

Practice Sheet 1

LC 744

LC 744 = Lower Bound ka advanced version



Problem Samajh Le

Given:

- Sorted array of characters `letters`
- Ek target character

Return:

👉 Smallest character strictly greater than target

👉 Agar koi bada character na mile → **wrap around** (return first character)



Example

```
letters = ['c', 'f', 'j']
target = 'a' → answer = 'c'
target = 'c' → answer = 'f'
target = 'd' → answer = 'f'
target = 'g' → answer = 'j'
target = 'j' → answer = 'c' (wrap around)
```



⚠ Important: Strictly greater (`>`), not `>=`



1

Brute Force Approach



Intuition

Array sorted hai

Left se scan karo

Jo first char target se bada mile → return

Agar na mile → first element return

Time Complexity

$O(n)$

Code (Java)

```
class Solution {  
    public char nextGreatestLetter(char[] letters, char target) {  
        for (char ch : letters) {  
            if (ch > target) {  
                return ch;  
            }  
        }  
        return letters[0]; // wrap around  
    }  
}
```

Brute me galti

- `>=` laga dete hain instead of `>`
- Wrap around bhool jaate hain

2 Better Approach (Binary Search Conceptually)

Yaha interviewer dekh raha hai:

| Kya tum lower bound samajhte ho?

We need:

👉 First element strictly greater than target

Yeh basically:

Upper Bound of target



3

Optimal Approach (Binary Search)



Intuition

Sorted array hai → Binary search use karenge.

We want:

Smallest index jahan $\text{letters}[j] > \text{target}$

So condition:

- If $\text{letters}[\text{mid}] \leq \text{target} \rightarrow$ right side jao
- Else → answer ho sakta hai → left me search karo



Visualization

```
letters = [c f j]
target = d
```

c \leq d → ignore

f $>$ d → candidate

Try left for smaller greater element



Optimal Code (Java)

```
class Solution {
    public char nextGreatestLetter(char[] letters, char target) {
        int low = 0, high = letters.length - 1;

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

```

        if (letters[mid] <= target) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }

    // Wrap around using modulo
    return letters[low % letters.length];
}
}

```

🔥 Ye Modulo Trick Kyu?

Agar target \geq last element

Example:

```

letters = [c f j]
target = j

```

Binary search ke baad:

```
low = 3
```

But array size = 3

Toh:

```
low % 3 = 0
```

Return letters[0]

🔥 Elegant wrap-around

✖ Common Interview Mistakes

1 >= vs >

Galat comparison

Wrong:

```
if (letters[mid] < target)
```

Correct:

```
if (letters[mid] <= target)
```

2 Wrap around bhol jana

Return:

```
return letters[low];
```

Galat hoga jab low == n

3 Infinite loop

Wrong:

```
while(low < high)
```

Edge cases me fail ho sakta hai

Use:

```
while(low <= high)
```

4 Off by one error

Binary search me:

- mid calculation
- high = mid - 1
- low = mid + 1

Ye precise hona chahiye

Edge Cases

Case	Example	Output
Single element	['a'], target='z'	'a'
All same	['c','c','c'], target='c'	'c'
Target smaller than all	target='a'	first element
Target equal to last	target='j'	first element
Target between elements	normal case	next greater

Interviewer Kya Test Kar Raha Hai?

- Lower bound concept
 - Strict comparison understanding
 - Wrap around handling
 - Edge case thinking
 - Clean binary search template
-

Pattern Recognition

LC 744 =

Upper Bound in sorted array

Ye same pattern lagega:

- Find first element > x
 - Smallest element greater than target
 - Next alphabet
 - Ceiling in sorted array
-

Google Crack Level Insight

Binary Search = 3 types

1. Exact match
2. Lower bound (\geq)
3. Upper bound ($>$)

Ye question type 3 hai.



Final Clean Version (Interview Safe)

```
class Solution {  
    public char nextGreatestLetter(char[] letters, char target) {  
        int n = letters.length;  
        int low = 0, high = n - 1;  
  
        while (low <= high) {  
            int mid = low + (high - low) / 2;  
  
            if (letters[mid] > target) {  
                high = mid - 1;  
            } else {  
                low = mid + 1;  
            }  
        }  
  
        return letters[low % n];  
    }  
}
```

LC 852

Bhai LC 852 🔥

Ye **binary search on pattern recognition** wala question hai. Interviewer dekh raha hai:

- Kya tum mountain pattern samajh rahe ho?

- Kya tum slope-based binary search laga sakte ho?
- Kya tum mid+1 safe use karna jaante ho?

Chal full breakdown karte hain — brute → better → optimal → mistakes → edge cases.



Problem Samjhoo

Mountain array:

Strictly increasing

Then

Strictly decreasing

Example:

```
arr = [0, 2, 5, 7, 6, 3, 1]
Peak index = 3 (value = 7)
```

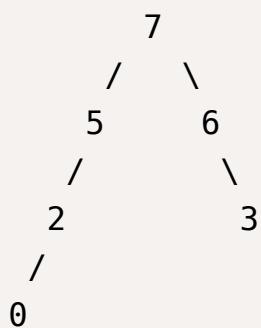
Guarantee:

- arr.length >= 3
 - Exactly one peak
-



Visualization

Graph socho:



Peak wahi hai jahan:

```
arr[i-1] < arr[i] > arr[i+1]
```

1 Brute Force Approach

Intuition

Har index check karo:

- Kya left se bada?
- Kya right se bada?

Time Complexity

$O(n)$

Code (Java)

```
class Solution {  
    public int peakIndexInMountainArray(int[] arr) {  
        for (int i = 1; i < arr.length - 1; i++) {  
            if (arr[i] > arr[i - 1] && arr[i] > arr[i + 1])  
                return i;  
        }  
        return -1; // won't happen due to constraints  
    }  
}
```

Brute Mistakes

- $i = 0$ se start kar dena ($arr[-1]$ crash)
- last index check karna ($arr[n]$ crash)

2 Better Approach (Linear but smarter thinking)

Observe:

- Jab tak increasing hai → peak nahi
- Jaise hi decreasing start → previous index peak

```
class Solution {  
    public int peakIndexInMountainArray(int[] arr) {  
        for (int i = 1; i < arr.length; i++) {  
            if (arr[i] < arr[i - 1]) {  
                return i - 1;  
            }  
        }  
        return -1;  
    }  
}
```

Still O(n)



3 Optimal Approach (Binary Search)



Real Intuition

Mountain array me 2 slopes hote hain:

Case 1: Mid left side me hai (increasing part)

```
arr[mid] < arr[mid+1]
```

Peak right side me hai.

Case 2: Mid right side me hai (decreasing part)

```
arr[mid] > arr[mid+1]
```

Peak left side ya mid ho sakta hai.

IMPORTANT

Yaha hum `mid+1` compare kar rahe hain

Iqliye:

```
while (low < high)
```

Use karna safe hota hai.

Optimal Code

```
class Solution {  
    public int peakIndexInMountainArray(int[] arr) {  
        int low = 0;  
        int high = arr.length - 1;  
  
        while (low < high) {  
            int mid = low + (high - low) / 2;  
  
            if (arr[mid] < arr[mid + 1]) {  
                // Increasing slope  
                low = mid + 1;  
            } else {  
                // Decreasing slope  
                high = mid;  
            }  
        }  
  
        return low; // or high  
    }  
}
```

Dry Run Example

```
arr = [0,2,5,7,6,3,1]
```

low=0 high=6

mid=3 → arr[3]=7

7 > 6 → decreasing

high=3

low=0 high=3

mid=1 → 2 < 5 → increasing

low=2

low=2 high=3

mid=2 → 5 < 7 → increasing

low=3

Now low=high=3

Answer = 3

🎯 Why This Works?

Binary search on **slope direction**

We are not searching value.

We are searching change in trend.

This is advanced binary search pattern 🔥

✗ Common Interview Mistakes

1 Using while($low \leq high$)

Galat ho sakta hai kyunki mid+1 access kar rahe ho.

Use:

```
while (low < high)
```

2 high = mid - 1 likh dena

Galat.

Correct:

```
high = mid;
```

Kyuki mid peak ho sakta hai.

3 Index out of bound

arr[mid+1] safe tabhi hai jab low < high use karo.

4 Sochna ki peak value return karna hai

Question index maangta hai.

🔥 Edge Cases

Case	Example
Smallest mountain	[0,1,0]
Peak near start	[0,5,3,1]
Peak near end	[0,1,2,5,4]



Pattern Recognition

LC 852 same pattern as:

- Find Peak Element (LC 162)
- Bitonic Array Peak
- Search in Mountain Array
- Maximum in Bitonic Array



Interview Insight

Interviewer test karta hai:

- Kya tum monotonic property detect kar sakte ho?
 - Kya tum trend-based binary search laga sakte ho?
 - Kya tum off-by-one control kar sakte ho?
-



Final Interview-Ready Code

```
class Solution {  
    public int peakIndexInMountainArray(int[] arr) {  
        int low = 0, high = arr.length - 1;  
  
        while (low < high) {  
            int mid = low + (high - low) / 2;  
  
            if (arr[mid] < arr[mid + 1]) {  
                low = mid + 1;  
            } else {  
                high = mid;  
            }  
        }  
  
        return low;  
    }  
}
```

Time: $O(\log n)$

Space: $O(1)$

LC 162

Bhai LC 162 🔥

Ye **Peak Index (852)** ka big brother hai.

Yaha twist ye hai ki **mountain guarantee nahi hai**.

Interviewer check karta hai:

- Kya tum local peak samajhte ho?

- Kya tum monotonic thinking bina sorted array ke laga sakte ho?
 - Kya tum binary search ko "pattern detection" ke liye use kar sakte ho?
-

Problem Samjho

Given an array:

Return **any peak element index.**

Peak element:

```
nums[i] > nums[i-1] AND nums[i] > nums[i+1]
```

Important:

- $\text{nums}[-1] = -\infty$
 - $\text{nums}[n] = -\infty$
 - At least one peak guaranteed
-

Example 1

