

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

CP — 2 pointers

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

→ $O(n)$

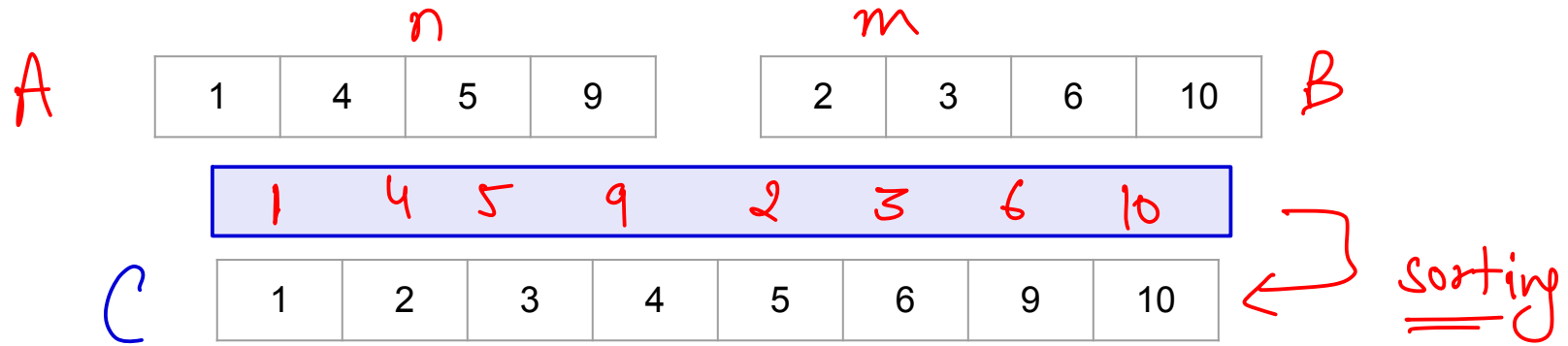
$O(n \log n)$

→ $O(n)$

$O(n^2 \log n)$

$n \rightarrow \underline{\underline{5000}}$

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



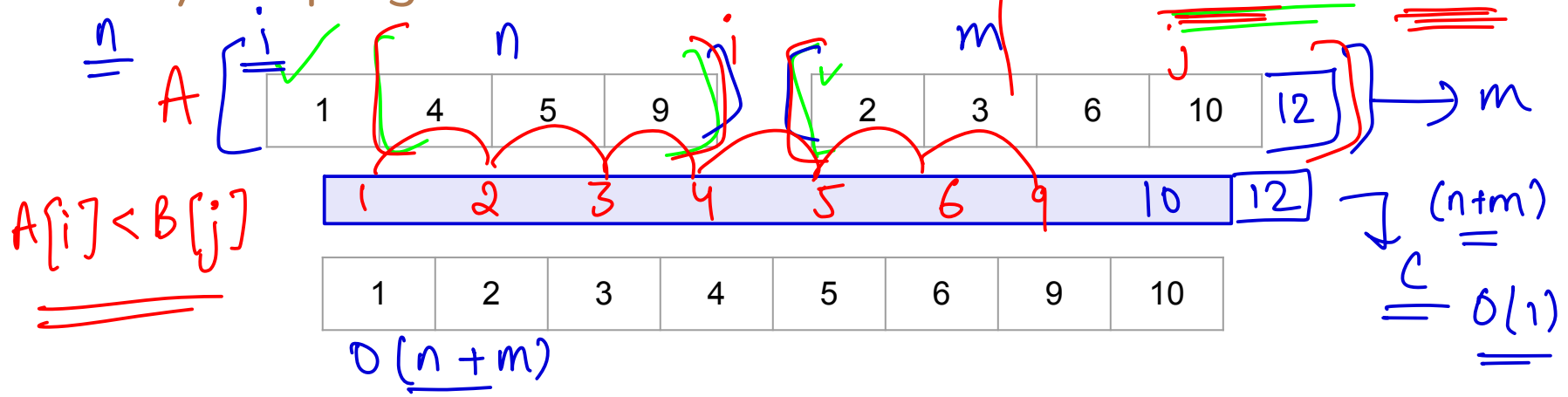
First Approach: Add all elements in an array and sort it

