1. Binary Search — Iterative Approach

Problem Statement

Given a sorted array `nums` and a target value `target`, return the **index of `target`** if it exists in the array. Otherwise, return `-1`.

Approach

Hum **binary search** algorithm use karte hain. Do pointers lete hain: `low` aur `high`. Har step pe:

- Agar `nums[mid] == target` hai, to `mid` return karo.
- Agar `nums[mid] < target`, to `low = mid+1` kar do.
- Agar `nums[mid] > target`, to `high = mid-1` kar do. Jab tak `low > high` nahi ho jata, tab tak repeat karte hain. Agar nahi mila, to `-1` return karte hain.

Patterns / Concepts

Binary Search Two Pointer Technique Divide and Conquer

Time Complexity

O(log n) — because we halve the search space at each step. **O(1)** — since we use only a few extra variables.

Example

Input: `nums = [2,3,4,5,6,7,8,9,56]`, `target = 56` **Output:** `your target element 56 is at index: 8`

Edge Cases

Edge Case 1: **Input:** `nums = []`, `target = 5` **Output:** `-1` (array is empty)

Edge Case 2: **Input:** `nums = [1]`, `target = 1` **Output:** `0` (single element found)

```
# Function to perform binary search iteratively

def BinarySearch(nums, target):
    n = len(nums)
    low = 0
    high = n - 1
    # Loop until low and high cross
    while low <= high:
```

```python
        mid = low + ((high - low) // 2)  # Calculate middle index safely
        if nums[mid] == target:
            return mid  # Target found at mid
        elif nums[mid] < target:
            low = mid + 1  # Search right half
        else:
            high = mid - 1  # Search left half
    return -1  # Target not found

# Example usage
nums = [2, 3, 4, 5, 6, 7, 8, 9, 56]
target = 56
output = BinarySearch(nums, target)
print(f"your target element {target} is at index:", output)
```

2. Lower Bound — Iterative Binary Search

Problem Statement

Given a sorted array `nums` and a target value `x`, return the **index of the first element in `nums` which is greater than or equal to `x`** (the lower bound).
If all elements are smaller than `x`, return `n` (size of array).

Approach

English

We use a modified **binary search**.
We maintain two pointers: `low` and `high`, and initialize `ans = n`.
At each step:

- If `nums[mid] >= x`, this could be our answer, so we update `ans = mid` and move `high = mid - 1`.
- Else, move `low = mid + 1`.

Loop until `low > high`. `ans` will be the index of lower bound.

Hinglish

Hum modified **binary search** use karte hain.
Do pointers lete hain: `low` aur `high`, aur ek `ans = n` initialize karte hain.
Har step pe:

- Agar `nums[mid] >= x` ho, to ye ek potential answer ho sakta hai, to `ans = mid` aur `high = mid -1` kar dete hain.
- Nahi to `low = mid+1` kar dete hain.

Jab tak `low > high` nahi ho jata, tab tak loop chalta hai. Final `ans` lower bound ka index hoga.

Patterns / Concepts

Binary Search
Lower Bound in Sorted Array
Divide and Conquer

Time Complexity

O(log n) — because we halve the search space at each step.

Space Complexity

O(1) — since we use only a few extra variables.

Example

Input:
`nums = [2, 3, 7, 8, 8, 8, 9, 10, 56]`, `x = 8`

Output:
`Your lowerbound for the el 8 is at index: 3 --> 8`

Edge Cases

Edge Case 1:
Input: `nums = []`, `x = 5`
Output: `0` (array is empty, conventionally returns size `0`)

Edge Case 2:
Input: `nums = [1, 2, 3]`, `x = 10`
Output: `3` (all elements are smaller than `x`, so returns `n`)

Code with Comments

```
def Lowerbound(nums, x, n):
    low = 0
    high = n - 1
    ans = n

    while low <= high:
        mid = (low + high) // 2

        if nums[mid] >= x:
            ans = mid
            high = mid - 1
        else:
            low = mid + 1

    return ans

nums = [2, 3, 7, 8, 8, 8, 9, 10, 56]
x = 8
n = len(nums)
```

```
output = Lowerbound(nums, x, n)
print(f"Your lowerbound for the el {x} is at index: {output} --> {nums[output]} ")
```

3. Upper Bound — Iterative Binary Search

Problem Statement

Given a sorted array `nums` and a target value `x`, return the **index of the first element in `nums` which is strictly greater than `x`** (the upper bound).
If no such element exists, return `n` (size of array).