```
nums = [1,2,3,1]
Peak = 2 (value 3)
```

Example 2

```
nums = [1,2,1,3,5,6,4]
Possible answers = 1 or 5
```

Multiple peaks allowed.

Brute Force Approach



Intuition

Har element check karo.

Time Complexity

$O(n)$

Code

```
class Solution {
    public int findPeakElement(int[] nums) {
        int n = nums.length;

        for (int i = 0; i < n; i++) {
            boolean left = (i == 0) || (nums[i] > nums[i - 1]);
            boolean right = (i == n - 1) || (nums[i] > nums[i + 1]);

            if (left && right) {
                return i;
            }
        }

        return -1;
    }
}
```



2

Optimal Approach (Binary Search)

 Ye important hai.

Yaha array sorted nahi hai.

Phir bhi binary search possible hai.

Why?

Because:

If:

```
nums[mid] < nums[mid + 1]
```

Toh right side me **definitely ek peak hoga**.

If:

```
nums[mid] > nums[mid + 1]
```

Toh left side me peak hoga (including mid).

Intuition Deeply Samjho

Socho tum slope pe khade ho.

- Agar right side upar ja rahi hai → peak right me hai
- Agar right side neeche ja rahi hai → peak left me hai

Since edges = $-\infty$

Peak always exist karega.

Optimal Code

```
class Solution {  
    public int findPeakElement(int[] nums) {  
        int low = 0;  
        int high = nums.length - 1;  
  
        while (low < high) {  
            int mid = low + (high - low) / 2;  
  
            if (nums[mid] < nums[mid + 1]) {  
                low = mid + 1;  
            } else {  
                high = mid;  
            }  
        }  
  
        return low;  
    }  
}
```

Time: $O(\log n)$

Space: $O(1)$



Dry Run

```
nums = [1,2,1,3,5,6,4]
```

low=0 high=6

mid=3 → 3 < 5 → go right

low=4

low=4 high=6

mid=5 → 6 > 4 → go left

high=5

low=4 high=5

mid=4 → 5 < 6 → go right

low=5

low=high=5

Return 5



Why high = mid ?

Because mid itself peak ho sakta hai.

If you do:

```
high = mid - 1
```

You might skip the peak.



Common Interview Mistakes

- 1 Thinking array sorted hona chahiye

Nahi chahiye.

2 Using while($\text{low} \leq \text{high}$)

Yaha $\text{mid}+1$ access kar rahe ho

So use:

```
while( $\text{low} < \text{high}$ )
```

3 Out of bound

Never check $\text{nums}[\text{mid}-1]$ or $\text{nums}[\text{mid}+1]$ blindly.

4 Trying to find maximum

Ye maximum search nahi hai.

Ye local peak search hai.

🔥 Edge Cases

Case	Example
Single element	[5]
Two elements	[1,2]
Strictly increasing	[1,2,3,4]
Strictly decreasing	[4,3,2,1]



Compare LC 852 vs LC 162

852	162
Mountain guaranteed	No guarantee
One peak	May be multiple
Strict increase then decrease	Any shape
Same binary logic works	Yes

Same slope logic 🔥

 Interview Insight

Interviewer test karta hai:

- Can you apply binary search without sorted array?
 - Can you reason about slope?
 - Can you prove peak always exists?
 - Can you avoid index errors?
-



Ultimate Clean Version

```
class Solution {  
    public int findPeakElement(int[] nums) {  
        int low = 0, high = nums.length - 1;  
  
        while (low < high) {  
            int mid = low + (high - low) / 2;  
  
            if (nums[mid] > nums[mid + 1]) {  
                high = mid;  
            } else {  
                low = mid + 1;  
            }  
        }  
  
        return low;  
    }  
}
```

LC 33

Bhai LC 33 🔥

Ye **interview classic** hai. Google, Amazon, Microsoft sab puchte hain.

Ye test karta hai:

- Sorted half detect karna

- Rotation logic
- Binary search me decision making
- Edge case handling

Chal full breakdown karte hain 

Problem Samjhoo

Array sorted tha ascending me.

Phir kisi pivot pe rotate kar diya gaya.

Example:

```
Original: [0,1,2,4,5,6,7]
Rotated: [4,5,6,7,0,1,2]
```

Target diya hai → index return karo

Nahi mila → -1

Time complexity must be **O(log n)**

1

Brute Force

Linear search.

```
class Solution {
    public int search(int[] nums, int target) {
        for(int i = 0; i < nums.length; i++) {
            if(nums[i] == target)
                return i;
        }
        return -1;
    }
}
```

Time: O(n)

Interview me reject ho jayega 



2

Optimal Approach (Binary Search)

Core idea:

Har iteration me:

Ek half hamesha sorted hota hai

Kyuki array originally sorted tha.



Intuition Deep Dive

Example:

[4,5,6,7,0,1,2]

Mid nikalo.

Check:

nums[low] <= nums[mid]

Agar true:

→ Left half sorted hai

Else:

→ Right half sorted hai

🎯 Step 1: Detect Sorted Half

```
if(nums[low] <= nums[mid])
    left sorted
else
    right sorted
```

🎯 Step 2: Check Target Us Half Me Hai?

If left sorted:

```
if(target >= nums[low] && target < nums[mid])
    high = mid - 1
else
    low = mid + 1
```

If right sorted:

```
if(target > nums[mid] && target <= nums[high])
    low = mid + 1
else
    high = mid - 1
```



Final Code (Java)

```
class Solution {
    public int search(int[] nums, int target) {
        int low = 0, high = nums.length - 1;

        while (low <= high) {
            int mid = low + (high - low) / 2;

            if (nums[mid] == target)
                return mid;

            // Left half sorted
            if (nums[low] <= nums[mid]) {

                if (target >= nums[low] && target < nums[mi
d]) {
                    high = mid - 1;
                } else {
                    low = mid + 1;
                }
            }

            // Right half sorted
```

```

        else {

            if (target > nums[mid] && target <= nums[high]) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }

        return -1;
    }
}

```

Time: $O(\log n)$

Space: $O(1)$



Dry Run

Example:

```

nums = [4,5,6,7,0,1,2]
target = 0

```

low=0 high=6

mid=3 → 7

Left sorted ($4 \leq 7$)

Target not in 4–7

→ go right

low=4 high=6

mid=5 → 1

Left sorted ($0 \leq 1$)

Target in 0–1

→ go left

```
high=4  
low=4 high=4  
mid=4 → 0  
Found.
```

✗ Common Mistakes

1 <= vs < confusion

Use:

```
nums[low] <= nums[mid]
```

Not <

2 Wrong boundary check

Be careful:

Left sorted case:

```
target >= nums[low]  
target < nums[mid]
```

Right sorted case:

```
target > nums[mid]  
target <= nums[high]
```

3 Infinite loop

Always:

```
while(low <= high)
```

4 Forgetting mid match check first

Always check:

```
if(nums[mid] == target)
```

Before sorted logic.

🔥 Edge Cases

Case	Example
Single element	[1]
Not rotated	[1,2,3,4]
Rotated at last	[2,3,4,1]
Target not present	return -1

💎 Interview Insight

Interviewer dekh raha hai:

- Kya tum pivot find kiye bina solve kar sakte ho?
 - Kya tum decision tree logically bana sakte ho?
 - Kya tum off-by-one errors avoid karte ho?
-

🧠 Pattern Recognition

LC 33 base hai for:

- LC 153 (Find Min in Rotated Array)
 - LC 81 (Rotated with Duplicates)
 - Search in Bitonic Array
 - Binary Search on Rotated Structures
-

🚀 Ultra Clean Template

Ye yaad rakh:

1. Check mid match
2. Detect sorted half
3. Check target in sorted half
4. Move accordingly

Bas.

LC 81

Bhai LC 81 🔥

Ye LC 33 ka **evil version** hai.

Difference kya hai?

👉 **Duplicates allowed**

Aur yahi pura game change karta hai 😈



Problem Samjho

Given:

Rotated sorted array

Duplicates ho sakte hain

Return:

`true` if target exists

`false` otherwise

Example

```
nums = [2,5,6,0,0,1,2]
target = 0 → true
target = 3 → false
```



Why This Is Hard?

LC 33 me:

At least one half always strictly sorted hota tha

But yaha duplicates ki wajah se:

nums[low] == nums[mid] == nums[high]

Toh tum decide hi nahi kar paoge kaunsa half sorted hai.

Binary search ka clear direction khatam ho jaata hai.

Worst case → O(n)

1 Brute Force

```
class Solution {  
    public boolean search(int[] nums, int target) {  
        for(int num : nums) {  
            if(num == target) return true;  
        }  
        return false;  
    }  
}
```

Time: O(n)



2 Optimal Modified Binary Search



Intuition

Steps:

1. If mid == target → return true
2. If left sorted ($\text{nums}[\text{low}] < \text{nums}[\text{mid}]$)
3. If right sorted ($\text{nums}[\text{mid}] < \text{nums}[\text{high}]$)
4. If duplicates block decision → shrink window

Important Duplicate Case

If:

```
nums[low] == nums[mid] == nums[high]
```

Then:

```
low++  
high--
```

Kyuki direction decide nahi ho sakta.



Final Code (Java)

```
class Solution {  
    public boolean search(int[] nums, int target) {  
        int low = 0, high = nums.length - 1;  
  
        while (low <= high) {  
            int mid = low + (high - low) / 2;  
  
            if (nums[mid] == target)  
                return true;  
  
            // Duplicate case  
            if (nums[low] == nums[mid] && nums[mid] == nums[high]) {  
                low++;  
                high--;  
            }  
  
            // Left sorted  
            else if (nums[low] <= nums[mid]) {  
  
                if (target >= nums[low] && target < nums[mid]) {  
                    high = mid - 1;  
                } else {  
                    low = mid + 1;  
                }  
            } else {  
                low = mid + 1;  
            }  
        }  
        return false;  
    }  
}
```

