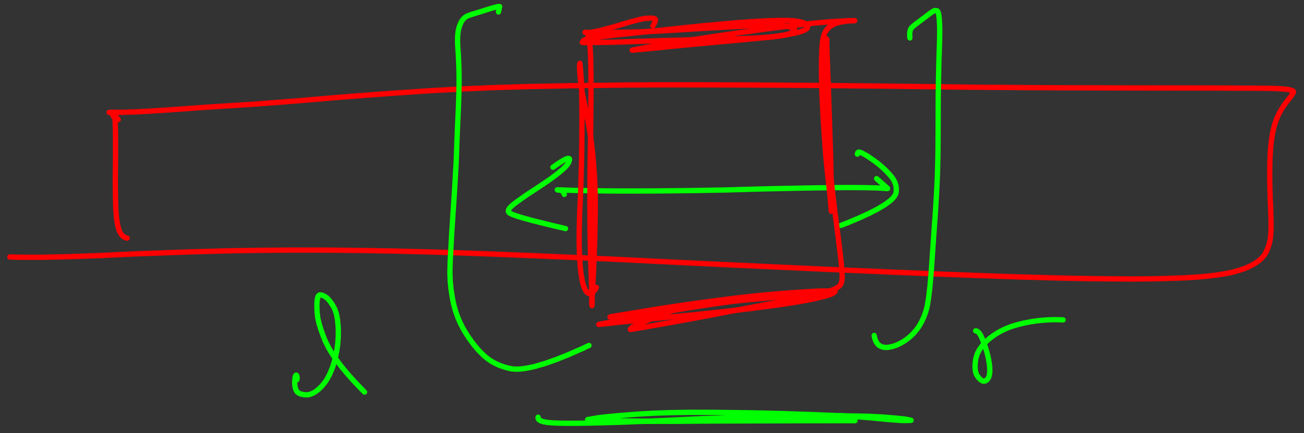
first $5 \cdot 10^5$ are 1

second $5 \cdot 10^5$ are $10^9$

# Good Segments Technique (Increasing)

- Given an array of positive integers find the length of longest subarray with sum <= K
- Given an array find the length of longest subarray with not more than K distinct elements

M.size(), → O(1)



no. of distinct elements from

l to r $\leq$ k

| 1 | 2 | 4 | 5 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|---|

k = 11

length = 4

sum = 10

if a subarray of length 4 works

↳ there also exists a subarray of length 3 that will work

↳ " of length 2

↳ " " " 1

# if a length of x works then all length of x-1, x-2 ----- 1 will also work

# $f(x) = \begin{cases} T & \text{if there exists a subarray of length } x \text{ whose sum } \le k \\ F \end{cases}$

find out the biggest x for which $f(x) = T$ → that is the answer

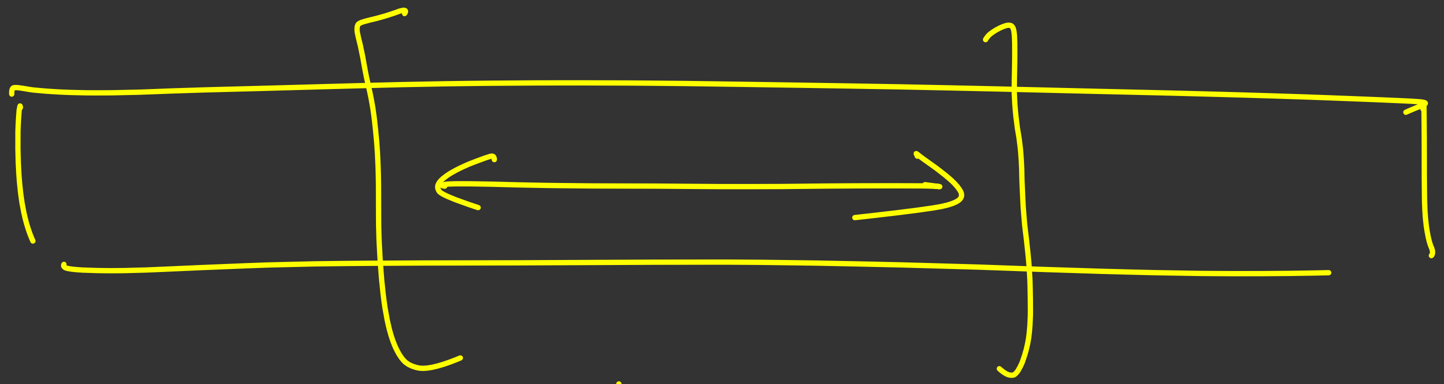
```
bool f(x) {
    sliding window approach to check all
    subarray of length x
}
```

$\underline{O(n)}$

$\boxed{O(n\log n)}$

```
while( left <= right) {
    mid = (left + right)/2
    if(f(mid))
        ans = max (ans, mid)
        left = mid+1
    else
        right = mid-1 }
```

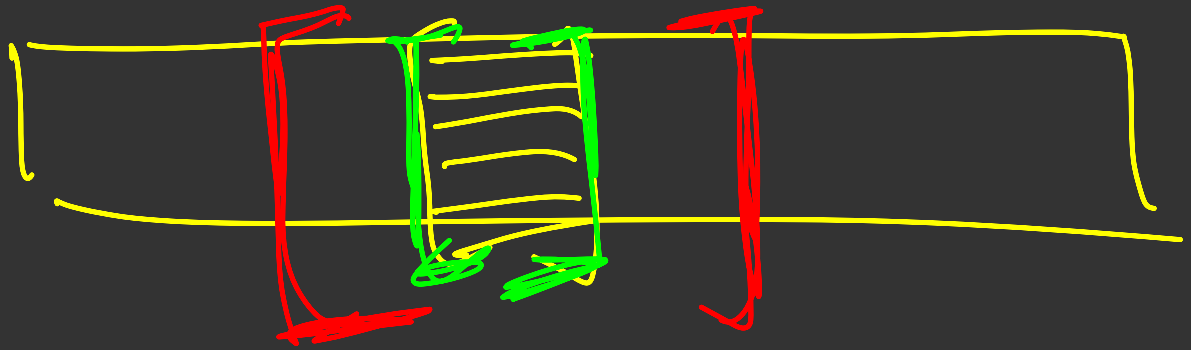
$\underline{O(\log n)}$



this works

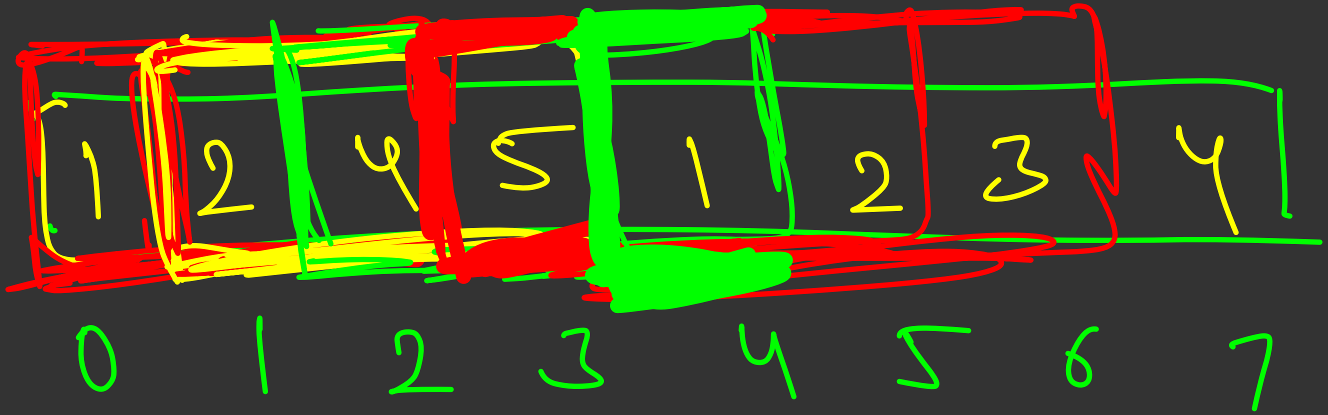
$\boxed{sum \le k}$

Good segment



Good segment : any subarray with sum ≤ k

Any segment inside a good segment will also be good.