Approach

English

We use a modified **binary search**.
We maintain two pointers: `low` and `high`, and initialize `ans = n`.
At each step:

- If `nums[mid] > x`, this could be our answer, so we update `ans = mid` and move `high = mid - 1`.
- Else, move `low = mid + 1`.

Loop until `low > high`. `ans` will be the index of upper bound.

Hinglish

Hum modified **binary search** use karte hain.
Do pointers lete hain: `low` aur `high`, aur ek `ans = n` initialize karte hain.
Har step pe:

- Agar `nums[mid] > x` ho, to ye ek potential answer ho sakta hai, to `ans = mid` aur `high = mid -1` kar dete hain.
- Nahi to `low = mid+1` kar dete hain.

Jab tak `low > high` nahi ho jata, tab tak loop chalta hai. Final `ans` upper bound ka index hoga.

Patterns / Concepts

Binary Search
Upper Bound in Sorted Array
Divide and Conquer

Time Complexity

O(log n) — because we halve the search space at each step.

Space Complexity

O(1) — since we use only a few extra variables.

Example

Input:
`nums = [2, 3, 4, 5, 6, 7, 8, 8, 8, 9, 56]`, `x = 8`

Output:
`Your upperbound for the el 8 is at index: 9`

Edge Cases

Edge Case 1:
Input: `nums = []`, `x = 5`
Output: `0` (array is empty, conventionally returns size `0`)

Edge Case 2:
Input: `nums = [1, 2, 3]`, `x = 10`
Output: `3` (all elements are smaller than or equal to `x`, so returns `n`)

Code with Comments

```
def upperbound(nums, x, n):
    low = 0
    high = n - 1
    ans = n

    while low <= high:
        mid = (low + high) // 2

        if nums[mid] > x:
            ans = mid
            high = mid - 1
        else:
            low = mid + 1

    return ans

nums = [2, 3, 4, 5, 6, 7, 8, 8, 8, 9, 56]
x = 8
n = len(nums)
output = upperbound(nums, x, n)
print(f"Your upperbound for the el {x} is at index:", output)
```

4. Find Pivot in Rotated Sorted Array — Iterative Binary Search

Problem Statement

Given a **rotated sorted array** `nums`, find the **index of the smallest element (pivot)** where the rotation happens.
If the array is not rotated, return `0`.

Approach / तरीका (English + Hinglish)

English

We use a modified **binary search** to locate the pivot point (smallest element).
We maintain two pointers: `low` and `high`.

- If `nums[low] < nums[high]`, the array is already sorted and pivot is at index `0`.
- Otherwise, we keep checking `mid`:
  - If `nums[mid] > nums[mid+1]`, then `mid+1` is pivot.
  - If `nums[mid] < nums[mid-1]`, then `mid` is pivot.
- If `nums[mid] >= nums[low]`, move `low = mid+1`.
- Else move `high = mid-1`.

Loop until `low > high`. If not found explicitly, pivot is at `0`.

Hinglish

Hum modified **binary search** use karte hain pivot point (smallest element) dhoondhne ke liye.
Do pointers lete hain: `low` aur `high`.

- Agar `nums[low] < nums[high]`, matlab array already sorted hai aur pivot `0` hai.
- Nahi to, `mid` check karte hain:
  - Agar `nums[mid] > nums[mid+1]`, to `mid+1` pivot hai.
  - Agar `nums[mid] < nums[mid-1]`, to `mid` pivot hai.
- Agar `nums[mid] >= nums[low]`, to `low = mid+1`.
- Nahi to `high = mid-1`.

Agar loop ke baad bhi nahi mila, to pivot `0` hai.

Patterns / Concepts

Binary Search
Find Minimum in Rotated Sorted Array
Divide and Conquer

Time Complexity

O(log n) — because we halve the search space at each step.

Space Complexity

O(1) — since we use only a few extra variables.

