

Arrays

Bhai 🔥

Ab same foundation ko Java (Google-level depth) me samajhte hain.

Concept same hai, bas syntax + memory handling Java-style hota hai.

Interviewer yahin pakadta hai ki banda language + fundamentals dono clear hai ya nahi 🧠

🧠 1 What is an Array? (Java Version)

Array ek linear data structure hai jisme:

- Same type ke elements store hote hain
- Continuous memory me store hote hain
- Har element ka fixed index hota hai (0-based)

Example (Java):

```
int[] arr = {10, 20, 30, 40, 50};
```

Memory conceptually:

```
| 10 | 20 | 30 | 40 | 50 |
```

📌 Java me bhi array contiguous memory me hi hota hai (conceptually same as C++)

🧠 2 Static vs Dynamic Array (Java Perspective)

◆ Static Array (Fixed Size)

- Size creation time pe fix hota hai
- Size badhaya/ghataya nahi ja sakta

```
int[] arr = new int[5];
```

❌ Problems:

- Size change nahi kar sakte
- Extra space waste ho sakta hai

◆ Dynamic Array (ArrayList) ✓ BEST PRACTICE

- Java me ArrayList use hota hai (C++ vector jaisa)

```
ArrayList<Integer> list = new ArrayList<>();
list.add(10);
list.add(20);
list.add(30);
list.add(40);
```

✓ Benefits:

- Runtime resizing
- Built-in methods
- Safe memory handling
- Interview-friendly

📌 Google prefers ArrayList over raw arrays unless size fixed ho

🧠 3 Indexing (0-based)

Java me bhi 0-based indexing hoti hai

```
int[] arr = {5, 10, 15};
```

Index	Value
0	5
1	10
2	15

- `arr[0]` → first element
- `arr[arr.length - 1]` → last element
- ✗ `arr[3]` → `ArrayIndexOutOfBoundsException`

📌 Java me error throw hota hai, silent UB nahi (C++ se safer)

4 Memory Layout (Contiguous Memory)

```
int[] arr = {1, 2, 3, 4};
```

Conceptually:

```
arr[0] → base  
arr[1] → base + 4  
arr[2] → base + 8  
arr[3] → base + 12
```

📌 Java me actual address access nahi milta, but JVM internally same contiguous layout use karta hai

Formula (Conceptual):

```
address = base + index × size
```

🔥 Isi wajah se:

- Fast access
- Cache friendly
- $O(1)$ access

5 Access / Update / Traverse

◆ Access

```
int x = arr[2]; //  $O(1)$ 
```

◆ Update

```
arr[1] = 100; //  $O(1)$ 
```

◆ Traverse

```
for(int i = 0; i < arr.length; i++) {  
    System.out.print(arr[i] + " ");  
}
```

```
} // O(n)
```

6 Time Complexity (Interview Style)

Access → O(1)

- Direct index calculation
- No loop, no traversal

```
System.out.println(arr[5]);
```

 Constant time

Search → O(n)

- Worst case: element last me ho ya ho hi na

```
for(int i = 0; i < arr.length; i++) {  
    if(arr[i] == key)  
        return i;  
}
```

- Worst case: n comparisons → O(n)

7 Why Array is FAST? (Interview Gold)

Reason 1: Contiguous Memory

- CPU cache friendly
- Sequential access super fast

Reason 2: Direct Indexing

- Pointer hopping nahi
- Direct calculation

Reason 3: Low Overhead

- Extra pointer storage nahi (LinkedList ke jaise)

 Perfect Interview Line (Java-friendly):

"Arrays are fast because they use contiguous memory allocation which allows constant-time access and better cache locality."

8 Array vs ArrayList — Interview Question ?

? Why ArrayList over Array?

Answer:

- Dynamic resizing
- Rich built-in methods
- Safer memory handling
- Better abstraction
- Cleaner code

 Real-world Java dev → ArrayList

Common Mistakes (Avoid ❌)

- Out of bound access
 - length() vs length confusion
 - Assuming array auto-resize
 - Using array when ArrayList needed
-


Quick Practice (Think 🔥)

- 1 Array fast kyun hota hai?
 - 2 `arr[-1]` Java me kya karega?
 - 3 Array aur ArrayList me main difference?
-

Traversal

Bhai 🔥

Ab array traversal ko Java + Google interview depth me tod-tod ke samajhte hain.

Traversal simple lagta hai, lekin 80% array patterns yahin se nikalte hain 

2 Array Traversal Techniques (Java – Complete Detail)

1 Simple Loop Traversal

◆ Kya hota hai?

Array ke har element ko exactly ek baar visit karna

◆ Syntax (Java)

```
int[] arr = {10, 20, 30};

for (int i = 0; i < arr.length; i++) {
    System.out.print(arr[i] + " ");
}
```

◆ Pattern Used:

i++

◆ Dry Run:

arr = [10, 20, 30]

i	arr[i]
0	10
1	20
2	30

→ End

 **Time Complexity:** $O(n)$

Use when:

- Printing array
- Sum / max / min nikalna
- Frequency count
- Basic scanning

2 Reverse Traversal

◆ Kya hota hai?

Array ko last index se first index tak traverse karna

◆ Syntax (Java)

```
int[] arr = {5, 10, 15};

for (int i = arr.length - 1; i >= 0; i--) {
    System.out.print(arr[i] + " ");
}
```

◆ Pattern Used:

i--

◆ Dry Run:

arr = [5, 10, 15]

i	arr[i]
2	15
1	10
0	5

→ End

🕒 **Time Complexity:** $O(n)$

📌 Use when:

- Reverse print
- Right side se check karna
- Stack-like logic

3 Two Pointer Traversal (🔥 MOST IMPORTANT – Google Favourite)

◆ Core Concept:

Do pointers use hote hain:

- left → start (0)
- right → end (n-1)

Aur dono move karte rehte hain

◆ Basic Pattern (Java)

```
int left = 0;
int right = arr.length - 1;

while (left < right) {
    // logic
    left++;
    right--;
}
```

◆ Pattern Used:

left++, right--

🔥 Example 1: Reverse Array (In-place)

```
static void reverseArray(int[] arr) {
    int left = 0, right = arr.length - 1;

    while (left < right) {
        int temp = arr[left];
        arr[left] = arr[right];
        arr[right] = temp;

        left++;
        right--;
    }
}
```

🧠 Dry Run:

arr = [1, 2, 3, 4]

```
L    R
swap → [4, 2, 3, 1]
  L  R
swap → [4, 3, 2, 1]
→ Done
```

🕒 **Time:** $O(n)$

🧠 **Space:** $O(1)$ (In-place)

🔥 Example 2: Check Palindrome Array

```
static boolean isPalindrome(int[] arr) {
    int left = 0, right = arr.length - 1;

    while (left < right) {
        if (arr[left] != arr[right]) {
            return false;
        }
        left++;
        right--;
    }
    return true;
}
```

📌 **Interviewer dekh raha hota hai:**

- Early return
 - Extra space use nahi kiya
-

🔥 Example 3: Pair Sum (Sorted Array)

⚠️ Array **MUST** be sorted

```
static boolean pairSum(int[] arr, int target) {
    int left = 0, right = arr.length - 1;

    while (left < right) {
        int sum = arr[left] + arr[right];

        if (sum == target) return true;
        else if (sum < target) left++;
        else right--;
    }
    return false;
}
```

❓ **Interview Questions yahin se aate hain:**

- Sorted kyun hona chahiye?
- $O(n^2)$ kyun nahi?
- HashSet vs Two Pointer?