k = 11

i = 0 , j = 2

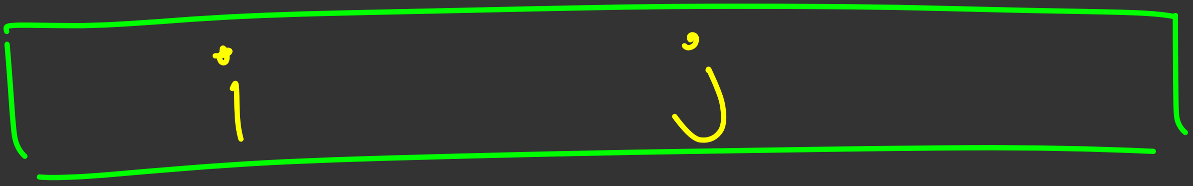
i = 1 , j = 3

i = 2 , j = 4

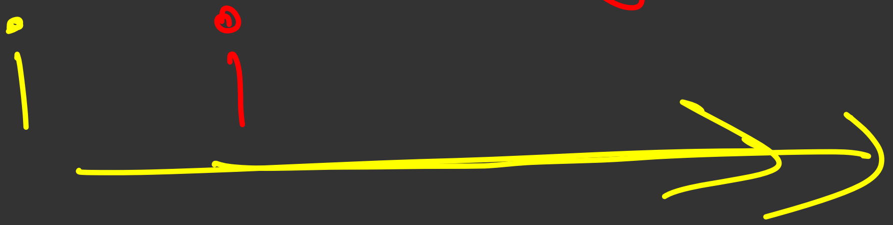
i = 3 , j = 6

i = 4 , j = 7

1

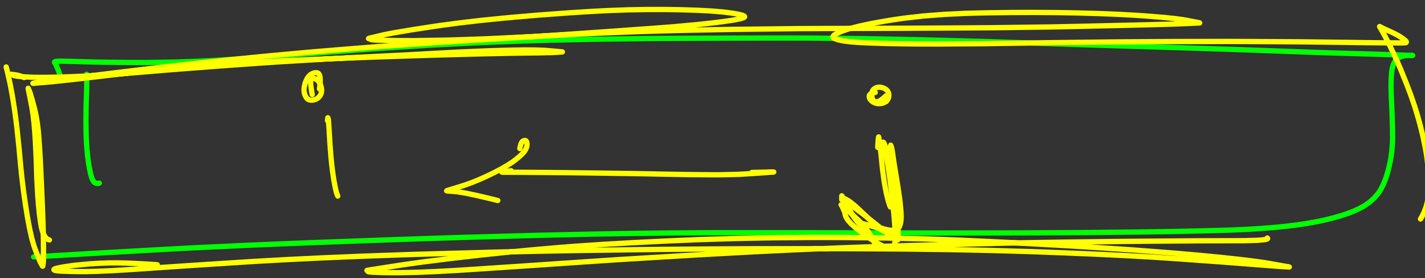


find biggest subarray that starts at



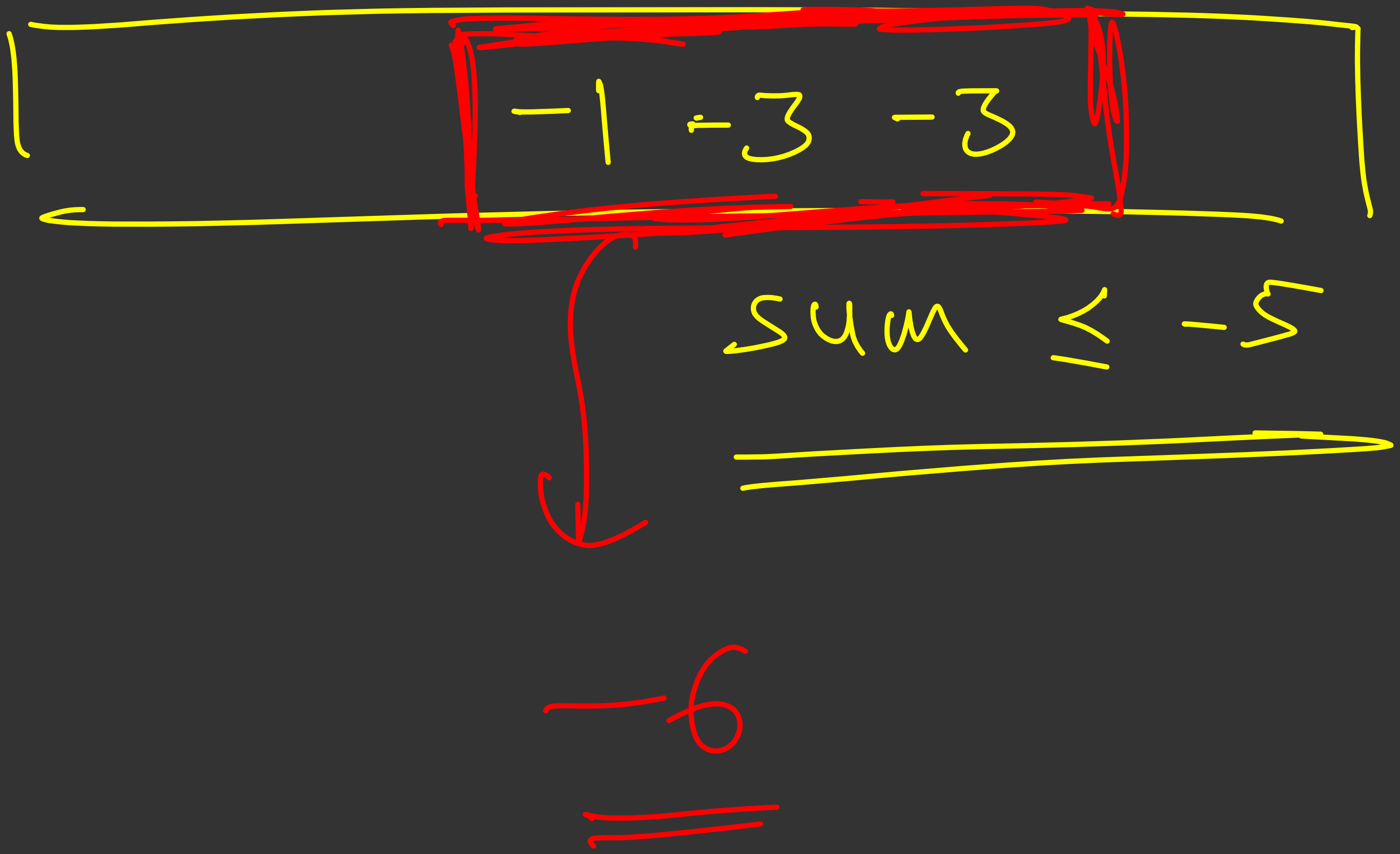
max

2

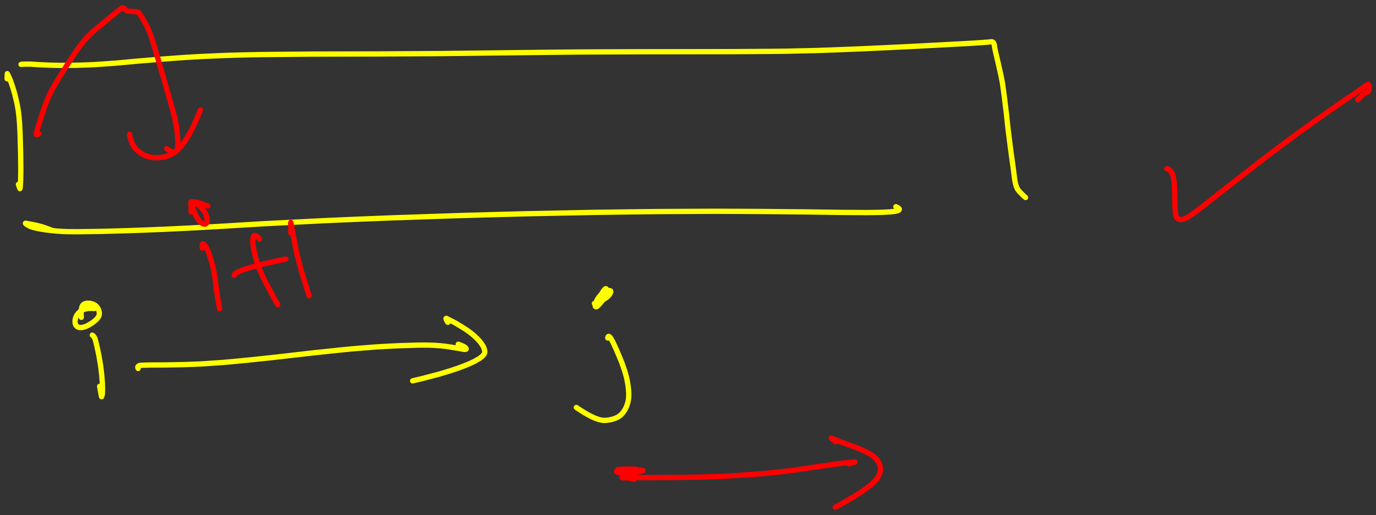


find bigest subarray that ends at

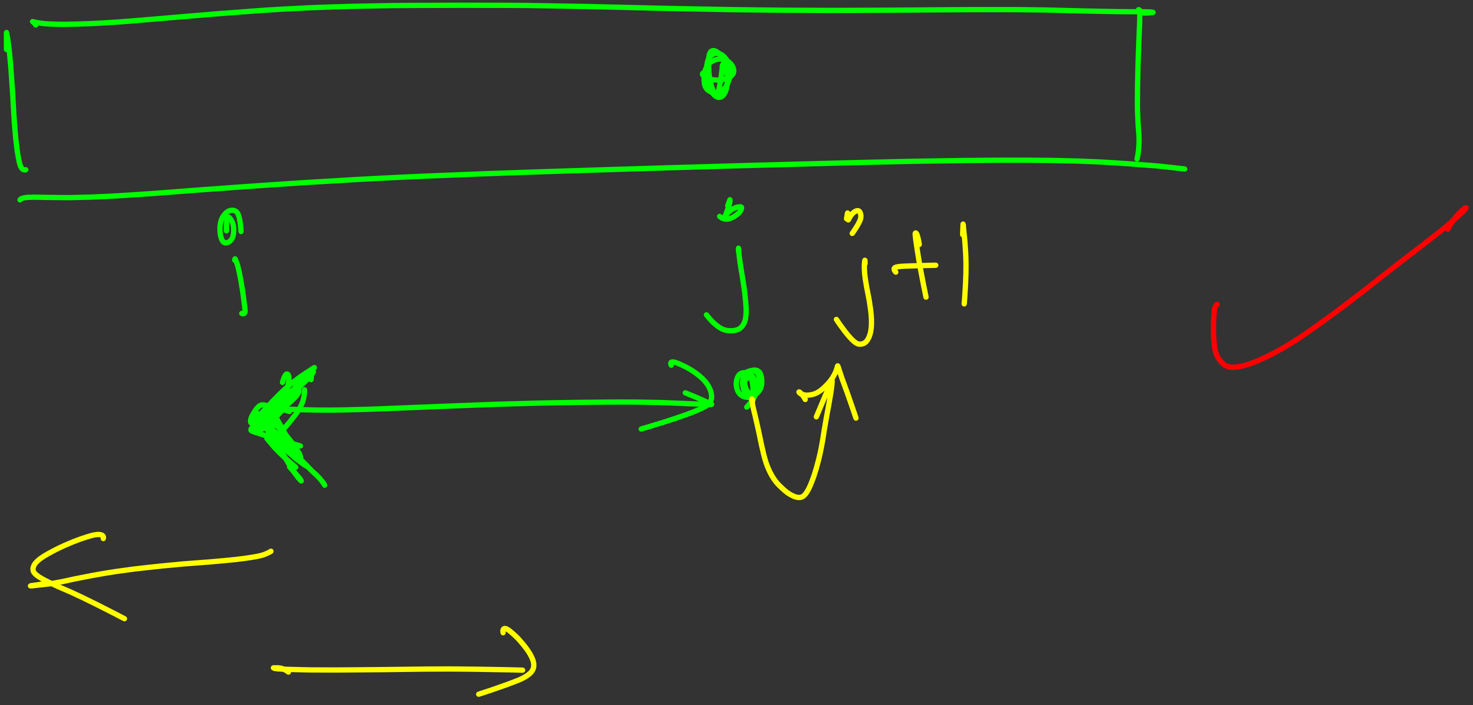
max

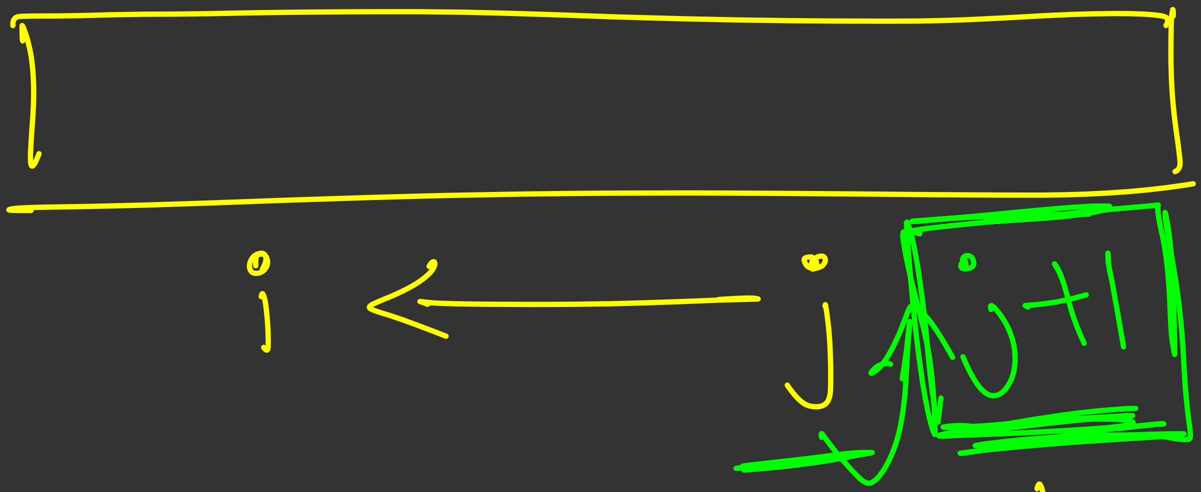

-1 -3 -3
sum ≤ -5
-6

①



②



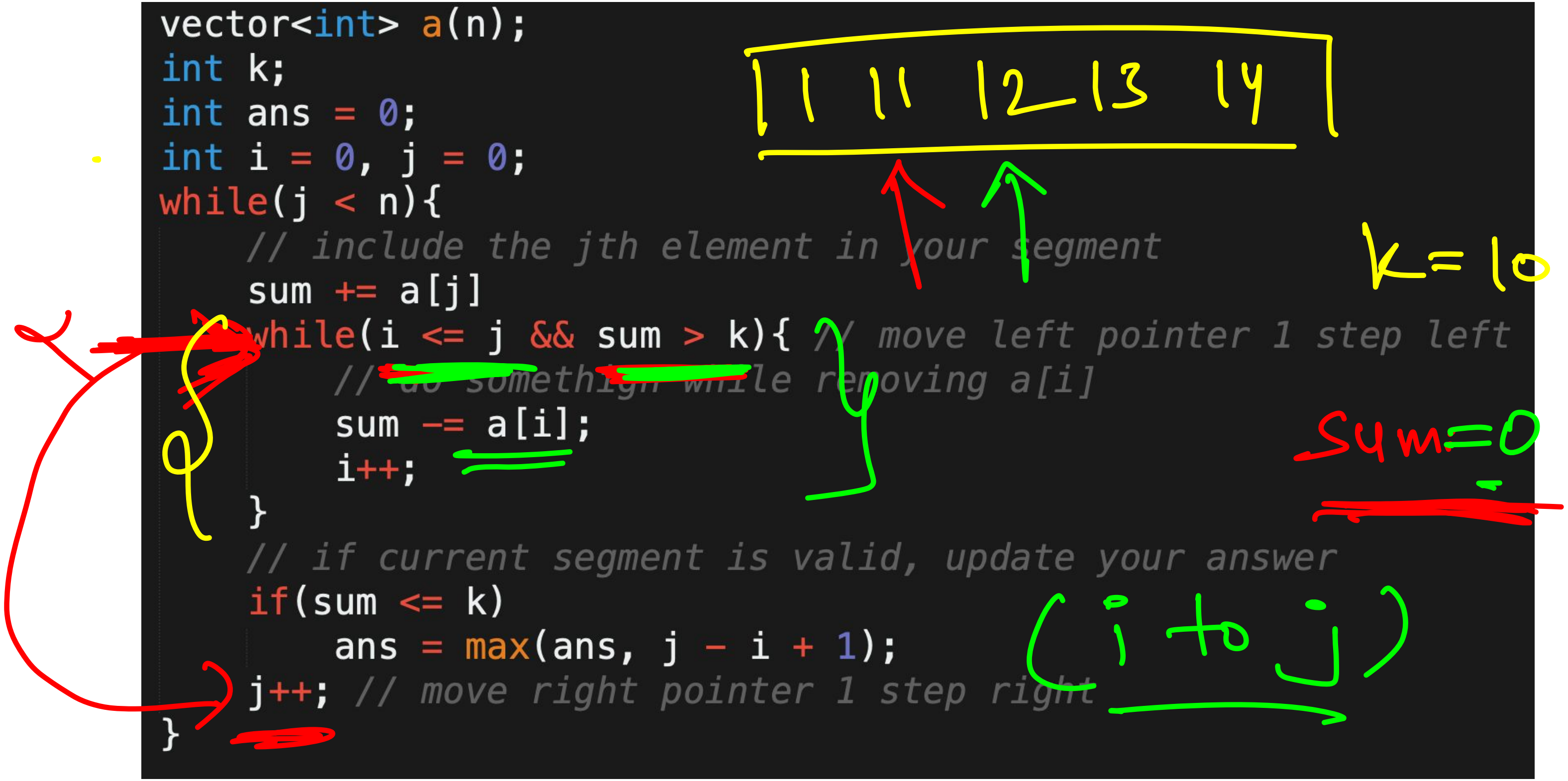


