Sliding windows

Two Pointers

variable size sliding window

fixed size sliding window

- Priyansh Agarwal

① Binary Searching on Answer

Arrays   o(nlogn)

↓

② using 2 pointers   o(n)

Cf Problem Tags

→ two pointer

✓✓ ad hoc 2 pointer problem

good segment category ←

Two Pointers

- Widely used in Competitive Programming ✓✓
- Optimization Technique $O(n\log n) \rightarrow O(n)$ using 2 pointers
- Most Two Pointer problems can be solved using Binary Search
- Useful for a lot of array based problems sudarray
- Super useful for interviews too

Binary Search ⟷ 2 pointers

Given 2 sorted arrays, merge them into one single array keeping the elements sorted

$O(n)$

1	4	5	9

$O(m)$

2	3	6	10

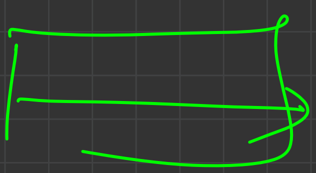
1	2	3	4	5	6	9	10

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

Second Approach: Use 2 pointers

$O(n+m)$

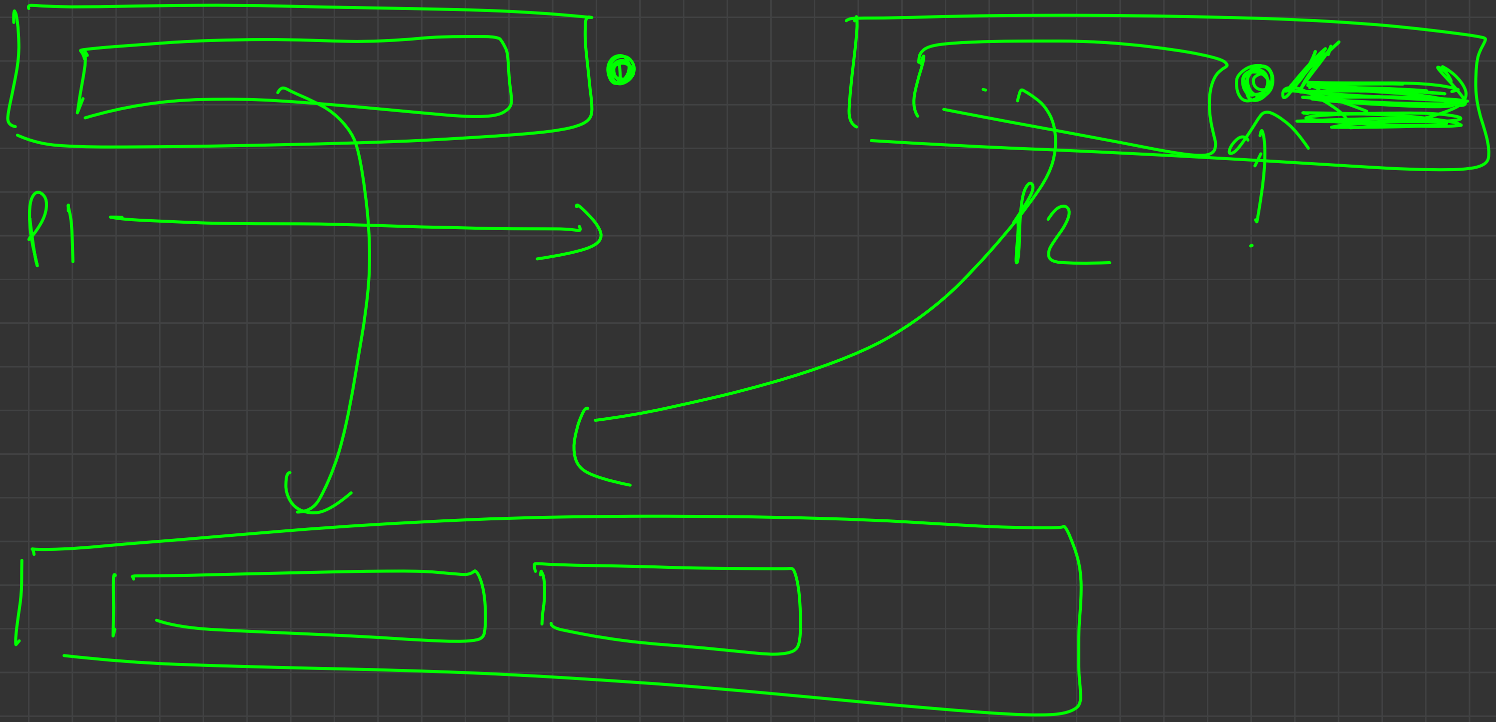
Merge Sort



Best Case	Avg Case	Worst Case
nlogn	nlogn	nlogn

Quick Sort

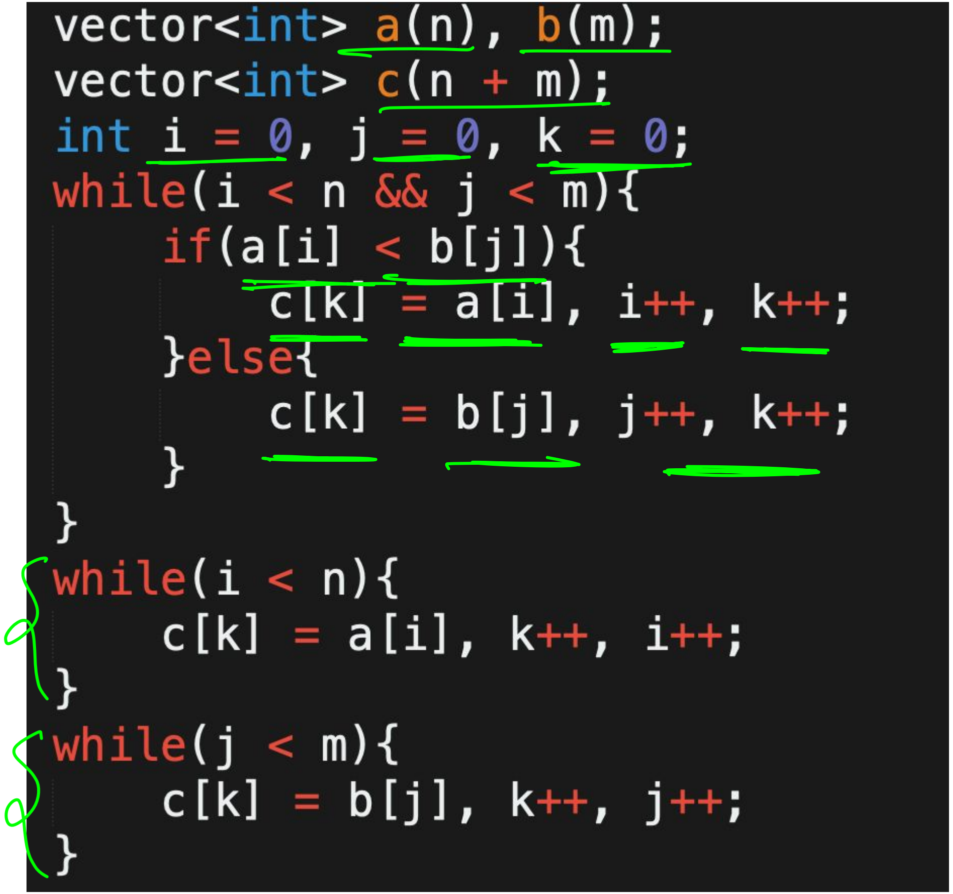
O(n)	O(nlogn)	$O(n^2)$


P1
P2

Solution using 2 pointers

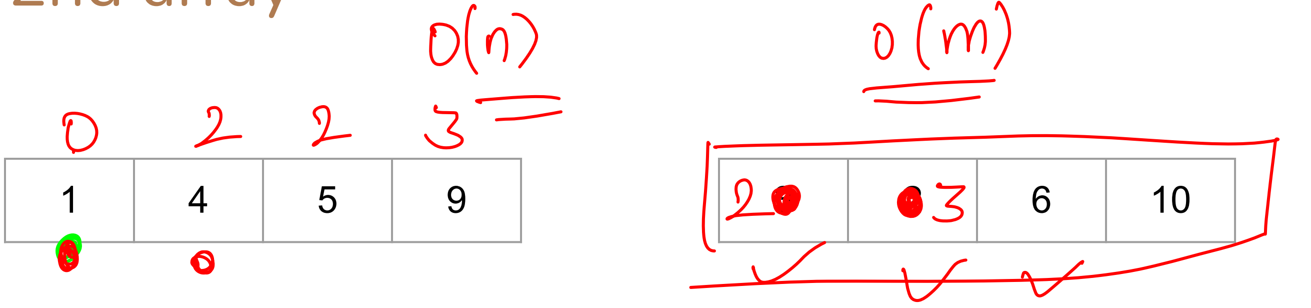
Maintain 2 Pointers, i and j both starting from the left ends of the arrays