Example

Input:
`nums = [6, 7, 8, 1, 2, 3, 4, 5]`

Output:
`The pivot is at index: 3`

Edge Cases

Edge Case 1:
Input: `nums = [1, 2, 3, 4, 5]`
Output: `0` (array is already sorted)

Edge Case 2:
Input: `nums = [2, 1]`
Output: `1` (pivot at last index)

Code with Comments

```python
def search(nums):
    n = len(nums)
    low = 0
    high = n - 1

    # If array is already sorted
    if nums[low] < nums[high]:
        return 0

    while low <= high:
        mid = (low + high) // 2

        # Check if mid is pivot
        if mid < n - 1 and nums[mid] > nums[mid + 1]:
            return mid + 1
        elif mid > 0 and nums[mid] < nums[mid - 1]:
            return mid

        # Decide which half to search
        if nums[mid] >= nums[low]:
            low = mid + 1
        else:
            high = mid - 1

    return 0  # Pivot not found explicitly, default to 0

nums = [6, 7, 8, 1, 2, 3, 4, 5]
pivot = search(nums)
print(f"The pivot is at index: {pivot}")
```

5. Find Floor and Ceil of a Number in a Sorted Array — Iterative Binary Search

Problem Statement

Given a **sorted array** `arr` of size `n` and a number `x`, find:

- **Floor:** the largest number in `arr` ≤ `x`
- **Ceil:** the smallest number in `arr` ≥ `x`
  If floor or ceil does not exist, return `-1` for that.

Approach / तरीका (English + Hinglish)

English

We use two separate binary searches:

- `findFloor`: keep track of the largest `arr[mid] ≤ x` seen so far. Move `low = mid+1` if `arr[mid] ≤ x`.
- `findCeil`: keep track of the smallest `arr[mid] ≥ x` seen so far. Move `high = mid-1` if `arr[mid] ≥ x`.

Return both values as a pair.

Hinglish

Hum do alag **binary search** chalate hain:

- `findFloor`: ab tak jo sabse bada element `≤ x` mila usse track karte hain. Agar `arr[mid] ≤ x`, to `low = mid+1`.
- `findCeil`: ab tak jo sabse chhota element `≥ x` mila usse track karte hain. Agar `arr[mid] ≥ x`, to `high = mid-1`.

Finally `floor` aur `ceil` pair me return karte hain.

Patterns / Concepts

Binary Search
Floor and Ceil in Sorted Array
Search Space Reduction

Time Complexity

O(log n) — two binary searches, each logarithmic.

Space Complexity

O(1) — no extra space.

Example

Input:
`arr = [3, 4, 4, 7, 8, 10]`, `x = 5`

Output:
`The floor and ceil are: 4 7`

Edge Cases

Edge Case 1:
Input: `arr = [1, 2, 3]`, `x = 0`
Output: `The floor and ceil are: -1 1` (no floor)

Edge Case 2:
Input: `arr = [5, 6, 7]`, `x = 10`
Output: `The floor and ceil are: 7 -1` (no ceil)

Code with Comments

```
def findFloor(arr, n, x):
    low = 0
    high = n - 1
    ans = -1

    while low <= high:
        mid = (low + high) // 2
        # possible floor
        if arr[mid] <= x:
            ans = arr[mid]
            low = mid + 1  # look further right
        else:
            high = mid - 1  # look left

    return ans

def findCeil(arr, n, x):
    low = 0
    high = n - 1
    ans = -1

    while low <= high:
        mid = (low + high) // 2
        # possible ceil
        if arr[mid] >= x:
            ans = arr[mid]
            high = mid - 1  # look further left
        else:
            low = mid + 1  # look right

    return ans

def getFloorAndCeil(arr, n, x):
```

```
    f = findFloor(arr, n, x)
    c = findCeil(arr, n, x)
    return (f, c)

arr = [3, 4, 4, 7, 8, 10]
n = 6
x = 5
ans = getFloorAndCeil(arr, n, x)
print("The floor and ceil are:", ans[0], ans[1])
```

6. Search in Rotated Sorted Array — Iterative Binary Search

Problem Statement

Given a **rotated sorted array** `nums` and a `target`, return the **index of the target** if it exists in the array.
If the target is not found, return `-1`.

Approach / तरीका (English + Hinglish)

English

We use a modified **binary search**.
At each step:

- If `nums[mid] == target`, return `mid`.
- If the left half `nums[low..mid]` is sorted:
  - If `target` lies in `nums[low..mid]`, move `high = mid-1`.
  - Else, move `low = mid+1`.
- Else, the right half `nums[mid..high]` is sorted:
  - If `target` lies in `nums[mid..high]`, move `low = mid+1`.
  - Else, move `high = mid-1`.

If the loop finishes without finding, return `-1`.

