Binary Search

Bhai 🔥

Content ekdum same rakha hai — ek shabd bhi change nahi kiya.

Bas **clean, readable, interview-ready format** me laga diya hai 👇

🚀 PHASE 1: Binary Search Basics

(Foundation – Java Version)

🧠 1️⃣ What is Binary Search

🔷 Definition

Binary Search → Efficient search in **sorted array**

Idea: Divide & Conquer

Har step me search space **half** kar dete hain

🔷 Intuition (Deep Understanding)

Socho tumhare paas sorted array hai:

```
[1, 3, 5, 7, 9, 11, 15]
```

Tumhe **9** find karna hai.

Linear search karein?

→ 1 by 1 check → **O(n)**

Binary search kya karega?

→ Middle check karega

→ Decide karega **left jaana hai ya right**

Har step me **half elements eliminate**

Isliye **time = O(log n)**

🔷 Why Sorted Array Required?

Kyuki binary search decision leta hai:

```
if arr[mid] < target → right jao
else → left jao
```

Ye decision tabhi valid hai jab array **sorted** ho.

Unsorted me ye logic toot jaata hai ❌

🔷 Variants

1️⃣ **Iterative** → Space **O(1)**

2️⃣ **Recursive** → Space **O(log n)**

🔷 Mid Calculation (VERY IMPORTANT)

❌ **Wrong**

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

Problem:

Agar

`low = 2_000_000_000`

`high = 2_000_000_000`

To sum **overflow** karega ❌

✅ **Correct**

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

Reason:

`(high - low)` kabhi overflow nahi karega

🔷 Time & Space Complexity

Variant	Time	Space
Iterative	O(log n)	O(1)

Variant	Time	Space
Recursive	O(log n)	O(log n)

⚠️ Common Bugs

1️⃣ Infinite loop

2️⃣ mid overflow

3️⃣ `low < high` vs `low <= high` confusion

4️⃣ wrong update (`low = mid` instead of `mid + 1`)

✅ Iterative Binary Search

(JAVA – Fully Commented)

```
public class BinarySearchBasic {

    public static int binarySearch(int[] arr, int target) {

        int low = 0;                    // starting index
        int high = arr.length - 1;      // ending index

        // loop tab tak chalega jab tak valid search space hai
        while (low <= high) {

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

            // agar mid element target ke equal hai → mil gaya
            if (arr[mid] == target) {
                return mid;             // index return karo
            }

            // agar mid element target se chhota hai
            else if (arr[mid] < target) {
                low = mid + 1;          // right half me jao
            }
```

```
            // agar mid element target se bada hai
            else {
                high = mid - 1;        // left half me jao
            }
        }

        // agar yaha tak aaye → element nahi mila
        return -1;
    }
}
```

🔍 Dry Run (Step-by-Step)

Array

```
[1, 3, 5, 7, 9, 11, 15]
```

Target = 9

Step 1

```
low = 0, high = 6
mid = 3 → arr[3] = 7
```

7 < 9 → right jao

`low = 4`

Step 2

```
low = 4, high = 6
mid = 5 → arr[5] = 11
```

11 > 9 → left jao

`high = 4`

Step 3

```
low = 4, high = 4
```

```
mid = 4 → arr[4] = 9
```

Match ✅

Return **4**

✅ Recursive Binary Search

(JAVA – Fully Commented)

```java
public class BinarySearchRecursive {

    public static int binarySearchRec(int[] arr, int low, int high, int target) {

        // base case: search space khatam
        if (low > high) {
            return -1;     // element nahi mila
        }

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

        // agar match mil gaya
        if (arr[mid] == target) {
            return mid;
        }

        // agar target bada hai mid se
        else if (arr[mid] < target) {
            return binarySearchRec(arr, mid + 1, high, target);
        }

        // agar target chhota hai mid se
        else {
            return binarySearchRec(arr, low, mid - 1, target);
        }
    }
}
```

🔥 IMPORTANT: `low <= high` vs `low < high`

Standard search me use:

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

Kyuki:

Agar `low == high`

To **ek element baaki hai**

Usko check karna **zaroori** hai

🏆 Interview Gold Line

> "Binary search eliminates half of the search space every iteration. Careful handling of mid and boundaries prevents infinite loops and overflow."

📚 Practice Problems

🟢 Easy

1️⃣ LeetCode 704 – Binary Search

2️⃣ LeetCode 374 – Guess Number Higher or Lower

🟡 Medium

3️⃣ LeetCode 35 – Search Insert Position

4️⃣ LeetCode 33 – Search in Rotated Sorted Array

🔴 Codeforces Practice

1️⃣ CF 706B – Interesting Drink

2️⃣ CF 371C – Hamburgers *(Binary Search on Answer intro)*

3️⃣ CF 1191C – Tokitsukaze and Discard Items

🧠 Mini Mental Checklist (Interview Me Bolna)

- Array sorted hai?

- mid safe calculate kiya?
- loop condition correct?
- low/high correctly update?
- infinite loop possible?

Bhai 🔥

Bilkul same content hai — ek word bhi change nahi kiya.

Bas **clean, structured, interview-ready format** me likh diya hai 👇

🚀 First & Last Occurrence, Upper / Lower Bound

🔷 Problem 1: First & Last Occurrence

Array sorted

Find **first** and **last index** of a target element

🔑 Idea

Use **binary search**

- Jab target mile → **answer store karo**
- **First** ke liye → **left search continue**
- **Last** ke liye → **right search continue**

✅ First Occurrence

(JAVA – Fully Commented)

```
public class FirstLastOccurrence {

    public static int firstOccurrence(int[] arr, int target) {

        int low = 0;                    // search space ka start
        int high = arr.length - 1;      // search space ka end
```

```
        int res = -1;                    // agar element na mile to -1 return karenge

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

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

            // agar mid pe target mil gaya
            if (arr[mid] == target) {

                res = mid;              // answer store karo
                high = mid - 1;         // LEFT side search karo (first occurrence ch
ahiye)
            }

            // agar mid value target se chhoti hai
            else if (arr[mid] < target) {

                low = mid + 1;          // right side search karo
            }

            // agar mid value target se badi hai
            else {

                high = mid - 1;         // left side search karo
            }
        }

        return res;                     // final first occurrence
    }
}
```

✅ Last Occurrence

(JAVA – Fully Commented)

```
public class FirstLastOccurrence {

    public static int lastOccurrence(int[] arr, int target) {

        int low = 0;                     // search space start
        int high = arr.length - 1;       // search space end
        int res = -1;                    // default -1 (not found)

        while (low <= high) {

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

            if (arr[mid] == target) {

                res = mid;               // answer store karo
                low = mid + 1;           // RIGHT side search karo (last occurrence c
hahiye)
            }

            else if (arr[mid] < target) {

                low = mid + 1;           // right jao
            }

            else {

                high = mid - 1;          // left jao
            }
        }

        return res;                      // final last occurrence
    }
}
```

🧪 Dry Run

```
arr = [1,2,2,2,3,4]
target = 2
```

First Occurrence

```
low=0, high=5
mid=2 → arr[2]=2 → res=2 → high=1

low=0, high=1
mid=0 → arr[0]=1 → low=1

low=1, high=1
mid=1 → arr[1]=2 → res=1 → high=0

Stop.
```

First = 1

Last Occurrence

```
low=0, high=5
mid=2 → res=2 → low=3

low=3, high=5
mid=4 → arr[4]=3 → high=3

low=3, high=3
mid=3 → arr[3]=2 → res=3 → low=4

Stop.
```

Last = 3

🔷 Problem 2: Lower Bound & Upper Bound

🔑 Definitions

- **Lower Bound** → first index where element **≥ target**
- **Upper Bound** → first index where element **> target**

✅ Lower Bound

(JAVA – Fully Commented)

```
public class Bounds {

    public static int lowerBound(int[] arr, int target) {

        int low = 0;                     // start index
        int high = arr.length;           // IMPORTANT: high = n (not n-1)
                                         // kyuki lower bound n bhi ho sakta hai

        while (low < high) {             // note: yaha <= nahi, sirf <

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

            // agar mid value >= target hai
            if (arr[mid] >= target) {

                high = mid;              // left part me search karo
            }

            else {

                low = mid + 1;           // right part me jao
            }
        }

        return low;                      // yahi first index hai jahan arr[i] >= target
    }
}
```

✅ Upper Bound

(JAVA – Fully Commented)

```
public class Bounds {

    public static int upperBound(int[] arr, int target) {

        int low = 0;                    // start
        int high = arr.length;          // end = n

        while (low < high) {

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

            // agar mid value target se badi hai
            if (arr[mid] > target) {

                high = mid;             // left search
            }

            else {

                low = mid + 1;          // right search
            }
        }

        return low;                     // first index jahan arr[i] > target
    }
}
```

🧪 Dry Run

```
arr = [1,2,2,2,3,4]
target = 2
```

- **Lower Bound**

  → first index where ≥ 2

→ index = **1**

- **Upper Bound**

  → first index where > 2

  → index = **4**

🔷 Problem 3: Count of Element

Formula 1

```
count = lastOccurrence - firstOccurrence + 1
```

Formula 2 (Better)

```
count = upperBound - lowerBound
```

Example

```
lowerBound = 1
upperBound = 4

count = 4 - 1 = 3 ✅
```

⚠️ Interview Traps

❌ First occurrence ≠ lower bound (conceptually different but often same result)

❌ Upper bound ≠ last occurrence

❌ Using `low <= high` in lowerBound template

❌ Infinite loop due to wrong update

❌ Off-by-one mistakes

🔷 Usage in Interviews

- Frequency problems

- Range queries
- Count elements in sorted array
- Median in two arrays
- Kth smallest problems

🔧 Practice Problems

🟡 LeetCode

- 34 — Find First and Last Position of Element
- 278 — First Bad Version
- 35 — Search Insert Position
- 540 — Single Element in a Sorted Array

🔵 Codeforces

- 706B — Interesting Drink
- 600B — Queries about less or equal elements
- 702C — Cellular Network

🏆 Interview Gold Line

> "Lower and Upper Bound are boundary-finding binary search patterns that allow range and frequency queries in O(log n) without scanning."

Bhai 🔥

Same content — ek word bhi change nahi.