Keep pushing the smaller of the 2 elements from the 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++;
}
```

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



1	4	5	9

2	3	6	10

O(n log m)

First Approach: Binary Search for each elements

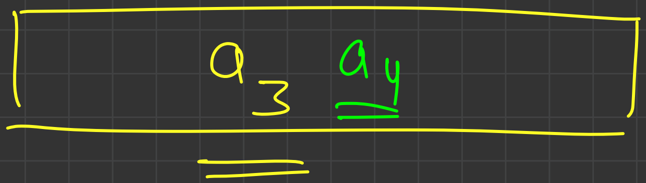
Second Approach: 2 pointers

seem ← $a[i] \le a[i+1]$

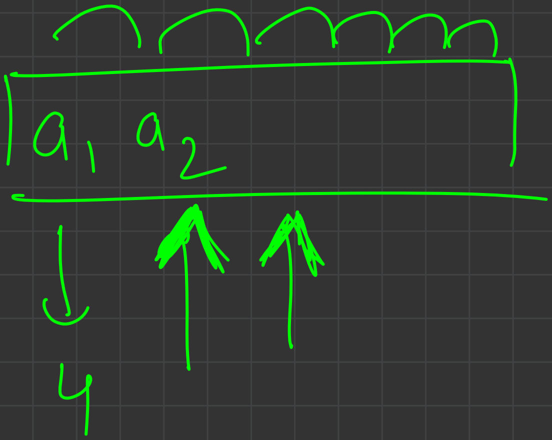
$a[i] \le a[i+1]$ $b[i] \le b[i+1]$

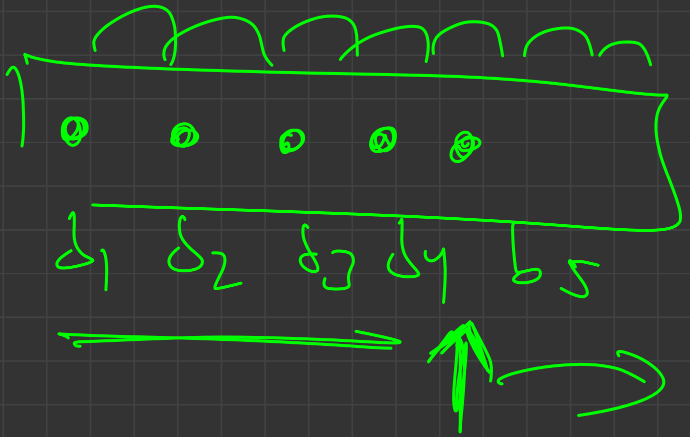
5 element

if 5 element in B are smaller than $a[i]$ how many element in B would be smaller than $a[i+1]$ ⇒ ≥ 5



if 6 elements are smaller than a_3





Solution using 2 pointers

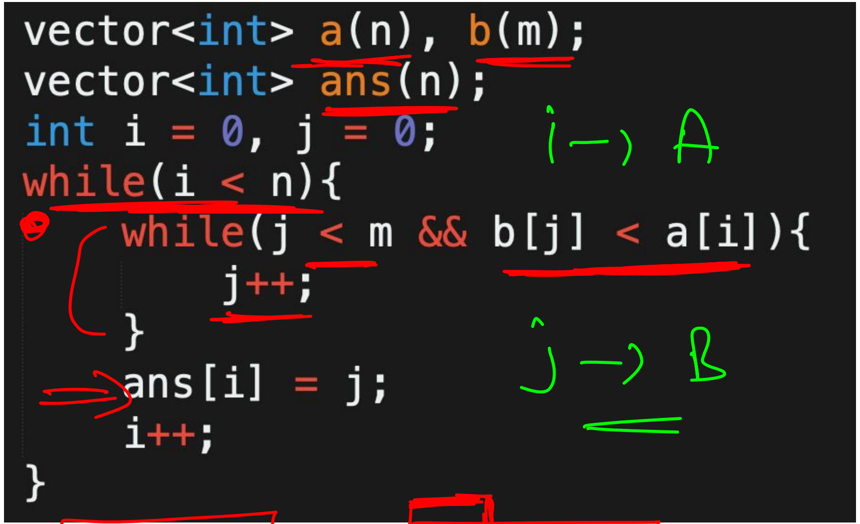
O(n+m)

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

≥ 5

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

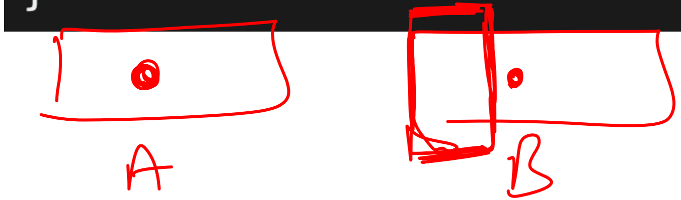
Having 2 pointers and both only move right. Time complexity?



