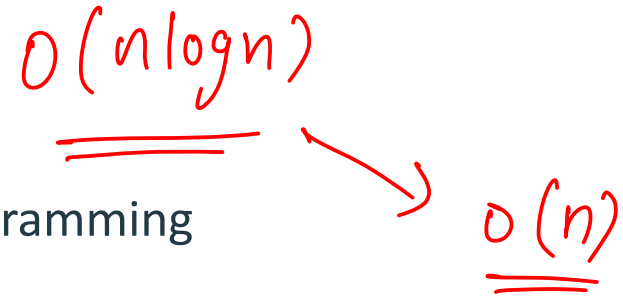
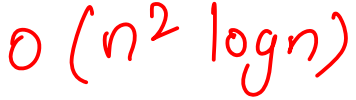
Two Pointers

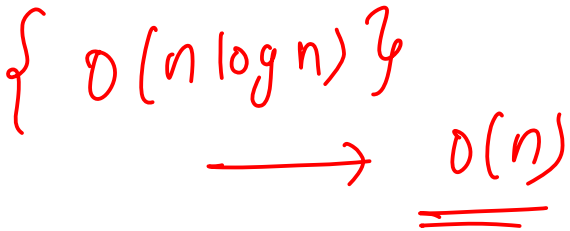
- Priyansh Agarwal

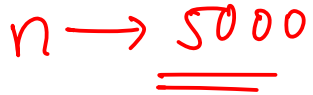
Two Pointers

- 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

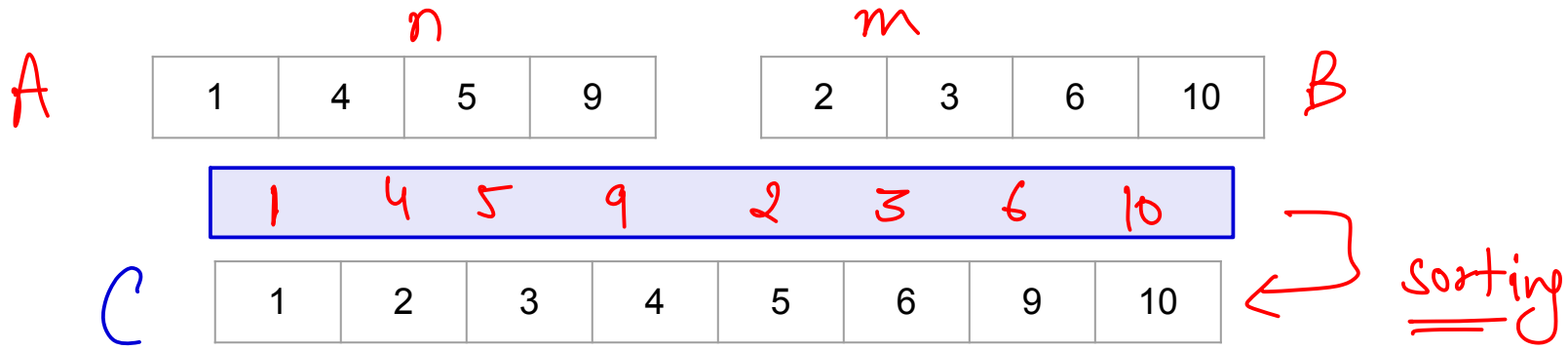








Given 2 sorted arrays, merge them into one single array keeping the elements sorted