Bas **clean, structured, interview-ready format** me set kar diya hai 👇

🚀 Search Insert Position (LC 35)

🔷 Problem

- Sorted array **arr**

- Target element **target**

Return index where:

1️⃣ **Target exists** → return index

2️⃣ **Target does not exist** → return index where it would be inserted

Examples

```
arr = [1,3,5,6], target = 5 → return 2
arr = [1,3,5,6], target = 2 → return 1
arr = [1,3,5,6], target = 7 → return 4
```

🔷 Key Observation

This is **exactly Lower Bound**:

👉 **First index i where arr[i] >= target**

- If target exists → `arr[i] == target`
- If target not exist → `i = correct insert position`

So:

> LC 35 = Lower Bound template

✅ Binary Search Idea

1️⃣ `low = 0, high = n`

2️⃣ While `low < high`

3️⃣ `mid = low + (high - low)/2`

4️⃣ If `arr[mid] >= target` → `high = mid`

5️⃣ Else → `low = mid + 1`

6️⃣ Return `low`

✅ Java Code

(Fully Commented – Line by Line)

}

```
public class SearchInsertPosition {

    public static int searchInsert(int[] arr, int target) {

        int low = 0;                     // search space ka start
        int high = arr.length;           // IMPORTANT: high = n (not n-1)
                                         // kyuki insert position n bhi ho sakta hai
                                         // example: target = 7 in [1,3,5,6]

        // yaha condition low < high hai (NOT <=)
        // kyuki hum boundary search kar rahe hain (lower bound pattern)
        while (low < high) {

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

            // agar mid value >= target hai
            // iska matlab potential answer left side me bhi ho sakta hai
            if (arr[mid] >= target) {

                high = mid;              // left part me shrink karo
            }

            // agar mid value target se chhoti hai
            // to insert position right side me hi hoga
            else {

                low = mid + 1;           // right half search karo
            }
        }

        // loop khatam hone par low == high
        // ye hi first index hai jahan arr[i] >= target
        return low;
    }
}
```

🧪 Detailed Dry Run

Example 1

```
arr = [1,3,5,6]
target = 5
```

Initial

```
low = 0
high = 4
```

Step 1

```
mid = 2
arr[2] = 5 >= 5
→ high = 2
```

Step 2

```
low = 0
high = 2
mid = 1
arr[1] = 3 < 5
→ low = 2
```

```
low = 2
high = 2
```

Loop ends.

Return 2 ✅

Example 2

```
arr = [1,3,5,6]
target = 2
```

Step 1

```
mid = 2 → arr[2]=5 >=2 → high=2
```

Step 2

```
mid = 1 → arr[1]=3 >=2 → high=1
```

Step 3

```
mid = 0 → arr[0]=1 <2 → low=1
```

Stop.

Return 1 ✅

Example 3

```
arr = [1,3,5,6]
target = 7
```

Every element `< 7`

Eventually:

```
low = 4
high = 4
```

Return 4 ✅ (insert at end)

⚠️ Interview Traps

❌ Using `low <= high` (wrong template for lower bound)

❌ Returning `mid` instead of `low`

❌ Using `high = arr.length - 1` (insert at end case break ho sakta hai)

❌ Forgetting overflow-safe mid

❌ Not understanding that this is **boundary search**

🔷 Pattern Recognition

- **LC 35** = Lower Bound
- **LC 278** = First True in boolean array
- **LC 69** = Sqrt(x) (boundary search)
- **LC 875** = Koko Eating Bananas (Binary Search on Answer)

🔧 Practice Problems

🟡 LeetCode

- 35 — Search Insert Position
- 278 — First Bad Version
- 69 — Sqrt(x)
- 875 — Koko Eating Bananas
- 1011 — Capacity to Ship Packages
- 410 — Split Array Largest Sum

🔵 Codeforces

- 706B — Interesting Drink
- 600B — Queries about less or equal elements
- 371C — Hamburgers (Binary Search on Answer)

🏆 Interview Gold Line

> "Search Insert Position is simply the lower bound pattern — find the first index where element ≥ target."

Bhai 🔥

Same content — ek word bhi change nahi kiya.

Bas **clean, structured, interview-ready format** me properly set kar diya hai 👇

🚀 Search in Rotated Sorted Array — COMPLETE

(JAVA VERSION)

🔷 Problem

Array **sorted** then **rotated** at unknown pivot

Find **target element index**

Examples

```
arr = [4,5,6,7,0,1,2], target = 0 → return 4
arr = [4,5,6,7,0,1,2], target = 3 → return -1
```

🔥 INTUITION (Very Important)

Normal Binary Search tab kaam karta hai jab array **fully sorted** ho.

Yaha array **rotated** hai — but important observation:

👉 **Har step pe at least ek half sorted hota hi hai**

Example:

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

Left part sorted **OR** right part sorted.

Strategy

1️⃣ Identify sorted half

2️⃣ Check target us half me hai ya nahi

3️⃣ Agar hai → wahi search karo

4️⃣ Nahi → doosra half

Binary Search ka logic still valid 🔥

Time Complexity: `O(log n)`

🔷 Step 1: Identify Sorted Half

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

If:

```
arr[low] <= arr[mid]
```

→ **Left half sorted**

Else:

→ **Right half sorted**

✅ Java Code

(LC 33 — No Duplicates)

```
class Solution {

    public int search(int[] arr, int target) {

        int low = 0;                  // starting index
        int high = arr.length - 1;   // ending index

        // standard binary search loop
        while (low <= high) {        // loop until search space valid

            // safe mid calculation (avoid overflow)
            int mid = low + (high - low) / 2;

            // if target found directly
            if (arr[mid] == target)
                return mid;          // return index immediately

            // CHECK WHICH HALF IS SORTED

            if (arr[low] <= arr[mid]) {
```

```
            // LEFT HALF IS SORTED

            // Check if target lies inside left sorted half
            if (arr[low] <= target && target < arr[mid]) {
                high = mid - 1;   // shrink to left half
            } else {
                low = mid + 1;    // search right half
            }

        } else {
            // RIGHT HALF IS SORTED

            // Check if target lies inside right sorted half
            if (arr[mid] < target && target <= arr[high]) {
                low = mid + 1;    // search right half
            } else {
                high = mid - 1;   // search left half
            }
        }
    }

    return -1;   // target not found
  }
}
```

🧪 Dry Run (Step-by-Step Clearly)

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

Iteration 1

```
low = 0
high = 6
mid = 3
```

```
arr[mid] = 7
```

Check sorted half:

```
arr[low] <= arr[mid]
4 <= 7 → YES → left sorted
```

Check target in left:

```
4 <= 0 < 7 → NO
```

So:

```
low = mid + 1 = 4
```

Iteration 2

```
low = 4
high = 6
mid = 5

arr[mid] = 1
```

Check sorted half:

```
arr[low] <= arr[mid]
0 <= 1 → YES → left sorted
```

Check target:

```
0 <= 0 < 1 → YES
```

So:

```
high = mid - 1 = 4
```

Iteration 3

```
low = 4
high = 4
mid = 4

arr[mid] = 0 → FOUND ✅
```

Return 4

🔷 Step 3: Rotated Array with Duplicates (LC 81)

Problem

If:

```
arr[low] == arr[mid] == arr[high]
```

Then we **cannot decide** which half sorted.

Solution

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

Search space shrink kar do.

⚠️ **Worst case becomes O(n)**

(because duplicates destroy strict ordering)

✅ Java Code

(Duplicates Allowed – LC 81)

```
class Solution {

    public boolean search(int[] arr, int target) {
```

```
        int low = 0;
        int high = arr.length - 1;

        while (low <= high) {

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

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

            // Duplicates case: cannot decide sorted half
            if (arr[low] == arr[mid] && arr[mid] == arr[high]) {
                low++;      // shrink from left
                high--;     // shrink from right
            }

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

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

            // Right half sorted
            else {

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

        return false;   // not found
    }
}
```

⚠️ Common Interview Traps

❌ Forgetting which half sorted

❌ Using `<` instead of `<=`

❌ Not handling duplicates

❌ Infinite loop due to wrong boundary updates

❌ Confusing with normal binary search

🔧 Practice Problems

(ONLY This Topic + Previous Topics)

🔥 Rotated Array

- LC 33 – Search in Rotated Sorted Array
- LC 81 – Search in Rotated Sorted Array II
- LC 153 – Find Minimum in Rotated Sorted Array
- LC 154 – Find Minimum in Rotated Sorted Array II

🔥 From Previous Binary Search Topics

- LC 35 – Search Insert Position
- LC 34 – First and Last Position
- LC 704 – Binary Search
- CF 706B – Interesting Drink (Lower Bound usage)

🏆 Interview Gold Line

> "At every step in a rotated sorted array, one half is always sorted. We identify the sorted half and check whether the target lies inside it — duplicates require shrinking the search space carefully."

Bhai 🔥

Same content — ek bhi word change nahi kiya.

Bas **clean, structured, interview-ready format** me properly set kar diya hai 👇

🚀 Find Minimum in Rotated Sorted Array (Pivot Finding) — JAVA

🔷 Problem

Array **sorted** then **rotated**

Find **minimum element**

Examples

```
arr = [4,5,6,7,0,1,2] → min = 0
arr = [3,4,5,1,2] → min = 1
```

With duplicates

```
arr = [2,2,2,0,1,2] → min = 0
```

🔥 INTUITION (Very Important)

Normal sorted array:

```
[1,2,3,4,5]
```

Rotated:

```
[4,5,1,2,3]
```

Minimum element hi pivot hai.

Key idea

👉 Minimum element always lies in the **unsorted part**

👉 **Compare mid with high**

👉 Decide which side pivot is in

Why compare with `high`?

Because **rightmost element** helps us detect whether `mid` lies in:

- left sorted part
- right sorted part (**which contains minimum**)

Time Complexity

- **No duplicates** → `O(log n)`
- **With duplicates** → Worst case `O(n)`

🔷 Step 1: No Duplicates (LC 153)

🔥 Logic Recap

While `low < high`:

1️⃣ Find `mid`

2️⃣ If `nums[mid] > nums[high]`

→ minimum lies in **right half**

→ `low = mid + 1`

3️⃣ Else

→ minimum lies in **left half including mid**

→ `high = mid`

❓ Why `high = mid` and NOT `mid - 1`?

Because **mid itself might be minimum**.