sum( i to j) $\le$ k

but sum( i-1 to j) $>$ k

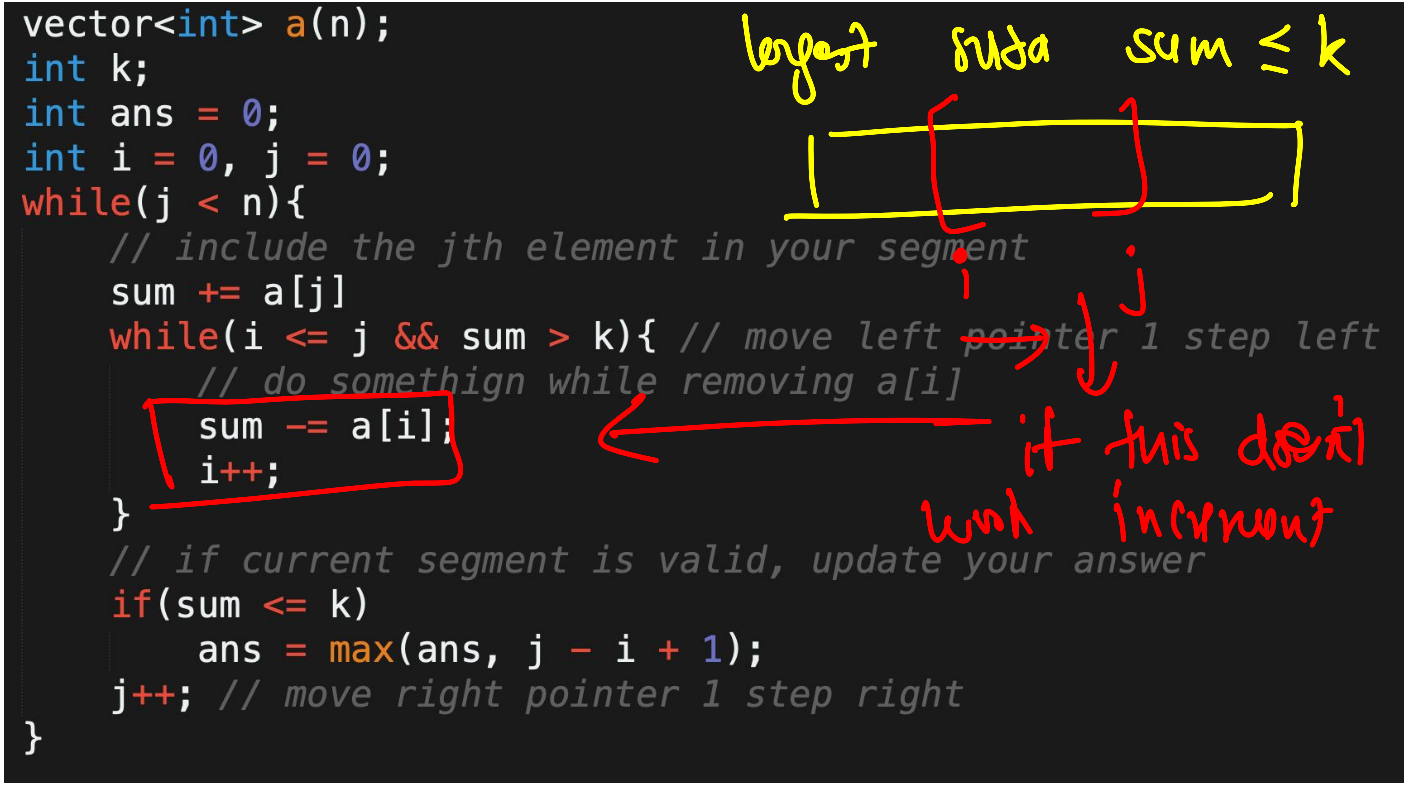
i → j →

# Good Segments Technique Problem 1



```
vector<int> a(n);
int k;
int ans = 0;
int i = 0, j = 0;
while(j < n){
    // include the jth element in your segment
    sum += a[j]
    while(i <= j && sum > k){ // move left pointer 1 step left
        // do somethign while removing a[i]
        sum -= a[i];
        i++;
    }
    // if current segment is valid, update your answer
    if(sum <= k)
        ans = max(ans, j - i + 1);
    j++; // move right pointer 1 step right