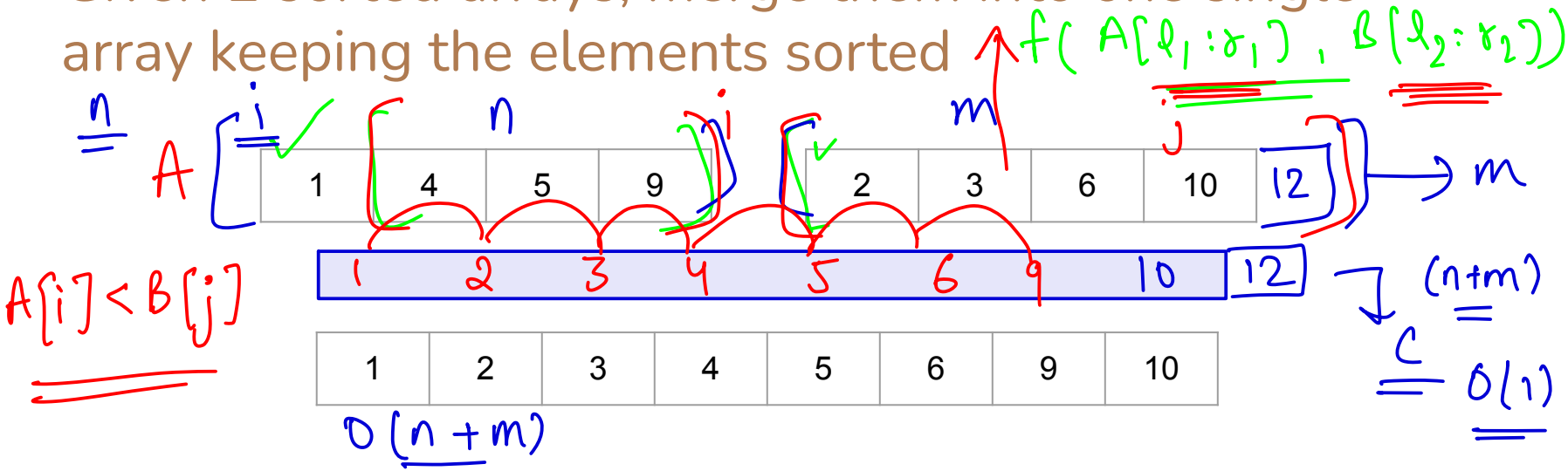
First Approach: Add all elements in an array and sort it

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

Second Approach: Use 2 pointers

2 pointers

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



First Approach: Add all elements in an array and sort it → [A[0:n-1] and B[0:m-1])

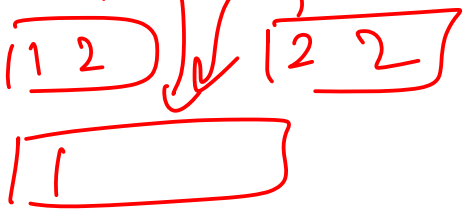
Second Approach: Use 2 pointers → (A[1:n-1] and B[0:m-1])

Solution using 2 pointers

3 pointer

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 arrays into the output array



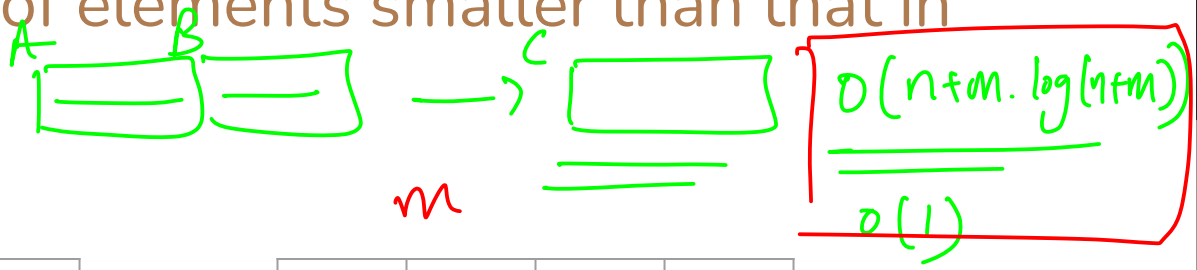

```
vector<int> a(n), b(m);
vector<int> c(n + m);
int i = 0, j = 0, k = 0;
while(i < n && j < m){
    if(a[i] < b[j]){
        c[k] = a[i], i++, k++;
    }else{
        c[k] = b[j], j++, k++;
    }
}
while(i < n){
    c[k] = a[i], k++, i++;
}
while(j < m){
    c[k] = b[j], k++, j++;
}
```

b[j] ≤ a[i]

O(n+m)

O(n) time
O(1) space

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 O(n log m)

Second Approach: 2 pointers { n → 10^7, m → 10^7 } O(n log m)