✅ Java Code (No Duplicates)

```
class Solution {

    public int findMin(int[] nums) {

        int low = 0;                // start of search space
```

```
        int high = nums.length - 1;       // end of search space

        // we use low < high (NOT <=)
        // because we are shrinking search space to single element
        while (low < high) {

            // safe mid calculation to prevent overflow
            int mid = low + (high - low) / 2;

            // CASE 1:
            // If mid element greater than high element
            // → pivot lies in right half
            if (nums[mid] > nums[high]) {
                low = mid + 1;  // eliminate left half including mid
            }

            // CASE 2:
            // nums[mid] <= nums[high]
            // → minimum lies in left half including mid
            else {
                high = mid;     // keep mid because it might be minimum
            }
        }

        // When loop ends, low == high
        // That index holds minimum element
        return nums[low];
    }
}
```

🧪 Dry Run (Step-by-Step)

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

Iteration 1

```
low = 0
high = 6
mid = 3

nums[mid] = 7
nums[high] = 2
```

```
7 > 2 → minimum in right half
low = mid + 1 = 4
```

Iteration 2

```
low = 4
high = 6
mid = 5

nums[mid] = 1
nums[high] = 2
```

```
1 <= 2 → minimum in left half
high = mid = 5
```

Iteration 3

```
low = 4
high = 5
mid = 4

nums[mid] = 0
nums[high] = 1
```

```
0 <= 1 → minimum left
high = 4
```

Now:

```
low = 4
high = 4
```

Loop stops.

Return `nums[4] = 0` ✅

🔷 Step 2: With Duplicates (LC 154)

Problem

If:

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

We **cannot decide** which side pivot lies.

Example:

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

Solution

👉 **Shrink search space safely**

```
high--
```

❓ Why safe?

Because if `nums[mid] == nums[high]`, removing `high` **does not remove unique minimum guarantee**.

Worst case becomes `O(n)` if all elements same.

✅ Java Code (Duplicates Allowed)

```
class Solution {

    public int findMin(int[] nums) {

        int low = 0;
        int high = nums.length - 1;

        while (low < high) {

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

            if (nums[mid] > nums[high]) {
                // pivot definitely in right half
                low = mid + 1;
            }

            else if (nums[mid] < nums[high]) {
                // pivot in left half including mid
                high = mid;
            }

            else {
                // nums[mid] == nums[high]
                // cannot decide → shrink safely
                high--;
            }
        }

        return nums[low];
    }
}
```

🧪 Dry Run (With Duplicates)

```
arr = [2,2,2,0,1,2]
```

Iteration 1

```
low=0
high=5
mid=2

nums[2]=2
nums[5]=2
```

```
Equal → high-- → high=4
```

Iteration 2

```
low=0
high=4
mid=2

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

```
2 > 1 → right half
low=mid+1=3
```

Iteration 3

```
low=3
high=4
mid=3

nums[3]=0
nums[4]=1
```

```
0 < 1 → left half
high=3
```

```
low=3
high=3
```

Return `nums[3]=0` ✅

🔷 When to Use / Interview Thoughts

This is **pivot finding logic**.

Used in:

- LC 153 — Find Minimum
- LC 154 — Find Minimum II
- LC 33 — Search Rotated
- LC 81 — Search Rotated II

Rotation Count

```
rotationCount = index_of_min
```

⚠️ Common Interview Traps

❌ Using `low <= high` instead of `low < high`

❌ Doing `high = mid - 1` (wrong — mid might be min)

❌ Not handling duplicates → infinite loop

❌ Comparing with `nums[low]` instead of `nums[high]`

❌ Overflow in mid calculation

🔧 Practice Problems

(ONLY This Topic + Previous Rotated)

🔥 Rotated Array Core

- LC 153 — Find Minimum in Rotated Sorted Array
- LC 154 — Find Minimum in Rotated Sorted Array II

- LC 33 — Search in Rotated Sorted Array
- LC 81 — Search in Rotated Sorted Array II

🔥 Binary Search Foundation

- LC 35 — Search Insert Position
- LC 34 — First and Last Position
- LC 704 — Binary Search
- CF 706B — Interesting Drink (Lower Bound logic)

🏆 Interview Gold Line

> "Minimum in a rotated sorted array is the pivot; comparing mid with high lets us discard half of the array safely — duplicates require shrinking the boundary carefully."

🚀 Binary Search on Answer (BSA) — COMPLETE (JAVA)

🔥 What is Binary Search on Answer?

✅ Normal Binary Search

Search **index**

✅ Binary Search on Answer

Search **optimal value**

You are **NOT searching array**.

You are searching the **minimum / maximum possible valid answer**.

🔷 When To Identify BSA?

If question says:

- Minimize maximum

- Maximize minimum
- Smallest capacity
- Minimum speed
- Maximum distance
- Allocate / distribute

⚡ **That is 90% BSA.**

🧠 Core Idea

1️⃣ Define answer search space

2️⃣ Create `isPossible(mid)`

3️⃣ Binary search on answer range

⏱️ Time Complexity

```
O(log(answer_range) * check_time)
```

🔷 Step 1: Define Search Space

Example (Allocate Pages)

```
arr = [10,20,30,40]
k = 2 students
```

Minimum possible answer?

At least largest book

```
low = max(arr)
```

Maximum possible answer?

One student reads everything

```
high = sum(arr)
```

🔷 Step 2: Design isPossible()

Assume answer = `mid`

Check:

Can we allocate books such that

no student gets more than `mid` pages?

- If yes → `mid` is valid
- If no → `mid` too small

🔷 Binary Search Template (MINIMIZE MAXIMUM)

If we want **smallest valid answer**:

```
if(isPossible(mid))
    high = mid - 1;
else
    low = mid + 1;
```

✅ Java Implementation — Allocate Pages

🔶 Step 1: isPossible()

```
public boolean isPossible(int[] arr, int k, int mid) {

    int students = 1;      // start with first student
    int currSum = 0;       // pages assigned to current student

    for (int i = 0; i < arr.length; i++) {
```

```
        // If adding this book exceeds limit
        if (currSum + arr[i] > mid) {
            students++;           // allocate to next student
            currSum = arr[i];     // start new allocation

            // If students exceed k → not possible
            if (students > k)
                return false;
        }
        else {
            currSum += arr[i];    // continue assigning
        }
    }

    return true;   // allocation successful
}
```

🔶 Step 2: Binary Search on Answer

```
import java.util.*;

class Solution {

    public int allocatePages(int[] arr, int k) {

        int low = 0;
        int high = 0;

        // Define search space
        for (int num : arr) {
            low = Math.max(low, num);   // minimum possible
            high += num;                // maximum possible
        }

        int ans = high;  // store best answer
```

```
        while (low <= high) {

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

            if (isPossible(arr, k, mid)) {
                ans = mid;        // mid works
                high = mid - 1;   // try smaller answer
            }
            else {
                low = mid + 1;    // increase capacity
            }
        }

        return ans;
    }

    public boolean isPossible(int[] arr, int k, int mid) {

        int students = 1;
        int currSum = 0;

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

            if (currSum + arr[i] > mid) {
                students++;
                currSum = arr[i];

                if (students > k)
                    return false;
            } else {
                currSum += arr[i];
            }
        }

        return true;
    }
}
```

🧪 Dry Run

```
arr = [10,20,30,40]
k = 2

low = 40
high = 100
```

Iteration 1

```
mid = 70
Possible? Yes
ans = 70
high = 69
```

Iteration 2

```
mid = 54
Possible? Yes
ans = 54
high = 53
```

Iteration 3

```
mid = 46
Possible? No
low = 47
```

Iteration 4

```
mid = 50
Possible? Yes
ans = 50
```

✅ Final Answer = 50

🔥 Important Concept

If question says:

🔷 Minimize Maximum

→ `high = mid - 1`

🔷 Maximize Minimum

→ `low = mid + 1`

🔥 General Template (Clean Version)

```
while (low <= high) {

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

    if (isPossible(mid)) {
        ans = mid;

        // For minimize
        high = mid - 1;

        // For maximize
        // low = mid + 1;
    }
    else {
        low = mid + 1;
    }
}
```

🧠 Most Important Triggers

Problem Type	Search Space
Capacity	max(arr) → sum(arr)
Speed	1 → max possible

Problem Type	Search Space
Distance	0 → max gap
Time	1 → max time
Chocolate / Wood cutting	0 → max height

⚠️ Interview Traps

❌ Wrong search space

❌ Not initializing low properly

❌ isPossible logic bug

❌ Confusing minimize vs maximize

❌ Forgetting ans variable

🔥 Must Practice (Only BSA Category)

- LC 410 — Split Array Largest Sum
- LC 875 — Koko Eating Bananas
- LC 1011 — Ship Packages
- GFG — Painters Partition
- Aggressive Cows (Classic)

🏆 Interview Gold Line

> "Binary Search on Answer converts optimization problems into a decision problem — define search space carefully and design a monotonic isPossible function."

🐄 Aggressive Cows — Maximize Minimum Distance (JAVA)

🔷 Problem

Given: **n stalls (positions on a line)**

Place **k cows**

🎯 Goal:

Maximize the **minimum distance** between any two cows

```
stalls = [1,2,4,8,9]
k = 3

Answer = 3
Placement: 1, 4, 8
```

🧠 Why This is Binary Search on Answer?

We are **NOT searching index**.

We are searching:

> Maximum possible minimum distance

That means:

- Try a candidate distance `d`
- Check if placement possible
- If yes → try bigger distance
- If no → try smaller distance

This is **maximize minimum pattern**.

🔷 Step 1: Define Search Space

Minimum possible distance?

```
low = 1
```

Maximum possible distance?