```

        high = mid - 1;
    } else {
        low = mid + 1;
    }
}

// Right sorted
else {

    if (target > nums[mid] && target <= nums[high]) {
        low = mid + 1;
    } else {
        high = mid - 1;
    }
}

return false;
}
}

```

Dry Run (Duplicate Trap)

```

nums = [1,1,1,1,1,2,1]
target = 2

```

low=0 high=6

mid=3 → nums[mid]=1

nums[low]==nums[mid]==nums[high]

→ low++ high--

Window shrink hota rahega

Eventually 2 mil jayega.

⚠ Why Worst Case O(n)?

Consider:

```
[1,1,1,1,1,1,1,1,1]
```

Har baar:

```
low++  
high--
```

Binary search degrade ho jata hai.

✖ Common Interview Mistakes

1 Duplicate case ignore karna

Ye sabse bada trap hai.

2 <= vs < galti

Use:

```
nums[low] <= nums[mid]
```

Not strict <

3 Infinite loop

Always:

```
while(low <= high)
```

4 Forgetting mid match check first

Always:

```
if(nums[mid] == target)
```

Top pe rakho.



Compare LC 33 vs LC 81

Feature	LC 33	LC 81
Duplicates	No	Yes
Always O(log n)	Yes	No
Clean direction	Yes	Sometimes blocked
Needs shrink trick	No	Yes



Interview Insight

Interviewer test karta hai:

- Can you adapt logic when assumptions break?
 - Can you handle ambiguous cases?
 - Do you know when binary search degrades?
-



Pattern Summary

1. Check mid
2. If duplicates → shrink
3. Detect sorted half
4. Check target in half
5. Move accordingly



Final Polished Version

```
class Solution {
    public boolean search(int[] nums, int target) {
        int low = 0, high = nums.length - 1;

        while (low <= high) {
```

```

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

        if (nums[mid] == target)
            return true;

        if (nums[low] == nums[mid] && nums[mid] == nums[high]) {
            low++;
            high--;
        }
        else if (nums[low] <= nums[mid]) {
            if (target >= nums[low] && target < nums[mid])
                high = mid - 1;
            else
                low = mid + 1;
        }
        else {
            if (target > nums[mid] && target <= nums[high])
                low = mid + 1;
            else
                high = mid - 1;
        }
    }

    return false;
}
}

```

Agar tu real Google crack mode me hai 🦸

Next logical step:

- LC 153 – Find Minimum in Rotated Sorted Array
- LC 154 – Find Minimum with Duplicates
- Complete Rotated Array Pattern Mastery

Bol bhai next kaunsa todna hai 🔥

LC 153

Bhai LC 153 🔥

Ye rotated array pattern ka **core concept** hai.

Agar ye solid ho gaya →

LC 33, 81, 154 sab easy lagne lagenge.



Problem Samjho

Given:

- Sorted ascending array
- Rotated at some pivot
- **No duplicates**

Return:

👉 Minimum element

Example 1

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

Example 2

```
nums = [4,5,6,7,0,1,2]  
Answer = 0
```



Key Observation

Minimum hi actually:

```
Rotation pivot element
```

Original sorted array me smallest first hota tha.

Rotation ke baad woh beech me aa jata hai.

1 Brute Force

```
class Solution {  
    public int findMin(int[] nums) {  
        int min = nums[0];  
        for(int num : nums) {  
            min = Math.min(min, num);  
        }  
        return min;  
    }  
}
```

Time: $O(n)$

Interview reject 



2 Optimal Binary Search



Deep Intuition

Observe:

If array not rotated:
 $\text{nums}[\text{low}] < \text{nums}[\text{high}]$

→ Minimum = $\text{nums}[\text{low}]$

Else:

Compare mid with high.

Case 1:

$\text{nums}[\text{mid}] > \text{nums}[\text{high}]$

Means:

- Minimum right side me hai

Example:

```
[4,5,6,7,0,1,2]
```

↑

So:

```
low = mid + 1
```

Case 2:

```
nums[mid] < nums[high]
```

Means:

- Minimum left side me ya mid ho sakta hai

So:

```
high = mid
```

Notice:

Not `mid - 1`

Because mid could be minimum.



Final Code (Java)

```
class Solution {
    public int findMin(int[] nums) {
        int low = 0, high = nums.length - 1;

        while (low < high) {
            int mid = low + (high - low) / 2;

            if (nums[mid] > nums[high]) {
```

```

        low = mid + 1;
    } else {
        high = mid;
    }
}

return nums[low];
}
}

```

Time: $O(\log n)$

Space: $O(1)$



Dry Run

```
nums = [4,5,6,7,0,1,2]
```

low=0 high=6

mid=3 → 7 > 2

→ minimum right

low=4

low=4 high=6

mid=5 → 1 < 2

→ minimum left

high=5

low=4 high=5

mid=4 → 0 < 1

→ high=4

low=high=4

Return $\text{nums}[4] = 0$



Why Compare With HIGH?

Because:

Rightmost part always sorted hota hai.

Mid ko high se compare karna easier hai pivot detect karne ke liye.

Common Mistakes

1 Using while($low \leq high$)

Wrong here.

Use:

```
while( $low < high$ )
```

2 $high = mid - 1$

Galat.

Correct:

```
high = mid
```

3 Comparing with $\text{nums}[low]$

Complex ho jata hai.

Compare with $\text{nums}[high]$ is cleaner.

4 Forgetting not-rotated case

Code automatically handle kar leta hai.

Example:

```
[1,2,3,4]
```

Eventually high move hote hote 0 pe aa jayega.

Edge Cases

Case	Example
Single element	[5]
Already sorted	[1,2,3,4]
Rotated once	[2,1]
Large rotation	[5,1,2,3,4]

Pattern Recognition

Ye same logic use hota hai:

- LC 154 (with duplicates)
 - Find rotation count
 - Search in rotated array
 - Find pivot index
-

Interview Insight

Interviewer dekh raha hai:

- Can you identify monotonic break?
 - Can you reason using high comparison?
 - Do you understand why mid can be answer?
 - Can you avoid off-by-one?
-

Difference From LC 33

LC 33	LC 153
Search target	Find minimum
Detect sorted half	Detect pivot
Move based on target	Move based on mid vs high

Ultra Clean Template

```
while(low < high):
    mid = ...
    if nums[mid] > nums[high]:
        low = mid + 1
    else:
        high = mid
```

Return nums[low]

Bas.

Agar tu serious mastery mode me hai 🦸

Next natural step:

- LC 154 – Find Minimum with Duplicates (harder)
- Rotation count problem
- Complete rotated array pattern sheet

Bol bhai next kaunsa todna hai 🔥

LC 875

Bhai LC 875 🔥

Ye **Binary Search on Answer** ka classic question hai.

Google, Amazon, Uber sab puchte hain.

Agar ye concept clear ho gaya →

Saare “minimum rate / minimum time / maximum capacity” wale questions easy ho jayenge.



Problem Samjho

Koko ke paas piles hain:

```
piles = [3,6,7,11]
h = 8 hours
```

Koko har hour me 🍌 bananas kha sakti hai.

Rules:

- Ek hour me sirf ek pile choose karegi
- Agar pile me k se kam bache \rightarrow pura khatam karegi
- Har hour exactly 1 pile pe kaam

Goal:

👉 Minimum k find karo such that Koko \leq h hours me finish kare.

🎯 Output

For example:

```
piles = [3,6,7,11]
h = 8
Answer = 4
```

🔥 Why This Is Binary Search on Answer?

Hum direct k nahi jaante.

But:

- Minimum k = 1
- Maximum k = max(piles)

Search space:

```
[1 ... maxPile]
```

We are searching smallest valid k.

1 Brute Force

Check every k from 1 to maxPile.

For each k:

- Total hours calculate karo

- Agar $\leq h \rightarrow$ return

Time complexity:

$O(\maxPile * n)$

Worst case me bahut slow 



2

Optimal – Binary Search on Answer



Core Insight

As k increases:

- Required hours decrease

Monotonic function hai.

$k \uparrow \rightarrow \text{hours} \downarrow$

So we can binary search minimum valid k.



How To Check Valid k?

For each pile:

Hours needed:

$\text{ceil}(pile / k)$

Important:

Java me:

$(pile + k - 1) / k$

Use karo instead of Math.ceil.



Code (Java)

```

class Solution {
    public int minEatingSpeed(int[] piles, int h) {
        int low = 1;
        int high = 0;

        for (int pile : piles) {
            high = Math.max(high, pile);
        }

        while (low < high) {
            int mid = low + (high - low) / 2;

            if (canFinish(piles, h, mid)) {
                high = mid;
            } else {
                low = mid + 1;
            }
        }

        return low;
    }

    private boolean canFinish(int[] piles, int h, int k) {
        long hours = 0;

        for (int pile : piles) {
            hours += (pile + k - 1) / k; // ceil division
        }

        return hours <= h;
    }
}

```

Time:

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

Dry Run

```
piles = [3,6,7,11]  
h = 8
```

Search space:

```
1 to 11
```

Try mid=6:

Hours:

3 → 1

6 → 1

7 → 2

11 → 2

Total=6 ≤ 8

Valid → try smaller

Try mid=3:

3 → 1

6 → 2

7 → 3

11 → 4

Total=10 > 8

Not valid → increase k

Eventually answer=4



Important Pattern

Ye type ke questions me:

You are searching for:

Minimum k such that condition true

Template:

```
if(condition(mid))
    high = mid
else
    low = mid + 1
```

✖ Common Interview Mistakes

1 Using double and Math.ceil

Slow + unnecessary.

Use:

```
(pile + k - 1) / k
```

2 Integer overflow

Use:

```
long hours
```

Because sum big ho sakte hai.

3 Using while(low <= high)

Yaha better hai:

```
while(low < high)
```

4 Wrong boundary update

Remember:

We want smallest valid k.

So:

```
high = mid
```

not mid-1.

🎯 Edge Cases

Case	Example
Single pile	[10], h=5
h = piles.length	answer = max(pile)
h very large	answer = 1
Very large values	overflow careful

💎 Interview Insight

Interviewer test karta hai:

- Can you detect monotonic function?
 - Can you convert word problem to binary search?
 - Can you handle ceil division correctly?
 - Can you avoid overflow?
-

🧠 Pattern Recognition

LC 875 same category:

- LC 1011 – Capacity To Ship Packages
- LC 1283 – Smallest Divisor Given Threshold
- LC 1482 – Minimum Days to Make Bouquets
- Allocate Books
- Aggressive Cows

Sabka same template 🔥



Final Interview Template

Binary search on answer

```
low = minimum possible  
high = maximum possible
```

```
while(low < high):  
    mid  
    if(valid(mid)):  
        high = mid  
    else:  
        low = mid + 1  
  
return low
```

Agar tu serious Google crack mode me hai 😈

Next karte hain:

- LC 1011 (Ship Packages)
- Allocate Books (classic)
- Aggressive Cows (harder intuition)
- Complete Binary Search on Answer Mastery

Bol bhai next kaunsa todna hai 🔥

LC 349

Bhai LC 349 🔥

Ye easy lagta hai, lekin interviewer **set logic, hashing aur deduplication clarity** test karta hai.



Problem Samjho

Given:

```
nums1 = [1,2,2,1]
nums2 = [2,2]
```

Return:

```
[2]
```

Important:

- Result me **unique elements** hone chahiye
- Order matter nahi karta

🎯 Core Understanding

We need:

- 👉 Common elements
- 👉 No duplicates in output

1 Brute Force Approach

💡 Intuition

Har element nums1 ka nums2 me search karo.

Use nested loops.