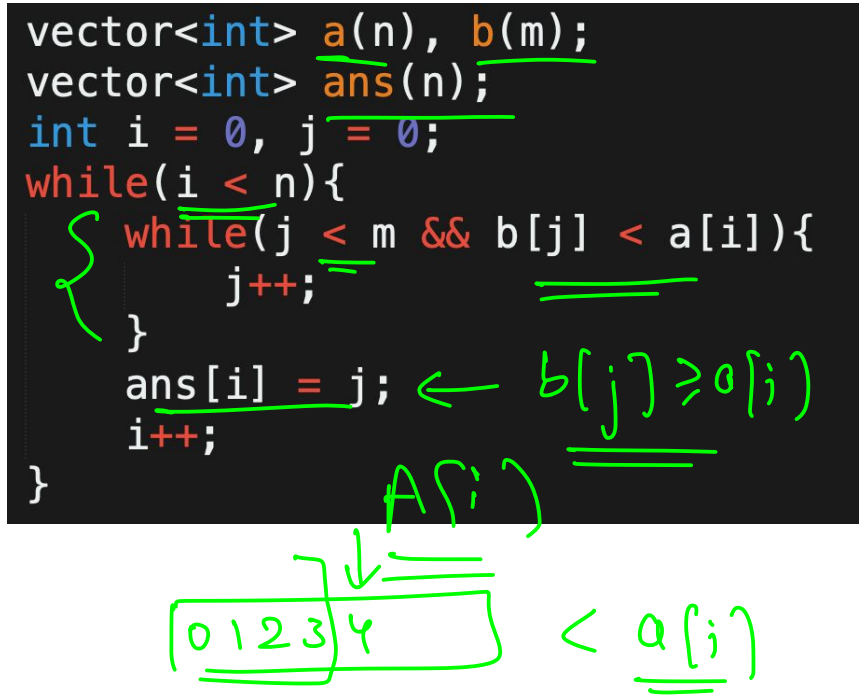
Solution using 2 pointers

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]

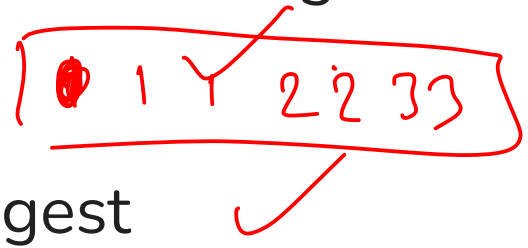
Having 2 pointers and both only move right. Time complexity?

```
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++;
}
```

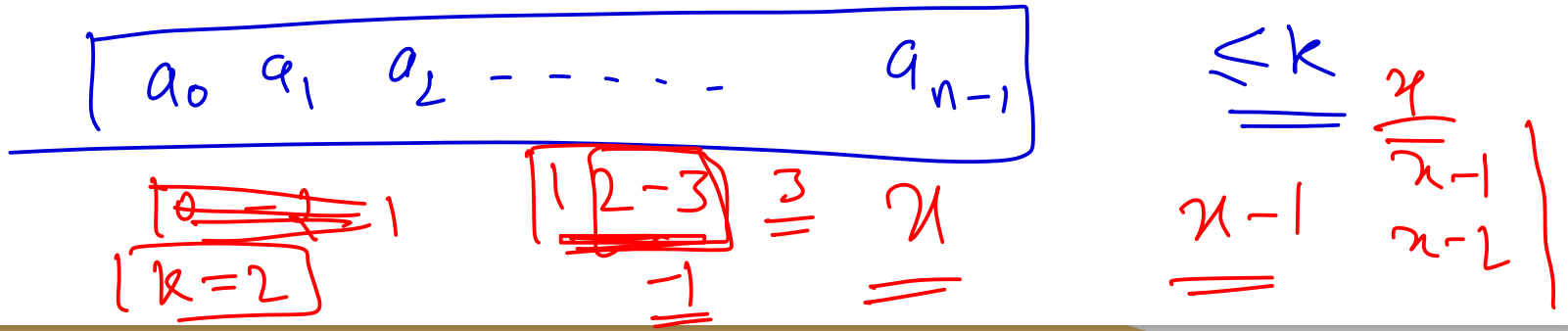


Good Segments Technique (Increasing)