```
high = max(stalls) - min(stalls)
```

Why?

Because worst case we place cows at extreme ends.

🔷 Step 2: Design isPossible(dist)

Greedy logic:

- Always place first cow at first stall
- Place next cow at first stall whose distance ≥ `dist`
- Continue until:
  - All cows placed → return true
  - End reached → return false

✅ Java Implementation — isPossible()

```
public boolean isPossible(int[] stalls, int k, int dist) {

    int count = 1;                 // First cow placed at first stall
    int lastPosition = stalls[0];  // Track last placed cow position

    // Start checking from second stall
    for (int i = 1; i < stalls.length; i++) {

        // If distance from last placed cow >= required distance
        if (stalls[i] - lastPosition >= dist) {

            count++;                    // Place cow
            lastPosition = stalls[i];   // Update last position

            // If we successfully placed all cows
            if (count == k)
                return true;
```

```
        }
    }

    // Not able to place all cows
    return false;
}
```

🔷 Step 3: Binary Search on Answer (Maximize Minimum)

Since we want to **maximize minimum distance**:

If possible → try bigger distance

```
low = mid + 1
```

✅ Full Java Code

```
import java.util.Arrays;

class Solution {

    public int aggressiveCows(int[] stalls, int k) {

        // Step 1: Sort stalls (VERY IMPORTANT)
        Arrays.sort(stalls);

        int low = 1;
        int high = stalls[stalls.length - 1] - stalls[0];
        int ans = -1;

        while (low <= high) {

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

```
            // Check if we can place cows with minimum distance = mid
            if (isPossible(stalls, k, mid)) {

                ans = mid;
                low = mid + 1;  // try to maximize distance
            }
            else {
                high = mid - 1; // reduce distance
            }
        }

        return ans;
    }

    public boolean isPossible(int[] stalls, int k, int dist) {

        int count = 1;
        int lastPosition = stalls[0];

        for (int i = 1; i < stalls.length; i++) {

            if (stalls[i] - lastPosition >= dist) {

                count++;
                lastPosition = stalls[i];

                if (count == k)
                    return true;
            }
        }

        return false;
    }
}
```

🧪 Complete Dry Run

```
stalls = [1,2,4,8,9]
k = 3

Sorted → already sorted
low = 1
high = 9 - 1 = 8
```

🔷 mid = 4

Check distance = 4

Place at 1

Next ≥ 5 → 8

Next ≥ 12 → none

Only 2 cows placed → ❌ Not possible

```
high = 3
```

🔷 mid = 2

Place at 1

Next ≥ 3 → 4

Next ≥ 6 → 8

3 cows placed → ✅ Possible

```
ans = 2
low = 3
```

🔷 mid = 3

Place at 1

Next ≥ 4 → 4

Next ≥ 7 → 8

3 cows placed → ✅ Possible

```
ans = 3
low = 4
```

Now:

```
low = 4
high = 3
```

Stop.

✅ Final Answer = 3

🧠 Why Greedy Works?

Because:

If we want to maximize minimum distance,

placing cows as early as possible

gives more space for future cows.

This makes the check function **monotonic**:

If distance = 3 possible

Then distance = 2 also possible

If distance = 4 not possible

Then distance = 5 also not possible

Monotonic property → Binary Search applicable.

🔥 Pattern Recognition

If question says:

- Maximize minimum
- Largest minimum
- Maximum spacing

- Largest distance

👉 **99% BSA.**

⚠️ Common Interview Mistakes

❌ Not sorting stalls

❌ Using wrong search space

❌ Forgetting maximize logic (`low = mid + 1`)

❌ Wrong greedy placement

❌ Infinite loop (wrong low/high update)

🔧 Must Practice (Same Pattern)

- GFG — Aggressive Cows
- LC 1552 — Magnetic Force Between Two Balls
- LC 410 — Split Array Largest Sum
- LC 875 — Koko Eating Bananas
- LC 1011 — Ship Packages

📚 Book Allocation / Painter's Partition — Minimize Maximum Load (JAVA)

🔷 Problem

Given:

- `n` books with `pages[i]`
- `k` students

🎯 Goal:

Allocate books **in order (contiguous)** such that

maximum pages assigned to any student is minimized

⚠️ Important Constraints

- Each student gets **contiguous books**
- Every book must be assigned

🧪 Example

```
pages = [10,20,30,40]
k = 2
```

Possible allocations:

```
[10,20,30] & [40] → max = 60 ✅
[10,20] & [30,40] → max = 70 ❌
```

✅ Answer = 60

🧠 Why This is Binary Search on Answer?

We are **NOT searching index**.

We are searching:

> Minimum possible value of maximum pages per student.

Monotonic Behaviour:

- If `maxLoad = 70` possible

  → `maxLoad = 80` also possible

- If `maxLoad = 50` not possible

  → `maxLoad = 40` also not possible

Monotonic property → **Binary Search applicable**

🔷 Step 1: Define Search Space

✅ Minimum possible answer

```
low = max(pages)
```

Why?

Because at least one student must read the largest book.

✅ Maximum possible answer

```
high = sum(pages)
```

Why?

Because one student can read all books.

🔷 Step 2: Greedy Check Function

Idea:

- Keep assigning books to current student
- If adding next book exceeds `maxLoad`
  - Assign new student
  - Increase student count
- If students > k → not possible

✅ Java Code — isPossible()

```
public boolean isPossible(int[] pages, int k, int maxLoad) {

    int students = 1;   // Start with first student
    int currentSum = 0; // Pages assigned to current student

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

        // If adding this book exceeds allowed maxLoad
        if (currentSum + pages[i] > maxLoad) {
```

```
            students++;
            currentSum = pages[i];

            if (students > k)
                return false;
        }
        else {
            currentSum += pages[i];
        }
    }

    return true;
}
```

🔷 Step 3: Binary Search on Answer (Minimize Maximum)

Since we want to **minimize maximum load**:

If possible → try smaller value

```
high = mid - 1
```

✅ Full Java Implementation

```
class Solution {

    public int bookAllocation(int[] pages, int k) {

        // Edge case: more students than books
        if (k > pages.length)
            return -1;

        int low = 0;
        int high = 0;
```

```
    // Calculate search space
    for (int page : pages) {
        low = Math.max(low, page);
        high += page;
    }

    int ans = high;

    while (low <= high) {

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

        if (isPossible(pages, k, mid)) {

            ans = mid;
            high = mid - 1;   // minimize
        }
        else {
            low = mid + 1;
        }
    }

    return ans;
}

public boolean isPossible(int[] pages, int k, int maxLoad) {

    int students = 1;
    int currentSum = 0;

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

        if (currentSum + pages[i] > maxLoad) {

            students++;
            currentSum = pages[i];

            if (students > k)
```

```
            return false;
        }
        else {
            currentSum += pages[i];
        }
    }

    return true;
  }
}
```

🧪 Complete Dry Run

```
pages = [10,20,30,40]
k = 2
```

Calculate Search Space

```
low = 40
high = 100
```

🔷 mid = 70

Allocation:

```
10 + 20 + 30 = 60
Add 40 → exceeds 70
→ New student
Total students = 2 → ✅ possible
```

```
ans = 70
high = 69
```

🔷 mid = 54

Allocation:

```
10 + 20 = 30
+30 = 60 → exceeds
→ New student (30)

+40 → exceeds
→ Third student ❌
```

```
Not possible
low = 55
```

Continue...

Eventually answer becomes:

✅ **60**

🧠 Why Greedy Works?

Because:

- We always assign minimum books needed before switching student.
- If smaller `maxLoad` possible, greedy will find it.
- This ensures monotonicity.

🔥 Pattern Recognition

If question says:

- Minimize maximum
- Distribute workload
- Allocate tasks
- Capacity / limit / load
- Split array

👉 Think **Binary Search on Answer**

⚠️ Common Interview Mistakes

❌ Not setting `low = max(pages)`

❌ Using `low = 0` (wrong lower bound)

❌ Forgetting contiguous allocation rule

❌ Confusing minimize vs maximize

❌ Infinite loop due to wrong update

🔧 Must Practice (Same Pattern)

🔥 LC 410 — Split Array Largest Sum

🔥 LC 1011 — Capacity To Ship Packages

🔥 LC 875 — Koko Eating Bananas

🔥 GFG — Painter's Partition

🔥 GFG — Book Allocation

🏆 Interview Gold Line

> "Book Allocation is a classic Minimize Maximum Load problem — define answer range carefully and validate using greedy contiguous allocation inside Binary Search on Answer."

🍌 Koko Eating Bananas — Minimize Eating Speed (JAVA)

🔷 Problem

Given:

- `piles[i]` → bananas in each pile
- `h` → total hours

🎯 Find:

Minimum integer speed `k` such that Koko can finish all bananas within `h` hours.

🧪 Example

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

Try k = 4

Hours needed:

```
3  → 1 hour
6  → 2 hours
7  → 2 hours
11 → 3 hours
```

Total = **8 hours** ✅

✅ Answer = 4

🧠 Why This is Binary Search on Answer?

We are **NOT searching index**.

We are searching:

> Minimum valid eating speed

Monotonic Behaviour:

- If speed = 6 works

  → speed = 7, 8, 9 also works

- If speed = 3 fails

  → speed = 2, 1 also fails

Monotonic property → **Apply Binary Search**

🔷 Step 1: Define Search Space

Minimum possible speed

```
low = 1
```

Maximum possible speed

```
high = max(piles)
```

Because at worst she eats entire largest pile in 1 hour.

🔷 Step 2: Hours Calculation (VERY IMPORTANT)

For each pile:

```
hours = ceil(pile / k)
```

⚠️ Avoid floating point in Java

Use integer ceiling formula:

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

This ensures rounding up.