Hinglish

Hum modified **binary search** use karte hain.
Har step pe:

- Agar `nums[mid] == target`, to `mid` return karo.
- Agar left half `nums[low..mid]` sorted hai:
  - Agar `target` left half me hai, to `high = mid-1`.
  - Nahi to `low = mid+1`.
- Warna right half `nums[mid..high]` sorted hai:
  - Agar `target` right half me hai, to `low = mid+1`.
  - Nahi to `high = mid-1`.

Agar nahi mila, to `-1` return karo.

Patterns / Concepts

Binary Search
Search in Rotated Sorted Array
Divide and Conquer

Time Complexity

O(log n) — because each step halves the search space.

Space Complexity

O(1) — only pointers used.

Example

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

Output:
`4`

Edge Cases

Edge Case 1:
Input: `nums = [1]`, `target = 0`
Output: `-1` (target not present)

Edge Case 2:
Input: `nums = [1,3]`, `target = 3`
Output: `1`

Code with Comments

```python
def search(nums, target):
    n = len(nums)
    low = 0
    high = n - 1

    while low <= high:
        mid = (low + high) // 2

        if nums[mid] == target:
            return mid

        # Left half is sorted
        if nums[low] <= nums[mid]:
            if nums[low] <= target <= nums[mid]:
```

```
                high = mid - 1
            else:
                low = mid + 1
        else:
            # Right half is sorted
            if nums[mid] <= target <= nums[high]:
                low = mid + 1
            else:
                high = mid - 1

    return -1
```

Find Square Root of a Number — Floor Value Using Binary Search

Problem Statement

Given a **non-negative integer** `n`, find the **floor value of square root of `n`**.
That is, return the largest integer `x` such that `x*x ≤ n`.

Approach

English

We use **binary search** to find the square root.
We search in the range `[0, n//2]` and keep track of the last `mid` whose square `≤ n` as the answer.

At each step:

- If `mid*mid > n`, move `high = mid-1`.
- Else, store `mid` in `ans` and move `low = mid+1`.

When loop ends, `ans` will hold the floor of square root.

Hinglish

Hum **binary search** use karke square root nikalte hain.
Search range hoti hai `[0, n//2]`. Jo `mid*mid ≤ n` hota hai, usse `ans` me store karte hain.

Har step pe:

- Agar `mid*mid > n`, to `high = mid-1`.
- Nahi to `ans = mid` aur `low = mid+1`.

Loop ke baad `ans` me square root ka floor hota hai.

Patterns / Concepts

Binary Search
Search Space Reduction

Square Root

Time Complexity

O(log n) — binary search halves the range each step.

Space Complexity

O(1) — no extra space.

Example

Input:
`n = 5`

Output:
`2` — since `2*2=4 ≤ 5` but `3*3=9 > 5`

Edge Cases

Edge Case 1:
Input: `n = 0`
Output: `0`

Edge Case 2:
Input: `n = 1`
Output: `1`

Code with Comments

```
def SqaureRoot(n):
    ans = 0
    low = 0
    high = n // 2

    while low <= high:
        mid = (low + high) // 2

        if mid * mid > n:
            high = mid - 1
        else:
            ans = mid
            low = mid + 1

    return ans

print(SqaureRoot(5))
```

Find Nth Root of a Number — Using Binary Search

Problem Statement

Given two integers `n` and `m`, find the integer **x** such that `x^n = m`.
If no such integer exists, return `-1`.

Approach

English

We use **binary search** to find the `n`th root of `m`.
We search in the range `[1, m]` and at each step:

- Compute `mid^n` (mid to the power n).
- If `mid^n == m`, we found the root and return `mid`.
- If `mid^n < m`, we search the right half (`low = mid+1`).
- If `mid^n > m`, we search the left half (`high = mid-1`).

If no such integer root is found after the loop, return `-1`.

Hinglish

Hum **binary search** use karke `n`th root of `m` nikalte hain.
Search range hoti hai `[1, m]`. Har step pe:

- `mid^n` calculate karte hain.
- Agar `mid^n == m`, to `mid` return kar dete hain.
- Agar `mid^n < m`, to `low = mid+1`.
- Agar `mid^n > m`, to `high = mid-1`.

Agar loop ke baad bhi nahi mila, to `-1` return karte hain.

Patterns / Concepts

Binary Search
Nth Root Search
Search Space Reduction

Time Complexity

O(log m) — binary search on `m`.

Space Complexity

O(1) — no extra space.