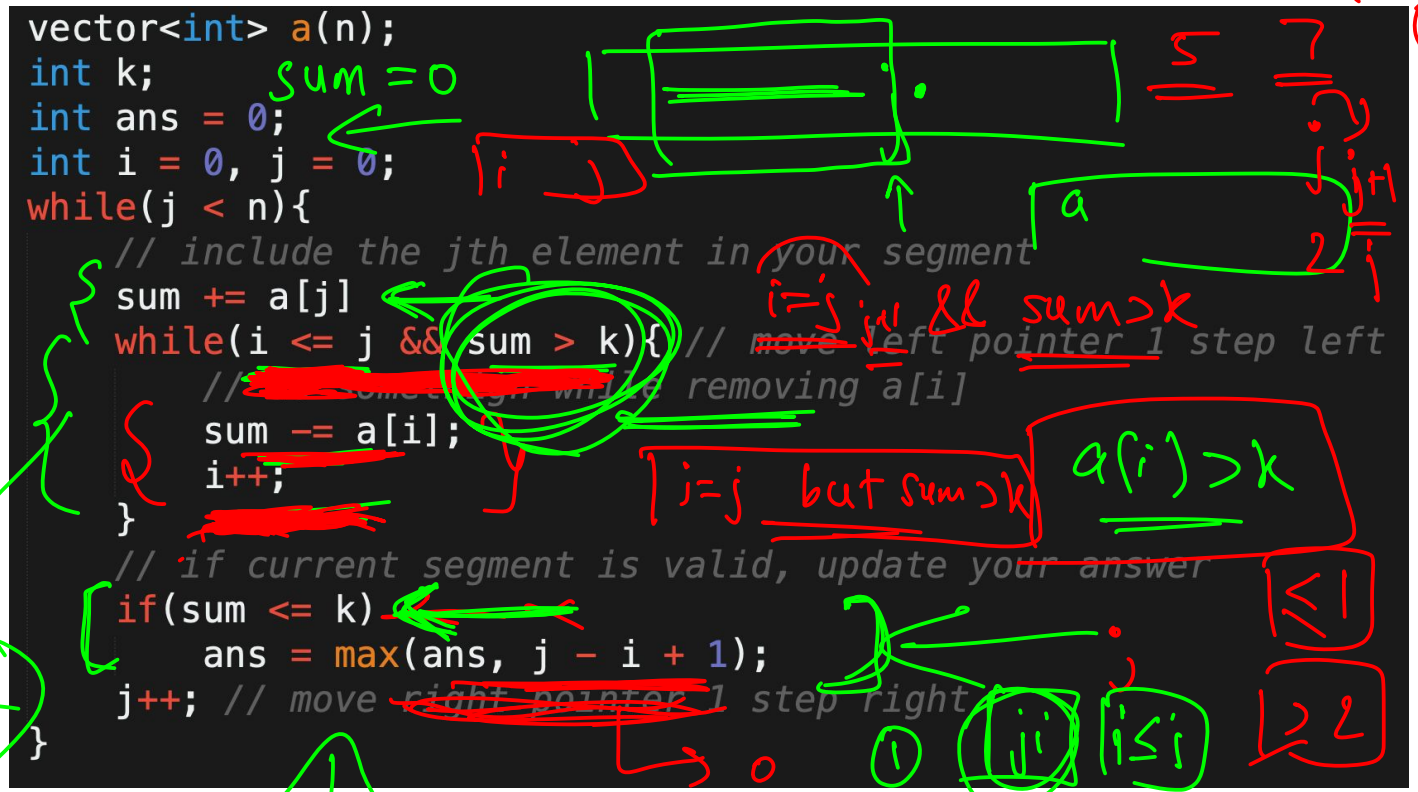
- Given an array of positive integers find the length of longest subarray with sum <= K



- Given an array find the length of longest subarray with not more than K distinct elements



Good Segments Technique Problem 1



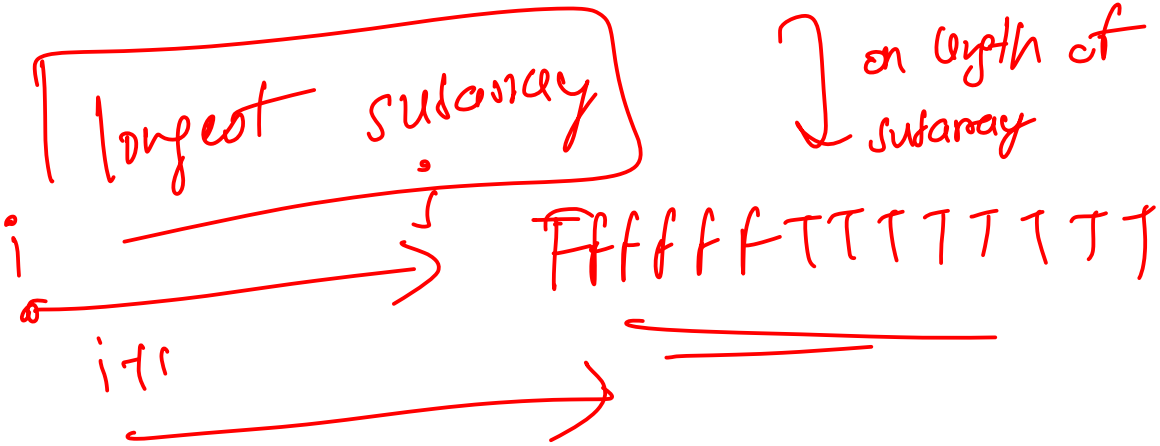
```
vector<int> a(n);
int k;
int ans = 0;
int i = 0, j = 0;
while(j < n){
    // include the jth element in your segment
    sum += a[j]
    while(i <= j && sum > k){ // move left pointer 1 step left
        // something while removing a[i]
        sum -= a[i];
        i++;
    }
    // if current segment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