✅ Java Code — isPossible()

```
public boolean isPossible(int[] piles, int h, int k) {

    long totalHours = 0; // use long to prevent overflow

    for (int pile : piles) {

        // Ceil division without floating point
```

```
            totalHours += (pile + k - 1) / k;

            // Early stop optimization
            if (totalHours > h)
                return false;
        }

        return totalHours <= h;
    }
```

🔷 Step 3: Binary Search on Answer (Minimize Speed)

Since we want to **minimize k**:

If possible → try smaller

```
high = mid - 1
```

✅ Full Java Implementation

```
class Solution {

    public int minEatingSpeed(int[] piles, int h) {

        int low = 1;
        int high = 0;

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

        int ans = high;

        while (low <= high) {
```

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

            if (isPossible(piles, h, mid)) {

                ans = mid;
                high = mid - 1;  // try smaller speed
            }
            else {
                low = mid + 1;   // increase speed
            }
        }

        return ans;
    }

    private boolean isPossible(int[] piles, int h, int k) {

        long totalHours = 0;

        for (int pile : piles) {

            totalHours += (pile + k - 1) / k;

            if (totalHours > h)
                return false;
        }

        return totalHours <= h;
    }
}
```

🧪 Full Dry Run

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

Find Search Range

```
low = 1
high = 11
```

🔷 mid = 6

Hours:

```
3  → 1
6  → 1
7  → 2
11 → 2
```

Total = 6 ≤ 8 ✅

```
ans = 6
high = 5
```

🔷 mid = 3

Hours:

```
3  → 1
6  → 2
7  → 3
11 → 4
```

Total = 10 ❌

```
low = 4
```

🔷 mid = 4

Hours:

```
3  → 1
6  → 2
```

```
7  → 2
11 → 3
```

Total = 8 ✅

```
ans = 4
high = 3
```

Loop ends.

✅ Final Answer = 4

🧠 Why Ceil Division Matters

If you do:

```
pile / k
```

That's floor division.

Example:

```
7 / 4 = 1 ❌
Actual hours = 2
```

So always use:

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

⚠️ This is a VERY common interview trap.

🔥 Pattern Recognition

If question says:

- Minimum rate
- Minimum speed

- Minimum divisor
- Within H hours
- Within threshold
- Capacity limit

👉 Think **Binary Search on Answer**

⚠️ Common Interview Mistakes

❌ Forgetting ceiling division

❌ Using double / floating point

❌ Wrong search range

❌ Not minimizing correctly

❌ Overflow in hour sum

🔧 Must Practice (Same Pattern)

🔥 LC 875 — Koko Eating Bananas

🔥 LC 1283 — Smallest Divisor Given Threshold

🔥 LC 1011 — Ship Packages Within D Days

🔥 LC 410 — Split Array Largest Sum

🔥 LC 1552 — Magnetic Force Between Two Balls

🏆 Interview Gold Line

> “Koko Eating Bananas is a Binary Search on Answer problem where we minimize eating speed by validating a candidate speed using ceiling-based hour calculation.”

🚢 Capacity to Ship Packages Within D Days (LC 1011) — JAVA

🔷 Problem

Given:

- `weights[i]` → weight of each package
- `D` → number of days

📌 Rules:

- Ship packages **in order**
- Cannot split a package
- Each day total load ≤ ship capacity `C`

🎯 Goal:

Find **minimum ship capacity C** to ship all packages within `D` days.

🧪 Example

```
weights = [1,2,3,4,5,6,7,8,9,10]
D = 5
```

Try capacity = 15

```
Day 1 → 1+2+3+4+5 = 15
Day 2 → 6+7 = 13
Day 3 → 8
Day 4 → 9
Day 5 → 10
```

Within 5 days ✅

✅ Answer = 15

🧠 Why This is Binary Search on Answer?

We are **NOT searching index**.

We are searching:

> Minimum valid ship capacity

Monotonic Behaviour:

- If capacity = 20 works

  → capacity = 25, 30 also works

- If capacity = 14 fails

  → capacity = 13, 12 also fails

Monotonic property → **Binary Search applicable**

🔷 Step 1: Define Search Space

✅ Minimum possible capacity

```
low = max(weights)
```

Ship must at least carry the heaviest package.

✅ Maximum possible capacity

```
high = sum(weights)
```

Ship could carry everything in one day.

🔷 Step 2: Greedy Check Function

Simulate shipping:

- Start `days = 1`
- Add packages until capacity exceeded
- If exceeded → new day
- If `days > D` → fail

✅ Java Code — isPossible()

```
private boolean isPossible(int[] weights, int D, int capacity) {

    int days = 1;
    int currentLoad = 0;

    for (int w : weights) {

        if (currentLoad + w > capacity) {

            days++;
            currentLoad = w;

            if (days > D)
                return false;
        }
        else {
            currentLoad += w;
        }
    }

    return true;
}
```

🔷 Step 3: Binary Search on Answer (Minimize Capacity)

Since we want to **minimize capacity**:

If possible → try smaller

```
high = mid - 1
```

✅ Full Java Implementation

```
class Solution {

    public int shipWithinDays(int[] weights, int D) {

        int low = 0;
        int high = 0;

        // Determine search space
        for (int w : weights) {
            low = Math.max(low, w);
            high += w;
        }

        int ans = high;

        while (low <= high) {

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

            if (isPossible(weights, D, mid)) {

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

        return ans;
    }

    private boolean isPossible(int[] weights, int D, int capacity) {

        int days = 1;
        int currentLoad = 0;

        for (int w : weights) {
```

```
            if (currentLoad + w > capacity) {
                days++;
                currentLoad = w;

                if (days > D)
                    return false;
            }
            else {
                currentLoad += w;
            }
        }

        return true;
    }
}
```

🧪 Full Dry Run

```
weights = [1,2,3,4,5,6,7,8,9,10]
D = 5
```

Search Space:

```
low = 10
high = 55
```

🔷 mid = 32

Possible? ✅

```
ans = 32
high = 31
```

🔷 mid = 20

Possible? ✅

```
ans = 20
high = 19
```

🔷 mid = 14

Possible? ❌

```
low = 15
```

🔷 mid = 17

Possible? ✅

```
ans = 17
high = 16
```

🔷 mid = 16

Possible? ✅

```
ans = 16
high = 15
```

🔷 mid = 15

Possible? ✅

```
ans = 15
high = 14
```

Loop ends.

✅ Final Answer = 15

🧠 Pattern Recognition

If question says:

- Minimum capacity
- Minimum load
- Allocate tasks
- Within D days
- Split array
- Partition problem

👉 Think **Binary Search on Answer**

🔥 Compare With Similar Problems

Problem	What We Minimize
Book Allocation	Max pages
Painter's Partition	Max board length
Ship Packages	Ship capacity
Split Array Largest Sum	Largest subarray sum
Koko	Eating speed

Same pattern. Only `isPossible` logic changes.

⚠️ Common Interview Mistakes

❌ Forgetting `low = max(weights)`

❌ Starting `low = 1` (wrong)

❌ Miscounting days

❌ Not resetting `currentLoad` properly

❌ Infinite loop due to wrong bounds

🏆 Interview Gold Line

> “Capacity to Ship Packages is a classic minimize-capacity Binary Search on Answer problem where we validate candidate capacities using greedy daily allocation.”

Ab sahi, clean aur interview-ready format me likh raha hoon 👇

🧠 2D Matrix Search — Complete Pattern Guide (LC 74 & LC 240)

✅ TYPE 1: Matrix Treated as Sorted 1D Array

(LeetCode 74 — Search a 2D Matrix)

🔷 Matrix Properties

- Har row sorted hai
- Har next row ka first element > previous row ka last element

Example:

```
[
 [1, 3, 5, 7],
 [10,11,16,20],
 [23,30,34,60]
]
```

Ye internally behave karta hai:

```
[1,3,5,7,10,11,16,20,23,30,34,60]
```

👉 Matlab **poora matrix globally sorted hai**

🔥 Core Idea

2D matrix ko 1D sorted array treat karo.

🔁 Index Mapping Formula

Agar matrix size = `m x n`

```
row = mid / n
col = mid % n
```

✅ Java Code

```
class Solution {

    public boolean searchMatrix(int[][] matrix, int target) {

        int m = matrix.length;
        int n = matrix[0].length;

        int low = 0;
        int high = m * n - 1;

        while (low <= high) {

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

            int row = mid / n;
            int col = mid % n;

            int value = matrix[row][col];

            if (value == target)
                return true;
            else if (value < target)
                low = mid + 1;
            else
                high = mid - 1;
        }

        return false;
```

```
    }
}
```

🧪 Dry Run

Target = `16`

```
low=0, high=11
mid=5 → value=11 → low=6

mid=8 → value=23 → high=7

mid=6 → value=16 ✅
```

⏱️ Complexity

- **Time:** O(log(m × n))
- **Space:** O(1)

✅ TYPE 2: Row + Column Sorted Matrix

(LeetCode 240 — Search a 2D Matrix II)

🔷 Matrix Properties

- Har row sorted
- Har column sorted
- Lekin globally sorted nahi

Example:

```
[
 [1, 4, 7, 11],
 [2, 5, 8, 12],
 [3, 6, 9, 16],
```

```
  [10,13,14,17]
]
```

Flatten karoge to sorted nahi milega ❌

🔥 Best Trick: Start from Top-Right

Start at `(0, n-1)`

Kyun?

- Left → chhota
- Down → bada

Har step me ek row ya column eliminate hota hai.

✅ Java Code

```
class Solution {

    public boolean searchMatrix(int[][] matrix, int target) {

        int m = matrix.length;
        int n = matrix[0].length;

        int row = 0;
        int col = n - 1;

        while (row < m && col >= 0) {

            int value = matrix[row][col];

            if (value == target)
                return true;
            else if (value > target)
                col--;      // move left
            else
                row++;      // move down
        }
```

```
        return false;
    }
}
```

🧪 Dry Run

Target = `5`

```
Start at 11
11 > 5 → left
7 > 5 → left
4 < 5 → down
5 == 5 ✅
```

⏱️ Complexity

- **Time:** O(m + n)
- **Space:** O(1)

🔥 Important Comparison

Type	Matrix Property	Approach	Time
LC 74	Fully sorted	Treat as 1D Binary Search	O(log(m×n))
LC 240	Row + Column sorted	Top-right elimination	O(m+n)

❗ Interview Trap

Har 2D matrix me binary search nahi lagta.

👉 Pehle matrix ki property check karo:

- Globally sorted? → **1D Binary Search**
- Only row + column sorted? → **Top-right Staircase Search**

🏆 Interview Gold Line

> "If matrix is globally sorted, apply binary search on virtual 1D index. If only row and column sorted, use top-right elimination technique."

Agar chaho to main iska **pattern recognition flowchart** bhi bana du — exam me 5 sec me decide karne wala 😄

Ab isko bhi clean, structured, interview-ready format me likh dete hain 👇

🧠 2D Matrix Search — Edge Cases & Interview Traps

✅ PART 1: LC 74 — Fully Sorted Matrix

(Matrix globally sorted hota hai → 1D Binary Search apply karte hain)

🔷 1️⃣ Empty Matrix

```
if (matrix == null || matrix.length == 0 || matrix[0].length == 0)
    return false;
```

⚠ Agar ye check nahi lagaya → `NullPointerException` ya `ArrayIndexOutOfBounds`.

🔷 2️⃣ Single Element Matrix

Example:

```
[[5]]
```

- Target = 5 → ✅ true
- Target = 3 → ❌ false

Yaha:

```
low = 0
high = 0
```

Binary search perfectly work karega —

bas loop condition `low <= high` honi chahiye.

🔷 3️⃣ Target Outside Global Range

Agar:

```
target < matrix[0][0]
target > matrix[m-1][n-1]
```

Toh direct return false kar sakte ho.