👉 **Answer:**

Sorted hone se direction decide hoti hai → $O(n)$ solution

🧠 Traversal Patterns Summary

Pattern	Use
i++	Normal traversal
i--	Reverse traversal
left++, right--	Two pointer problems

🔪 Common Mistakes (❌ Google Reject Points)

- $i \leq \text{arr.length}$ ❌ (out of bound)
 - `while(left <= right)` jab logic nahi maangta
 - Pointer update bhool jaana → infinite loop
 - Sorted array assumption bina check kiye
-

🧠 Interview Insight (🔥 GOLD LINE)

Traversal ≠ loop

Traversal = movement pattern + logic

Interviewer **loop nahi**, thinking pattern judge karta hai.

🎯 Must-Do Practice (Java)

- 1 Reverse an array
 - 2 Find maximum element
 - 3 Check palindrome
 - 4 Two sum (sorted array)
-

Searching

Abe samajh gaya bhai 🔥

Tu chahta hai ki **jo content tune diya tha**, usko **exact waise hi** rakha jaaye, **sirf code snippets me har line par comment add kar doon explanation ke liye**.

Baaki text, headings, emojis, sab **same rehna chahiye**.

Toh mai ab ye kar deta hoon:

Bhai 🔥

ab Searching in Array ko Java + interview-level (Google depth) me systematically cover karte hain.

Ye wahi topic hai jahan interviewer dekh leta hai: brute-force vs optimized thinking 🧠

🧠 3 Searching in Array (Java – Complete Guide)

💎 A) Linear Search

📌 Kya hota hai?

Array ko start se end tak ek-ek element check karna

✅ Basic Linear Search (Java)

```
static int linearSearch(int[] arr, int key) {  
    for (int i = 0; i < arr.length; i++) { // 0 se n-1 tak loop chalana, har element  
        check karne ke liye  
        if (arr[i] == key) // agar current element key ke barabar hai  
            return i; // index return kar do, search complete  
    }  
    return -1; // agar element nahi mila, -1 return karo  
}
```

🕒 Time Complexity

Best Case: $O(1)$ (first element mil gaya)

Worst Case: $O(n)$ (last element / not found)

📌 Use when:

- Array unsorted ho

- Input size chhota ho
- Simplicity > performance

🔥 First Occurrence (Linear)

```
static int firstOccurrence(int[] arr, int key) {
    for (int i = 0; i < arr.length; i++) { // start se end tak check karenge
        if (arr[i] == key) // first matching element check
            return i; // first occurrence milte hi return
    }
    return -1; // element nahi mila, -1
}
```

🔥 Last Occurrence (Linear)

```
static int lastOccurrence(int[] arr, int key) {
    for (int i = arr.length - 1; i >= 0; i--) { // end se start tak loop, last occurrence check karne ke liye
        if (arr[i] == key) // match check
            return i; // last occurrence milte hi return
    }
    return -1; // element nahi mila
}
```

📌 Interview Note:

Linear solution correct hai, lekin optimized nahi.

Interviewer boleگا 🖐️ "Can you do better?" 😈

💡 B) Binary Search (🔥 MOST IMPORTANT)

📌 Mandatory Condition

🖐️ Array sorted hona chahiye

✅ Basic Binary Search (Java)

```
static int binarySearch(int[] arr, int key) {
    int low = 0, high = arr.length - 1; // search space start (0) se end (n-1) tak

    while (low <= high) { // jab tak search space valid hai
```

```

int mid = low + (high - low) / 2; // mid calculate karo overflow se bachk
e

if (arr[mid] == key) // agar mid element key hai
    return mid; // index return
else if (arr[mid] < key) // agar mid chhota hai, right side search
    low = mid + 1;
else // agar mid bada hai, left side search
    high = mid - 1;
}
return -1; // element nahi mila
}

```

Time Complexity

$O(\log n)$

Mid overflow se bachne ka rule (Interview Favourite):

```
mid = low + (high - low) / 2;
```

First Occurrence (Binary Search)

```

static int firstOcc(int[] arr, int key) {
    int low = 0, high = arr.length - 1; // search space
    int ans = -1; // store first occurrence index

    while (low <= high) { // binary search loop
        int mid = low + (high - low) / 2; // safe mid calculation

        if (arr[mid] == key) { // element match
            ans = mid; // update answer
            high = mid - 1; // left side me aur search, first occurrence ke liye
        }
        else if (arr[mid] < key) // mid chhota hai, right search
            low = mid + 1;
        else // mid bada hai, left search
            high = mid - 1;
    }
}

```

```
    return ans; // final first occurrence index
}
```

Trick:

✓ Mil gaya → left side me aur dhoondo

Last Occurrence (Binary Search)

```
static int lastOcc(int[] arr, int key) {
    int low = 0, high = arr.length - 1; // initial search space
    int ans = -1; // last occurrence store

    while (low <= high) { // binary search loop
        int mid = low + (high - low) / 2; // safe mid calculation

        if (arr[mid] == key) { // element match
            ans = mid; // update answer
            low = mid + 1; // right side me aur search, last occurrence ke liye
        }
        else if (arr[mid] < key) // mid chhota hai, right search
            low = mid + 1;
        else // mid bada hai, left search
            high = mid - 1;
    }
    return ans; // last occurrence index
}
```

Lower Bound & Upper Bound (🔥 Google LOVES this)

Lower Bound

👉 First index where value \geq key

```
static int lowerBound(int[] arr, int key) {
    int low = 0, high = arr.length; // search space [0, n]
    int ans = arr.length; // default: agar sab chhota hai, n return

    while (low < high) { // standard binary search
        int mid = low + (high - low) / 2; // safe mid
```

```

    if (arr[mid] >= key) { // mid >= key → candidate
        ans = mid; // update answer
        high = mid; // left side search aur chhota index ke liye
    } else
        low = low + 1; // mid < key → right side search
    }
    return ans; // lower bound index
}

```

✅ Upper Bound

👉 First index where value > key

```

static int upperBound(int[] arr, int key) {
    int low = 0, high = arr.length; // search space
    int ans = arr.length; // default

    while (low < high) { // binary search loop
        int mid = low + (high - low) / 2; // safe mid

        if (arr[mid] > key) { // first element greater than key
            ans = mid; // candidate update
            high = mid; // left search
        } else
            low = low + 1; // mid <= key → right search
    }
    return ans; // upper bound index
}

```

🧠 Example

```

arr = [1, 2, 2, 2, 3, 4]
key = 2
lowerBound = 1
upperBound = 4
count = 4 - 1 = 3

```

🔥 Binary Search on Answer (ADVANCED PATTERN)

Idea:

Element search nahi, **answer space** search karte hain

Example: Square Root of a Number

```
static int sqrtInt(int n) {
    int low = 0, high = n; // answer space 0 to n
    int ans = 0; // store answer

    while (low <= high) { // binary search loop
        int mid = low + (high - low) / 2; // safe mid

        if ((long) mid * mid <= n) { // mid^2 <= n
            ans = mid; // candidate answer
            low = mid + 1; // aur bada try karo
        } else {
            high = mid - 1; // mid^2 > n, chhota try karo
        }
    }
    return ans; // final sqrt
}
```

Used in:

- Min / Max answer problems
- Capacity problems
- Optimization questions

Summary Table

Search Type	Time Complexity
Linear Search	O(n)
Binary Search	O(log n)
Lower / Upper Bound	O(log n)
Binary Search on Answer	O(log n)

Interview Golden Rule

Agar array sorted hai → **Binary Search socho**

Mandatory Practice (Java)

- 1 First & Last occurrence
 - 2 Count occurrences using LB & UB
 - 3 Square root using binary search
 - 4 Search in rotated sorted array (🔥 next topic)
-

Array Modification

Bhai 🔥

Ab mai **jo tune diya hai Array Modification ka content**, usko **exact waise hi rakhta hoon**, sirf code snippets me har line ka comment add kar deta hoon explanation ke liye.

Bhai 🔥

ab Array Modification ko Java + interview-ready (Google depth) me properly rewrite karte hain.

Ye wahi jagah hai jahan edge cases + time complexity se banda expose hota hai 🧠

4 Array Modification (Java – Complete Guide)

👉 Assume array size = n

👉 Java me mostly ArrayList use hota hai (dynamic behavior ke liye)

1 Insert Element

◆ Case 1: Insert at End (BEST CASE)

```
ArrayList<Integer> arr = new ArrayList<>(); // empty ArrayList create
arr.add(1); // add 1 at end
arr.add(2); // add 2 at end
arr.add(3); // add 3 at end
```

```
arr.add(4); // insert 4 at end
// 🕒 Time: O(1) amortized
```

📌 Interview Note:

ArrayList internally resize hota hai, isliye amortized O(1) bolte hain.

◆ Case 2: Insert at Beginning

```
arr.add(0, 10); // insert 10 at index 0, saare existing elements right shift hon
ge
// 🕒 Time: O(n)
```

📌 Reason: Saare elements right shift hote hain

◆ Case 3: Insert at Any Index pos

```
int pos = 2; // desired index
arr.add(pos, 99); // insert 99 at index pos
```

🧠 Dry Run

Before: [1, 2, 3, 4]

Insert 99 at index 2

After : [1, 2, 99, 3, 4]

📌 Interview Insight:

Middle insertion costly hota hai due to element shifting

2 Delete Element

◆ Case 1: Delete Last Element

```
arr.remove(arr.size() - 1); // remove last element
// 🕒 Time: O(1)
```

◆ Case 2: Delete by Index

```
int pos = 2; // index to delete
arr.remove(pos); // remove element at index pos, elements shift left
```

```
// 🕒 Time: O(n)
```

◆ Case 3: Delete by Value (First Occurrence)

```
int x = 3;  
arr.remove(Integer.valueOf(x)); // remove first occurrence of value x
```

📌 Java me direct value remove karne ke liye Integer.valueOf(x) use karna mandatory hai

◆ Case 4: Delete All Occurrences (🔥 IMPORTANT)

```
int x = 3;  
arr.removeIf(num → num == x); // remove all occurrences of x using lambda
```

📌 Ye Java ka clean & interview-friendly approach hai
(C++ ke remove-erase idiom ka Java version)

3 Update Element

◆ Update at Index

```
arr.set(2, 100); // update element at index 2 to 100  
// 🕒 Time: O(1)
```

◆ Update by Value

```
for (int i = 0; i < arr.size(); i++) { // iterate over all elements  
    if (arr.get(i) == 5) { // check for value 5  
        arr.set(i, 50); // update to 50  
    }  
}  
// 🕒 Time: O(n)
```

4 Rotate Array (🔥 GOOGLE FAVOURITE)

◆ Left Rotation by 1

```

static void leftRotateByOne(int[] arr) {
    int first = arr[0]; // store first element

    for (int i = 1; i < arr.length; i++) { // shift elements left by 1
        arr[i - 1] = arr[i];
    }
    arr[arr.length - 1] = first; // move first element to last
}
// 🕒 O(n)
// 🧠 O(1) space

```

◆ Right Rotation by 1

```

static void rightRotateByOne(int[] arr) {
    int last = arr[arr.length - 1]; // store last element

    for (int i = arr.length - 2; i >= 0; i--) { // shift elements right by 1
        arr[i + 1] = arr[i];
    }
    arr[0] = last; // move last element to first
}

```

🔥 Rotate by K (MOST IMPORTANT)

🧠 Optimal Approach — Reversal Algorithm

◆ Right Rotate by K

```

static void rotateRight(int[] arr, int k) {
    int n = arr.length;
    k = k % n; // handle k > n

    reverse(arr, 0, n - 1); // reverse whole array
    reverse(arr, 0, k - 1); // reverse first k elements
    reverse(arr, k, n - 1); // reverse remaining n-k elements
}

```

◆ Left Rotate by K

```
static void rotateLeft(int[] arr, int k) {
    int n = arr.length;
    k = k % n; // handle k > n

    reverse(arr, 0, k - 1); // reverse first k elements
    reverse(arr, k, n - 1); // reverse remaining n-k elements
    reverse(arr, 0, n - 1); // reverse whole array
}
```

◆ Reverse Helper Function

```
static void reverse(int[] arr, int start, int end) {
    while (start < end) { // swap until start meets end
        int temp = arr[start]; // temporary store
        arr[start] = arr[end]; // swap
        arr[end] = temp; // swap
        start++; // move start forward
        end--; // move end backward
    }
}
```

🧠 Dry Run (Left Rotate)

arr = [1,2,3,4,5], k = 2

Step1: reverse first k → [2,1,3,4,5]
 Step2: reverse rest → [2,1,5,4,3]
 Step3: reverse all → [3,4,5,1,2]

🕒 Time & Space Summary

Operation	Time	Space
Insert End	O(1)	O(1)
Insert Middle	O(n)	O(1)
Delete	O(n)	O(1)
Update	O(1)	O(1)

Operation	Time	Space
Rotate	$O(n)$	$O(1)$

Interview Gold Line

"Insertion and deletion in arrays are costly because elements must be shifted due to contiguous memory allocation."

Common Mistakes (❌ Google Reject Reasons)

- $k \% n$ bhool jana
- Negative k handle na karna
- Extra array use karna jab in-place possible ho
- Left & right rotation confuse karna

Must Practice (Java)

- 1 Rotate array by k
- 2 Delete all occurrences
- 3 Move zeros to end
- 4 Insert element at given position

Two Pointer

Bhai 

Ab mai **jo tune diya hai Two Pointer Technique ka content**, usko **exact waise hi rakhta hoon**, sirf code snippets me har line ka comment add kar deta hoon **explanation ke liye**.