```
class Solution {
    public int[] intersection(int[] nums1, int[] nums2) {
        List<Integer> result = new ArrayList<>();

        for(int i = 0; i < nums1.length; i++) {
            for(int j = 0; j < nums2.length; j++) {
                if(nums1[i] == nums2[j] && !result.contains
(nums1[i])) {
                    result.add(nums1[i]);
                }
            }
        }
    }
}
```

```

    }

    int[] ans = new int[result.size()];
    for(int i = 0; i < result.size(); i++)
        ans[i] = result.get(i);

    return ans;
}
}

```

Time:

$O(n * m)$

Interview me acceptable nahi ✗

2 Better Approach – Sorting + Two Pointer

Idea

1. Dono arrays sort karo
 2. Two pointer lagao
 3. Common element mila → store karo
 4. Duplicate skip karo
-

Code

```

import java.util.*;

class Solution {
    public int[] intersection(int[] nums1, int[] nums2) {
        Arrays.sort(nums1);
        Arrays.sort(nums2);

```

```

int i = 0, j = 0;
Set<Integer> set = new HashSet<>();

while(i < nums1.length && j < nums2.length) {
    if(nums1[i] == nums2[j]) {
        set.add(nums1[i]);
        i++;
        j++;
    }
    else if(nums1[i] < nums2[j]) {
        i++;
    }
    else {
        j++;
    }
}

int[] result = new int[set.size()];
int index = 0;
for(int num : set)
    result[index++] = num;

return result;
}
}

```

Time:

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



3

Optimal Approach – HashSet (Best)

🔥 Most preferred in interview



Intuition

1. nums1 ke elements set me daal do

2. nums2 traverse karo
3. Agar element set me hai → result set me daal do

Automatically duplicates handle ho jayenge.



Code

```
import java.util.*;  
  
class Solution {  
    public int[] intersection(int[] nums1, int[] nums2) {  
        Set<Integer> set1 = new HashSet<>();  
        Set<Integer> result = new HashSet<>();  
  
        for(int num : nums1)  
            set1.add(num);  
  
        for(int num : nums2) {  
            if(set1.contains(num))  
                result.add(num);  
        }  
  
        int[] ans = new int[result.size()];  
        int index = 0;  
  
        for(int num : result)  
            ans[index++] = num;  
  
        return ans;  
    }  
}
```

Time:

$O(n + m)$

Space:

$O(n)$

Best approach 

Common Mistakes

1 Forgetting uniqueness

Using ArrayList instead of Set → duplicates aa jayenge.

2 Confusing with LC 350

LC 350 me:

- Duplicates allowed in output

LC 349 me:

- Only unique

3 Returning List instead of int[]

Conversion zaroori hai.

Edge Cases

Case	Example
No intersection	[1,2,3], [4,5]
One empty array	[]
All elements same	[1,1,1], [1]
Negative numbers	[-1,-2], [-2]

Pattern Recognition

LC 349 category:

- Set operations
- Hashing

- Two pointer after sorting
-



Compare LC 349 vs LC 350

Feature	LC 349	LC 350
Unique result	Yes	No
Use Set	Yes	No
Use Map for freq	No	Yes



Interview Insight

Interviewer dekh raha hai:

- Kya tum hashing use kar sakte ho?
 - Kya tum uniqueness samajhte ho?
 - Kya tum time complexity optimize karte ho?
-



Cleanest Interview Version

```
class Solution {
    public int[] intersection(int[] nums1, int[] nums2) {
        Set<Integer> set1 = new HashSet<>();
        Set<Integer> result = new HashSet<>();

        for(int num : nums1)
            set1.add(num);

        for(int num : nums2)
            if(set1.contains(num))
                result.add(num);

        return result.stream().mapToInt(Integer::intValue).
toArray();
    }
}
```

```
    }  
}
```

Agar tu mastery mode me hai 😈

Next logical step:

- LC 350 – Intersection II (frequency version)
- Two pointer mastery sheet
- HashMap pattern deep dive

Bol bhai next kya todna hai 🔥

LC 350

Bhai LC 350 🔥

Ye LC 349 ka **frequency version** hai.

Yaha twist:

- 👉 Output me duplicates allowed
- 👉 Jitni baar dono arrays me common aaye utni baar result me hona chahiye



Problem Samjhו

Example:

```
nums1 = [1,2,2,1]  
nums2 = [2,2]
```

Output:

```
[2,2]
```

Kyuki:

- 2 nums1 me 2 baar
- 2 nums2 me 2 baar
 - Minimum frequency = 2

Key Concept

Result frequency =

```
min(freq in nums1, freq in nums2)
```

1 Brute Force (Not Good)

Nested loops + visited marking.

Time:

```
O(n * m)
```

Interview reject 

2 Optimal Approach – HashMap (Best)

Intuition

1. nums1 ka frequency map bana lo
2. nums2 traverse karo
3. Agar element map me present hai AND freq > 0:
 - result me add karo
 - freq--

Code (Java)

```
import java.util.*;  
  
class Solution {  
    public int[] intersect(int[] nums1, int[] nums2) {  
        Map<Integer, Integer> map = new HashMap<>();
```

```

        for(int num : nums1) {
            map.put(num, map.getOrDefault(num, 0) + 1);
        }

        List<Integer> result = new ArrayList<>();

        for(int num : nums2) {
            if(map.containsKey(num) && map.get(num) > 0) {
                result.add(num);
                map.put(num, map.get(num) - 1);
            }
        }

        int[] ans = new int[result.size()];
        for(int i = 0; i < result.size(); i++)
            ans[i] = result.get(i);

        return ans;
    }
}

```

Time:

$O(n + m)$

Space:

$O(n)$

Best solution 

Dry Run

```

nums1 = [1,2,2,1]
nums2 = [2,2]

```

Map:

```
1 → 2  
2 → 2
```

Traverse nums2:

2 → freq=2 → add → freq=1

2 → freq=1 → add → freq=0

Result:

```
[2,2]
```

3 Two Pointer Approach (When Arrays Sorted)

If arrays already sorted:

1. Use two pointer
2. If equal → add and move both
3. Else move smaller

Code

```
import java.util.*;  
  
class Solution {  
    public int[] intersect(int[] nums1, int[] nums2) {  
        Arrays.sort(nums1);  
        Arrays.sort(nums2);  
  
        int i = 0, j = 0;  
        List<Integer> result = new ArrayList<>();  
  
        while(i < nums1.length && j < nums2.length) {  
            if(nums1[i] == nums2[j]) {  
                result.add(nums1[i]);  
            }  
            else if(nums1[i] < nums2[j]) {  
                i++;  
            }  
            else {  
                j++;  
            }  
        }  
        return result.stream().mapToInt(Integer::intValue).  
    }  
}
```

```

        i++;
        j++;
    }
    else if(nums1[i] < nums2[j]) {
        i++;
    }
    else {
        j++;
    }
}

int[] ans = new int[result.size()];
for(int k = 0; k < result.size(); k++)
    ans[k] = result.get(k);

return ans;
}
}

```

Time:

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

Use when:

- Arrays sorted
- Memory limited

✗ Common Mistakes

1 Forgetting to decrease frequency

Then duplicates extra aa jayenge.

2 Confusing with LC 349

LC 349:

- Unique only → Set

LC 350:

- Frequency count → Map

3 Using Set in LC 350

Wrong answer dega.

🎯 Edge Cases

Case	Example
No intersection	[1,2], [3,4]
One empty	[]
All same	[1,1,1], [1,1]
Negative numbers	[-1,-2], [-2,-2]



Follow-up (Interview Important)

Interviewer pooch sakte hai:

1 If nums1 small and nums2 huge?

→ Build map of smaller array.

2 If arrays sorted?

→ Two pointer.

3 If nums2 on disk (cannot load fully)?

→ Chunk processing + binary search if sorted.



Compare LC 349 vs 350

Feature	LC 349	LC 350
Unique result	Yes	No
Data structure	Set	HashMap
Frequency needed	No	Yes

 Interview Insight

Ye question test karta hai:

- Frequency counting skill
 - HashMap mastery
 - Two pointer understanding
 - Space vs time tradeoff
-



Cleanest Interview Version

```
class Solution {
    public int[] intersect(int[] nums1, int[] nums2) {
        Map<Integer, Integer> map = new HashMap<>();

        for(int num : nums1)
            map.put(num, map.getOrDefault(num, 0) + 1);