```
if (target < matrix[0][0] || target > matrix[m-1][n-1])
    return false;
```

Ye optional optimization hai.

🔷 4️⃣ Integer Overflow in mid

❌ Wrong:

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

✅ Correct:

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

Large inputs me overflow avoid karta hai.

🔷 5️⃣ Index Mapping Mistake (Very Common Bug)

❌ Galat:

```
row = mid / m;
```

```
col = mid % m;
```

✅ Sahi:

```
row = mid / n;
col = mid % n;
```

👉 Always divide by **number of columns (n)**.

Kyun?

Because 1D flattening row-wise hoti hai.

🧠 PART 2: LC 240 — Row + Column Sorted Matrix

(Yaha globally sorted nahi hota → Staircase approach)

🔷 1️⃣ Wrong Starting Point

❌ Top-left se start karoge → 2 direction possible

Decision ambiguous ho jayega.

✅ Hamesha start karo:

- **Top-right**

  OR

- **Bottom-left**

Yaha har step me ek row ya column eliminate hota hai.

🔷 2️⃣ Single Row Matrix

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

Ye basically 1D array hai.

- Binary search → O(log n)
- Staircase → O(n)

Interview me better bolo:

👉 “Since single row, we can apply binary search.”

🔷 3️⃣ Single Column Matrix

```
[
 [1],
 [3],
 [5]
]
```

Same logic — treat as 1D sorted array.

🔷 4️⃣ Duplicates in Matrix

Example:

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

Algorithm still works ✅

Because decision strictly `<` and `>` comparison pe based hai.

Duplicates logic break nahi karta.

🔷 5️⃣ Target Between Values but Not Present

Example:

```
[
 [1,4,7],
 [8,10,12],
 [13,15,18]
```

```
]
target = 9
```

Staircase safely eliminate karega and exit karega.

No infinite loop.

🔷 6️⃣ Worst Case Time

Worst case me diagonal traverse karna pad sakta hai.

Time = **O(m + n)**

Space = **O(1)**

🔥 Advanced Interview Trap

Kabhi matrix aisa milega:

- Rows sorted hain
- Columns sorted nahi hain

Toh:

❌ Staircase fail karega

❌ Full 1D binary search bhi nahi chalega

👉 Tab approach:

Har row me binary search

Time Complexity:

```
O(m log n)
```

⚡ Fast Identification Table

Matrix Type	Correct Approach	Time
Fully sorted	1D Binary Search	O(log(m×n))
Row + Col sorted	Staircase	O(m+n)
Only Row sorted	Row-wise Binary Search	O(m log n)

Matrix Type	Correct Approach	Time
Random matrix	No binary search	Depends

❗ Most Common Interview Mistakes

❌ Wrong `mid` formula

❌ Wrong row/col mapping

❌ Boundary checks bhool jana

❌ Galat matrix type pe galat approach laga dena

❌ Infinite loop (wrong `low/high` update)

🏆 Interview Gold Line

> "Before applying binary search on a 2D matrix, always identify whether it is globally sorted or only row-column sorted. The matrix property decides the algorithm."

Agar chaho to next step me main iska **pattern recognition decision tree** bana deta hoon — jo interview me 10 second me approach decide kara de 🚀

Chalo isko bhi proper structured + interview-ready format me likhte hain 👇

🧠 Kth Smallest Element in Sorted Matrix (LC 378)

> Matrix row-wise + column-wise sorted hai
>
> Goal: **kth smallest element** find karna

🔷 Matrix Property

Example:

```
1   5   9
10  11  13
```

```
12  13  15
```

- Har row sorted
- Har column sorted
- But globally 1D sorted nahi hai

🔥 Core Idea — Binary Search on Value

Ye index search nahi hai ❌

Ye **value space search** hai ✅

Hum element ka position nahi dhundh rahe.

Hum **answer ki value** dhundh rahe hain.

🔷 Search Space Define Karna

```
low  = matrix[0][0]        → smallest value
high = matrix[n-1][n-1]     → largest value
```

Example:

```
low = 1
high = 15
```

Ab binary search chalega value range pe.

🧠 Key Question During BS

For any `mid`:

👉 Matrix me kitne elements ≤ mid hain?

- Agar `count >= k` → mid valid ho sakta hai
- Agar `count < k` → mid chhota hai

🧪 Dry Run (Step-by-Step)

Matrix:

```
1   5   9
10  11  13
12  13  15
```

k = 8

🔶 Step 1

```
low = 1
high = 15
mid = 8
```

Elements ≤ 8 → {1,5}

Count = 2

2 < 8 ❌

```
low = 9
```

🔶 Step 2

```
low = 9
high = 15
mid = 12
```

Elements ≤ 12 →

{1,5,9,10,11,12}

Count = 6

6 < 8 ❌

```
low = 13
```

🔶 Step 3

```
low = 13
high = 15
mid = 14
```

Elements ≤ 14 →

{1,5,9,10,11,13,12,13}

Count = 8

8 ≥ 8 ✅

Possible answer = 14

Try smaller:

```
high = 13
```

🔶 Step 4

```
low = 13
high = 13
mid = 13
```

Elements ≤ 13 → 8

8 ≥ 8 ✅

Answer = 13

```
high = 12
```

Loop ends.

✅ Final Answer = 13

🚀 Most Important Part: Efficient Counting

Brute force counting = $O(n^2)$ ❌

We need O(n) per check.

🔥 Smart Trick — Start From Bottom-Left

Initialize:

```
i = n-1
j = 0
```

Why bottom-left?

- Agar value ≤ mid

  → upar ke saare elements bhi ≤ mid

  → count += (i + 1)

  → move right (j++)

- Agar value > mid

  → upar move (i--)

Example Visualization

For mid = 13:

We sweep diagonally upward/right and eliminate rows/columns efficiently.

Time per count = O(n)

⏱️ Time Complexity

Binary search runs on value range:

```
O(log(max - min))
```

Each step count = O(n)

Final:

```
O(n log(max-min))
```

Space:

```
O(1)
```

✅ Full Clean Java Code

```
class Solution {

    public int kthSmallest(int[][] matrix, int k) {

        int n = matrix.length;

        int low = matrix[0][0];
        int high = matrix[n - 1][n - 1];

        int ans = low;

        while (low <= high) {

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

            int count = countLessEqual(matrix, mid);

            if (count >= k) {
                ans = mid;
                high = mid - 1;   // try smaller
            } else {
                low = mid + 1;    // increase value
            }
        }

        return ans;
    }

    private int countLessEqual(int[][] matrix, int mid) {

        int n = matrix.length;
        int i = n - 1;
```

```
        int j = 0;
        int count = 0;

        while (i >= 0 && j < n) {

            if (matrix[i][j] <= mid) {
                count += (i + 1);
                j++;
            } else {
                i--;
            }
        }

        return count;
    }
}
```

⚠️ Edge Cases

✔️ Duplicates allowed

✔️ k = 1

✔️ k = n*n

✔️ 1×1 matrix

Sab cases me logic same chalega.

🔥 Compare With Similar Matrix Problems

Problem	Approach
LC 74	1D Binary Search
LC 240	Staircase
LC 378	Binary Search on Value + Counting

🏆 Interview Gold Line

> "Since the matrix is row and column sorted but not globally sorted, we apply Binary Search on value space and count elements ≤ mid in O(n) using bottom-left traversal."

Agar chaho to next level me main iska **Heap based solution (Min Heap approach O(k log n))** bhi explain kar deta hoon — jo alternative interview approach hota hai 🔥

Chalo isko bhi ekdum **clean structured + interview ready format** me likhte hain 👇

🏔️ Peak Element — Find Local Maximum (LC 162)

🔷 Problem

Given array `nums[]`

Find **any peak element index**

📌 Definition

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

Boundary condition:

```
nums[-1] = -∞
nums[n]  = -∞
```

🔷 Examples

```
[1,2,3,1] → peak = 3 (index 2)

[1,2,1,3,5,6,4]
→ peak = 2 (index 1)
→ peak = 6 (index 5)
```

Multiple peaks allowed ✅

🔥 Core Intuition (Most Important)

Array sorted nahi hai ❌

Fir bhi Binary Search lag sakta hai ✅

Trick: Observe the slope

Compare:

```
nums[mid]  vs  nums[mid + 1]
```

Case 1️⃣ Increasing Slope 📈

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

Matlab graph upar ja raha hai.

👉 Peak **right side me hoga**

Kyun?

Agar continuously increase kare → end me last element peak hoga

Ya beech me kahin peak mil jayega

So:

```
low = mid + 1
```

Case 2️⃣ Decreasing Slope 📉

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

Matlab graph neeche ja raha hai.

👉 Peak **left side me hoga (mid included)**

So:

```
high = mid
```

🧠 Why It Always Works?

Because:

- Increasing slope → peak right me guaranteed
- Decreasing slope → peak left me guaranteed

Binary search kisi na kisi local peak par converge karega.

✅ Clean Java Code

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

        int low = 0;
        int high = nums.length - 1;

        // IMPORTANT: low < high (NOT <=)
        while (low < high) {

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

            if (nums[mid] < nums[mid + 1]) {

                // Increasing slope → peak right side
                low = mid + 1;

            } else {

                // Decreasing slope → peak left side (mid included)
                high = mid;
            }
        }
```

```
        return low;  // or high (both same)
    }
}
```

🧪 Dry Run

Example 1

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

Initial:

```
low = 0
high = 3
```

Iteration 1

```
mid = 1
2 < 3 → slope up
low = 2
```

Iteration 2

```
low = 2
high = 3
mid = 2
3 > 1 → slope down
high = 2
```

Now:

```
low = 2
high = 2
```

Loop ends.

Return **2** ✅

🔥 Mountain Array Version (LC 852)

Strictly increasing → then strictly decreasing

Same logic works.

```
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]) {
                low = mid + 1;     // increasing
            } else {
                high = mid;        // decreasing
            }
        }

        return low;
    }
}
```

⏱️ Complexity

```
Time  : O(log n)
Space : O(1)
```

⚠️ Important Interview Traps

❌ Using `low <= high`

- mid+1 out of bounds ho sakta hai
- infinite loop ka risk

Always use:

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

❌ Returning value instead of index

Question usually index maangta hai.