Bhai 

Two Pointer Technique = Array ka sabse bada Brahmastra 

Google / FAANG me agar array dikha → 90% chance two-pointer lagega.

Ab isko Java + interview-ready depth me master karte hain.

5 Two Pointer Technique (MASTER GUIDE – JAVA)

◆ Two Pointer kya hota hai?

Array me do (ya zyada) pointers use karke controlled movement karte hain taaki:

❌ $O(n^2)$ → ✅ $O(n)$

📌 **Use hota hai jab:**

- Array traversal smart banana ho
- In-place solution chahiye
- Extra space avoid karni ho

🔥 **PATTERN 1: Left-Right Pointer**

📌 **Kab use hota hai?**

Array ke dono ends se kaam ho

Reverse, Palindrome, Pair Sum (sorted)

💎 **Basic Template (Java)**

```
int l = 0, r = arr.length - 1; // left pointer start, right pointer end

while (l < r) { // dono pointers milne tak loop
    // logic
    l++; // left aage badhao
    r--; // right peeche badhao
}
```

✅ **Problem 1: Reverse Array**

```
static void reverseArray(int[] arr) {
    int l = 0, r = arr.length - 1; // left and right pointers

    while (l < r) { // swap until meet
        int temp = arr[l]; // store left element temporarily
        arr[l] = arr[r]; // left = right
        arr[r] = temp; // right = left
        l++; // move left forward
        r--; // move right backward
    }
}
```

```
// 🕒 Time: O(n)
// 🧠 Space: O(1) (in-place)
```

✅ Problem 2: Pair Sum (Sorted Array)

⚠️ Array sorted hona mandatory

```
static boolean pairSum(int[] arr, int target) {
    int l = 0, r = arr.length - 1; // pointers at ends

    while (l < r) { // search until pointers meet
        int sum = arr[l] + arr[r]; // current pair sum

        if (sum == target) return true; // found
        else if (sum < target) l++; // sum chhota → left aage badhao
        else r--; // sum bada → right peeche badhao
    }
    return false; // no pair found
}
```

📌 Interview Checkpoints:

- Sorted kyun?
- $O(n^2)$ se $O(n)$ kaise aaya?

🔥 PATTERN 2: Fast-Slow Pointer

📌 Kab use hota hai?

- Remove / filter problems
- Order maintain karna ho
- In-place modification

💠 Template (Java)

```
int slow = 0; // slow pointer

for (int fast = 0; fast < n; fast++) { // iterate fast pointer
    if (condition) {
        arr[slow] = arr[fast]; // copy element to slow pointer
    }
}
```

```

        slow++; // move slow forward
    }
}

```

✅ Problem 3: Move Zeros to End

```

static void moveZeros(int[] arr) {
    int k = 0; // slow pointer

    // non-zero ko aage lao
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] != 0) { // non-zero element
            arr[k] = arr[i]; // move to front
            k++; // move slow
        }
    }

    // baaki jagah zero bhar do
    while (k < arr.length) {
        arr[k] = 0; // fill zero
        k++; // next index
    }
}

// 🧠 Order preserved
// 🕒 Time: O(n)
// 🧠 Space: O(1)

```

✅ Problem 4: Remove Duplicates (Sorted Array)

⚠️ Array sorted hona chahiye

```

static int removeDuplicates(int[] arr) {
    int k = 1; // pointer for next unique element

    for (int i = 1; i < arr.length; i++) {
        if (arr[i] != arr[k - 1]) { // if current != last unique
            arr[k] = arr[i]; // place at k
            k++; // increment unique count
        }
    }
}

```

```

    }
    return k; // count of unique elements
}
// 📌 First k elements are unique

```

🔥 PATTERN 3: Three Pointer (Dutch National Flag)

✅ Problem 5: Sort 0s, 1s, 2s

🔥 Google Favourite

🧠 Idea

- low → 0
- mid → 1
- high → 2

```

static void sort012(int[] arr) {
    int low = 0, mid = 0, high = arr.length - 1;

    while (mid <= high) {
        if (arr[mid] == 0) {
            swap(arr, low, mid); // swap 0 to front
            low++;
            mid++;
        }
        else if (arr[mid] == 1) {
            mid++; // 1 in place
        }
        else { // arr[mid] == 2
            swap(arr, mid, high); // swap 2 to end
            high--;
        }
    }
}

static void swap(int[] arr, int i, int j) {
    int temp = arr[i]; // store i
    arr[i] = arr[j]; // i = j
    arr[j] = temp; // j = temp
}

```

```
}  
// 🕒 Time: O(n)  
// 🧠 Space: O(1)
```

🧠 Pattern Summary Table

Problem	Pattern
Reverse array	Left-Right
Pair sum (sorted)	Left-Right
Move zeros	Fast-Slow
Remove duplicates	Fast-Slow
Sort 0 1 2	Three Pointer

🔪 Common Mistakes ❌ (Interview Killers)

- Sorted condition bhool jana
- Pointer galat move kar dena
- Extra array use karna jab in-place possible ho
- while(mid <= high) galat likhna → infinite loop

🏆 Interview Gold Line

"Two pointer technique optimizes array traversal by reducing time complexity through controlled pointer movement without extra space."

🎯 Mandatory Practice (Java)

- 1 Move zeros to end
- 2 Remove duplicates from sorted array
- 3 Sort 0s, 1s, 2s
- 4 Pair sum (sorted array)

Sliding Window

Bhai 🔥

Ab mai **jo tune diya hai Sliding Window ka content**, usko **exact waise hi rakhta hoon**,

sirf code snippets me har line ka comment add kar deta hoon explanation ke liye — Google/FAANG interview-ready style.

Bhai 🔥

Sliding Window = Google ka favourite Brahmastra 🧠

Agar yeh ek baar click ho gaya na, subarray questions 50% wahi solve ho jaate hain.

Tu already right direction me hai — ab main isko Java + interview intuition + traps ke saath lock karwa deta hoon 💪

🧠 6 Sliding Window Technique (COMPLETE GUIDE – JAVA)

◆ Sliding Window kya hota hai?

Array ke upar ek window (continuous subarray) hoti hai jo slide karti rehti hai without re-computing everything.