Example

Input:
`n = 2`, `m = 9`

Output:
`3` — since `3^2 = 9`

Edge Cases

Edge Case 1:
Input: `n = 2`, `m = 10`
Output: `-1` (no integer square root of 10)

Edge Case 2:
Input: `n = 1`, `m = 5`
Output: `5` (any number to power 1 is itself)

Code with Comments

```
def nthRoot(n, m):
    low = 1
    high = m

    while low <= high:
        mid = (low + high) // 2
        power = mid ** n

        if power == m:
            return mid  # found exact root
        elif power < m:
            low = mid + 1  # look in right half
        else:
            high = mid - 1  # look in left half

    return -1  # no integer root found

n = 2
m = 9
print(nthRoot(n, m))
```

Find Minimum Eating Speed — Using Binary Search

Problem Statement

Given an array `nums` where `nums[i]` represents the size of the `i-th` pile of bananas and an integer `h` representing the number of hours available, find the **minimum integer eating speed** `k` such that all bananas can be eaten within `h` hours.

You can eat at most `k` bananas from a pile in an hour, and you must completely finish one pile before moving to the next.

Approach / तरीका (English + Hinglish)

English

We use **binary search** to find the smallest valid speed `k`.
We search in the range `[1, max(nums)]`.
At each step:

- Assume current `mid` as the eating speed.
- Calculate total hours required at this speed using `totalTime()`.
- If total hours ≤ `h`, store `mid` as answer and try to find smaller speed (`high = mid-1`).
- Otherwise, search in higher speeds (`low = mid+1`).

When the loop ends, `ans` is the minimum valid eating speed.

Hinglish

Hum **binary search** use karke minimum valid speed `k` nikalte hain.
Search range hoti hai `[1, max(nums)]`.
Har step pe:

- Current `mid` ko speed maan ke total hours calculate karte hain.
- Agar total hours ≤ `h`, to `mid` ko `ans` me store karke aur chhoti speed try karte hain (`high = mid-1`).
- Warna badi speed check karte hain (`low = mid+1`).

Loop ke baad `ans` me minimum valid speed hoti hai.

Patterns / Concepts

Binary Search
Search on Answer Space
Greedy Checking

Time Complexity

O(n * log(max(nums))) — binary search on range and linear check at each step.

Space Complexity

O(1) — no extra space.

Example

Input:
`nums = [3,6,7,11]`, `h = 8`

Output:
`4` — minimum speed to finish all piles in 8 hours.

Edge Cases

Edge Case 1:
Input: `nums = [1]`, `h = 1`
Output: `1` (only one pile)

Edge Case 2:
Input: `nums = [30,11,23,4,20]`, `h = 6`
Output: `23`

Code with Comments

```python
import math

# Helper: calculate total hours at speed k
def totalTime(nums, k):
    hours = 0
    for num in nums:
        hours += math.ceil(num / k)
    return hours

# Main: binary search on k
def minEatingSpeed(nums, h):
    low = 1
    high = max(nums)
    ans = high

    while low <= high:
        mid = (low + high) // 2
        totalhours = totalTime(nums, mid)

        if totalhours <= h:
            ans = mid  # try to minimize
            high = mid - 1
        else:
            low = mid + 1

    return ans
```

Minimum Days to Make m Bouquets — Binary Search on Answer

Problem Statement

You are given an integer array `bloomDay`, where `bloomDay[i]` is the day the `i-th` flower blooms.
You are also given integers `m` and `k`.

You need to make exactly `m` bouquets, and each bouquet needs exactly `k` **adjacent** bloomed flowers.
Return the **minimum number of days** you need to wait to be able to make `m` bouquets.
If it is impossible, return `-1`.

Example Question Context

This is a **LeetCode Hard problem** — a classic **binary search on answer space** pattern.
We have to find the smallest day such that it is possible to pick `m` bouquets of `k` adjacent flowers on or before that day.

Approach / तरीका (English + Hinglish)

English

We binary search on the answer: the day.
Range of possible days is `[min(bloomDay), max(bloomDay)]`.
At each step:

- Assume `mid` day.
- Check if we can make `m` bouquets on `mid` day using `canMakeBouquets(mid)`:
  - Iterate `bloomDay`, count adjacent bloomed flowers on or before `mid`.
  - Every time `k` adjacent flowers are found, increment `bouquets`.
  - Reset counter if an unbloomed flower is encountered.
- If `m` bouquets can be made, try smaller day (`high = mid-1`).
- Else, try larger day (`low = mid+1`).