}
```

# Good Segments Technique Problem 1



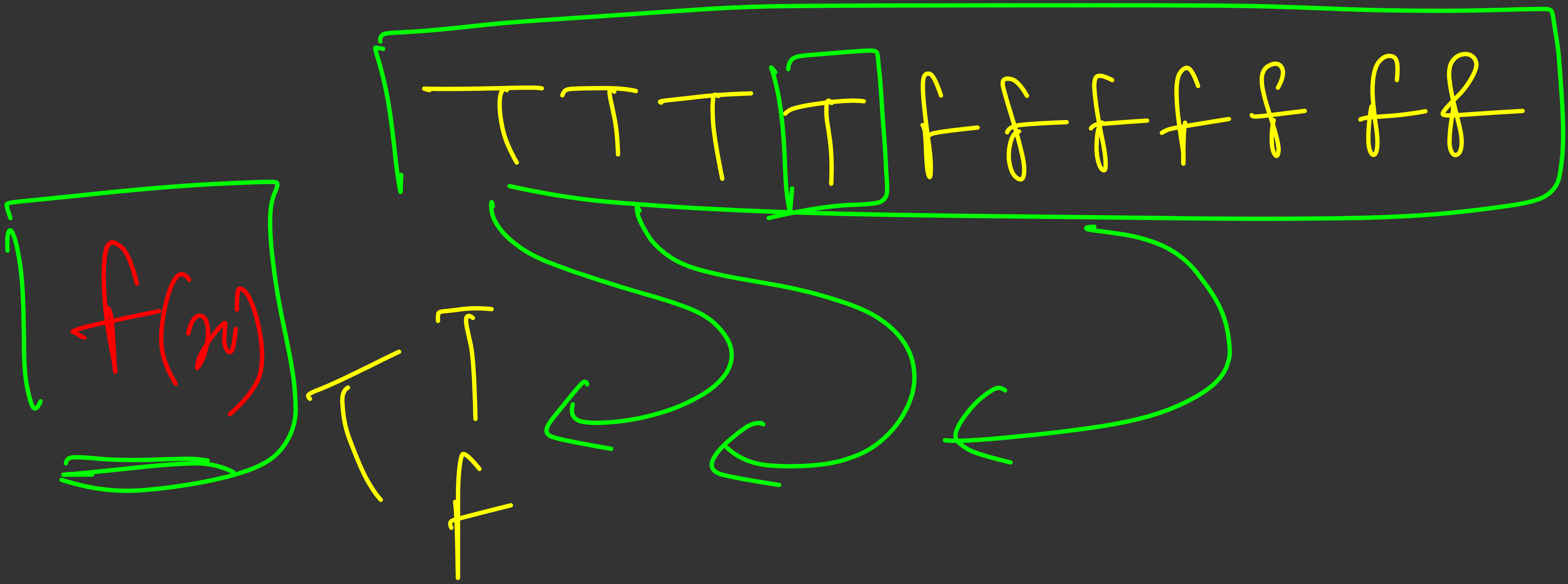
```
vector<int> a(n);
int k;
int ans = 0;
int i = 0, j = 0;
while(j < n){
    // include the jth element in your segment
    sum += a[j]
    while(i <= j && sum > k){ // move left pointer 1 step left
        // do somethign while removing a[i]
        sum -= a[i];
        i++;
    }
    // if current segment is valid, update your answer
    if(sum <= k)
        ans = max(ans, j - i + 1);
    j++; // move right pointer 1 step right
}
```