```
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++;
}
```

i → A

j → B



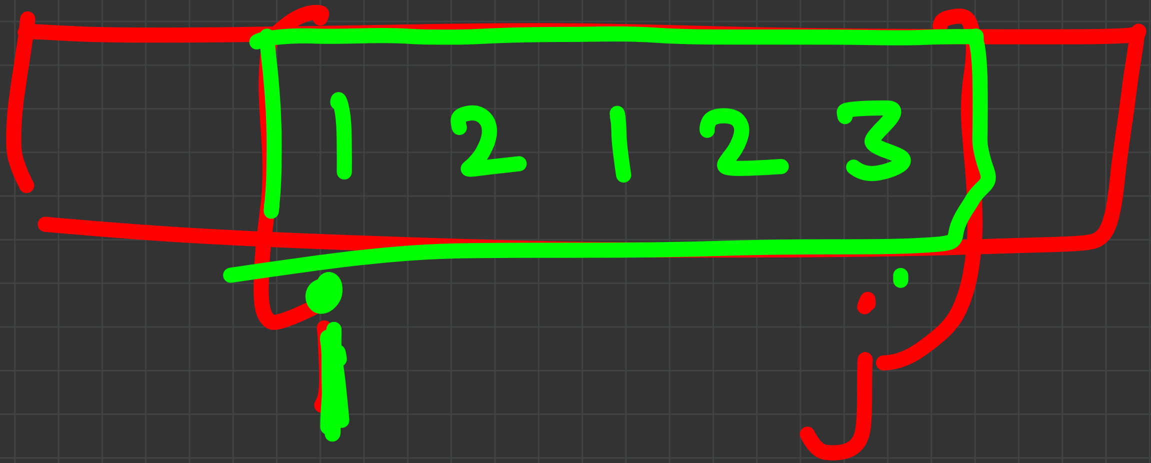
V.V. Intuitive

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

Good Segments Technique (Increasing)

- Given an array of positive integers find the ~~length of longest~~ no. of subarrays with sum <= K
- Given an array find the length of longest subarray with not more than K distinct elements



3

2 3

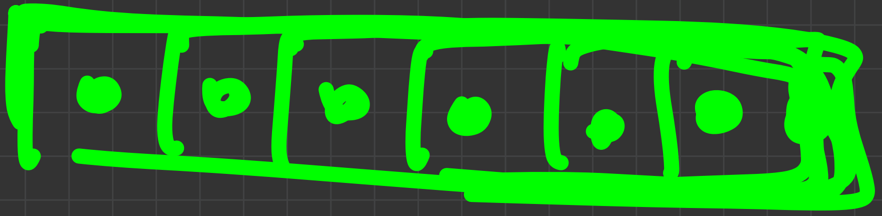
1 2 3

2 1 2 3

1 2 1 2 3

$$\text{sum} \le k$$

$$j-i+1$$



array of size n

n

2	1	1	3	4	2

$k = 7$

longest subarray such that sum $\leq k$

Binary Search solution ①

Two Pointers solution ②

① Binary Searching Solution

→ Binary search on length of subarray

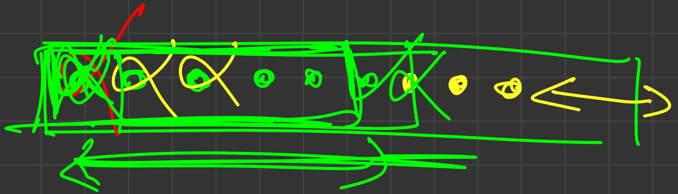
there exist a subarray of size x with sum $\leq k$

T	T	T	T	F	F	F	F
1	2	3	4	5	6	7	8

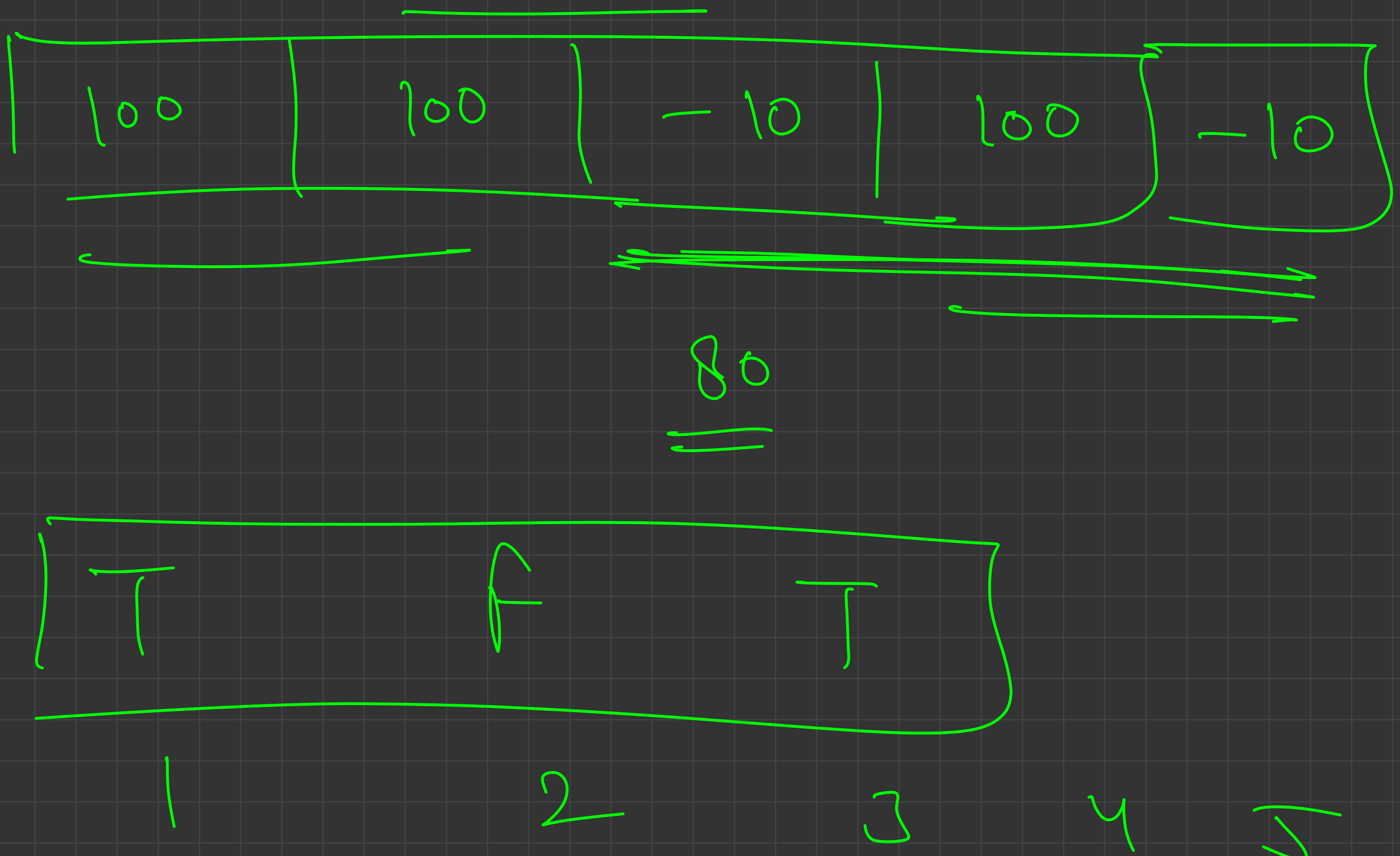
guess a length X and then check if there exists a subarray of size X with sum ≤ k

bool possible (int x)

O(n) time