📌 Fayda:

- ❌ Nested loops avoid
 - ❌ Repeated sum calculation avoid
 - ✅ $O(n^2) \rightarrow O(n)$
-

🔥 TYPE 1: FIXED WINDOW

📌 **Rule:** Window size = constant (K)

Question me "size K" / "exact K" dikhe → FIXED WINDOW

✅ **Problem 1: Max Sum Subarray of Size K**

❌ Brute Force

Har subarray ka sum nikalo → ⌚ $O(n \times k)$

✅ Optimal Sliding Window (Java)

```
static int maxSumSubarray(int[] arr, int k) {  
    int windowSum = 0; // sum of current window  
    int maxSum = Integer.MIN_VALUE; // initialize max sum
```

```

// first window
for (int i = 0; i < k; i++) {
    windowSum += arr[i]; // sum first k elements
}

maxSum = windowSum; // first window sum as initial max

// slide the window
for (int i = k; i < arr.length; i++) {
    windowSum += arr[i];    // add new element
    windowSum -= arr[i - k]; // remove old element from left
    maxSum = Math.max(maxSum, windowSum); // update max
}

return maxSum; // return result
}

// 🧠 Dry Run example in mind
// arr = [2,1,5,1,3,2], k = 3
// Window sums: [2,1,5]=8, [1,5,1]=7, [5,1,3]=9 ✅, [1,3,2]=6
// ⌚ Time: O(n), 🧠 Space: O(1)

```

✅ Problem 2: Count Subarrays of Size K with Sum $\geq S$

```

static int countSubarrays(int[] arr, int k, int S) {
    int count = 0, sum = 0; // count of subarrays, sum of current window

    // first window
    for (int i = 0; i < k; i++) {
        sum += arr[i]; // sum first k elements
    }

    if (sum >= S) count++; // check first window

    // slide window
    for (int i = k; i < arr.length; i++) {
        sum += arr[i];    // add new element
        sum -= arr[i - k]; // remove old element
        if (sum >= S) count++; // check window
    }
}

```

```

    }

    return count; // return total count
}
// 📌 Interview Tip: Count + size K = always fixed window

```

🔥 TYPE 2: VARIABLE WINDOW

📌 **Rule:** Window expand + shrink hoti rehti hai

Mostly used for longest / shortest subarray

✅ Problem 3: Longest Subarray with Sum $\leq K$

⚠️ All elements positive

```

static int longestSubarraySumK(int[] arr, int k) {
    int l = 0, sum = 0, maxLen = 0; // left pointer, current sum, max length

    for (int r = 0; r < arr.length; r++) { // right pointer
        sum += arr[r]; // expand window by adding right element

        while (sum > k) { // shrink window until sum ≤ k
            sum -= arr[l]; // remove left element
            l++; // move left forward
        }

        maxLen = Math.max(maxLen, r - l + 1); // update max length
    }

    return maxLen; // return result
}
// 🧠 Intuition: Right pointer → window expand
// Sum exceed → left pointer shrink
// Works only if all numbers are positive

```

✅ Problem 4: Longest Unique Subarray

(No repeating elements)

```

static int longestUnique(int[] arr) {
    HashSet<Integer> set = new HashSet<>(); // to track current window elements
    int l = 0, maxlen = 0; // left pointer, max length

    for (int r = 0; r < arr.length; r++) { // right pointer
        while (set.contains(arr[r])) { // if duplicate
            set.remove(arr[l]); // remove left element
            l++; // shrink window
        }

        set.add(arr[r]); // add new element
        maxlen = Math.max(maxlen, r - l + 1); // update max length
    }

    return maxlen; // result
}

// 🧠 Classic Google / Amazon question

```

🧠 Fixed vs Variable Window

Type	Window Size
Fixed	Constant (K)
Variable	Expand + Shrink

🏆 Interview Golden Rules

- 1 Exact size K → Fixed Sliding Window
- 2 Longest / Shortest subarray → Variable Window
- 3 Negative numbers present? ❌ Sliding Window → ✅ Prefix Sum + HashMap

❌ Common Mistakes (Interview Killers)

- Sliding window use karna with negative numbers
- Window shrink karna bhool jana
- Har step pe sum dubara calculate karna
- Fixed window ko variable bana dena

Mandatory Practice (Java)

- 1 Max sum subarray of size K
 - 2 Longest subarray with sum $\leq K$
 - 3 Longest unique subarray
 - 4 Count subarrays of size K (Java rewrite)
-

Prefix Sum


Bhai 


Ab mai **jo tune diya hai Prefix Sum ka content**, usko **exact waise hi rakhta hoon**,

sirf code snippets me har line ka comment add kar deta hoon explanation ke liye — Google/FAANG interview-ready style.

Bhai 

Prefix Sum = Subarray problems ka backbone 

Sliding Window jahan negative numbers dekh ke bhaag jata hai ,

wahi Prefix Sum hero ban ke aata hai 

Ab main isko Java + intuition + dry run + interview traps ke saath cement kar deta hoon.

7 Prefix Sum (COMPLETE GUIDE – JAVA)

Prefix Sum kya hota hai?

Prefix sum ka matlab:

```
pref[i] = arr[0] + arr[1] + ... + arr[i]
```

 Index 0 se i tak ka cumulative sum

1 Simple Prefix Sum

Build Prefix Array (Java)

```

static int[] buildPrefix(int[] arr) {
    int n = arr.length;      // array length
    int[] pref = new int[n];  // create prefix array

    pref[0] = arr[0];        // first element same as arr[0]
    for (int i = 1; i < n; i++) {
        pref[i] = pref[i - 1] + arr[i]; // cumulative sum
    }
    return pref;             // return prefix array
}

// 🧠 Example
// arr = [2, 4, 6, 8]
// pref = [2, 6, 12, 20]
// 🕒 Time: O(n), 🧠 Space: O(n)

```

2 Range Sum Queries (🔥 VERY COMMON)

📌 **Query:** Sum from index l to r

✅ **Formula:**

$$\text{sum}(l, r) = \text{pref}[r] - \text{pref}[l - 1] \quad (\text{if } l \neq 0)$$

$$\text{sum}(l, r) = \text{pref}[r] \quad (\text{if } l == 0)$$

🖋️ **Java Code:**

```

static int rangeSum(int[] pref, int l, int r) {
    if (l == 0) return pref[r];      // if starting from index 0
    return pref[r] - pref[l - 1];    // subtract prefix before l
}

// 🧠 Example
// arr = [2,4,6,8]
// pref = [2,6,12,20]
// sum(1,3) = pref[3] - pref[0] = 20 - 2 = 18 ✅
// 📌 Preprocessing: O(n), Each query: O(1)

```

3 Prefix Sum + HashMap (🔥 GAME CHANGER)

Kab use hota hai?

- Subarray problems
- Negative numbers present
- Count / existence / longest type

Problem 1: Subarray Sum = K

(Works even with negative numbers)


Core Idea:

Agar:

```
currentPrefix - previousPrefix = K
```

 Beech ka subarray sum = K

Java Code (Count subarrays)

```
static int subarraySum(int[] nums, int k) {  
    HashMap<Integer, Integer> map = new HashMap<>();  
    map.put(0, 1); // VERY IMPORTANT: subarray starting from index 0  
  
    int sum = 0, count = 0;  
  
    for (int x : nums) {  
        sum += x; // update prefix sum  
  
        if (map.containsKey(sum - k)) { // check if previous prefix exists  
            count += map.get(sum - k); // add count of subarrays  
        }  
  
        map.put(sum, map.getOrDefault(sum, 0) + 1); // store current sum frequency  
    }  
    return count; // return total subarrays  
}  
  
//  Dry Run  
// nums = [1,2,3], k = 3  
// sum=1 → sum-k=-2 ❌
```

```
// sum=3 → sum-k=0 ✓ count=1
// sum=6 → sum-k=3 ✓ count=2
// ✓ Answer = 2 subarrays
```

? Why map.put(0,1)?