longest subarray sum ≤ k

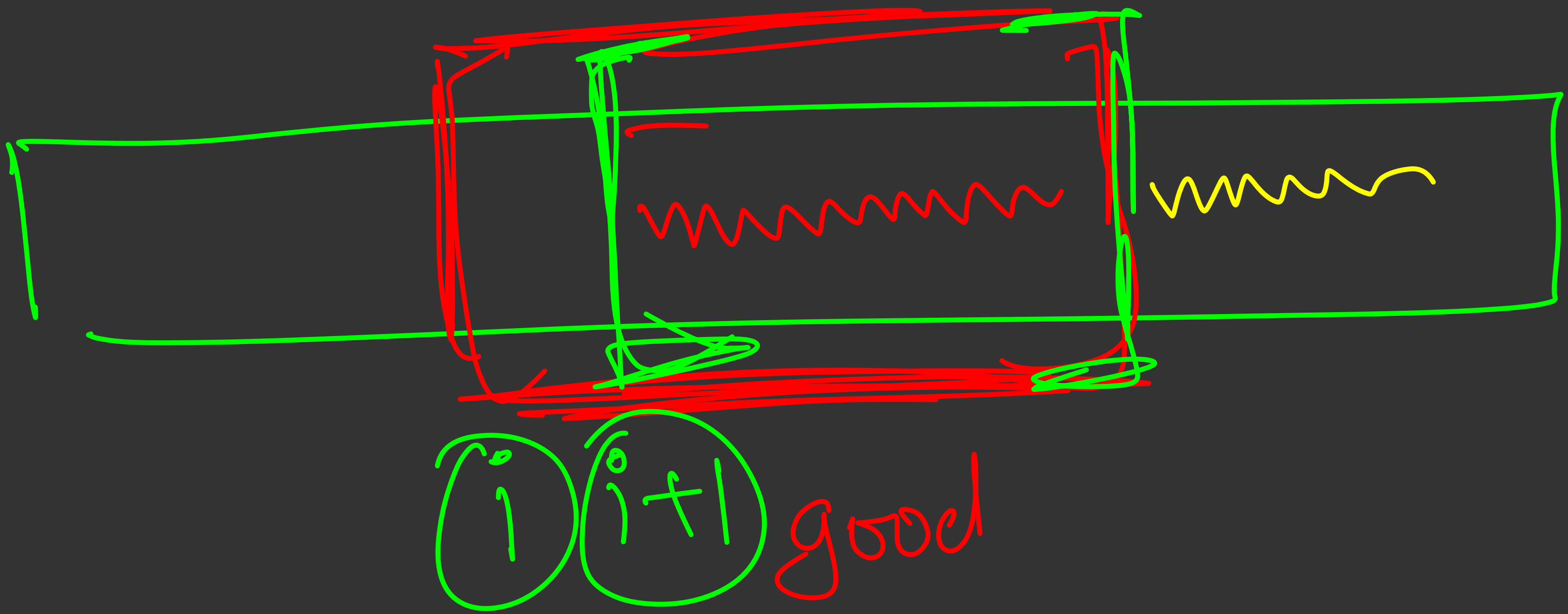
if this doesn't work increment

## ① Binary Search on Answer

monotonic predicate function



② 2 Pointer



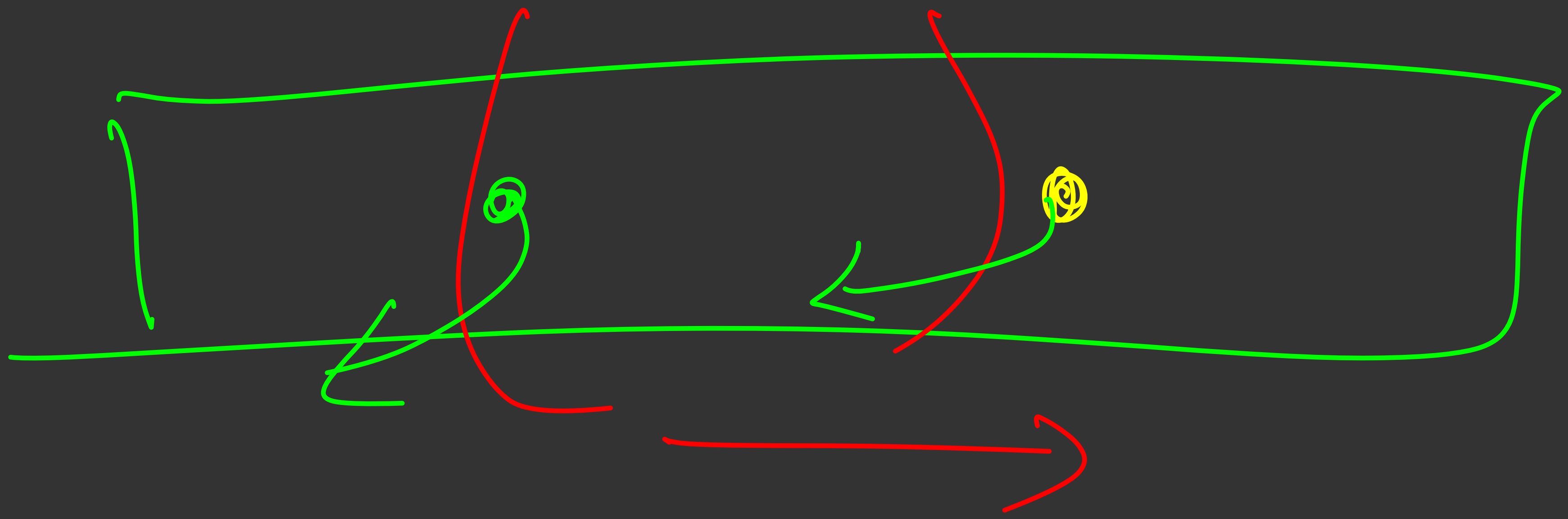
Good segments

① fixed size sliding window

② variable size sliding window

storing some info related to current window to get your answer for the current window

# to able to remove and add element into the window efficiently



~~1~~ 2 2 3 4 5

✓ Set ① ✓

✓ multiset ② ✗

vector ③ ✗ ✓

map ④

① relevant info ✓

② Adding new element

③ Removing some element ✓

{ ~~1:1~~ 2:2 3:1 4:1, 5:1 }

map ← insert

```
m[val]++;
```

→ remove

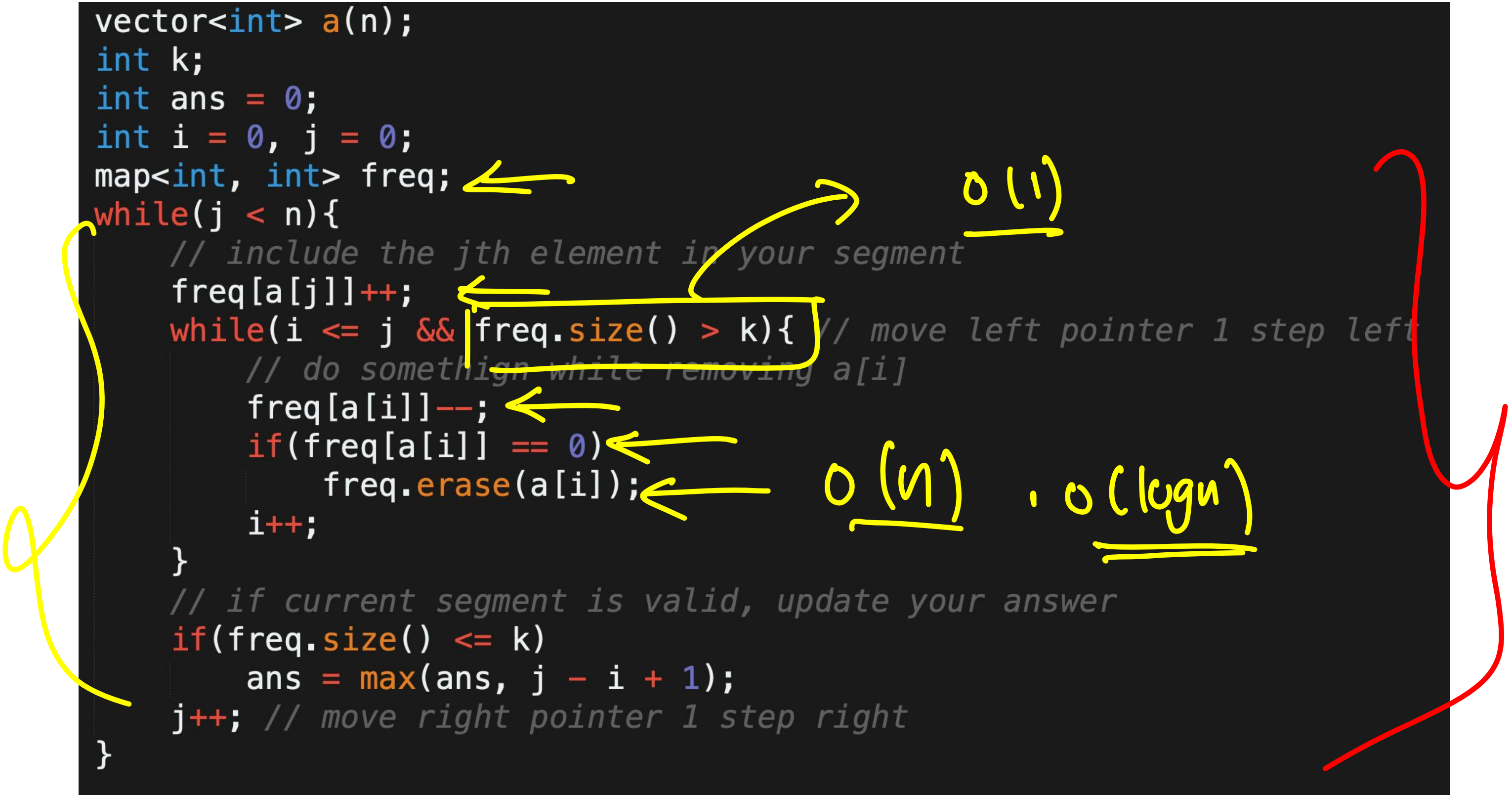
```
m[val]--;
if (m[val]==0)
    m.erase(val);
```

distinct element → m.size();

problem → data structure

data structure → problem

# Good Segments Technique Problem 2



```
vector<int> a(n);
int k;
int ans = 0;
int i = 0, j = 0;
map<int, int> freq;
while(j < n){
    // include the jth element in your segment
    freq[a[j]]++;
    while(i <= j && freq.size() > k){ // move left pointer 1 step left
        // do somethign while removing a[i]
        freq[a[i]]--;
        if(freq[a[i]] == 0)
            freq.erase(a[i]);
        i++;
    }
    // if current segment is valid, update your answer
    if(freq.size() <= k)
        ans = max(ans, j - i + 1);
    j++; // move right pointer 1 step right
}
```