```
start = 1 , end = n, ans = 0
while (start ≤ end)
    mid = (start + end)/2
    if (possible(mid)) {
        ans = max(mid, ans)
        start = mid + 1 }
    else
        end = mid - 1
```


100
100
-10
100
-10
80
T
F
T
1
2
3
4
5

[T T T T F F F F T T T T T

T T T T F F F F F

Binary Searching Solution

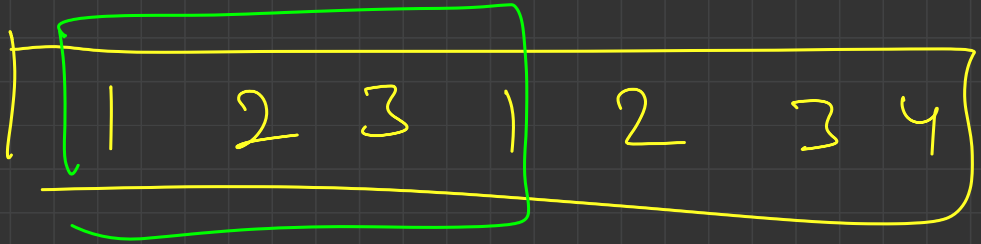
$\underline{O(n \log n)}$

↓

O(n) solution using 2
pointer

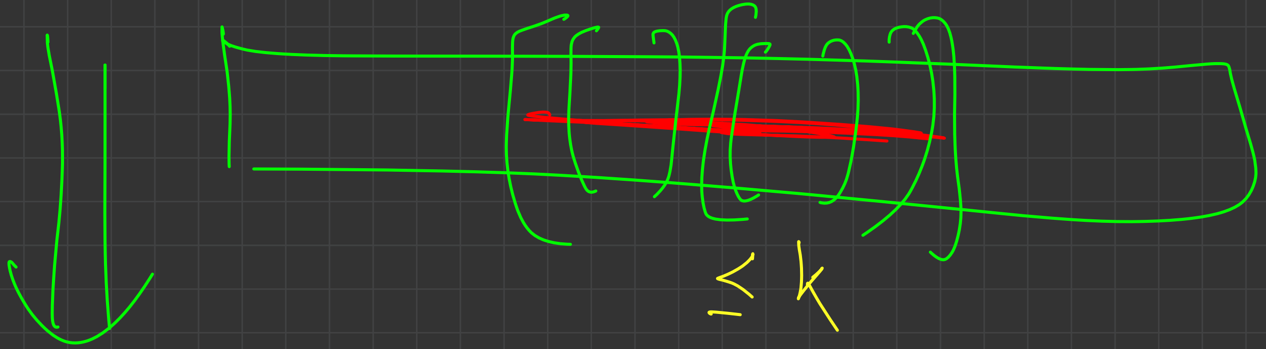
Good segment :

k = 7

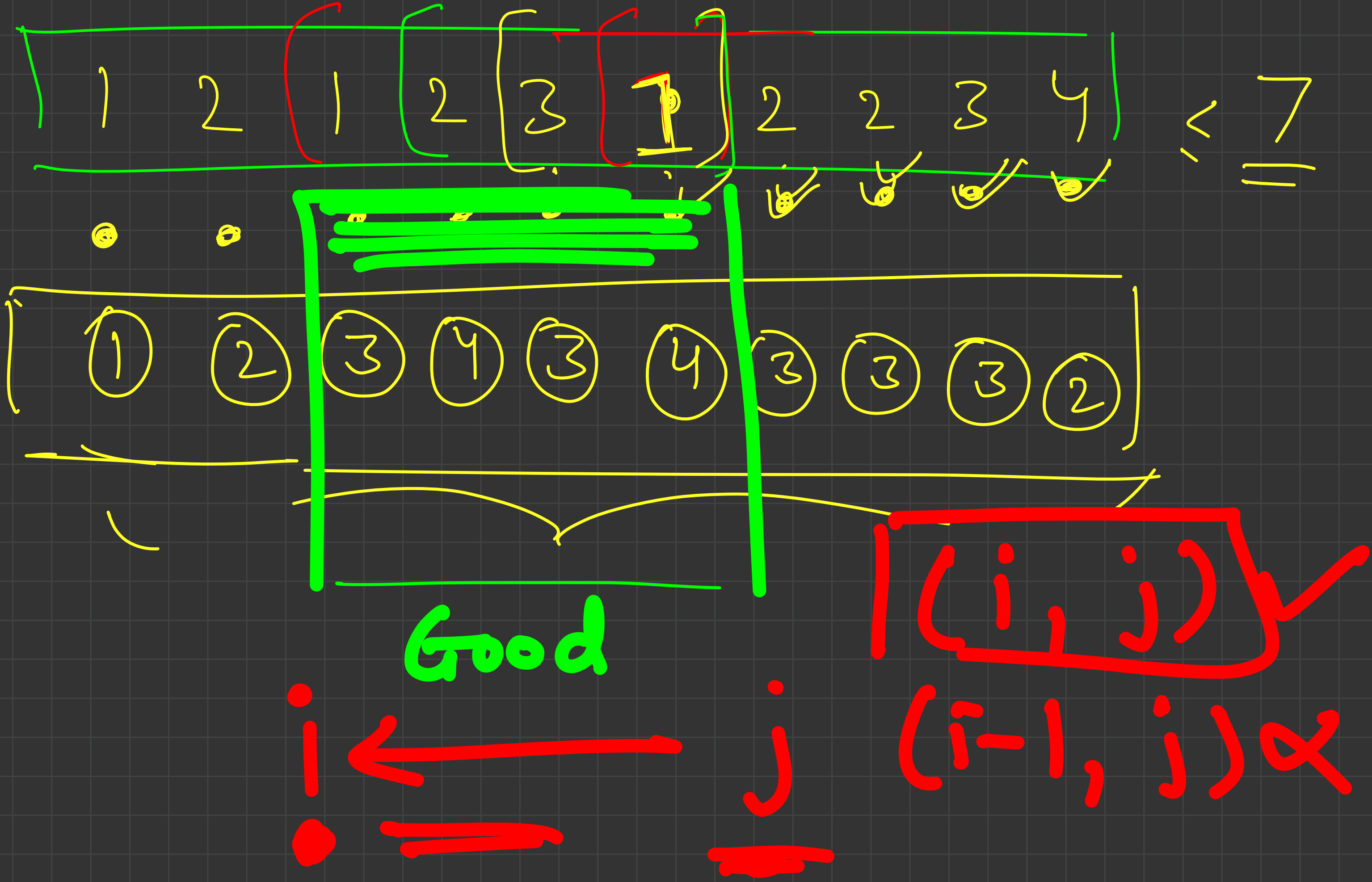


1 2 3 1 2 3 4

① If subarray X is good then all subarrays containing X will be good



② If a subarray X is good then all segment contained in X are good


1 2 1 2 3 1 2 2 3 4
< 7
1 2 3 4 3 4 3 3 3 2
Good
(i, j)
(i-1, j)
i
j

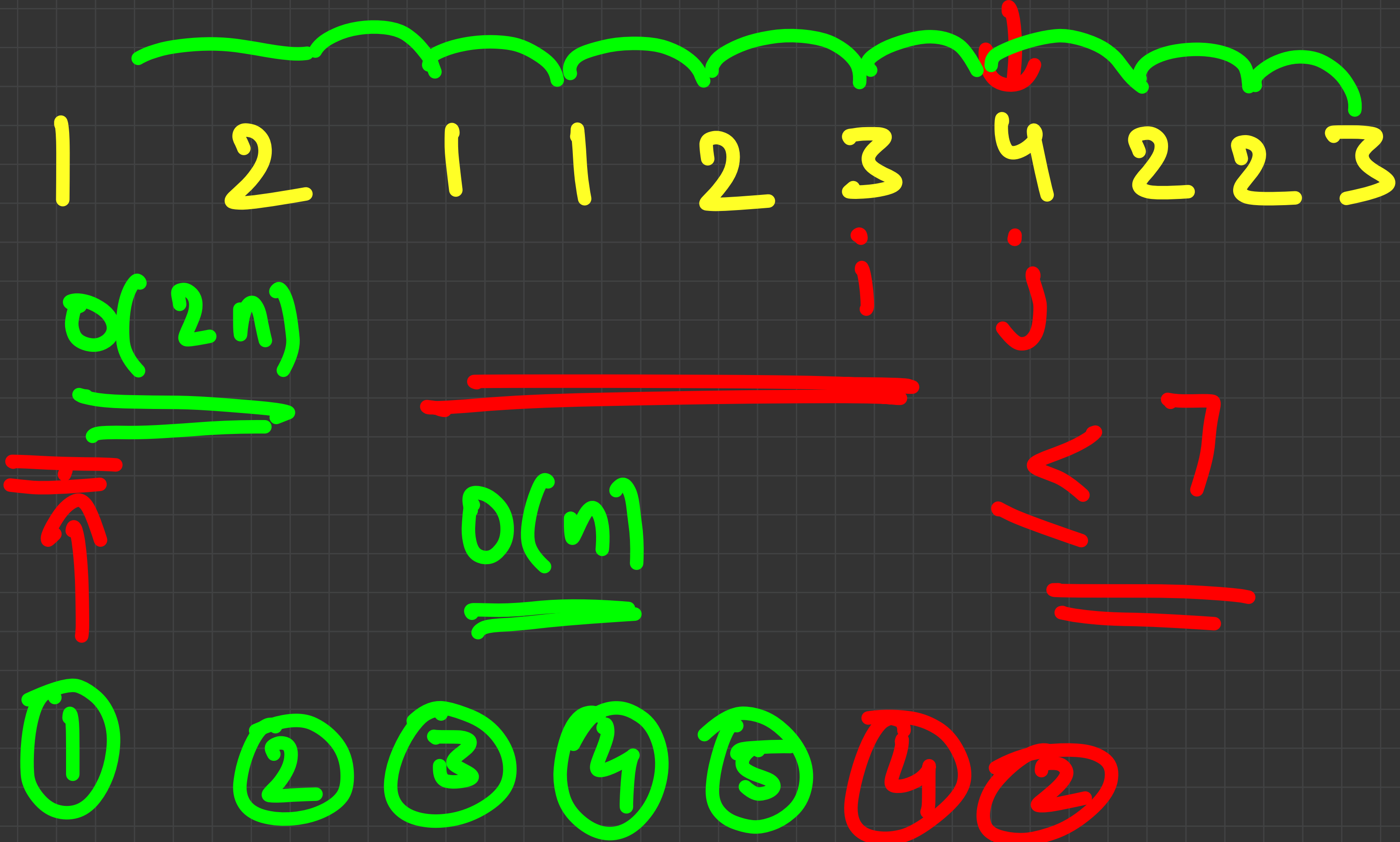
←

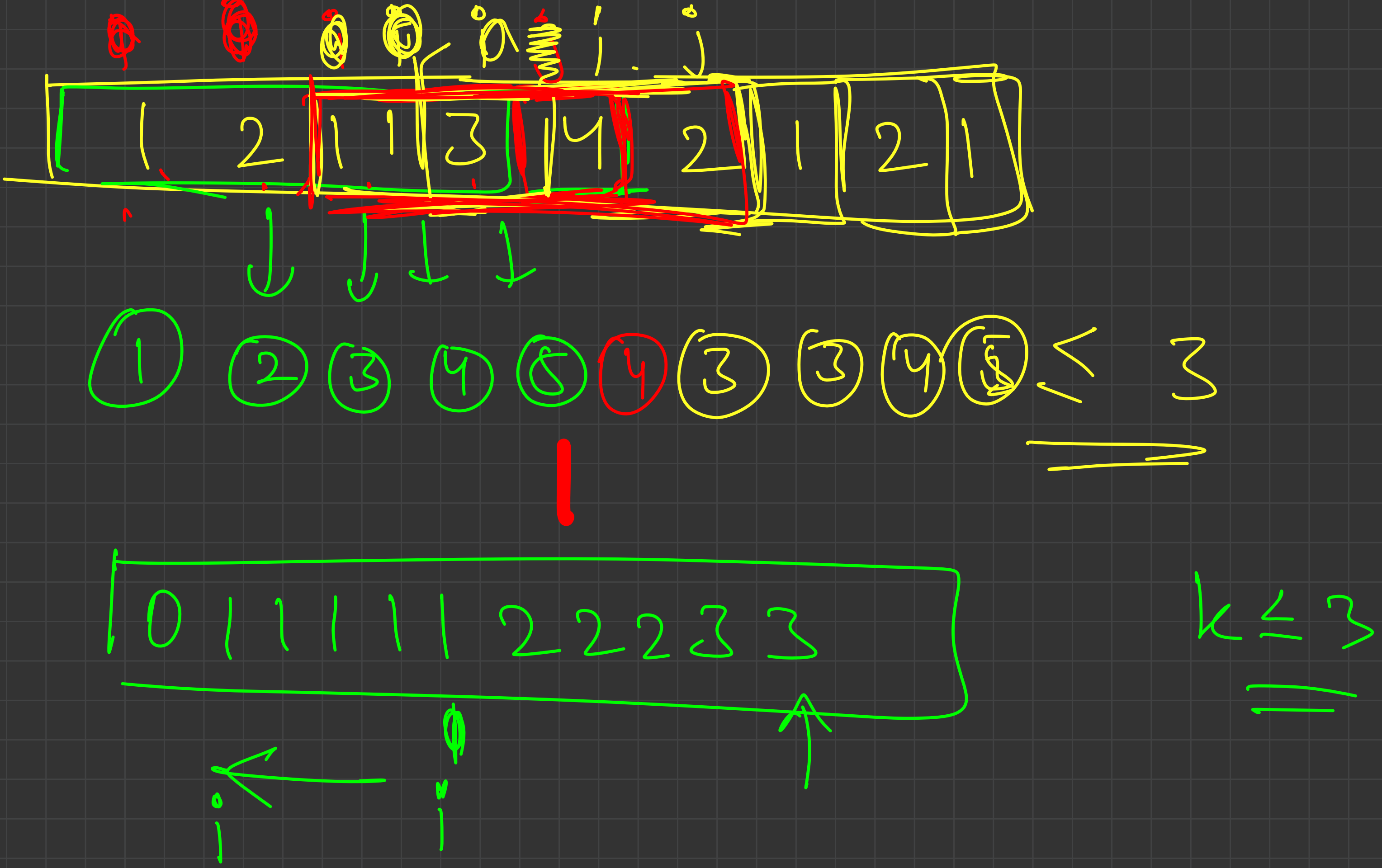
✓

$\underline{j+1}$

$(i-1, j) > k$

$(i-1, j+1) > k$


1 2 1 1 2 3 4 2 2 3
i
j
O(2n)
O(n)
< 7
1
2
3
4
5
4
2

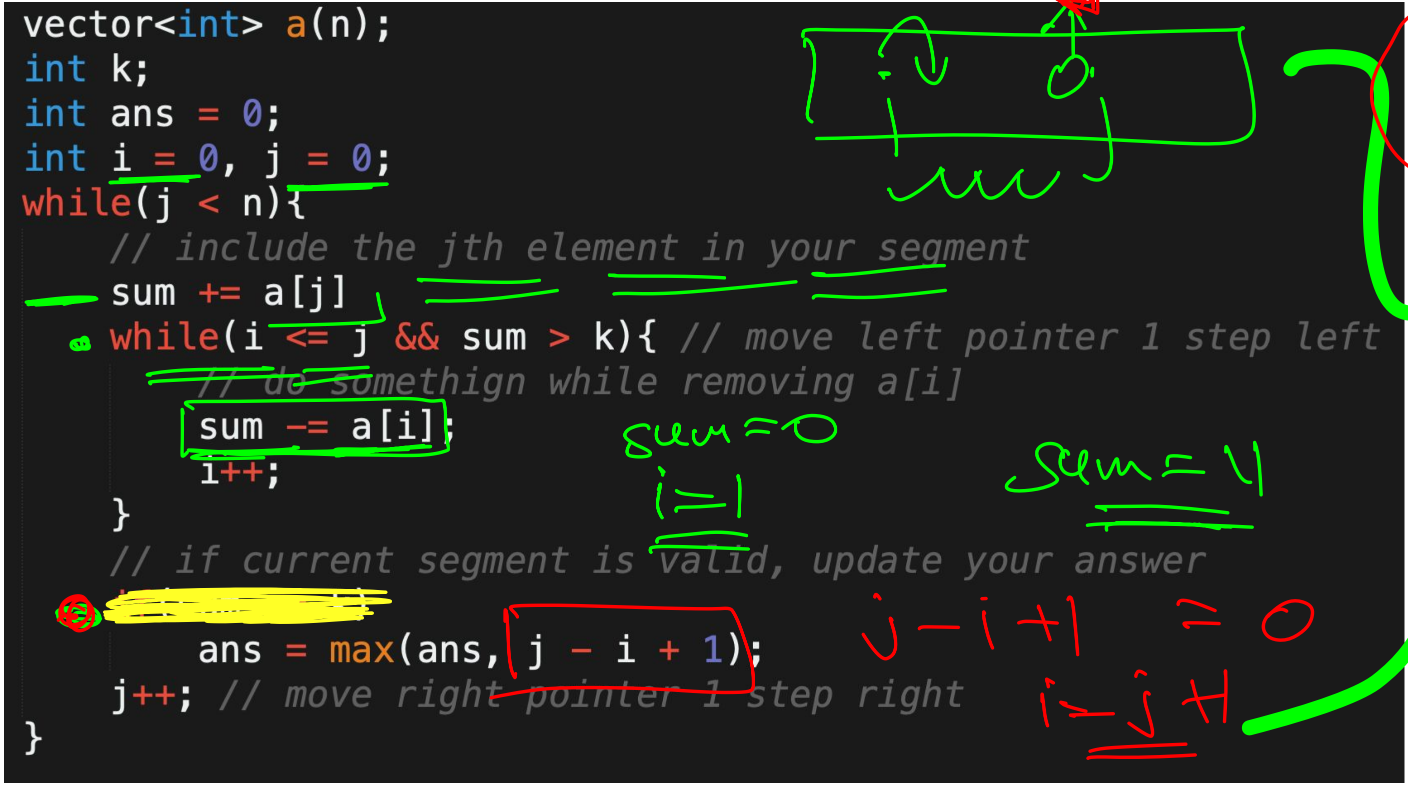


i
j
1 2 1 1 3 4 2 1 2 1
1 2 3 4 5 4 3 3 4 5 ≤ 3
0 1 1 1 1 1 2 2 2 3 3
i
j
k ≤ 3

Good Segments Technique Problem 1

11	12	13	14	15

j i