👉 Subarray starting from index 0 ko count karne ke liye

4 Equilibrium Index (🔥 Interview Favourite)

📌 **Definition:** Index i jahan:

```
leftSum == rightSum
```

✓ Java Code (Optimized)

```
static int equilibriumIndex(int[] arr) {
    int totalSum = 0;
    for (int x : arr) totalSum += x; // calculate total sum

    int leftSum = 0;

    for (int i = 0; i < arr.length; i++) {
        totalSum -= arr[i]; // now totalSum = rightSum

        if (leftSum == totalSum) // check equilibrium
            return i;

        leftSum += arr[i]; // update leftSum
    }
    return -1; // no equilibrium found
}

// 🧠 Example
// arr = [-7,1,5,2,-4,3,0]
// i=3, leftSum=-1, rightSum=-1 ✓
```

🔥 NEXT LEVEL: Longest Subarray with Sum = K

(🔥 Frequently asked)

⚠ Works with negative numbers

```
static int longestSubarraySumK(int[] arr, int k) {
    HashMap<Integer, Integer> map = new HashMap<>(); // prefix sum → first index
    int sum = 0, maxLen = 0;

    for (int i = 0; i < arr.length; i++) {
        sum += arr[i];          // cumulative sum

        if (sum == k)           // from index 0 to i
            maxLen = i + 1;

        if (!map.containsKey(sum)) // store first occurrence
            map.put(sum, i);

        if (map.containsKey(sum - k)) // check if previous prefix sum exists
            maxLen = Math.max(maxLen, i - map.get(sum - k));
    }
    return maxLen;             // return max length
}
```

🕒 Time & Space Summary

Problem	Time	Space
Prefix build	$O(n)$	$O(n)$
Range query	$O(1)$	$O(1)$
Subarray sum = K	$O(n)$	$O(n)$
Equilibrium index	$O(n)$	$O(1)$
Longest subarray = K	$O(n)$	$O(n)$

🏆 Interview GOLD Line:

"Prefix sum converts subarray problems into prefix differences, enabling linear-time solutions even in presence of negative numbers."

❌ Common Mistakes (Interview Killers)

- map.put(0,1) bhool jana ❌

- Negative numbers ke saath sliding window use karna ❌
- Integer overflow → use long if needed
- Prefix sum store karte waqt index overwrite karna

🎯 Mandatory Practice (Java)

- 1 Subarray sum equals K
- 2 Range sum queries
- 3 Equilibrium index
- 4 Longest subarray with sum K

Kadane

Bhai 🔥

Main ab teri **Kadane ka content** ko **Java-focused, interview-ready, aur step-by-step commented** version me rewrite kar raha hoon.

Saath hi **2D Kadane tak extend karenge**, jahan matrix ke max submatrix sum ka solution milega.

🧠 Kadane's Algorithm – FINAL INTERVIEW VERSION (JAVA)

1 Standard Kadane (1D Maximum Subarray Sum)

```
public int maxSubArray(int[] nums) {  
    int currSum = 0;                // Current subarray sum tracker  
    int maxSum = Integer.MIN_VALUE; // Store global maximum sum, handles all-negative array  
  
    for (int x : nums) {            // Traverse each element  
        currSum += x;               // Extend current subarray by adding current element  
        maxSum = Math.max(maxSum, currSum); // Update global max if needed  
  
        if (currSum < 0)             // If current sum drops below 0
```

```

        currSum = 0;                // Start a new subarray (drop negative pre
fix)
    }
    return maxSum;                  // Return max subarray sum
}

```

🕒 Time Complexity: $O(n)$

🧠 Space Complexity: $O(1)$

Dry Run:

`nums = [-2,1,-3,4,-1,2,1,-5,4]`

Best subarray = `[4,-1,2,1]` → `Answer = 6`

Edge Case: All Negative Numbers → handled because `maxSum = Integer.MIN_VALUE`

🏆 **Interview One-Liner:**

"Kadane decides at each index whether to extend the previous subarray or start a new one."

2 Circular Maximum Subarray Sum

```

// Standard Kadane helper
private int kadane(int[] nums) {
    int curr = 0, maxSum = Integer.MIN_VALUE;
    for (int x : nums) {
        curr += x;
        maxSum = Math.max(maxSum, curr);
        if (curr < 0) curr = 0;
    }
    return maxSum;
}

// Circular subarray maximum sum
public int maxSubarraySumCircular(int[] nums) {
    int normalMax = kadane(nums);    // Max sum without wrapping

    int totalSum = 0;
    for (int i = 0; i < nums.length; i++) {

```

```

        totalSum += nums[i];
        nums[i] = -nums[i];           // Invert elements
    }


    int minSub = kadane(nums);         // Max of inverted array → gives minimum subarray of original
    int circularMax = totalSum + minSub; // Circular max = totalSum - minSub

    if (circularMax == 0)              // All negative case (minSub = -totalSum)
        return normalMax;

    return Math.max(normalMax, circularMax); // Return max of normal and circular
}

```

Example:

`nums = [5,-3,5]` → Normal = 7, Circular = 10 

3 2D Kadane – Maximum Submatrix Sum

Idea:

Reduce 2D matrix problem to multiple 1D Kadane problems:

1. Fix top and bottom rows
2. Collapse rows into a 1D array (column sums)
3. Apply standard Kadane on this 1D array

```

public int maxSumRectangle(int[][] matrix) {
    int rows = matrix.length;
    int cols = matrix[0].length;
    int maxSum = Integer.MIN_VALUE;

    // Fix top row
    for (int top = 0; top < rows; top++) {
        int[] temp = new int[cols]; // Collapsed row sums

```

```

// Extend bottom row
for (int bottom = top; bottom < rows; bottom++) {
    for (int c = 0; c < cols; c++) {
        temp[c] += matrix[bottom][c]; // Sum of elements from top to bottom for each column
    }

    // Apply 1D Kadane on collapsed array
    int currMax = kadane(temp);
    maxSum = Math.max(maxSum, currMax);
}
}
return maxSum;
}

// Reuse 1D Kadane helper
private int kadane(int[] arr) {
    int currSum = 0, maxSum = Integer.MIN_VALUE;
    for (int x : arr) {
        currSum += x;
        maxSum = Math.max(maxSum, currSum);
        if (currSum < 0) currSum = 0;
    }
    return maxSum;
}

```

🕒 Time Complexity: $O(\text{rows}^2 \times \text{cols})$

🧠 Space Complexity: $O(\text{cols})$

Dry Run Example:

```

matrix = [
    [1, 2, -1],
    [-3, 4, 5],
    [2, -1, 2]
]
Collapsed arrays and Kadane → maximum submatrix sum = 10

```

✅ FAANG / Google Variations (Kadane)

1. Maximum Product Subarray (track min & max)
 2. Maximum Sum with At Most One Deletion
 3. K Concatenation Maximum Sum
 4. Maximum Subarray with Size $\geq K$
 5. 2D Maximum Submatrix Sum
-

❌ Common Mistakes

- Setting `maxSum = 0` → fails for all-negative arrays
 - Forgetting **circular case logic**
 - Confusing **Kadane vs Sliding Window**
 - Forgetting to invert for circular sum
-

🏆 Interview Gold Line

“Kadane’s algorithm is an optimal linear-time DP solution for maximum subarray problems, extendable to circular and 2D variants.”