Return the minimum day found or `-1`.

Hinglish

Hum **day** pe binary search karte hain.
Possible day range: `[min(bloomDay), max(bloomDay)]`.
Har step pe:

- Assume karte hain ki answer `mid` day hai.
- Check karte hain `canMakeBouquets(mid)` se ki `m` bouquets ban sakte hain ya nahi.
  - `bloomDay` iterate karke `k` adjacent bloomed flowers count karte hain.
  - Agar mil gaye, to `bouquets` increment karte hain aur flowers reset karte hain.
- Agar possible hai, to chhoti day try karte hain (`high = mid-1`), warna badi day (`low = mid+1`).

Patterns / Concepts

Binary Search on Answer
Greedy Check
Search Space Reduction

Time Complexity

O(n * log(max(bloomDay))) — binary search over days and linear scan at each check.

Space Complexity

O(1) — only counters used.

Example

Input:
`bloomDay = [1,10,3,10,2]`, `m = 3`, `k = 1`

Output:
`3` — minimum day to make 3 bouquets of 1 adjacent flower each.

Edge Cases

Edge Case 1:
Input: `bloomDay = [1,2,4,9,3,4,1]`, `m = 2`, `k = 4`
Output: `-1` (not enough flowers)

Edge Case 2:
Input: `bloomDay = [1,1,1,1,1]`, `m = 1`, `k = 4`
Output: `1`

Code with Comments

```python
from typing import List

def minDays(bloomDay: List[int], m: int, k: int) -> int:
    # If there are not enough flowers to form m bouquets
    if m * k > len(bloomDay):
        return -1

    # Helper: can we make m bouquets by day?
    def canMakeBouquets(day: int) -> bool:
        bouquets = 0
        flowers = 0
        for bloom in bloomDay:
            if bloom <= day:
                flowers += 1
                if flowers == k:
                    bouquets += 1
                    flowers = 0
            else:
                flowers = 0
        return bouquets >= m

    low, high = min(bloomDay), max(bloomDay)
    result = -1

    # Binary search on days
    while low <= high:
        mid = (low + high) // 2
```

```
        if canMakeBouquets(mid):
            result = mid
            high = mid - 1
        else:
            low = mid + 1

    return result
```

Smallest Divisor Given a Threshold — Binary Search on Answer

Problem Statement

You are given an integer array `nums` and an integer `threshold`.
You must choose an integer `divisor` such that the sum of `ceil(nums[i] / divisor)` for all `i` ≤ `threshold`.
Find the **smallest such divisor**.

This is a classic **search on answer space** problem — the smaller the divisor, the larger the sum, and vice versa.

Approach

English

We binary search on the possible divisor in the range `[1, max(nums)]`.
At each step:

- Assume `mid` as the current divisor.
- Calculate the sum of ceilings for all elements divided by `mid`.
- If sum ≤ threshold, store `mid` as current best (`k`) and search for a smaller divisor (`high = mid-1`).
- Else, search for a larger divisor (`low = mid+1`).

When the loop ends, `k` holds the smallest valid divisor.

Hinglish

Hum possible divisor par **binary search** karte hain (`[1, max(nums)]`).
Har step pe:

- `mid` ko current divisor maante hain.
- Sab elements ko `mid` se divide karke unka ceiling sum nikalte hain.
- Agar sum ≤ threshold hai, to `mid` ko `k` me store karke aur chhota divisor try karte hain (`high = mid-1`).
- Warna bada divisor try karte hain (`low = mid+1`).

Loop ke baad `k` minimum valid divisor hota hai.

Patterns / Concepts

Binary Search on Answer
Greedy Check
Search Space Reduction

Time Complexity

O(n * log(max(nums))) — binary search over divisors and linear scan at each check.

Space Complexity

O(1) — constant space used.