O(1)

O(n) . O(logn)

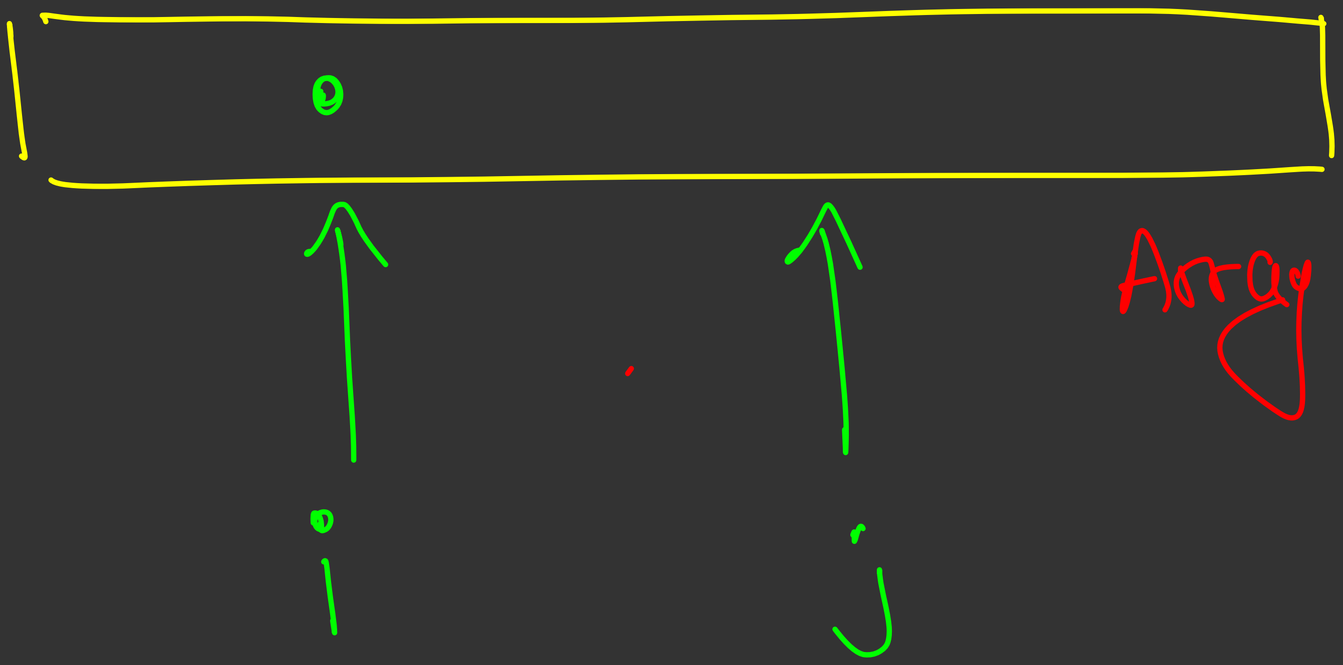
# Good Segments Technique (Decreasing)

- Given an array of positive integers find the length of smallest subarray with sum of elements >= K

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 1 | 1 | 4 | 1 | 2 | 1 | 1 |

Good segment → subarray with sum $\geq k$

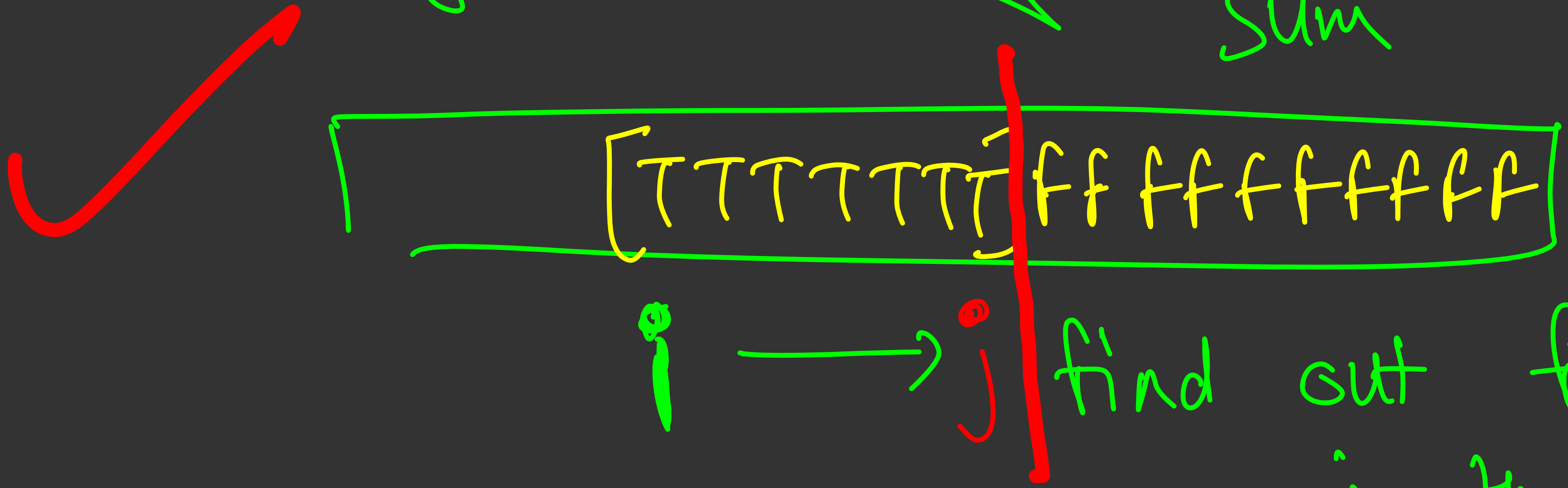
$k = 5$



sum ( i to j ) $\geq$ k   sum(i to j-1) < k

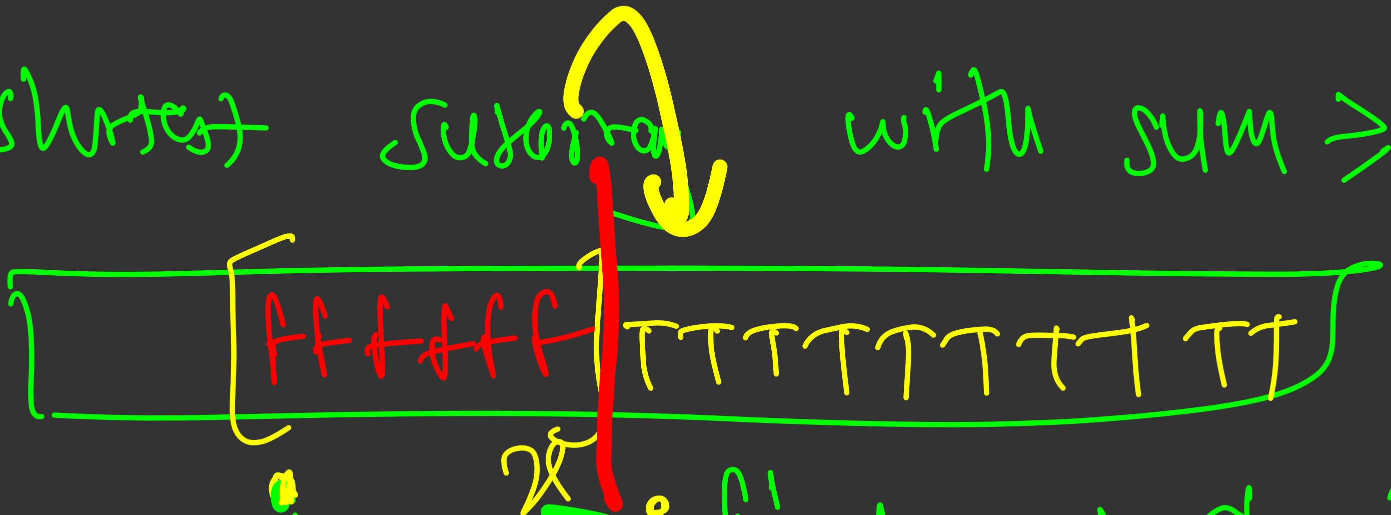
sum ( i to j+1 ) $\geq$ k   sum( i to j+2 ) $\geq$ k