Subarray

Bhai 🔥

Tu already Subarray ke FAANG-level skeleton par hai, ab main isko **ultimate MASTER GUIDE** bana raha hoon, jisme **Java + line-by-line intuition + interview traps + pattern recognition** sab ek jagah ho 💪

101 Subarray Problems – COMPLETE MASTER GUIDE (Java + Interview)


🧠 Subarray kya hota hai?

Continuous part of an array

```
arr = [1,2,3]
```

Subarrays:

```
[1], [1,2], [1,2,3], [2], [2,3], [3]
```

 **Important:** Continuous hona compulsory

 [1,3] → subsequence, NOT subarray

1 All Subarrays

Total Subarrays Formula:

```
[  
n * (n + 1) / 2  
]
```

Reason:


- 1st element → n subarrays
- 2nd element → n-1
- ...
- last element → 1

Brute Force (understanding ke liye):

```
for(int i = 0; i < n; i++) {  
    for(int j = i; j < n; j++) {  
        // subarray from i to j  
    }  
}
```

 $O(n^2)$ → Mostly educational

2 Subarray Sum Problems

 **A) Subarray Sum = K (Negative allowed)**

GOLD PATTERN: Prefix Sum + HashMap

LeetCode 560 (Google favourite)

```

public int subarraySum(int[] nums, int k) {
    HashMap<Integer, Integer> map = new HashMap<>();
    map.put(0, 1); // VERY IMPORTANT: subarrays starting at index 0

    int sum = 0, count = 0;

    for (int x : nums) {
        sum += x;

        if (map.containsKey(sum - k))
            count += map.get(sum - k); // found previous prefix sum

        map.put(sum, map.getOrDefault(sum, 0) + 1);
    }
    return count;
}

```

🕒 $O(n)$ | 🧠 $O(n)$

⚠️ **Interview Trap:**

❌ `map.put(0,1)` bhool gaye → Wrong answer

✅ B) Subarray Sum = K (Only Positive)

Technique: Sliding Window

```

public int subarraySumPositive(int[] nums, int k) {
    int left = 0, sum = 0, count = 0;

    for (int right = 0; right < nums.length; right++) {
        sum += nums[right];

        while (sum > k)
            sum -= nums[left++];

        if (sum == k)
            count++;
    }
}

```

```
    return count;
}
```

 **Note:** Negative numbers → Sliding Window FAIL

3 Longest Subarray Problems

✅ A) Longest Subarray with Sum = K (Negative allowed)

Pattern: Prefix Sum + Earliest Index

```
public int longestSubarraySumK(int[] nums, int k) {
    HashMap<Integer, Integer> map = new HashMap<>();
    int sum = 0, maxLen = 0;

    for (int i = 0; i < nums.length; i++) {
        sum += nums[i];

        if (sum == k)
            maxLen = i + 1;

        if (map.containsKey(sum - k))
            maxLen = Math.max(maxLen, i - map.get(sum - k));

        map.putIfAbsent(sum, i); // store earliest index
    }
    return maxLen;
}
```

 **Trap:**

 Latest index store karna → WRONG

 Always store earliest index

✅ B) Longest Subarray with Sum ≤ K (Only positive numbers)

Pattern: Sliding Window (expand + shrink)

| Same logic as "Longest subarray ≤ K" problem

4 Count Subarrays with Conditions

✓ A) Count Subarrays with Sum = K

| Same as LC 560 → Prefix Sum + HashMap

✓ B) Count Subarrays with XOR = K

Pattern: Prefix XOR + HashMap

```
public int subarrayXor(int[] arr, int k) {
    HashMap<Integer, Integer> map = new HashMap<>();
    map.put(0, 1);

    int xr = 0, count = 0;

    for (int x : arr) {
        xr ^= x; // prefix XOR

        if (map.containsKey(xr ^ k))
            count += map.get(xr ^ k);

        map.put(xr, map.getOrDefault(xr, 0) + 1);
    }
    return count;
}
```

✓ C) Count Subarrays with Equal 0s & 1s

Trick: 0 → -1, 1 → +1

```
public int countEqualZeroOne(int[] arr) {
    HashMap<Integer, Integer> map = new HashMap<>();
    map.put(0, 1);

    int sum = 0, count = 0;

    for (int x : arr) {
        sum += (x == 0 ? -1 : 1);
        count += map.getOrDefault(sum, 0);
    }
}
```

```

        map.put(sum, map.getOrDefault(sum, 0) + 1);
    }
    return count;
}

```

📌 Same logic → Longest equal 0s & 1s

🧠 PATTERN RECOGNITION (INTERVIEW GOLD)

Condition	Technique
All positive	Sliding Window
Negative allowed	Prefix Sum + HashMap
Count subarrays	HashMap frequency
Longest subarray	Prefix Sum + earliest index
XOR based	Prefix XOR + HashMap

⚠️ COMMON INTERVIEW MISTAKES

- Sliding window on negative numbers ❌
- Prefix sum map initialization miss ❌
- Count vs longest logic mix ❌
- Latest index store karna ❌

🏆 INTERVIEW ONE-LINER

"Subarray problems are optimized using sliding window when elements are positive, and prefix-sum hashing when negatives or counting is involved."

🚀 FINAL FAANG PRACTICE LIST

- 1 LC 560 – Subarray Sum Equals K
- 2 LC 525 – Equal 0s & 1s
- 3 LC 974 – Subarray Sum Divisible by K
- 4 LC 1248 – Nice Subarrays
- 5 GFG – Subarray with Given Sum

Bhai 🔥

Chal main tujhe **Subarray Problems ka core idea** ekdum **step-by-step, story format** me samjha deta hoon, taaki tu **interview me instantly pattern recognize** kar sake 💪

Subarray Problems – Core Idea Story

1 Subarray = Continuous Part of Array

Soch:

“Array ek railway track hai, aur subarray ek train hai jo track ke consecutive stations se guzarti hai.”

- `[1, 2, 3]` → Train ka route: `[1]`, `[1,2]`, `[1,2,3]`, `[2]`, `[2,3]`, `[3]`
- Agar koi station skip hua → `[1,3]` → **ye subarray nahi, ye subsequence hai**

Key Insight:

Continuous hona compulsory → yahi sliding window aur prefix sum ka logic build hota hai.

2 Sliding Window (Positive Numbers)

Story:

“Tumhare paas ek magical window hai, jo array pe move karti hai.

Tumhe chahiye window ka **sum ya condition**. Window grow kare ya shrink kare jaise tum chaaho, lekin only positive numbers me.”

- Window expand → `right pointer` move
- Sum exceed → `left pointer` shrink
- Sum match → record count / length

Example:

arr = `[2,1,5,1,3,2]`, k = 3 → Max sum window

- Window `[2,1,5]` → sum = 8
- Move right → `[1,5,1]` → sum = 7

- `[5,1,3]` → sum = 9  Max

Key Insight:

Sliding window → $O(n)$ solution, **only works with all positives**.