```cpp
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
}
```

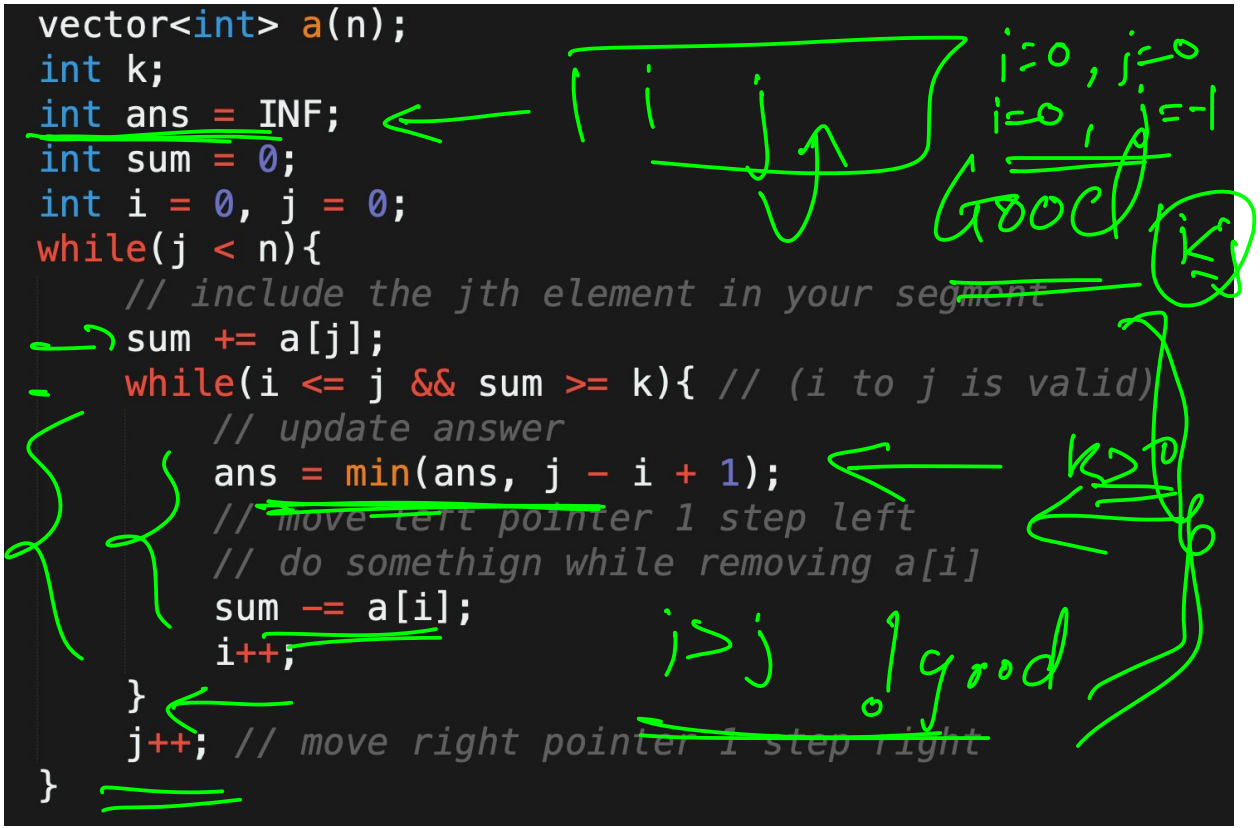
Good Segments Technique (Decreasing)

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



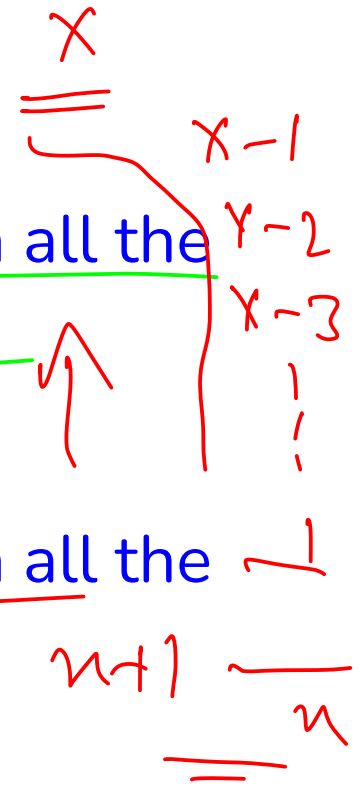
Good Segments Technique Problem 3

```
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
}
```



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~~

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