① longest subarray with sum $\leq k$



i → j find out farthest j that works

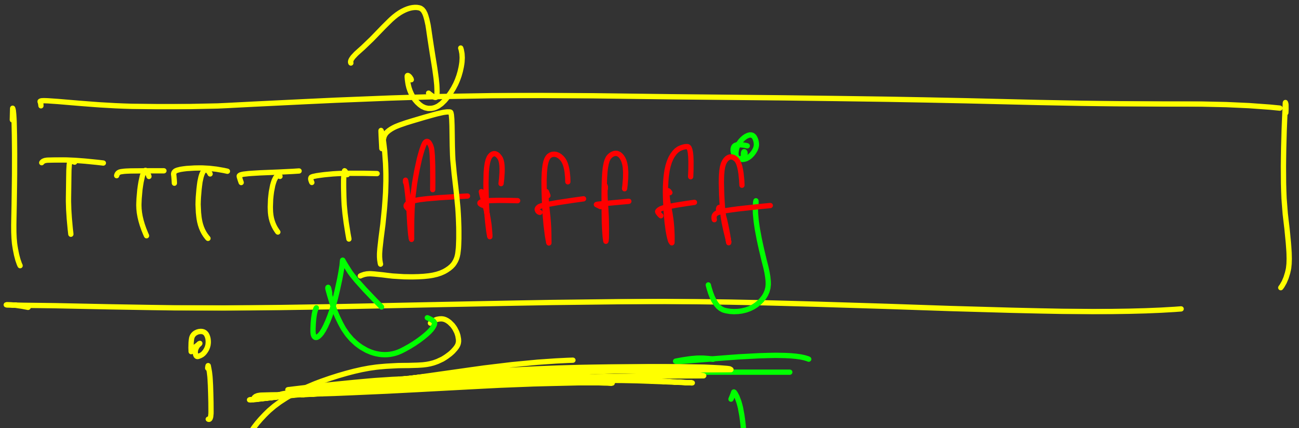
② shortest subarray with sum $\geq k$



i → j find out closest j that works

↳ find out the farthest $x$ which doesn't work

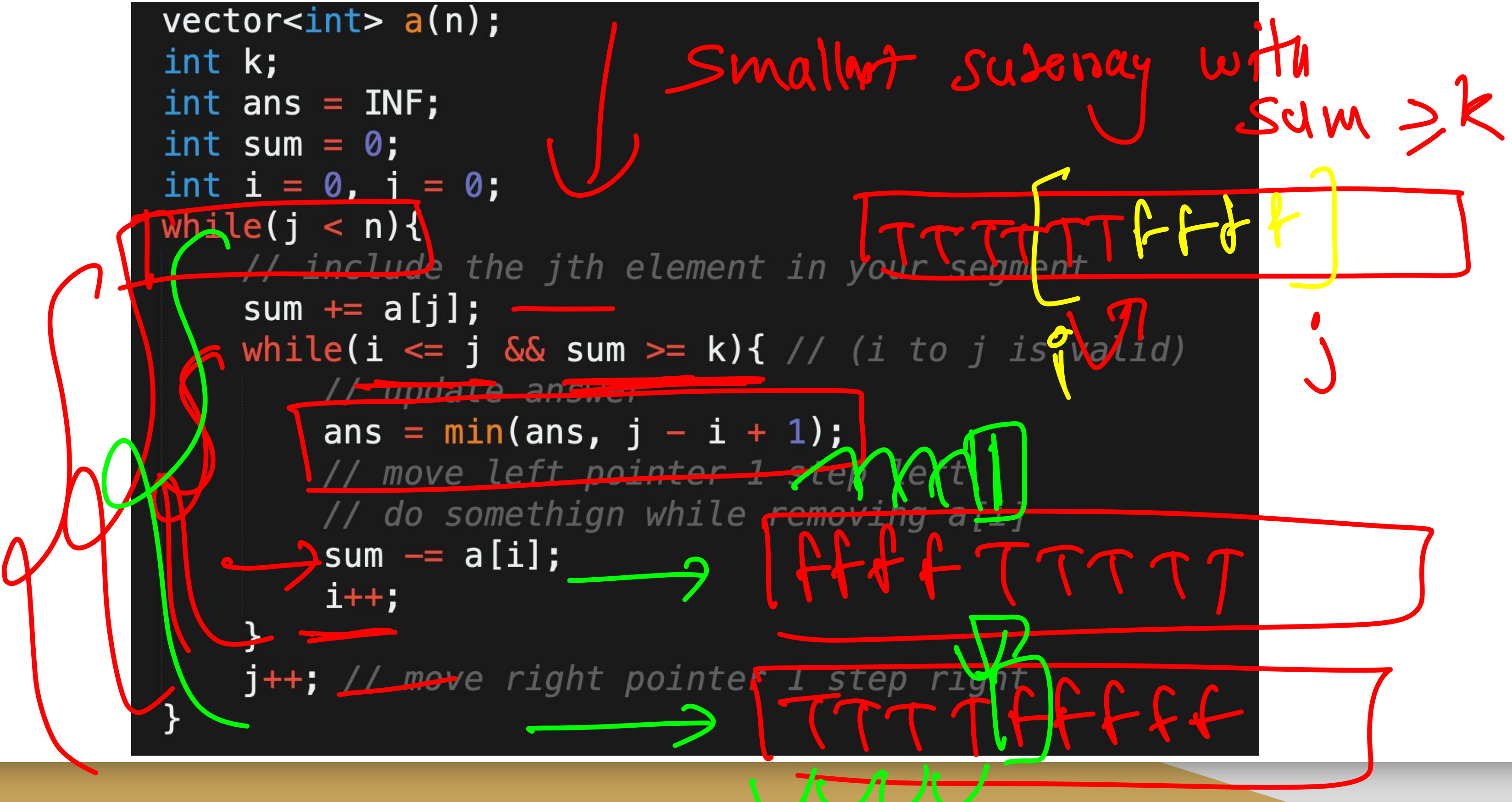
then $j = x+1$



Smallest subarray with sum $\geq k$

ending at j

# Good Segments Technique Problem 3

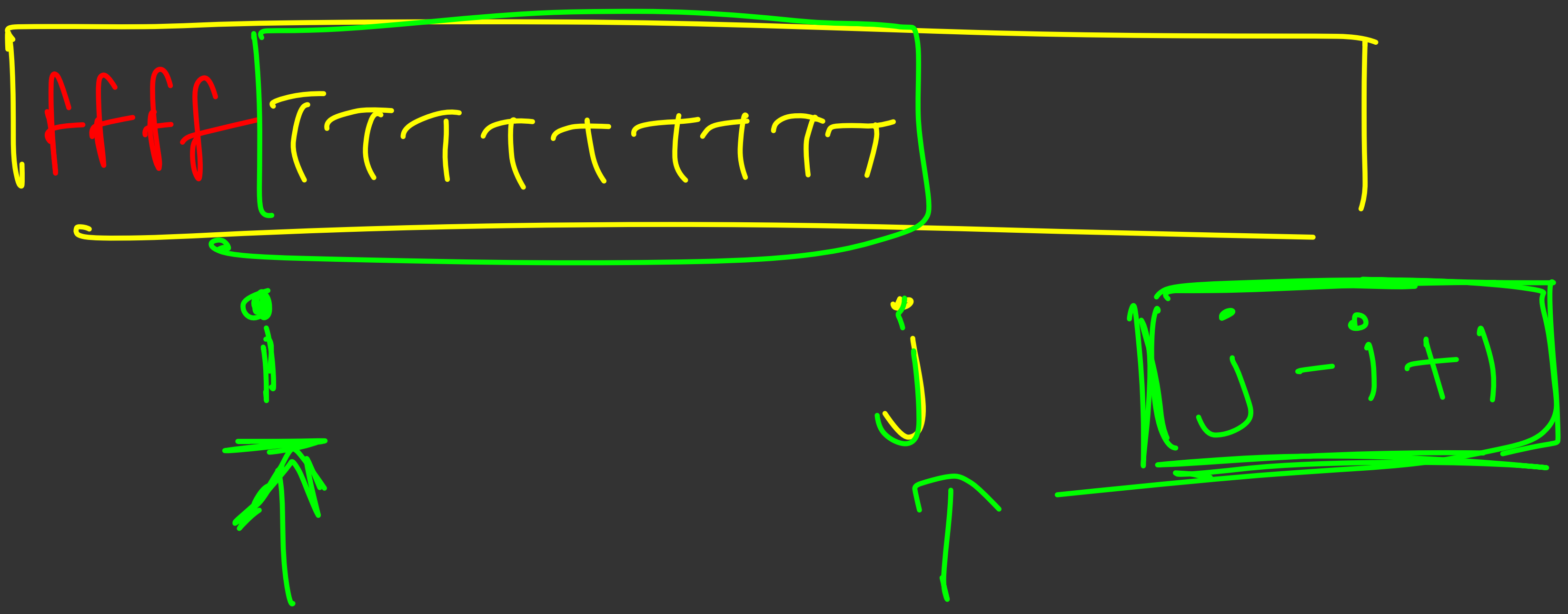


```
vector<int> a(n);
int k;
int ans = INF;
int sum = 0;
int i = 0, j = 0;
while(j < n){
    // include the jth element in your segment
    sum += a[j];
    while(i <= j && sum >= k){ // (i to j is valid)
        // update answer
        ans = min(ans, j - i + 1);
        // move left pointer 1 step left
        // do somethign while removing a[i]
        sum -= a[i];
        i++;
    }
    j++; // move right pointer 1 step right
}
```