❌ Forgetting edge cases

🔷 Edge Cases

1️⃣ Single element

```
[5] → peak = index 0
```

2️⃣ Strictly increasing

```
[1,2,3,4]
Peak = last element
```

3️⃣ Strictly decreasing

```
[5,4,3,2]
Peak = first element
```

4️⃣ Multiple peaks

Any valid peak allowed.

🧠 Why Binary Search Possible Without Sorting?

Because we don't need full order.

We just need **direction of slope**.

That directional information is enough to eliminate half search space.

🔥 Compare With Other BS Patterns

Problem	Logic
Classic BS	Compare with target
Peak Element	Compare with mid+1
Rotated Array	Compare with sorted half
Mountain Array	Compare slope

🏆 Interview Gold Line

> "By comparing nums[mid] with nums[mid+1], we detect slope direction and eliminate half the search space, guaranteeing convergence to a local peak in O(log n)."

Agar next level jaana hai to bata:

- 🔥 2D Peak Element (LC 1901)
- 🔥 Prove mathematically why peak always exists
- 🔥 Google style tricky peak variations
- 🔥 Strictly local maxima vs global maxima difference

Bol kaunsa deep dive chahiye 🚀

Chalo isko bhi ekdum **interview-ready, structured, crystal-clear version** me finalize karte hain 👇

🔥 Median of Two Sorted Arrays (LC 4) — COMPLETE MASTER GUIDE

🔷 Problem

Given two **sorted arrays** `nums1[]` and `nums2[]`

Find median of combined array in:

```
O(log(min(n, m)))
```

⚠️ You are NOT allowed to merge arrays.

🧠 Core Intuition (Most Important Part)

Agar merge karte:

```
[1,2] + [3,4] → [1,2,3,4]
Median = middle element
```

But merging costs:

```
O(n + m) ❌
```

We need:

```
O(log(min(n, m))) ✅
```

💡 Real Trick: Partition Instead of Merge

We don't merge arrays.

We **cut (partition)** both arrays such that:

```
Left side size  = (n + m + 1) / 2
Right side size = remaining
```

And condition must hold:

```
max(left part) <= min(right part)
```

If this is satisfied → we found correct partition → median ready.

🔥 Visual Understanding

Example:

```
A = [1,3]
B = [2]
```

Partition like this:

```
A: [1 | 3]
B: [2 | ]
```

Left side:

```
[1,2]
```

Right side:

```
[3]
```

Check:

```
max(left) = 2
min(right) = 3
2 <= 3 ✅
```

Median = 2

🧠 Why Binary Search Works?

We binary search on partition index `i` of smaller array.

Let:

```
i = partition in A
j = partition in B
```

Such that:

```
i + j = (n + m + 1) / 2
```

Now check:

```
Aleft  <= Bright
Bleft  <= Aright
```

If true → correct partition.

If not → adjust binary search.

🚀 Always Binary Search on Smaller Array

Very important:

```
if (A.length > B.length)
    swap arrays
```

Why?

To guarantee:

```
O(log(min(n,m)))
```

✅ Clean Java Implementation

```
class Solution {

    public double findMedianSortedArrays(int[] A, int[] B) {

        // Ensure A is smaller array
        if (A.length > B.length) {
```

```
    return findMedianSortedArrays(B, A);
}

int n = A.length;
int m = B.length;

int low = 0;
int high = n;

while (low <= high) {

    int i = low + (high - low) / 2;
    int j = (n + m + 1) / 2 - i;

    int Aleft  = (i == 0) ? Integer.MIN_VALUE : A[i - 1];
    int Aright = (i == n) ? Integer.MAX_VALUE : A[i];

    int Bleft  = (j == 0) ? Integer.MIN_VALUE : B[j - 1];
    int Bright = (j == m) ? Integer.MAX_VALUE : B[j];

    // Correct partition found
    if (Aleft <= Bright && Bleft <= Aright) {

        if ((n + m) % 2 == 0) {

            int leftMax  = Math.max(Aleft, Bleft);
            int rightMin = Math.min(Aright, Bright);

            return (leftMax + rightMin) / 2.0;

        } else {

            return Math.max(Aleft, Bleft);
        }

    }
    // Move left
    else if (Aleft > Bright) {
```

```
            high = i - 1;
        }
        // Move right
        else {
            low = i + 1;
        }
    }

    return 0.0; // never reached logically
  }
}
```

🧪 Step-by-Step Dry Run

Example:

```
nums1 = [1,2]
nums2 = [3,4]
```

Total length = 4

Left side size:

```
(4 + 1) / 2 = 2
```

We search correct partition.

Eventually partition becomes:

```
A: [1,2 | ]
B: [ | 3,4]
```

Left = [1,2]

Right = [3,4]

Median:

```
(2 + 3) / 2 = 2.5
```

🔥 Why Use Integer.MIN_VALUE / MAX_VALUE?

Edge case:

If partition at extreme:

```
i = 0 → no left element in A
i = n → no right element in A
```

Instead of special handling every time:

We simulate:

```
No left → -∞
No right → +∞
```

Very common interview trick.

🎯 Time & Space Complexity

```
Time  : O(log(min(n,m)))
Space : O(1)
```

⚠️ Common Interview Mistakes

❌ Not searching on smaller array

Leads to wrong complexity.

❌ Off-by-one in partition formula

Correct formula:

```
j = (n + m + 1) / 2 - i
```

❌ Forgetting odd vs even case

```
Odd  → max(left)
Even → average(max(left), min(right))
```

❌ Using (a + b) / 2 instead of (a + b) / 2.0

Integer division error.

🧠 Deep Understanding Summary

We are NOT finding element position.

We are finding a **valid partition** such that:

```
All left elements <= all right elements
```

Binary search helps adjust partition direction efficiently.

🔥 Compare With Other BSA Problems

Problem	What we binary search on
Koko	Eating speed
Book Allocation	Max load
Ship Packages	Capacity
Kth Smallest Matrix	Value
Median of Two Arrays	Partition index

This one is special because:

👉 Binary search on partition index

👉 Not directly on values

🏆 Interview Gold Line

> "Instead of merging arrays, we binary search the partition of the smaller array such that left half contains exactly half elements and satisfies max(left)

> ≤ min(right)."

If you want next level:

- 🔥 Why partition proof works mathematically
- 🔥 Edge cases like empty array
- 🔥 Tricky interviewer variations
- 🔥 Convert this into template memory trick

Bol kya next deep dive karein 🚀

Theek hai 😌 ab main isko **clean, interview-ready proper format** me likh raha hoon — headings, structure, clarity sab perfect.

🔥 Square Root / Nth Root — Binary Search on Answer

🧠 Core Concept (Sabse Important)

Ye **Binary Search on Answer** problem hai.

Hum kisi sorted array pe search nahi kar rahe ❌

Hum **numeric range pe search kar rahe hain** ✅

Example:

```
sqrt(8)
```

Answer 0 se 8 ke beech hi hoga.

Toh:

```
low = 0
high = 8
```

Har step pe check:

```
mid * mid <= 8 ?
```

- Agar **true** → answer right side me bhi ho sakta hai
- Agar **false** → answer left side me hoga

Yahi Binary Search on Answer ka core idea hai.

1️⃣ Integer Square Root (LeetCode 69)

🎯 Goal

Return **floor value** of `sqrt(x)`

Example:

```
Input: 8
Output: 2
```

🧠 Intuition (Hindi)

Hume **largest number chahiye jiska square ≤ x ho.**

Mathematically:

```
mid² ≤ x
```

Agar satisfy kare:

- `ans = mid`
- Right side explore karo (shayad bada valid mil jaye)

Agar nahi:

- Left side move karo

⚠️ Important: Overflow Avoid Karna

❌ Galti:

```
mid * mid <= x
```

Ye overflow kar sakta hai (int range exceed)

✅ Sahi tarika:

```
mid <= x / mid
```

✅ Clean Java Code

```
class Solution {

    public int mySqrt(int x) {

        if (x == 0 || x == 1)
            return x;

        int low = 1;
        int high = x;
        int ans = 0;

        while (low <= high) {

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

            // Overflow safe check
            if (mid <= x / mid) {

                ans = mid;        // possible answer
                low = mid + 1;    // try bigger value

            } else {

                high = mid - 1;   // mid too large
            }
        }

        return ans;
    }
}
```

🧪 Dry Run (x = 8)

```
low=1, high=8
mid=4 → 16>8 → high=3
mid=2 → 4<=8 → ans=2, low=3
mid=3 → 9>8 → high=2
Stop → ans=2
```

2️⃣ Nth Root (Integer Version)

🎯 Problem

Find largest integer `k` such that:

```
k^n ≤ m
```

Example:

```
m = 27
n = 3
Answer = 3
```

🧠 Intuition

Same pattern:

```
mid^n <= m ?
```

- True → right side explore
- False → left side move

⚠️ Important: Power Overflow

Power calculate karte waqt overflow ho sakta hai.

Safe approach:

- `long` use karo
- Ya loop me break kar do agar value `m` se exceed ho jaye

✅ Clean Java Code

```
class Solution {

    public int nthRoot(int n, int m) {

        int low = 1;
        int high = m;
        int ans = 1;

        while (low <= high) {

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

            long power = 1;

            // Compute mid^n safely
            for (int i = 0; i < n; i++) {
                power *= mid;

                if (power > m)
                    break;
            }

            if (power <= m) {

                ans = mid;
                low = mid + 1;

            } else {

                high = mid - 1;
            }
        }
```

```
        return ans;
    }
}
```

🧪 Dry Run (m=27, n=3)

```
low=1, high=27
mid=14 → overflow >27 → high=13
mid=7 → >27 → high=6
mid=3 → =27 → ans=3, low=4
mid=5 → >27 → high=4
mid=4 → >27 → high=3
Stop → ans=3
```

3️⃣ Floating Point Square Root (Precision Version)

🎯 Goal

Find real value like:

```
sqrt(8) = 2.828427...
```

Yaha integer nahi milega.