        List<Integer> result = new ArrayList<>();

        for(int num : nums2) {
            int count = map.getOrDefault(num, 0);
            if(count > 0) {
                result.add(num);
                map.put(num, count - 1);
            }
        }

        return result.stream().mapToInt(i -> i).toArray();
    }
}
```

Agar tu mastery mode me hai 😎

Next logical step:

- 4 Sum

- 3 Sum
- Two pointer full mastery sheet
- HashMap advanced pattern
- Sliding window deep dive

Bol bhai next kya todna hai 🔥

LC 448

Bhai LC 448 🔥

Ye question **index mapping + cyclic sort + in-place marking** ka classic hai.

Google level me ye isliye puchte hain kyunki ye check karta hai:

- Kya tum constraint use kar sakte ho?
- Kya tum extra space avoid kar sakte ho?
- Kya tum array ko hash table ki tarah treat kar sakte ho?



Problem Samjhו

Given:

```
nums length = n
Elements range: [1, n]
Some appear twice, some once
Find missing numbers
```

Example:

```
nums = [4,3,2,7,8,2,3,1]
n = 8

Missing = [5,6]
```



Important Constraint

Elements are in range:

```
1 ≤ nums[i] ≤ n
```

Ye hi trick hai 🔥

1 Brute Force

For each number 1 to n:

Check if exists in array.

Time:

```
O(n2)
```

Reject ✗

2 Better – Using HashSet



1. Sab elements set me daal do
 2. 1 se n tak check karo kaun missing hai
-



```
import java.util.*;  
  
class Solution {  
    public List<Integer> findDisappearedNumbers(int[] nums)  
    {  
        Set<Integer> set = new HashSet<>();  
  
        for(int num : nums)  
            set.add(num);  
    }  
}
```

```

List<Integer> result = new ArrayList<>();

for(int i = 1; i <= nums.length; i++) {
    if(!set.contains(i))
        result.add(i);
}

return result;
}
}

```

Time:

$O(n)$

Space:

$O(n)$

Good but not optimal.

3 Optimal – In-Place Marking (Most Important)

 Ye real interview solution hai.

Core Idea

Since numbers are 1 to n:

Each number should ideally be at index:

`value - 1`

We can mark visited numbers by making that index negative.

How?

For each num:

```
index = abs(num) - 1  
nums[index] = -abs(nums[index])
```

After marking:

Any index i where $\text{nums}[i]$ is positive \rightarrow missing number $= i + 1$



Code

```
import java.util.*;  
  
class Solution {  
    public List<Integer> findDisappearedNumbers(int[] nums)  
{  
    List<Integer> result = new ArrayList<>();  
  
    for(int i = 0; i < nums.length; i++) {  
        int index = Math.abs(nums[i]) - 1;  
  
        if(nums[index] > 0) {  
            nums[index] = -nums[index];  
        }  
    }  
  
    for(int i = 0; i < nums.length; i++) {  
        if(nums[i] > 0) {  
            result.add(i + 1);  
        }  
    }  
  
    return result;  
}
```



Dry Run

```
nums = [4,3,2,7,8,2,3,1]
```

Marking:

4 → index 3 → make negative

3 → index 2 → negative

2 → index 1 → negative

7 → index 6 → negative

8 → index 7 → negative

2 → already negative

3 → already negative

1 → index 0 → negative

After marking:

```
[-4,-3,-2,-7,8,2,-3,-1]
```

Check positive:

Index 4 → 8 positive → missing 5

Index 5 → 2 positive → missing 6

Answer:

```
[5,6]
```

🔥 Why It Works?

Because:

- Values map directly to valid indices
- We use sign as visited flag
- No extra space needed

✗ Common Mistakes

1 Forgetting abs()

Very common bug:

After marking negative, next iteration wrong index aa sakte hai.

Always:

```
Math.abs(nums[i])
```

2 Modifying array without checking positive

Always check:

```
if(nums[index] > 0)
```

3 Returning index instead of index+1

Remember:

```
Missing number = i + 1
```

🎯 Edge Cases

Case	Example
All present	[1,2,3,4]
All same	[1,1]
Single element	[1]
Missing multiple	[2,2]

💎 Interview Insight

Interviewer dekh raha hai:

- Can you use constraints smartly?
- Can you convert value to index?
- Can you solve without extra space?

- Do you understand in-place marking?
-

Pattern Recognition

Ye category hai:

- Cyclic Sort Pattern
 - Index Mapping Problems
 - Missing / Duplicate detection
 - First Missing Positive (Harder version)
-

Alternative: Cyclic Sort Approach

If you rearrange numbers:

Put each number at correct index.

Then check mismatch.

Time: $O(n)$

Space: $O(1)$

Big Picture

LC 448 build karta hai base for:

- LC 41 – First Missing Positive (Hard)
 - Find Duplicate Number
 - Set Mismatch
 - Cyclic Sort Pattern Mastery
-

Final Clean Version

```
class Solution {  
    public List<Integer> findDisappearedNumbers(int[] nums)  
{
```

```

        for(int i = 0; i < nums.length; i++) {
            int index = Math.abs(nums[i]) - 1;
            if(nums[index] > 0)
                nums[index] = -nums[index];
        }

        List<Integer> result = new ArrayList<>();

        for(int i = 0; i < nums.length; i++) {
            if(nums[i] > 0)
                result.add(i + 1);
        }

        return result;
    }
}

```

Time: O(n)

Space: O(1) (excluding output)

LC 41

Bhai ye **LC 41 – First Missing Positive** FAANG favourite hai 🔥

Ye simple lagta hai but trick hidden hai. Yaha interviewer dekh raha hota hai:

- In-place thinking
- Index mapping idea
- O(n) time + O(1) space approach

Chal teen approach full detail me samajhte hain 👇



Problem

Given an unsorted array, find **smallest missing positive integer**.

Example:

Input: [1,2,0]

Output: 3

Input: [3,4,-1,1]

Output: 2

Input: [7,8,9,11,12]

Output: 1



Intuition (Most Important)

Agar array length = n hai

To answer **range me hoga [1 to n+1]**

Kyu?

- Agar 1 se n tak sab present hai \rightarrow answer = n+1
- Warna jo missing hai woh isi range me hoga

Example:

[1,2,3] \rightarrow answer = 4

[2,3,4] \rightarrow answer = 1

Iska matlab:

Hume sirf 1 to n tak ke numbers important hain.



1 Brute Force (Sorting)

Idea

Sort karo, fir 1 se check karte jao.

Steps

1. Sort array

2. `expected = 1`
3. Loop through:
 - agar number == expected → `expected++`
4. Return `expected`

Code (Java)

```
import java.util.Arrays;

class Solution {
    public int firstMissingPositive(int[] nums) {
        Arrays.sort(nums);
        int expected = 1;

        for (int num : nums) {
            if (num == expected) {
                expected++;
            }
        }
        return expected;
    }
}
```

Complexity

- Time: $O(n \log n)$
- Space: $O(1)$

Problems

- Sorting not allowed ideally
- Interviewer wants $O(n)$

Better (HashSet)

Idea

Store all numbers in HashSet

Then check from 1 to n

Code

```
import java.util.HashSet;

class Solution {
    public int firstMissingPositive(int[] nums) {
        HashSet<Integer> set = new HashSet<>();

        for (int num : nums) {
            if (num > 0) {
                set.add(num);
            }
        }

        for (int i = 1; i <= nums.length; i++) {
            if (!set.contains(i)) {
                return i;
            }
        }

        return nums.length + 1;
    }
}
```



Complexity

- Time: $O(n)$
- Space: $O(n)$

✗ Problem

Space allowed nahi hai ideally



3

Optimal (Cyclic Sort Concept) 🔥 🔥

Ye interview killer hai.



Core Idea

Har number x ko index $x-1$ par rakh do.

Example:

```
Value 1 → index 0  
Value 2 → index 1  
Value 3 → index 2
```

Step 1: Place numbers in correct position

Condition:

```
nums[i] > 0  
nums[i] <= n  
nums[i] != nums[nums[i]-1]
```

Tab swap karo.

Example

```
[3,4,-1,1]  
n = 4
```

Process:

```
3 → index 2  
swap
```

```
[-1,4,3,1]
```

```
4 → index 3
```

```
swap  
[-1,1,3,4]
```

```
1 → index 0  
swap
```

```
[1,-1,3,4]
```

Now check:

index 1 → should be 2 but -1 hai

Answer = 2

Code (Java)

```
class Solution {  
    public int firstMissingPositive(int[] nums) {  
        int n = nums.length;  
  
        int i = 0;  
        while (i < n) {  
            int correctIndex = nums[i] - 1;  
  
            if (nums[i] > 0 && nums[i] <= n  
                && nums[i] != nums[correctIndex]) {  
  
                int temp = nums[i];  
                nums[i] = nums[correctIndex];  
                nums[correctIndex] = temp;  
            } else {  
                i++;  
            }  
        }  
  
        for (i = 0; i < n; i++) {  
            if (nums[i] != i + 1) {  
                return i + 1;  
            }  
        }  
    }  
}
```

```
        }
    }

    return n + 1;
}
}
```

Complexity

- Time: $O(n)$
- Space: $O(1)$
- Every element swapped at most once

Common Mistakes (Interview me log fail yahi hote)

1. Negative numbers handle nahi karte

Ignore karna hai.

2. Duplicate case infinite loop

Istiyे condition:

```
nums[i] != nums[correctIndex]
```

Very important.

3. Index out of bound

Check:

```
nums[i] <= n
```

warna crash.

4. $n+1$ case bhool jana

Agar sab correct position me hai

Return n+1



Interview Thinking Pattern

Jab bhi question bole:

- Missing number
- Positive number
- O(n)
- O(1) space



"Can I use index as hash?"

Ye pattern important hai:

- LC 41
- LC 448
- LC 287
- LC 442

Same family questions.



Edge Cases

[]

→ 1

[1]

→ 2

[-1, -2]

→ 1

[1, 1]

→ 2

[2]

→ 1

🔥 FAANG Level Insight

Ye question test karta hai:

- In-place manipulation
- Cyclic sort
- Index mapping
- Boundary handling

Agar tumne ye clean solve kar diya

Interviewer impressed.

Agar chahe to mai iska dry run table format me bana du

ya iska Python / C++ version bhi de du

Bata bhai next kaunsa todna hai 😎🔥

Remove Duplicates

Bhai ye **LC 26 – Remove Duplicates from Sorted Array**

two-pointer ka sabse fundamental question hai 🔥

Google / Amazon me ye concept baar-baar aata hai.

🔥 Problem

Sorted array diya hai.

Duplicates remove karo **in-place**.

Return → unique elements ki count k

Aur first k positions me unique elements hone chahiye.

Example:

Input: [1,1,2]

Output: 2

Array: [1,2,_]

Input: [0,0,1,1,1,2,2,3,3,4]

Output: 5

Array: [0,1,2,3,4,_,_,_,_,_]



Intuition (Important)

Array sorted hai.

Matlab:

Duplicates hamesha side-by-side honge.

So hume bas:

👉 "Last unique element" track karna hai.



Brute Force (Extra Array)

Idea

New array me unique elements daalo.

Code

```
class Solution {  
    public int removeDuplicates(int[] nums) {  
        if (nums.length == 0) return 0;  
  
        int[] temp = new int[nums.length];  
        int j = 0;  
        temp[j++] = nums[0];  
  
        for (int i = 1; i < nums.length; i++) {  
            if (nums[i] != nums[i - 1]) {  
                temp[j++] = nums[i];  
            }  
        }  
        return j;  
    }  
}
```

```

        temp[j++] = nums[i];
    }
}

for (int i = 0; i < j; i++) {
    nums[i] = temp[i];
}

return j;
}

```

⌚ Complexity

- Time: $O(n)$
- Space: $O(n)$

✖ Interview me space allowed nahi hai.

🟡 2 Better (Two Pointer – Clean Version)

🧠 Core Idea

Ek pointer `i` unique element track karega.

Ek pointer `j` traversal karega.

Visualization

```

[1,1,2,2,3]
  i
  j

```

Jab `nums[j] != nums[i]`

→ `i++`

→ `nums[i] = nums[j]`

Code

```

class Solution {
    public int removeDuplicates(int[] nums) {
        if (nums.length == 0) return 0;

        int i = 0; // last unique index

        for (int j = 1; j < nums.length; j++) {
            if (nums[j] != nums[i]) {
                i++;
                nums[i] = nums[j];
            }
        }

        return i + 1;
    }
}

```

Complexity

- Time: $O(n)$
- Space: $O(1)$
- Optimal solution



3 Ultra Clean Variant (More Intuitive)

Same logic but thoda readable:

```

class Solution {
    public int removeDuplicates(int[] nums) {
        int n = nums.length;
        if (n == 0) return 0;

        int uniqueIndex = 0;

        for (int current = 1; current < n; current++) {
            if (nums[current] != nums[uniqueIndex]) {

```

```

        uniqueIndex++;
        nums[uniqueIndex] = nums[current];
    }
}

return uniqueIndex + 1;
}
}

```

Why This Works?

Because array sorted hai.

So duplicate kab aayega?

Jab `nums[j] == nums[i]`

Unique kab?

Jab `nums[j] > nums[i]`

Simple.

Dry Run

Input:

```
[0,0,1,1,1,2,2,3,3,4]
```

Steps:

```

i=0

j=1 → same → skip
j=2 → 1 ≠ 0 → i=1 → nums[1]=1
j=3 → same → skip
j=4 → same → skip
j=5 → 2 ≠ 1 → i=2 → nums[2]=2
j=6 → same
j=7 → 3 ≠ 2 → i=3 → nums[3]=3

```

```
j=8 → same  
j=9 → 4 ≠ 3 → i=4 → nums[4]=4
```

Return 5.



Common Mistakes

✗ 1. i ko 1 se start kar dena

Galat.

`i = 0` se start karna.

✗ 2. Return i instead of i+1

Bahut log yaha galti karte hain.

`i` index hai.

Count = `i + 1`

✗ 3. Sorted property ignore karna

Ye solution unsorted array pe kaam nahi karega.



Interview Insight

Interviewer dekh raha hota hai:

- Two pointer understanding
 - In-place modification
 - Edge case handling
-

💎 Edge Cases

```
[]
```

```
→ 0
```

```
[1]
```

```
→ 1  
  
[1,1,1]  
→ 1
```

```
[1,2,3]  
→ 3
```

🎯 Pattern Family

Ye pattern use hota hai:

- LC 26
- LC 80 (Allow at most 2 duplicates)
- Move Zeroes
- Remove Element
- Merge Sorted Array

Agar chahe to mai LC 80 bhi tod ke samjha du
wo thoda tricky hai 🔥

Next kaunsa marna hai bhai? 😎

LC 977

Bhai ye **LC 977 – Squares of a Sorted Array**

Two Pointer ka ekdam classic question hai 🔥

Google / Amazon me frequently aata hai.

🔥 Problem

Sorted array diya hai (ascending order).

Har element ka square karo.

Return sorted squares array.

Example:

```
Input: [-4,-1,0,3,10]
Output: [0,1,9,16,100]
```

```
Input: [-7,-3,2,3,11]
Output: [4,9,9,49,121]
```

Intuition

Important observation 

Array sorted hai **but squares sorted nahi honge.**

Example:

```
[-4,-1,0,3,10]
Squares:
[16,1,0,9,100]
```

Negative numbers ka square bada ho sakte hai.

 Largest square hamesha left ya right end pe milega.

Iqliye Two Pointer use karenge.

1 Brute Force

Idea

1. Har element ka square karo
2. Sort kar do

Code

```
import java.util.Arrays;

class Solution {
    public int[] sortedSquares(int[] nums) {
        int n = nums.length;
```

```

        for (int i = 0; i < n; i++) {
            nums[i] = nums[i] * nums[i];
        }

        Arrays.sort(nums);
        return nums;
    }
}

```

Complexity

- Time: $O(n \log n)$
- Space: $O(1)$

 Sorting avoid karna chahiye.



2

Better (Extra Array + Two Pointer)

Core Idea

Compare:

$|nums[left]|$ vs $|nums[right]|$

Jo bada hoga uska square end me daalenge.

Why end se fill karte hain?

Because biggest square pehle milega.

Code

```

class Solution {
    public int[] sortedSquares(int[] nums) {
        int n = nums.length;
        int[] result = new int[n];

```

```

        int left = 0;
        int right = n - 1;
        int index = n - 1;

        while (left <= right) {
            int leftSquare = nums[left] * nums[left];
            int rightSquare = nums[right] * nums[right];

            if (leftSquare > rightSquare) {
                result[index] = leftSquare;
                left++;
            } else {
                result[index] = rightSquare;
                right--;
            }
            index--;
        }

        return result;
    }
}

```

Complexity

- Time: $O(n)$
- Space: $O(n)$
- Optimal solution

Most Intuitive Explanation (Visualization)

Input:

`[-7, -3, 2, 3, 11]`

Step:

```
left= -7 (49)
right= 11 (121)
```

Put 121 at end

Then:

```
left= -7 (49)
right= 3 (9)
```

Put 49

Then:

```
left= -3 (9)
right= 3 (9)
```

Put 9

Continue...

Final:

```
[4,9,9,49,121]
```



Common Mistakes

✗ 1. Directly square karke return kar dena

Sorted nahi hoga.

✗ 2. Compare nums[left] and nums[right]

Galat.

Compare:

```
abs(nums[left]) and abs(nums[right])
```

ya squares compare karo.

3. Front se fill karna

Galat order ban jayega.

Always fill from end.



Edge Cases

[0]

→ [0]

[-1]

→ [1]

[-3, -2, -1]

→ [1, 4, 9]

[1, 2, 3]

→ [1, 4, 9]



Pattern Learning

Ye same thinking use hoti hai:

- Container With Most Water
 - Trapping Rain Water
 - Two Sum Sorted
 - Remove Duplicates
 - Move Zeroes
-



Interview Insight

Interviewer dekh raha hota hai:

- Two pointer mastery
 - Observation power
 - Absolute value thinking
 - Sorted property ka use
-

Agar chahe to mai iska in-place attempt bhi discuss kar sakta hu (tricky hota hai).

Next kaunsa todna hai bhai? 😎🔥

LC 88

Bhai ye **LC 88 – Merge Sorted Array**

Two pointer ka ekdam FAANG favourite sawaal hai 🔥

Ye simple lagta hai but trick hidden hai.

🔥 Problem

Do sorted arrays diye gaye hain:

- `nums1` size = $m + n$
- First m elements valid hain
- Last n elements 0 (empty space)
- `nums2` size = n

Goal:

Merge `nums2` into `nums1` in sorted order.

Example

```
nums1 = [1,2,3,0,0,0], m = 3
nums2 = [2,5,6], n = 3
```

Output:

```
[1,2,2,3,5,6]
```



Most Important Intuition

Agar hum front se merge karenge → overwrite ho jayega.

So kya karein?

👉 Back se merge karo.

Kyun?

Because nums1 ke end me empty space already hai.



1

Brute Force (Extra Array)

Idea

New array me merge karo, phir copy karo.