Example

Input:
`nums = [1,2,5,9]`, `threshold = 6`

Output:
`5` — smallest divisor such that `ceil(1/5) + ceil(2/5) + ceil(5/5) + ceil(9/5) = 6`

Edge Cases

Edge Case 1:
Input: `nums = [1,2,3]`, `threshold = 6`
Output: `1` (dividing by 1 is always valid)

Edge Case 2:
Input: `nums = [1000000]`, `threshold = 1`
Output: `1000000` (must divide largest number itself)

Code with Comments

```
import math

def smallestDivisor(nums, threshold):
    # Helper: calculate sum of ceilings for a given divisor
    def calculateSum(nums, divisor):
        sum_ = 0
        for number in nums:
            sum_ += math.ceil(number / divisor)
        return sum_

    low = 1
    high = max(nums)
    k = high  # initial answer

    # Binary search on possible divisors
    while low <= high:
        mid = (low + high) // 2
        if calculateSum(nums, mid) <= threshold:
            k = mid  # found a smaller valid divisor
```

```
            high = mid - 1
        else:
            low = mid + 1

    return k
```

Capacity to Ship Packages Within D Days — Binary Search on Answer

Problem Statement

You are given an array `weights` where `weights[i]` is the weight of the `i-th` package and an integer `days`.
You must ship all the packages within `days` days.
You can ship packages in the same order, and on each day you can ship as many packages as you like, as long as the total weight does not exceed the ship's capacity.

Find the **minimum ship capacity** needed to ship all the packages within `days`.

Example Question Context

This is a **classic binary search on answer space problem** — as ship capacity increases, required days decrease.
We need to find the smallest capacity such that all packages can be shipped within `days`.

Approach / तरीका (English + Hinglish)

English

We binary search on ship's capacity in the range `[max(weights), sum(weights)]`:

- For each mid-capacity, use `calculateDays()` to determine how many days it would take.
- If days ≤ given `days`, try a smaller capacity (`high = mid-1`).
- Else, try a larger capacity (`low = mid+1`).

When the loop ends, `ans` is the smallest valid ship capacity.

Hinglish

Hum ship ki **capacity** par binary search karte hain (`[max(weights), sum(weights)]`):

- Har mid-capacity ke liye `calculateDays()` se check karte hain kitne din lagenge.
- Agar din ≤ given `days`, to chhoti capacity try karte hain (`high = mid-1`).
- Warna badi capacity try karte hain (`low = mid+1`).

Loop ke baad `ans` me minimum valid capacity hoti hai.

Patterns / Concepts

Binary Search on Answer
Greedy Check
Search Space Reduction

Time Complexity

O(n * log(sum(weights) - max(weights))) — binary search over capacities and linear scan at each check.

Space Complexity

O(1) — constant space.

Example

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

Output:
`15` — minimum ship capacity to deliver all in 5 days.

Edge Cases

Edge Case 1:
Input: `weights = [1,1,1,1]`, `days = 4`
Output: `1`

Edge Case 2:
Input: `weights = [10,50,100,100]`, `days = 2`
Output: `150`

Code with Comments

```python
from typing import List

class Solution:
    def shipWithinDays(self, weights: List[int], days: int) -> int:
        # Helper: calculate required days for given capacity
        def calculateDays(weights, capacity):
            requiredDays = 1
            load = 0
            for weight in weights:
                if load + weight > capacity:
                    requiredDays += 1
                    load = weight
                else:
                    load += weight
            return requiredDays

        low = max(weights)  # ship must carry at least the heaviest package
        high = sum(weights)  # worst case: all in one day
        ans = high
```

```
    # Binary search on capacity
    while low <= high:
        mid = (low + high) // 2
        if calculateDays(weights, mid) <= days:
            ans = mid  # try to minimize
            high = mid - 1
        else:
            low = mid + 1

    return ans
```

Find K-th Missing Positive Number — Binary Search

Problem Statement

You are given a **sorted array** `arr` of unique positive integers and an integer `k`.
You need to find the **k-th missing positive number**.

Example Question Context

Since the array is strictly increasing and contains positive numbers, the difference between `arr[i]` and `(i+1)` gives the count of missing numbers **before index i**.
We need to find the smallest index where missing numbers ≥ `k` and calculate the result accordingly.

Approach / तरीका (English + Hinglish)

English

We perform binary search on indices `[0, n-1]`:

- For each `mid`, compute how many numbers are missing before `arr[mid]`:
  `missing = arr[mid] - (mid+1)`
- If `missing < k`, then look in the right half (`low = mid+1`) because we need more missing numbers.
- If `missing ≥ k`, then look in the left half (`high = mid-1`).
- Finally, `high+1+k` (or equivalently `low+k`) gives the k-th missing positive number.