```
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
    [redacted]
        ans = max(ans, j - i + 1);
    j++; // move right pointer 1 step right
}
```

sum ≤ 10

sum = 0

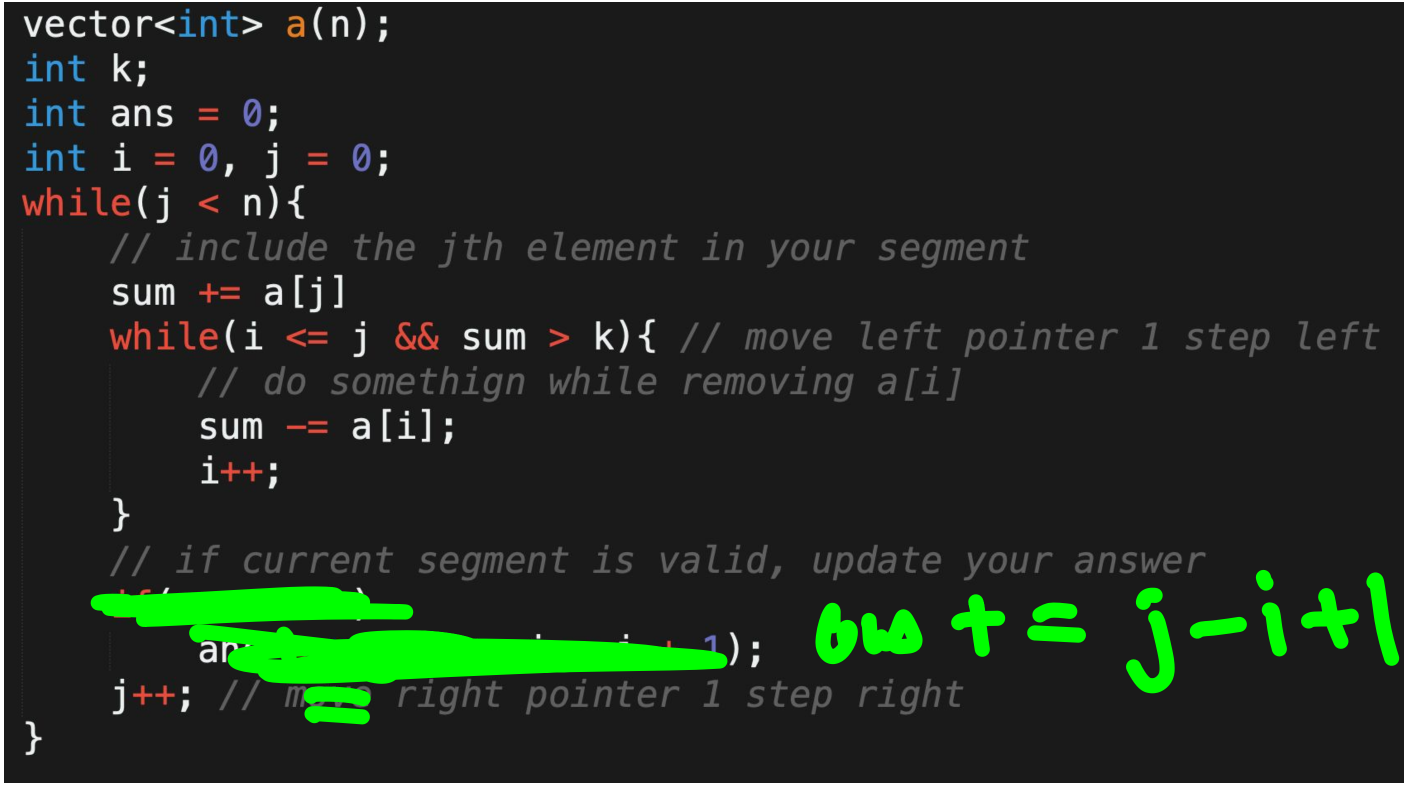
i = 1

sum = 4

j - i + 1 = 0

i = j + 1

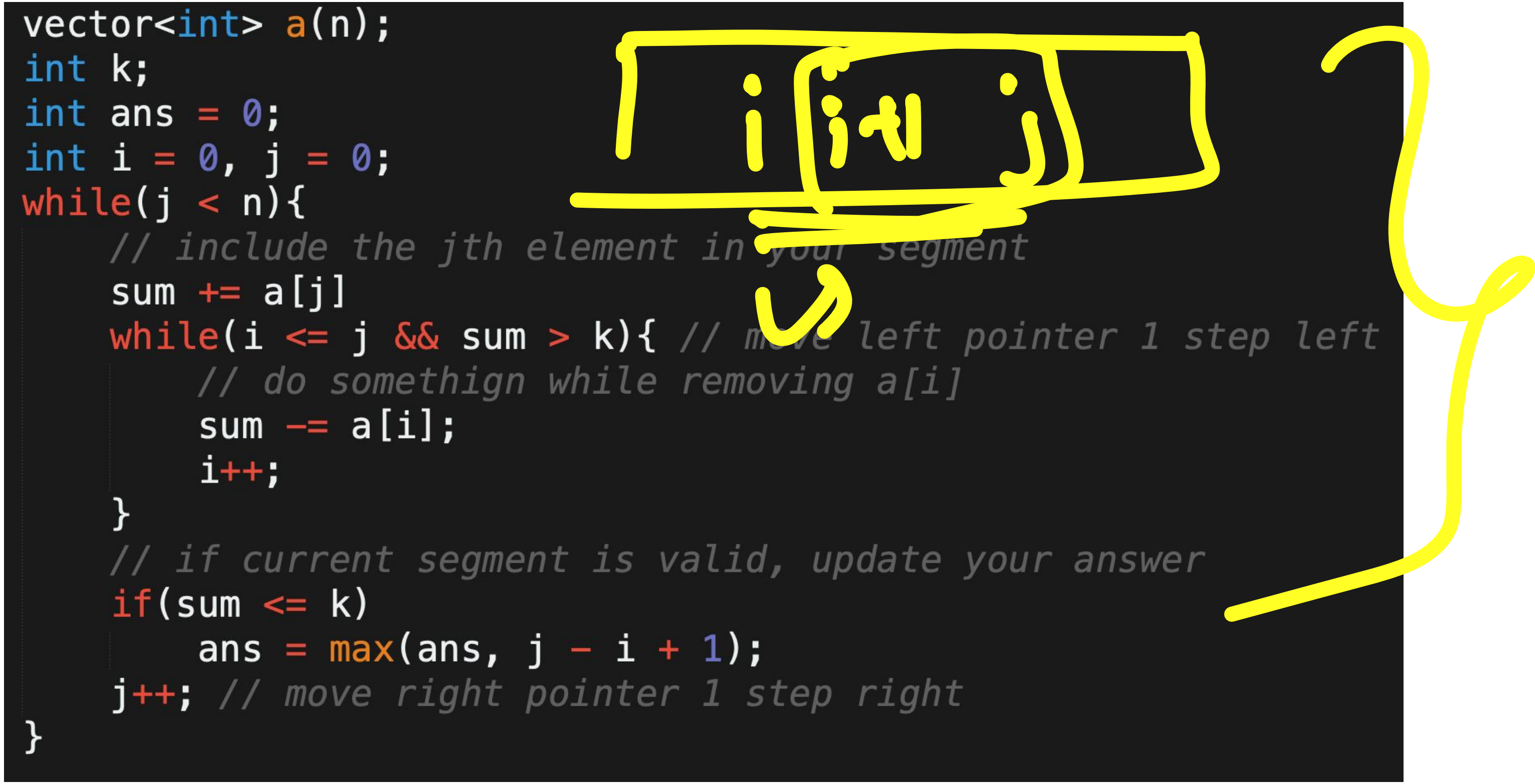
Good Segments Technique Problem 1