```
class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        int[] temp = new int[m + n];

        int i = 0, j = 0, k = 0;

        while (i < m && j < n) {
            if (nums1[i] <= nums2[j]) {
                temp[k++] = nums1[i++];
            } else {
                temp[k++] = nums2[j++];
            }
        }

        while (i < m) temp[k++] = nums1[i++];
        while (j < n) temp[k++] = nums2[j++];

        for (int x = 0; x < m + n; x++) {
            nums1[x] = temp[x];
        }
    }
}
```

```
    }  
}
```

⌚ Time: $O(m+n)$

📦 Space: $O(m+n)$

✖ Extra space allowed nahi hai ideally.



2 Optimal (Backwards Two Pointer) 🔥

🧠 Core Idea

Three pointers:

```
i = m - 1 (last valid element of nums1)  
j = n - 1 (last of nums2)  
k = m + n - 1 (last index of nums1)
```

Compare from back.

Code (Interview Perfect)

```
class Solution {  
    public void merge(int[] nums1, int m, int[] nums2, int n) {  
        int i = m - 1;  
        int j = n - 1;  
        int k = m + n - 1;  
  
        while (i >= 0 && j >= 0) {  
            if (nums1[i] > nums2[j]) {  
                nums1[k] = nums1[i];  
                i--;  
            } else {  
                nums1[k] = nums2[j];  
                j--;  
            }  
            k--;  
        }  
        while (j >= 0) {  
            nums1[k] = nums2[j];  
            j--;  
            k--;  
        }  
    }  
}
```

```

        }
        k--;
    }

    // If nums2 still has elements
    while (j >= 0) {
        nums1[k] = nums2[j];
        j--;
        k--;
    }
}

```

Why This Works?

Because:

- Largest element hamesha end pe aayega.
- End safe hai (empty space).

Overwrite ka tension nahi.

Dry Run

```

nums1 = [1,2,3,0,0,0]
nums2 = [2,5,6]

```

Step by step:

```

i=2 (3)
j=2 (6)
k=5

6 > 3 → nums1[5] = 6

```

Next:

```
i=2 (3)  
j=1 (5)  
k=4
```

```
5 > 3 → nums1[4] = 5
```

Next:

```
i=2 (3)  
j=0 (2)  
k=3
```

```
3 > 2 → nums1[3] = 3
```

Continue...

Final:

```
[1,2,2,3,5,6]
```



Common Mistakes

✗ 1. Front se merge karna

Data overwrite ho jayega.

✗ 2. nums1 ke leftover handle karna

Jarurat nahi.

Agar nums1 ke elements bach gaye → already correct place pe hain.

✗ 3. nums2 leftover bhool jana

Very common mistake.

Always:

```
while (j >= 0)
```

Edge Cases

```
nums1 = [0], m=0  
nums2 = [1], n=1  
→ [1]
```

```
nums1 = [1], m=1  
nums2 = [], n=0  
→ [1]
```

```
nums1 = [2,0], m=1  
nums2 = [1], n=1  
→ [1,2]
```

Interview Insight

Interviewer test karta hai:

- In-place thinking
- Reverse thinking ability
- Pointer control
- Edge case handling

Pattern Family

Ye pattern use hota hai:

- Merge Two Sorted Lists
- Merge Intervals
- Sorted Array problems
- K Sorted Arrays

Agar chahe to mai iska variation bhi bata du:

👉 Merge without extra space (Gap Method)

Next kaunsa todna hai bhai? 😎🔥

LC 905

Bhai ye **LC 905 – Sort Array By Parity**

easy lagta hai, but two-pointer foundation strong karta hai 🔥



Problem

Array diya hai.

Even numbers pehle aane chahiye,

odd numbers baad me.

Order matter nahi karta.

Example

Input: [3,1,2,4]

Output: [2,4,3,1]

Even first. Bas.



Intuition

Even check kaise?

`num % 2 == 0 → even`

`num % 2 == 1 → odd`

Goal:

👉 Even left side me shift karo

👉 Odd right side me



Brute Force (Extra Array)

Idea

1. Ek array me sab even daalo
2. Phir sab odd daalo

Code

```
class Solution {
    public int[] sortArrayByParity(int[] nums) {
        int[] result = new int[nums.length];
        int index = 0;

        // Add even numbers
        for (int num : nums) {
            if (num % 2 == 0) {
                result[index++] = num;
            }
        }

        // Add odd numbers
        for (int num : nums) {
            if (num % 2 != 0) {
                result[index++] = num;
            }
        }

        return result;
    }
}
```

 Time: O(n)

 Space: O(n)



2 Better (Two Pointer – Stable Variant)

Even milte hi left me push karo.

```
class Solution {
    public int[] sortArrayByParity(int[] nums) {
```

```

int insertPos = 0;

for (int i = 0; i < nums.length; i++) {
    if (nums[i] % 2 == 0) {
        int temp = nums[i];
        nums[i] = nums[insertPos];
        nums[insertPos] = temp;
        insertPos++;
    }
}

return nums;
}

```

Idea

`insertPos` batata hai next even kaha jayega.

3 Optimal (Classic Two Pointer from Both Ends 🔥)

🧠 Core Idea

left → even dhundega
right → odd dhundega

Swap when wrong position.

Code

```

class Solution {
    public int[] sortArrayByParity(int[] nums) {
        int left = 0;
        int right = nums.length - 1;

```

```

        while (left < right) {

            if (nums[left] % 2 > nums[right] % 2) {
                int temp = nums[left];
                nums[left] = nums[right];
                nums[right] = temp;
            }

            if (nums[left] % 2 == 0) left++;
            if (nums[right] % 2 == 1) right--;
        }

        return nums;
    }
}

```

Why This Works?

Because:

```

even % 2 = 0
odd % 2 = 1

```

So condition:

```
nums[left] % 2 > nums[right] % 2
```

Means:

left odd & right even → swap.

Elegant trick 🔥

Dry Run

Input:

```
[3,1,2,4]
```

Step:

```
left=0 (3 odd)  
right=3 (4 even)
```

```
swap → [4,1,2,3]
```

Move pointers...

Final:

```
[4,2,1,3]
```

Valid.



Common Mistakes

✗ 1. Sorting use karna

Unnecessary.

✗ 2. Single pointer without swap logic

Odd elements overwrite ho sakte hain.

✗ 3. Pointer move galat order me

Always after swap check both sides.



Edge Cases

```
[1]  
→ [1]
```

```
[2]  
→ [2]
```

[2, 4, 6]

→ same

[1, 3, 5]

→ same

🎯 Interview Insight

Interviewer check karta hai:

- Partition logic samajhte ho?
- In-place swap kar sakte ho?
- Two pointer control?

🔥 Pattern Family

Ye same pattern use hota hai:

- Move Zeroes
- Dutch National Flag
- Sort Colors
- Partition Array
- QuickSort partition step

Agar chahe to mai iska variation bhi bata du:

👉 Even index pe even numbers (LC 922)

👉 Stable version

Next kaunsa todna hai bhai? 😎🔥

LC 15

Bhai ye **LC 15 – 3Sum**

Google / Amazon / Meta ka favourite hai 🔥

Ye question tumhari **sorting + two pointer + duplicate handling** skill test karta hai.

Agar ye clean solve kar diya → strong DSA signal.

Problem

Given array `nums`, find all unique triplets:

```
nums[i] + nums[j] + nums[k] == 0
```

Conditions:

- $i \neq j \neq k$
 - No duplicate triplets
-

Example

```
Input: [-1,0,1,2,-1,-4]
```

```
Output:
```

```
[[ -1, -1, 2 ],  
 [ -1, 0, 1 ]]
```



Core Intuition

3 numbers ka sum 0.

Brute force → 3 loops

Better approach → Reduce to 2Sum

Important Trick:

Sort the array

Fix one element

Remaining array me 2Sum solve karo

1 Brute Force (Triple Loop)

Idea

Try all triplets.

```
class Solution {  
    public List<List<Integer>> threeSum(int[] nums) {  
        Set<List<Integer>> set = new HashSet<>();  
        int n = nums.length;  
  
        for (int i = 0; i < n; i++) {  
            for (int j = i+1; j < n; j++) {  
                for (int k = j+1; k < n; k++) {  
                    if (nums[i] + nums[j] + nums[k] == 0) {  
                        List<Integer> triplet =  
                            Arrays.asList(nums[i], nums[j],  
                                         nums[k]);  
                        Collections.sort(triplet);  
                        set.add(triplet);  
                    }  
                }  
            }  
        }  
  
        return new ArrayList<>(set);  
    }  
}
```

⌚ Time: $O(n^3)$

📦 Space: $O(n)$

✖ Too slow.

2 Better (Sorting + HashSet for 2Sum)

Idea

For each i :

- $\text{target} = -\text{nums}[i]$
- Find 2Sum using HashSet

Time: $O(n^2)$

Space: $O(n)$

Still not optimal.

3 Optimal (Sorting + Two Pointer 🔥)

🧠 Core Idea

1. Sort array
 2. Fix i
 3. Use left & right pointer
-

Why sorting?

- Duplicate easily skip kar sakte hain
 - Two pointer use kar sakte hain
-

💎 Algorithm

```
Sort array

for i in 0 → n-3
    if i > 0 and nums[i] == nums[i-1]
        continue // skip duplicate

    left = i+1
    right = n-1

    while left < right
        sum = nums[i] + nums[left] + nums[right]
```

```

if sum == 0
    add triplet
    skip duplicates
    left++
    right--

else if sum < 0
    left++

else
    right--

```

🔥 Code (Interview Perfect)

```

import java.util.*;

class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        List<List<Integer>> result = new ArrayList<>();
        Arrays.sort(nums);
        int n = nums.length;

        for (int i = 0; i < n - 2; i++) {

            if (i > 0 && nums[i] == nums[i - 1])
                continue; // skip duplicate i

            int left = i + 1;
            int right = n - 1;

            while (left < right) {
                int sum = nums[i] + nums[left] + nums[right];
                if (sum == 0) {
                    result.add(Arrays.asList(
                        nums[i], nums[left], nums[right]

```

```

        ));

        // skip duplicates
        while (left < right && nums[left] == nums[left + 1])
            left++;
        while (left < right && nums[right] == nums[right - 1])
            right--;

        left++;
        right--;
    }
    else if (sum < 0) {
        left++;
    }
    else {
        right--;
    }
}

return result;
}
}

```

Complexity

- Time: $O(n^2)$
- Space: $O(1)$ (excluding output)
- This is optimal.

Dry Run

Input:

```
[-1, 0, 1, 2, -1, -4]
```

Sorted:

```
[-4, -1, -1, 0, 1, 2]
```

i = 0 → -4

Find 2Sum = 4 → no pair

i = 1 → -1

left=2 right=5

- $1 + (-1) + 2 = 0$ ✓

Next:

- $1 + 0 + 1 = 0$ ✓

Duplicates skip.

Done.



Most Common Mistakes

✗ 1. Duplicate skip nahi karna

Interview fail.