3 Prefix Sum + HashMap (Negatives Allowed)

Story:

"Sliding window negative numbers me fail hota hai... ab tumhare paas ek magic notebook (HashMap) hai, jisme tum prefix sums likh sakte ho.




Agar tumhe subarray sum = K chahiye, tum check kar sakte ho:

`currentPrefixSum - previousPrefixSum = K`

- `map.put(0,1)` → Subarray starting at index 0 ka account rakhna
- Iterate array:
 - `sum += nums[i]`
 - Agar `(sum - K)` exist karta map me → Count badhao
 - Update map: `map.put(sum, map.getOrDefault(sum,0)+1)`

Example:

arr = `[1,2,3]`, k = 3

1. sum = 1 → 1 - 3 = -2 
2. sum = 3 → 3 - 3 = 0  Count = 1
3. sum = 6 → 6 - 3 = 3  Count = 2

Key Insight:

Prefix sum + hashmap → Linear time, **negatives allowed**.

Sliding window → fail with negatives.

4 Longest Subarray (Prefix + Earliest Index)

Story:

"Ab tumhare paas time machine (earliest index) hai.

Tum chahte ho longest subarray sum = K → tum pehle sum ka earliest occurrence note kar lo.

Jab future me `sum - K` mile → longest subarray = distance current index - earliest index"

- `map.putIfAbsent(sum, i)` → Earliest sum store
- `maxLen = max(maxLen, i - map.get(sum - K))`

Key Interview Insight:

Hamesha **earliest index** store karo → Latest index galat hai

5 Count Subarrays with XOR

Story:

"XOR wali subarray ek secret door hai...

Tumhare paas prefix XOR ka magic key hai.

Formula: `prefixXOR ^ K = previousPrefixXOR` → Agar exist kare map me → Count += frequency"

- Same concept as sum, bas XOR property use hoti hai
- Prefix XOR → HashMap

6 Count Equal 0s & 1s

Story:

"0 aur 1 ki subarray problem → tum 0 ko -1 treat kar do, 1 ko +1.

Ab sum = 0 → equal number of 0s & 1s."

- Prefix sum + HashMap logic same
- Elegant trick → convert problem to **sum = 0**

7 Pattern Recognition Table

Condition	Technique	Story Analogy
All positive	Sliding Window	Moving magical window
Negative allowed	Prefix Sum + HashMap	Magic notebook of sums
Count subarrays	HashMap frequency	Counting spells
Longest subarray	Prefix Sum + earliest index	Time machine (earliest occurrence)

Condition	Technique	Story Analogy
XOR based	Prefix XOR + HashMap	Secret door with XOR key

8 Core Philosophy (Interview GOLD)

1. **Positive numbers** → Sliding window (expand + shrink)
2. **Negative numbers / count / longest** → Prefix sum + HashMap
3. **Index matters?** → Always store earliest occurrence
4. **XOR / special transformation?** → Convert to prefix + hashmap problem

Interview one-liner:

"Subarray problems are optimized using sliding window when elements are positive, and prefix-sum hashing when negatives or counting is involved."

Bhai, ye story format tumhare **brain me ek map** bana dega:

- Sliding window → moving window
- Prefix sum → magic notebook
- Earliest index → time machine
- XOR → secret door

Ab tu **kisi bhi subarray question** ka **pattern instantly recognize** kar lega 🧠🔥

Interview Insights

Bhai 🔥🔥

Ye jo content tune diya hai, main ise **engineer-level, FAANG-ready, ultra-polished** form me tujhe explain kar deta hoon — jaise interviewer sochta hai, step by step with **intuition + Java examples + mindset** 100

🧠 Selection vs Rejection – Engineer Mindset

Interviewer yahi dekhta hai:

“Tu sirf code likh raha hai ya **soch ke trade-offs + constraints samajh raha hai?**”

- ❌ Code likhne wale → reject
- ✅ Problem ko **optimize, edge case proof, explainable** banane wale → select

1 Space Optimization

Idea: Extra memory kam karo, input array ko reuse karo jab possible ho

Example: Prefix Sum In-Place

```
// Instead of creating extra prefix array
for (int i = 1; i < n; i++)
    arr[i] += arr[i - 1]; // reuse input array
```

Interview Line:

“We can optimize space by reusing the input array.”

Thought Process:

- Input modify allowed?
- Order matter karega baad me?

2 In-Place Algorithm

Idea: Input array ko **directly modify karo**, extra array avoid karo

Common In-Place Problems:

- Reverse array
- Rotate array
- Remove duplicates
- Sort colors (Dutch National Flag)

Java Tip:

```
Collections.reverse(list);
```

Interview Trap:

| ❌ Assume mat kar input modify kar sakte ho

Correct Line:

| "Can I modify the input array, or should I preserve it?"

3 Overflow Handling 🔥🔥🔥

Top 5 rejection reason

Integer mid calculation:

```
// WRONG – overflow risk
int mid = (l + r) / 2;

// CORRECT
int mid = l + (r - l) / 2;
```

Sum Overflow:

```
long sum = 0; // always use long for cumulative sums
```

Use Cases:

- Prefix sum
- Kadane
- Binary search on answer
- Subarray sums

Interview Line:

| "I'll use long to avoid integer overflow."

4 Edge Cases (MOST IMPORTANT)

Mentally check always:

- Empty array

- Size = 1
- All same elements
- All negative
- Sorted / reverse sorted

Example: Kadane

```
int maxSum = Integer.MIN_VALUE; // handles all-negative arrays
```

Interview Line:

| "Let's consider edge cases like all-negative arrays."

5 Negative Numbers Handling

Problem: Sliding Window fail hota hai, negative sum break kar dete hain

Solution Techniques:

- Prefix sum + HashMap
- Kadane
- Binary search carefully

GOLD RULE:

Input Type	Best Technique
All positive	Sliding window
Negative present	Prefix sum + HashMap
Max subarray	Kadane

6 Duplicate Handling

Check:

- Duplicates allowed?
- Unique answer?
- First / last occurrence?

Example:

```
if (!map.containsKey(x))  
    map.put(x, i); // store first occurrence
```

Shows:

- Constraint awareness
- Correct index selection

! Common Interview Killers 💀

- ❌ Constraints clarify nahi ki
- ❌ Overflow ignore kar diya
- ❌ Edge case bhool gaye
- ❌ Extra space jab in-place possible tha

| Yahi pe 90% log reject hote hain

🏆 GOLDEN INTERVIEW LINES (Use consciously)

- 1 "I'll optimize this to constant space."
- 2 "This approach works only for positive numbers."
- 3 "We must handle integer overflow here."
- 4 "Let's consider the edge cases."
- 5 "Time complexity is $O(n)$ and space is $O(1)$."

Benefit: Confidence + Clarity dikhata hai, sirf code nahi

🧠 FINAL INTERVIEW MINDSET

| Correct + Efficient + Explainable = Selected

| Sirf code likhna enough nahi

| Explain karna = Engineer