```
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
    [redacted]
    j++; // m[redacted] right pointer 1 step right
}
```

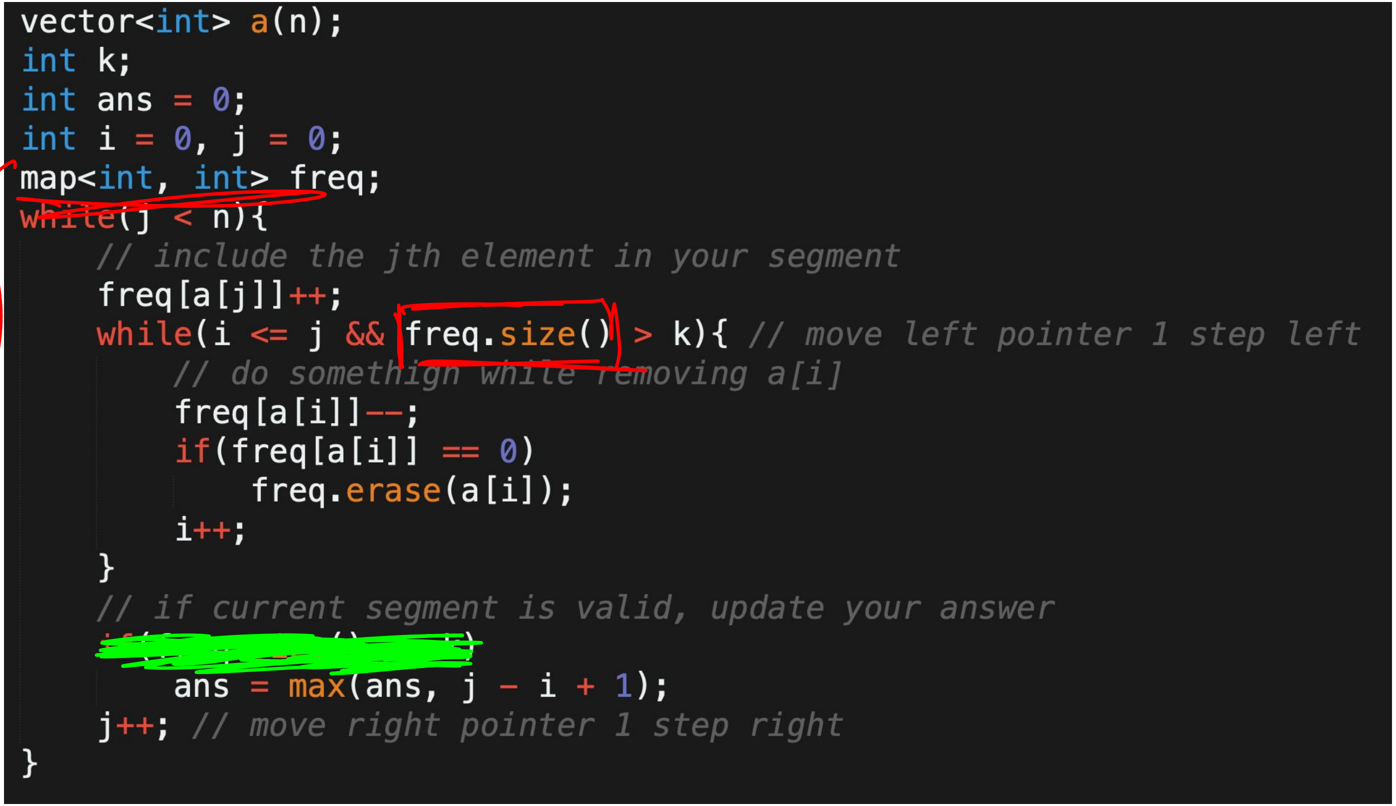
ans += j-i+1

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 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
    [redacted]
        ans = max(ans, j - i + 1);
    j++; // move right pointer 1 step right
}
```