```
if (i > 0 && nums[i] == nums[i-1])
```

Mandatory.

✗ 2. Triplet milne ke baad left++ right-- bhool jana

✗ 3. Duplicate skip after match nahi karna

```
while (nums[left] == nums[left+1])
```

Very important.

4. Not sorting

Two pointer work nahi karega.

Edge Cases

[]

→ []

[0]

→ []

[0, 0, 0]

→ [[0, 0, 0]]

[1, 2, -2, -1]

→ []



Interview Insight

Interviewer test karta hai:

- 2Sum reduction thinking
- Sorting importance
- Duplicate handling
- Pointer movement logic

Pattern Family

- 2Sum
- 3Sum Closest
- 4Sum
- KSum
- Two Pointer problems

FAANG Level Thinking

Jab bhi:

- "3 numbers"
- "Unique triplets"
- "Sum target"

👉 Think:

Fix one + Two Pointer.

Agar chahe to mai:

- 3Sum Closest
- 4Sum
- General KSum template
- ya interview edge-case thinking

bhi tod ke samjha du 🔥

Next kaunsa Google level todna hai bhai? 😎

LC 287

Bhai ye **LC 287 – Find the Duplicate Number**

Aur specifically **Floyd's Cycle Detection (Tortoise & Hare)** wala approach

ye Google level question hai 🔥🔥🔥

Ye question dikhne me array ka hai

but actually ye **linked list cycle problem** hai.

Agar tum ye samajh gaye → interview me strong signal.



Problem

Array `nums` given hai:

- Size = $n + 1$
- Elements range = $[1, n]$

- Exactly 1 duplicate number
- Multiple times appear ho sakte hai

Return duplicate number.

Example:

Input: [1,3,4,2,2]

Output: 2

Input: [3,1,3,4,2]

Output: 3



Important Constraints

- Modify array nahi kar sakte
- O(1) space
- O(n) time

Sorting allowed nahi

HashSet allowed nahi

To kaise karein?



Brute Force

Double loop check.

$O(n^2)$

Reject.



Better (HashSet)

```
class Solution {
    public int findDuplicate(int[] nums) {
        HashSet<Integer> set = new HashSet<>();
```

```

        for (int num : nums) {
            if (set.contains(num))
                return num;
            set.add(num);
        }
        return -1;
    }
}

```

⌚ O(n)

📦 O(n)

Space allowed nahi.

3 Optimal – Floyd Cycle Detection

Core Mind-Blowing Intuition

Array ko graph ki tarah treat karo.

Each index → next index:

```
i → nums[i]
```

Example:

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

Graph:

```

0 → 1
1 → 3
3 → 2
2 → 4
4 → 2 (cycle starts here)

```

Cycle create ho gaya.

Duplicate number hi cycle ka entry point hai.

🔥 Why?

Because two indexes pointing to same number

matlab two arrows same node pe ja rahe hain

→ cycle banega.



Linked List Analogy

Ye exactly same hai:

Linked list with cycle

Use Floyd's algorithm:

- Slow moves 1 step
- Fast moves 2 steps

They will meet inside cycle.



Phase 1: Detect meeting point

```
slow = nums[slow]
fast = nums[nums[fast]]
```

Jab tak equal na ho.



Phase 2: Find cycle start

- Reset slow = 0
- Move both 1 step

Jaha milenge → duplicate

🔥 Code (Interview Perfect)

```
class Solution {
    public int findDuplicate(int[] nums) {

        int slow = nums[0];
        int fast = nums[0];

        // Phase 1: detect cycle
        do {
            slow = nums[slow];
            fast = nums[nums[fast]];
        } while (slow != fast);

        // Phase 2: find entrance
        slow = nums[0];

        while (slow != fast) {
            slow = nums[slow];
            fast = nums[fast];
        }

        return slow;
    }
}
```

🧪 Dry Run

Input:

```
[1,3,4,2,2]
```

Phase 1:

```
slow = 1
fast = 1
```

```
slow = nums[1] = 3
fast = nums[nums[1]] = nums[3] = 2

slow = nums[3] = 2
fast = nums[nums[2]] = nums[4] = 2
```

They meet at 2.

Phase 2:

```
slow = nums[0] = 1
fast = 2

slow = nums[1] = 3
fast = nums[2] = 4

slow = nums[3] = 2
fast = nums[4] = 2
```

Meet at 2.

Answer = 2.



Why It Works (Deep Intuition)

Cycle math:

Distance before cycle = L

Cycle length = C

After meeting:

Reset slow to start

Move both 1 step

They meet at cycle start.

Mathematically proven property of Floyd.



Common Mistakes

1. Starting slow = 0

Wrong.

Start:

```
slow = nums[0]
fast = nums[0]
```

2. Using while instead of do-while in phase 1

First movement required.

3. Thinking this is sorting problem

It's graph problem.



Complexity

Time: $O(n)$

Space: $O(1)$



Interview Insight

Interviewer test karta hai:

- Can you convert array to graph?
- Do you know cycle detection?
- Can you reason mathematically?



Pattern Family

Same idea used in:

- Linked List Cycle II
- Happy Number
- Detect cycle problems



FAANG Level Thinking

Jab constraints bole:

- $n+1$ elements
- values 1 to n
- exactly one duplicate
- no modification
- constant space

👉 Immediately think:

Pigeonhole principle + cycle detection.

Agar chahe to mai iska:

- Binary Search on Answer approach
- Marking negative approach
- Mathematical proof visualization

bhi tod ke samjha du 🔥

Bata bhai next Google breaker kaunsa hai? 😎

LC 42

Bhai ye **Trapping Rain Water (LC 42)**

Google / Meta / Amazon ka classic hai 🔥

Aur sahi bola tune —

ye sliding window nahi hai ✗

Ye problem **prefix/suffix max + two pointer** category ki hai.



Problem

Given elevation map:

```
height = [0,1,0,2,1,0,1,3,2,1,2,1]
```

Return total water trapped.

Output:

6



Core Intuition (Golden Line)

Water at index i depends on:

```
min(max height on left, max height on right) - height[i]
```

Formula:

```
water[i] = min(leftMax[i], rightMax[i]) - height[i]
```



Why?

Water tabhi rukega jab dono sides pe wall ho.



1

Brute Force

Har index pe:

- Left max find karo
- Right max find karo

Time: $O(n^2)$ ❌



2

Prefix + Suffix Method (Allowed ✓)



Idea

Precompute:

```
leftMax[i] → max from 0 to i  
rightMax[i] → max from i to n-1
```

Phir formula apply karo.



Code (Java)

```
class Solution {  
    public int trap(int[] height) {  
        int n = height.length;  
        if (n == 0) return 0;  
  
        int[] leftMax = new int[n];  
        int[] rightMax = new int[n];  
  
        // Build leftMax  
        leftMax[0] = height[0];  
        for (int i = 1; i < n; i++) {  
            leftMax[i] = Math.max(leftMax[i-1], height[i]);  
        }  
  
        // Build rightMax  
        rightMax[n-1] = height[n-1];  
        for (int i = n-2; i >= 0; i--) {  
            rightMax[i] = Math.max(rightMax[i+1], height  
[i]);  
        }  
  
        // Calculate water  
        int water = 0;  
        for (int i = 0; i < n; i++) {  
            water += Math.min(leftMax[i], rightMax[i]) - he  
ight[i];  
        }  
  
        return water;  
    }  
}
```

```
    }  
}
```



Complexity

Time: $O(n)$

Space: $O(n)$



Dry Run Example

```
height = [4,2,0,3,2,5]
```

leftMax:

```
[4,4,4,4,4,5]
```

rightMax:

```
[5,5,5,5,5,5]
```

Water:

```
min(4,5)-4 = 0  
min(4,5)-2 = 2  
min(4,5)-0 = 4  
min(4,5)-3 = 1  
min(4,5)-2 = 2  
min(5,5)-5 = 0
```

Total = 9

3 Optimal (Two Pointer – No Extra Space 🔥🔥🔥)

Same logic but without arrays.

Core Observation

Water depends on smaller side.

If:

```
leftMax < rightMax
```

Then water at left determined by leftMax.

So move left pointer.



Code

```
class Solution {  
    public int trap(int[] height) {  
  
        int left = 0;  
        int right = height.length - 1;  
  
        int leftMax = 0;  
        int rightMax = 0;  
  
        int water = 0;  
  
        while (left < right) {  
  
            if (height[left] < height[right]) {  
  
                if (height[left] >= leftMax)  
                    leftMax = height[left];  
                else  
                    water += leftMax - height[left];  
  
                left++;  
            } else {  
  
                if (height[right] >= rightMax)  
                    rightMax = height[right];  
            }  
        }  
    }  
}
```

```

        else
            water += rightMax - height[right];

        right--;
    }
}

return water;
}
}

```

Why This Works?

Because:

Water height limited by smaller boundary.

So smaller side decide karega.



Common Mistakes

1. Thinking it's sliding window

Nahi hai.

No fixed window size.

2. Forgetting boundary condition

First & last index pe water 0 hota hai.

3. Negative water add kar dena

Formula wrong lagaya toh negative aa sakta hai.

4. Two pointer me wrong comparison

Correct logic:

```
height[left] < height[right]
```

Edge Cases

```
[ ]
```

```
→ θ
```

```
[1]
```

```
→ θ
```

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

```
→ θ
```

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

```
→ θ
```

```
[2,θ,2]
```

```
→ 2
```

Interview Insight

Interviewer test karta hai:

- Prefix thinking
 - Boundary concept
 - Two pointer logic
 - Can you optimize space?
-

Pattern Family

- Container With Most Water
 - Max Area
 - Prefix/Suffix arrays
 - Histogram problems
-

Recognition Trick

Jab question bole:

- Water
- Height bars
- Trapped

👉 Immediately think:

```
min(leftMax, rightMax) - height
```

Agar chahe to mai iska stack based solution bhi bata du (thoda advanced hota hai).

Next kaunsa FAANG breaker kare? 😎🔥