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

Second Approach: Use 2 pointers

2 pointers

Given 2 sorted arrays, merge them into one single array keeping the elements sorted $f(A[l_1:r_1], B[l_2:r_2])$



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

→ $[A[0:n-1] \text{ and } B[0:m-1]]$

Second Approach: Use 2 pointers

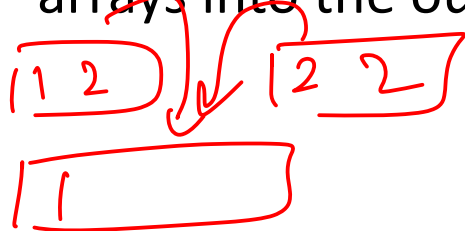
$[A[0:n-1] \text{ and } B[0:m-1]]$

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



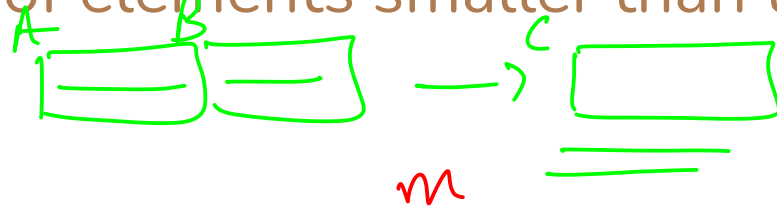
```
vector<int> a(n), b(m);  
vector<int> c(n + m);  
int i = 0, j = 0, k = 0;  
while(i < n && j < m){  
    → if(a[i] < b[j]){  
        c[k] = a[i], i++, k++;  
    } else{  
        c[k] = b[j], j++, k++;  
    }  
}  
while(i < n){  
    c[k] = a[i], k++, i++;  
}  
while(j < m){  
    c[k] = b[j],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

Handwritten annotations:

- Red checkmarks above `a(n)`, `b(m)`, and `c(n + m)`.
- Green checkmark above `int i = 0, j = 0, k = 0;`.
- Green arrow pointing down from the `while(i < n && j < m){}` loop.
- Red arrow pointing right from the `if(a[i] < b[j])` condition.
- Red bracket grouping the `if` and `else` blocks.
- Red arrow pointing up from the `b[j] < a[i]` condition.
- Red bracket grouping the `while(i < n){}` and `while(j < m){}` loops.
- Green underline under `o(n+m)`.

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

$O(n)$ time
 $O(1)$ space



$O(n+m \cdot \log(n+m))$
 $O(1)$

n

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

m

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



First Approach: Binary Search for each elements $O(n \log m)$

Second Approach: 2 pointers

$\left\{ \begin{array}{l} n \rightarrow 10^7 \\ m \rightarrow 10^7 \end{array} \right\}$ $O(n \log m)$

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 $a[i + 1] \geq a[i]$

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++;  
}
```

Ans:)
↓
[0 1 2 3] 4 < a[i]

Good Segments Technique (Increasing)

- Given an array of positive integers find the length of longest subarray with sum $\leq K$

1 4 2 2 3 3

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

$a_0 \ a_1 \ a_2 \ \dots \ a_{n-1}$

~~$[0, 1, 2]$~~
 $[2=2]$

~~$[1, 2, 3]$~~
 3
 2
 1

$\leq K$
 2
 $n-1$
 $n-2$

Good Segments Technique Problem 1

-1 -1 -1 -1 -1
-3, 8, -1 | k=4
②

```
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
        // sum -= a[i]; 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
}
```

Handwritten notes and diagrams:

- sum = 0** (green arrow pointing to `sum += a[j]`)
- i = j** (green arrow pointing to `i <= j`)
- a** (green arrow pointing to `a[j]`)
- 2** (green arrow pointing to `j++`)
- i=j && sum > k** (green circle around `sum > k`)
- move left pointer 1 step left** (green arrow pointing to `i++`)
- sum -= a[i];** (green arrow pointing to `sum -= a[i];`)
- i++;** (green arrow pointing to `i++;`)
- j=j but sum > k** (green arrow pointing to `j++`)
- a(i) > k** (green arrow pointing to `sum > k`)
- if(sum <= k)** (green arrow pointing to `if(sum <= k)`)
- ans = max(ans, j - i + 1);** (green arrow pointing to `ans = max(ans, j - i + 1);`)
- j++;** (green arrow pointing to `j++;`)
- 0 1 2** (green arrows pointing to `i`, `j`, and `j - i + 1`)
- 1 2 2** (green arrows pointing to `i`, `j`, and `j - i + 1`)