Good Segments Technique (Decreasing)

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

T T T T F F F F

F F T T T T T

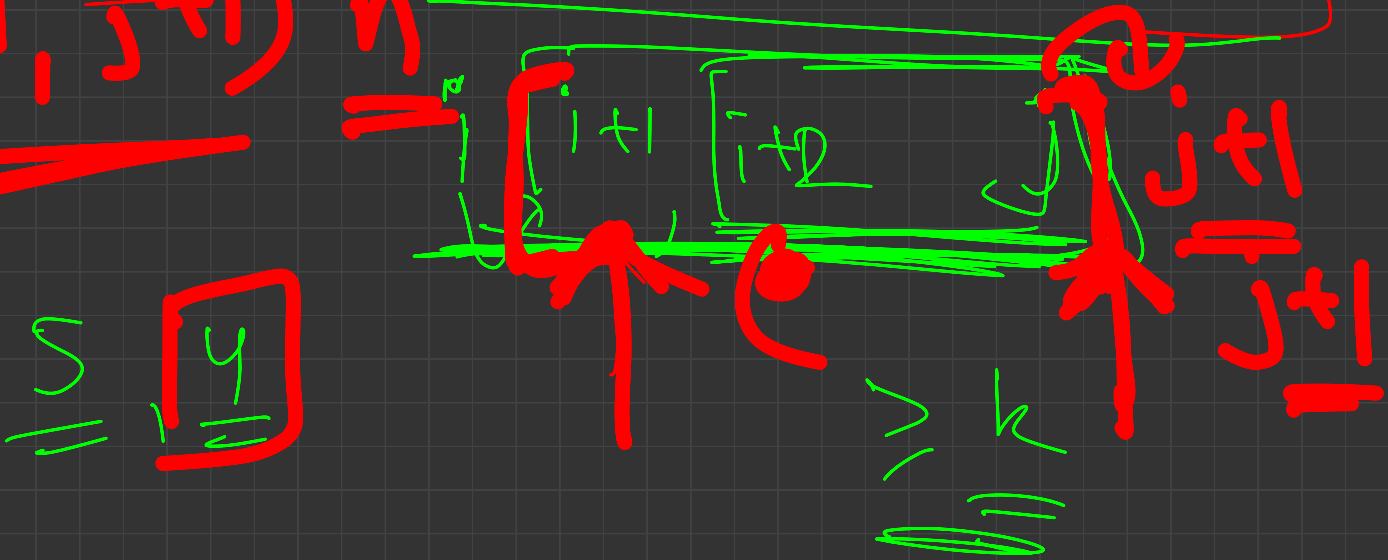
Good Segments Technique (Decreasing)

- Given an array of positive integers find the ~~length~~ no of ~~of smallest~~ subarrays with sum of elements >= K