Smallest subarray with sum $\geq k$

① find longest subarray which has sum ≤ k

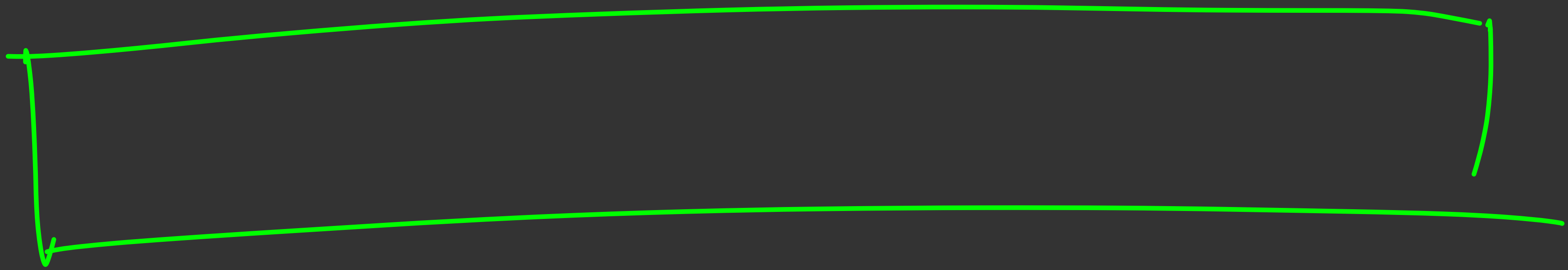
# no. of subarray with sum ≤ k



$$\text{sum}(i \text{ to } j) \leq k$$

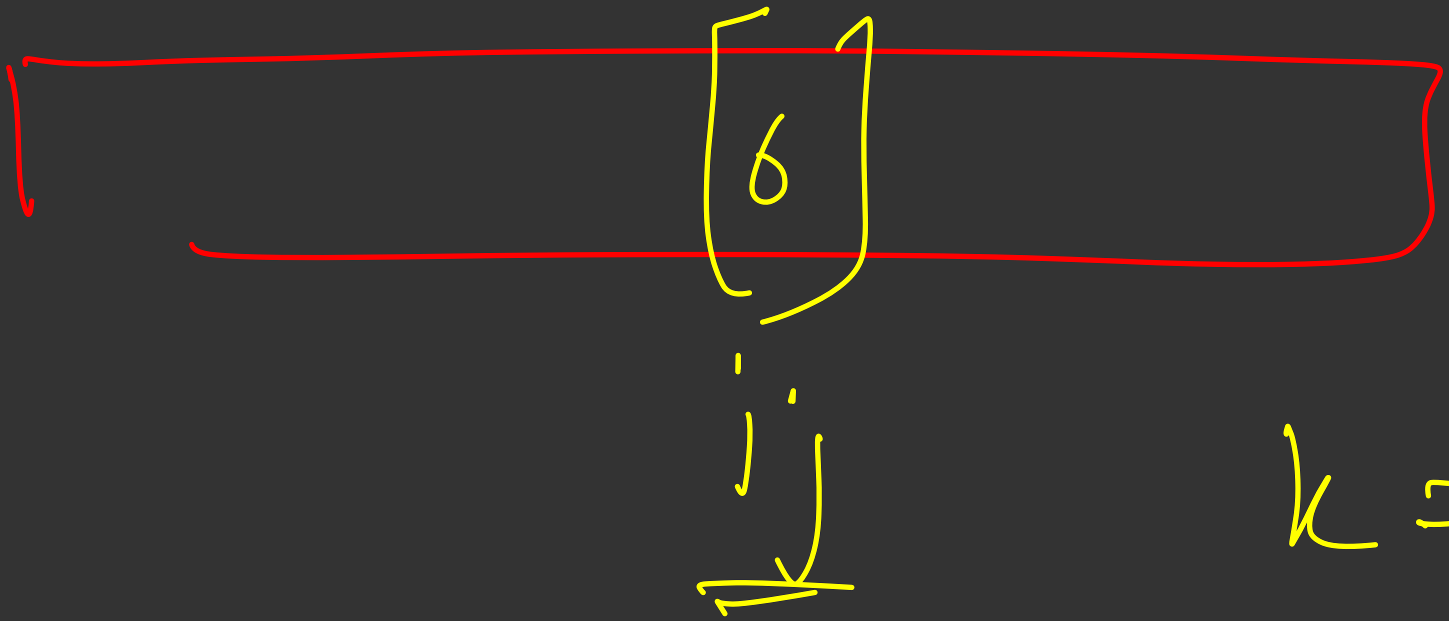
$$\text{sum}(i-1 \text{ to } j) > k$$

find best subarray



(find best subarray ending at j for every j) best no. of good subarrays

( find out no. of good subarrays ending at j ) and add them up



k = 8

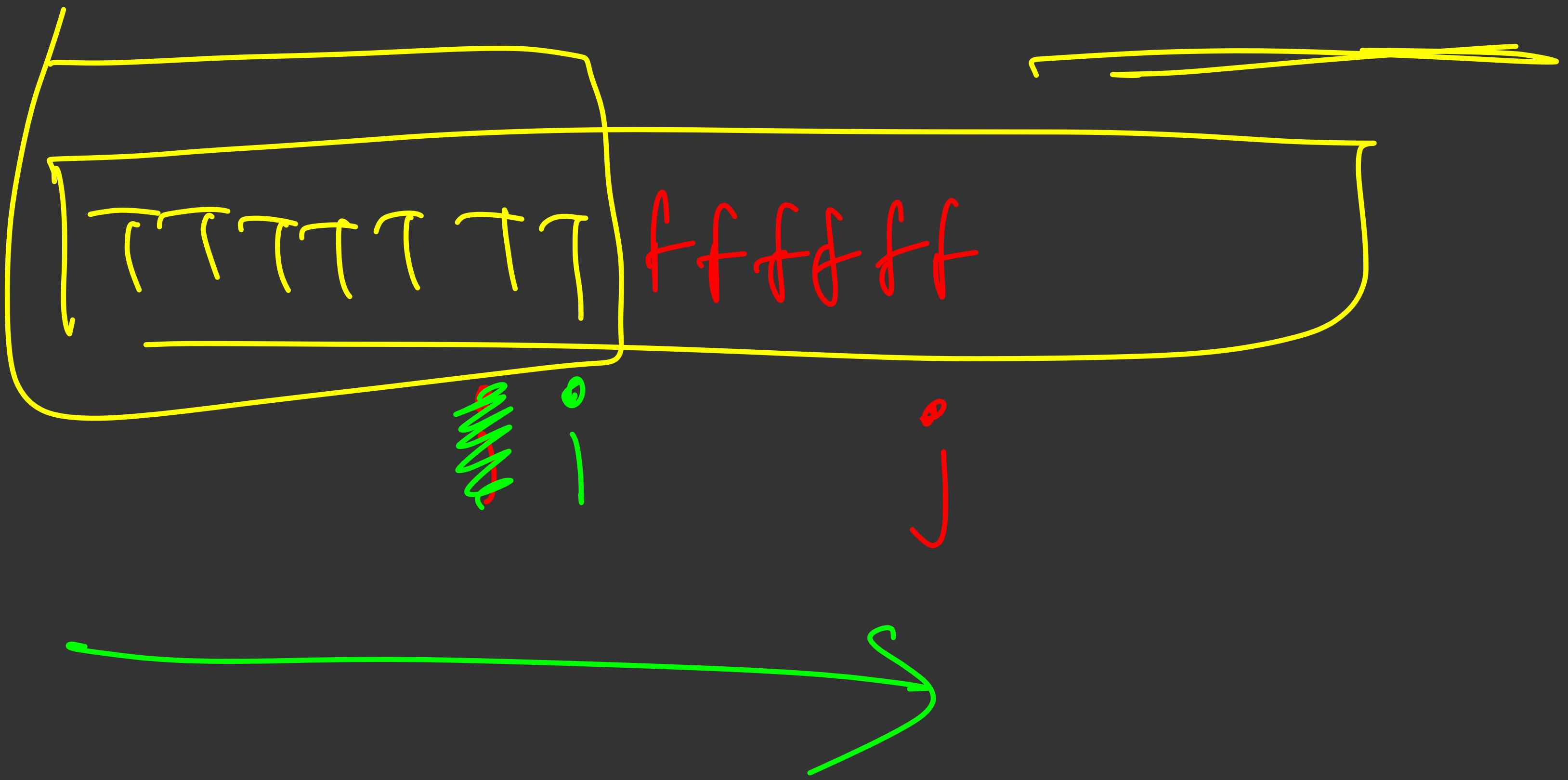
# Good Segments Technique General Trick

- Condition 1: If Segment [L:R] is good then all the segments enclosed within in will be good
  - Increasing
- Condition 2: If Segment [L:R] is good then all the segments enclosing it will be good
  - Decreasing technique
- Do not use binary search for these problems now!

# Good Segments Technique (Number of Segments?)

- How to find number of good segments?
  - Let’s solve the first problem.
    - ■ Number of subarrays with sum <= K
- Simple! Just ~~multiply~~ add (j - i + 1) for every j

find no. of subarray with sum ≥ k


T T T T T T T T T
f f f f f f
i
j