Good Segments Technique Problem 2

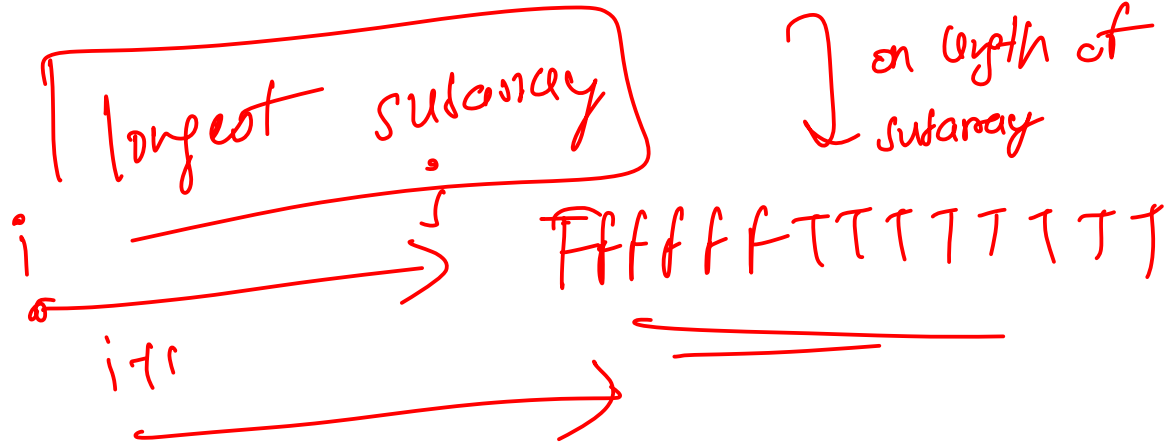
```
vector<int> a(n);
int k;
int ans = 0;
int i = 0, j = 0;
map<int, int> freq;
while(j < n){
    // include the jth element in your segment
    freq[a[j]]++;
    while(i <= j && freq.size() > k){ // move left pointer 1 step left
        // do something while removing a[i]
        freq[a[i]]--;
    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

$\text{freq}[a] > 0$ NO
GOOD

$k \geq 1$

Good Segments Technique (Decreasing)

- Given an array of positive integers find the length of smallest subarray with sum of elements $\geq K$



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 something while removing a[i]
        sum -= a[i];
        i++;
    }
    j++; // move right pointer 1 step right
}
```

Handwritten notes and diagrams:

- A green box around the `i` and `j` variables in the code, with an arrow pointing to the `while(i <= j && sum >= k)` condition.
- Handwritten text: `i=0, j=0` and `i=0, j=-1` with arrows pointing to the initial values of `i` and `j`.
- Handwritten text: `Good` with a green checkmark.
- Handwritten text: `k > 0` with a green checkmark.
- Handwritten text: `i > j ! good` with a green checkmark.
- Handwritten text: `i <= j` with a green checkmark.

Good Segments Technique General Trick

- Condition 1: If Segment $[L:R]$ is good then all the segments enclosed within it 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!

x
 x
 $x-1$
 $x-2$
 $x-3$
⋮

x $x+1$ x

Good Segments Technique (Number of Segments?)

- How to find number of good segments?

- Let's solve the first problem.

- Number of subarrays with sum $\leq K$

- Simple! Just multiple $(j - i + 1)$ for every i

while (j < n)

{ if (sum > K) {

ans += (j - i + 1)

increasing

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

Good Segments Technique (Number of Segments?)

- How to find number of good segments?

- Let's solve the first problem.

$(i \quad j)$

- Number of subarrays with sum $\leq K$

$(i-1 \quad j)$

- Simple! Just multiple $(j - i + 1)$ for every i

$(i-2 \quad j)$

$$ans = \min(ans, j - i + 1)$$

$$ans \pm \underline{\underline{(i+1)}}$$

$$\begin{matrix} i \\ n-j \\ j \\ i+1 \\ \underline{\underline{i+1}} \\ n+1 \\ n+2 \\ n+3 \end{matrix}$$