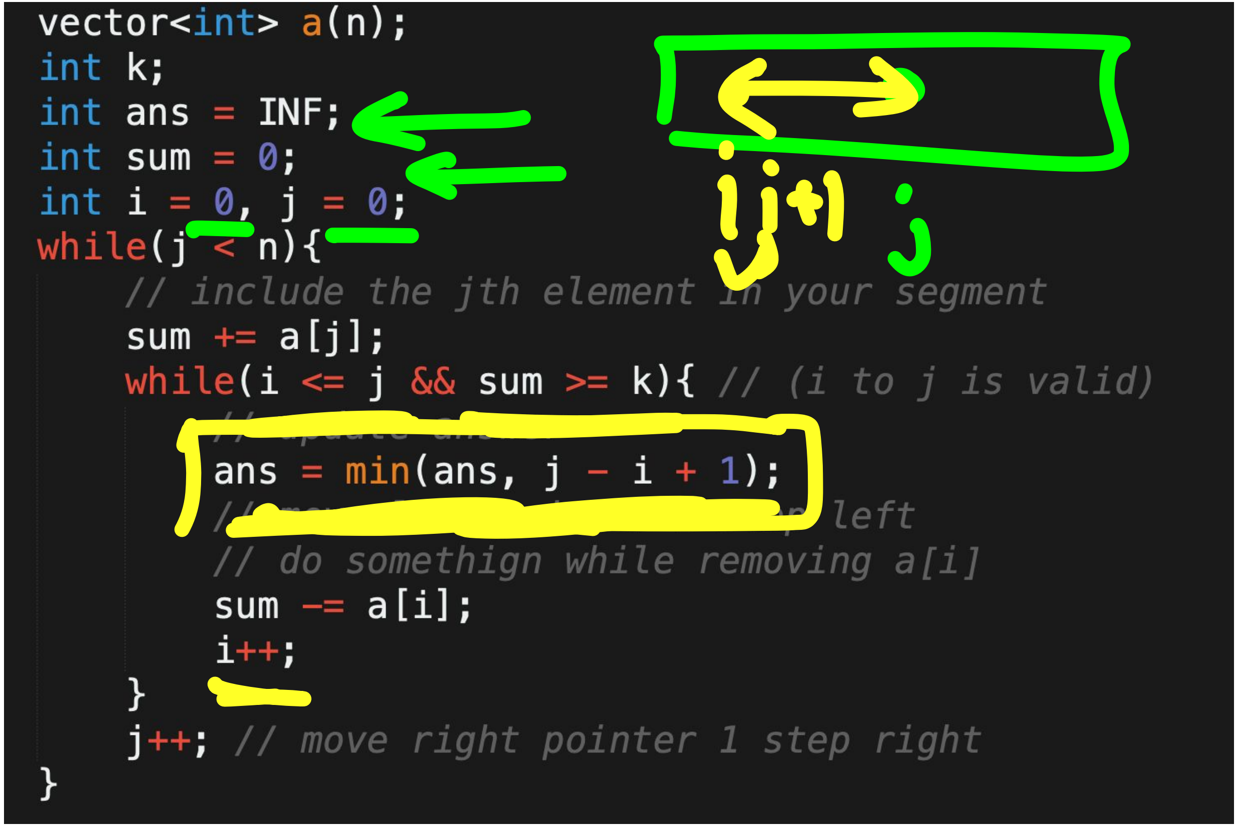
(i+1, j) n-1 ✓ (i+1, j) was working

(i+1, j+1) will also work

(i+1, j+1) n

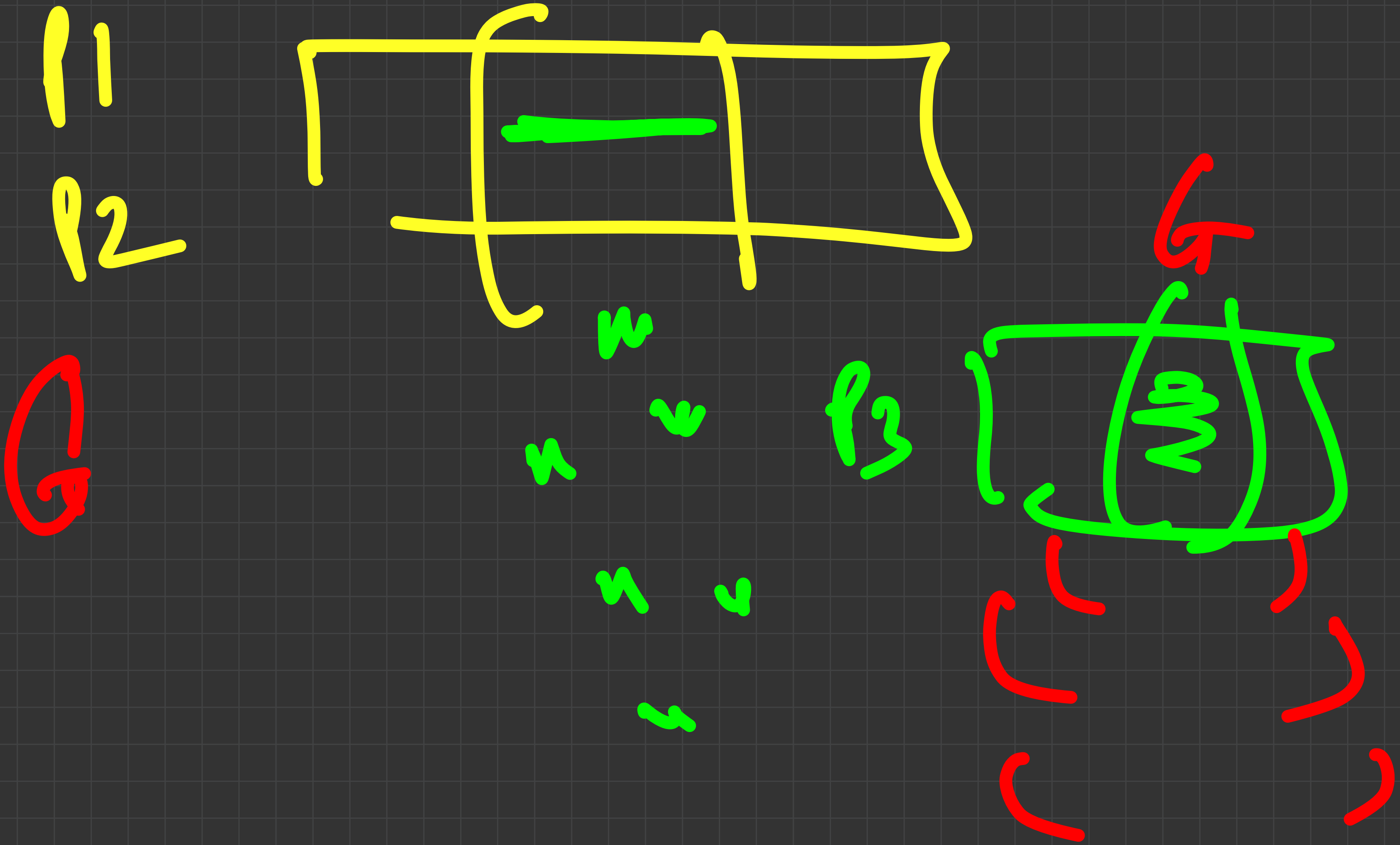


Good Segments Technique Problem 3



```
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
}
```

j+1 j


P1
P2
G
w
u
P3
G

Good Segments Technique General Trick

- Condition 1: If Segment [L:R] is good then all the segments enclosed within in will be good
  - ~~Use increasing technique~~
- Condition 2: If Segment [L:R] is good then all the segments enclosing it will be good
  - ~~Use 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 multiple (j - i + 1) for every i

Sliding Window

Class 2

- Useful for array based problems - subarray
- When to use?
- Optimization Technique
- Use of 2 pointers.
- Super useful for interviews too

Given an array, what is the maximum sum of a subarray of size k

Given an array, find the first negative number in every subarray of size k

Given an array, find the median of each subarray of size k