🧠 Idea

Binary search tab tak chalao jab tak:

```
(high - low > epsilon)
```

Example:

```
epsilon = 1e-6
```

✅ Java Code (Precision 1e-6)

```
class Solution {

    public double sqrtPrecision(double x) {

        double low = 0;
        double high = x;
        double eps = 1e-6;

        while (high - low > eps) {

            double mid = (low + high) / 2;

            if (mid * mid <= x) {
                low = mid;
            } else {
                high = mid;
            }
        }

        return low;
    }
}
```

🧪 Dry Run (x=8)

```
low=0, high=8
mid=4 → 16>8 → high=4
mid=2 → 4<=8 → low=2
mid=3 → 9>8 → high=3
mid=2.5 → 6.25<=8 → low=2.5
...
Eventually → 2.828427
Stop when high - low < 1e-6
```

🎯 Binary Search on Answer Template

Step	Meaning
Define range	low & high
Define condition	mid^n ≤ target
Move right	if valid
Move left	if too large

⚠️ Interview Traps

❌ mid * mid overflow

❌ infinite loop in precision

❌ forgetting x = 0 case

❌ returning high instead of ans

❌ not early-breaking power loop

🏆 Golden Line (Interview Ready)

> "Square root / Nth root is a classic Binary Search on numeric range where we search for the largest value mid such that mid^n ≤ target."

Agar chaho to main iska **one-page revision sheet version** bhi bana du jo tum Striver sheet ke saath revise kar sako 💪

Perfect 🔥

Ab main isko **clean, structured, interview-ready document format** me likh raha hoon — taaki tum directly revise ya bol sako interview me.

🚀 Square Root / Nth Root — Interview Mindset Guide

🔥 1️⃣ Interviewer Kya Test Kar Raha Hota Hai?

Ye sirf `sqrt()` ka question nahi hota.

✅ 1. Binary Search on Answer Samajh Aaya Ya Nahi?

- Koi sorted array nahi diya.
- Phir bhi binary search use kar rahe ho.
- Yeh **key insight** hai.

> Insight: “Answer numeric range me lie karta hai, not in an array.”

✅ 2. Overflow Handling

Interviewer dekhta hai tum likhte kya ho:

❌ Risky:

```
mid * mid <= x
```

✅ Smart:

```
mid <= x / mid
```

Agar tum proactively overflow mention karte ho → strong signal.

✅ 3. Edge Case Awareness

Most rejections yahin hote hain.

⚠️ 2️⃣ Most Common Interview Mistakes

❌ Mistake 1: Overflow

Large input:

```
x = 2,147,483,647
```

`mid * mid` overflow karega.

✔️ Always use:

```
mid <= x / mid
```

❌ Mistake 2: Infinite Loop (Precision Case)

Floating point me log likh dete hain:

```
while (low <= high)   // ❌ wrong
```

Correct:

```
while (high - low > eps)
```

❌ Mistake 3: Edge Cases Ignore Karna

Interviewer deliberately dega:

- `x = 0`
- `x = 1`
- `n = 1`
- `m = 1`

Agar crash hua → reject.

❌ Mistake 4: Wrong Search Space

Log likhte hain:

```
low = 0;
high = x;
```

Better (integer sqrt):

```
low = 1;
high = x;
```

Floating case me agar `x < 1` ho:

```
high = 1;
```

🧠 3️⃣ Important Edge Cases

🔷 Case 1: x = 0

Return 0 immediately.

🔷 Case 2: x = 1

Return 1.

🔷 Case 3: Perfect Square

```
x = 16 → answer must be 4
```

Check that you return exact value.

🔷 Case 4: Very Large Number

```
x = 2147395600
```

Ye `46340²` hai.

Overflow handling mandatory.

🔷 Case 5: Nth Root Not Exact

```
M = 10
N = 3
```

Cube root floor = 2.

Clarify with interviewer:

- Exact root chahiye?

- Floor chahiye?
- 1 return karna hai agar exact na ho?

🎯 4️⃣ Interview Strategy (Very Important)

Question milte hi bolo:

> "Since the answer lies in a numeric range, I'll apply Binary Search on Answer."

Phir bolo:

> "I need to find the largest mid such that $mid^n \le target$."

Clear direction. Strong confidence.

🔥 5️⃣ Common Follow-Up Questions

🔷 Q1: Time Complexity?

Binary Search:

```
O(log range)
```

Nth root:

```
O(log M × N)
```

🔷 Q2: Power Calculation Optimize?

- Early break when power > m
- Fast exponentiation (Binary Exponentiation)

🔷 Q3: What if Input is Double?

Use precision-based loop:

```
while (high - low > eps)
```

🔷 Q4: Newton's Method vs Binary Search?

Binary Search	Newton's Method
Simple	Faster convergence
Deterministic	Harder to implement
Safe	Requires careful math

🧩 6️⃣ Deep Insight (Senior-Level Thinking)

Binary Search on Answer tab kaam karta hai jab:

1. Answer numeric range me ho
2. Monotonic property exist kare

Yaha monotonic property:

If:

```
mid² ≤ x
```

Then:

```
All smaller values are also valid
```

Isliye binary search work karta hai.

🚨 Final Interview Checklist

Before submission, check:

- ✔ x = 0 handled?
- ✔ x = 1 handled?
- ✔ Overflow safe?
- ✔ Correct search space?

- ✔️ Infinite loop risk?
- ✔️ Returning correct variable (`ans`)?

🏆 Golden Interview Line

> "Square root problems are classic Binary Search on Answer problems where we exploit the monotonic property of mid" with respect to the target."

Agar chaho to next level me main tumhe:

- 🔥 Real interview tricky variants
- 🔥 Newton's Method implementation
- 🔥 Binary Search on Answer ke 5 advanced pattern questions

bhi structured format me bana deta hoon.

Perfect 👍

Ab main tumhe **Binary Search ka complete interview mindset** deta hoon —

- Interviewer kya test karta hai
- Kaise sochna hai
- Edge cases kya hote hain
- Most common mistakes
- Kaise impress karna hai

Clean structured format me 👇

🚀 Binary Search — Complete Interview Guide

🔥 1️⃣ Interviewer Kya Test Kar Raha Hota Hai?

Binary Search ka question usually simple dikhta hai...

But interviewer actually yeh test karta hai:

✅ 1. Kya tum Monotonic Property pehchan sakte ho?

Binary Search tabhi chalega jab:

- Array sorted ho

  OR

- Answer space monotonic ho

Monotonic matlab:

```
FFFFFTTTTT
ya
TTTTTFFFFF
```

Ek clear transition point hona chahiye.

Agar tum bolte ho:

> "Since the function is monotonic, we can apply binary search."

Interviewer impressed.

✅ 2. Boundary Handling

Most log fail hote hain yahin.

Interviewer dekh raha hota hai:

- low/high sahi define kiya?
- loop condition sahi?
- mid calculation safe?
- infinite loop ka risk?

✅ 3. Off-by-One Handling

Binary Search ka 80% error yahin hota hai.

🧠 2️⃣ Kaise Sochna Hai (Thinking Process)

Question milte hi khud se pucho:

Step 1:

Kya data sorted hai?

→ Haan → Direct binary search

→ Nahi → Kya answer monotonic hai?

Step 2:

Main kya search kar raha hoon?

- Exact value?
- First occurrence?
- Last occurrence?
- Lower bound?
- Upper bound?
- Minimum valid answer?
- Maximum valid answer?

Binary Search ka template problem ke hisaab se change hota hai.

Step 3:

Search Space define karo

Binary Search hamesha do cheez pe hota hai:

- Index space
- Answer space

⚠️ 3️⃣ Most Common Interview Mistakes

❌ Mistake 1: Wrong mid formula

Wrong:

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

Correct:

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

Overflow avoid karta hai.

❌ Mistake 2: Infinite Loop

Galat update:

```
low = mid;
high = mid;
```

Yeh stuck ho sakta hai.

Correct pattern:

```
low = mid + 1;
high = mid - 1;
```

❌ Mistake 3: Loop Condition Confusion

Do main patterns hote hain:

Pattern 1:

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

Pattern 2:

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

Mix mat karo.

❌ Mistake 4: Wrong Answer Return

Log return kar dete hain:

```
return low;
```

But question kabhi kabhi demand karta hai:

- high
- ans
- mid

Always verify.

❌ Mistake 5: Edge Cases Ignore

🧩 4️⃣ Important Edge Cases

🔷 Empty array

```
nums.length == 0
```

🔷 Single element

```
[5]
```

🔷 Target not present

Return kya karna hai?

- 1?
- insertion index?

Clarify.

🔷 All elements same

```
[2,2,2,2]
```

🔷 Duplicate elements

First occurrence / last occurrence carefully handle karo.

🔷 Large numbers (Overflow case)

Always safe mid formula use karo.

🎯 5️⃣ Binary Search Variants (Interview Favorite)

1️⃣ Classic Search

Return index of target.

2️⃣ First Occurrence

Condition change hota hai:

Target milne ke baad left move karo.

3️⃣ Last Occurrence

Target milne ke baad right move karo.

4️⃣ Lower Bound

Smallest index jahan:

```
nums[i] >= target
```

5️⃣ Upper Bound

Smallest index jahan:

```
nums[i] > target
```

6️⃣ Binary Search on Answer

Examples:

- Square root
- Capacity to ship packages
- Koko eating bananas
- Aggressive cows
- Min days to make bouquets

Yaha sorted array nahi hota

Monotonic condition hoti hai.

🏆 6️⃣ How to Impress Interviewer

Instead of coding immediately:

Bolna start karo:

> “I’ll first check if the problem has a monotonic property.”

Phir:

> “I’ll define the search space clearly.”

Phir:

> “I’ll use overflow-safe mid calculation.”

Phir complexity bolo:

```
Time: O(log n)
Space: O(1)
```

Structured thinking = strong signal.

🔥 7️⃣ Deep Understanding (Senior Level)

Binary Search ka real meaning hai:

> Searching on a monotonic function.

Array sirf ek special case hai.

Mathematically:

You are finding the boundary where:

```
Condition changes from false to true
```

Yahi real binary search hai.

🚨 Final Interview Checklist

Before submission:

- ✔️ Sorted/monotonic confirmed?
- ✔️ Correct search space?
- ✔️ Overflow-safe mid?
- ✔️ Infinite loop safe?
- ✔️ Edge cases handled?
- ✔️ Correct return value?

🥇 Golden Interview Line

> "Binary Search is not about searching in arrays — it is about exploiting monotonicity to reduce search space logarithmically."

Agar tum chaho to main next:

- 🔥 Binary Search ke 10 tricky patterns
- 🔥 15 must-do interview problems
- 🔥 Visual intuition breakdown

- 🔥 Complete Java template pack

bhi bana sakta hoon structured format me.