Hinglish

Hum binary search karte hain index par (`[0, n-1]`):

- Har `mid` par missing numbers nikalte hain:
  `missing = arr[mid] - (mid+1)`
- Agar `missing < k`, to right half me search karte hain (`low = mid+1`).
- Agar `missing ≥ k`, to left half me search karte hain (`high = mid-1`).
- End me `high+1+k` (ya `low+k`) answer deta hai.

Patterns / Concepts

Binary Search
Search Space Reduction
Counting Missing Elements

Time Complexity

O(log n) — binary search.

Space Complexity

O(1) — constant space.

Example

Input:
`arr = [2,3,4,7,11]`, `k = 5`

Output:
`9` — the 5-th missing positive number is 9.

Edge Cases

Edge Case 1:
Input: `arr = [1,2,3,4]`, `k = 2`
Output: `6`

Edge Case 2:
Input: `arr = [5,6,7,8]`, `k = 1`
Output: `1`

Code with Comments

```
def findKthPositive(arr, k):
    n = len(arr)
    low = 0
    high = n - 1

    # Binary search for smallest index where missing numbers ≥ k
    while low <= high:
        mid = (low + high) // 2
        missing = arr[mid] - (mid + 1)

        if missing < k:
            low = mid + 1  # need more missing numbers
        else:
            high = mid - 1  # too many, go left

    # k-th missing positive is before arr[low]
    return low + k
```

Aggressive Cows — Maximum Minimum Distance (Binary Search)

Problem Statement

You are given an array `arr` where each element represents a stall position along a straight line, and an integer `cows` representing the number of cows to place.
Place the cows in the stalls such that the **minimum distance between any two cows is maximized**.
Return that maximum minimum distance.

Example Question Context

This is a classic **binary search on answer space** problem.
We want to maximize the minimum distance between cows, so we check (using greedy) whether it is possible to place all cows with at least `dist` distance apart.

Approach / तरीका (English + Hinglish)

English

We binary search on possible minimum distances (`dist`) in range `[0, max(arr)]`:

- Sort the stalls.
- At each step, assume `mid` as the minimum distance.
- Use `canWePlaceCows()` to check if it is possible to place all cows with at least `mid` distance:
  - Place first cow at first stall.
  - For each stall, if the distance from the last placed cow ≥ `mid`, place next cow.
  - If at least `cows` cows are placed, return `True`.
- If possible, try to increase `dist` (`low = mid+1`).
- Else, decrease `dist` (`high = mid-1`).

Finally, `high` holds the maximum minimum distance.

Hinglish

Hum possible minimum distance (`dist`) par binary search karte hain (`[0, max(arr)]`):

- Pehle stalls ko sort karte hain.
- Har step pe `mid` ko distance assume karke check karte hain:
  - Pehli cow ko pehli stall par rakho.
  - Agli stall tab chunni jab `last placed cow` se distance ≥ `mid` ho.
  - Agar `cows` place ho jayein, to `True` return karo.
- Agar possible ho, to aur bada distance try karo (`low = mid+1`), warna kam karo (`high = mid-1`).

Loop ke baad `high` me answer hota hai.

Patterns / Concepts

Binary Search on Answer
Greedy Placement
Search Space Reduction

Time Complexity

O(n * log(max(arr))) — binary search + linear check each step.

Space Complexity

O(1) — constant space.

Example

Input:
`arr = [0,3,4,7,10]`, `cows = 4`

Output:
`3` — maximum minimum distance possible is 3.

Edge Cases

Edge Case 1:
Input: `arr = [1,2,3,4,5]`, `cows = 2`
Output: `4`

Edge Case 2:
Input: `arr = [0,10]`, `cows = 2`
Output: `10`

Code with Comments

```
def canWePlaceCows(arr, dist, cows):
    countCows = 1
    last = arr[0]
    for num in arr:
        if num - last >= dist:
            countCows += 1
            last = num
        if countCows >= cows:
            return True
    return False

def aggresiveCows(arr, cows):
    arr.sort()
    low = 0
    high = max(arr)

    while low <= high:
        mid = (low + high) // 2
        if canWePlaceCows(arr, mid, cows):
```

```
            low = mid + 1  # try for bigger distance
        else:
            high = mid - 1  # reduce distance

    return high

# Example call
arr = [0,3,4,7,10]
cows = 4
print(aggresiveCows(arr, cows))  # Output: 3
```

Book Allocation Problem — Binary Search on Answer

Problem Statement

You are given an array `arr` where `arr[i]` is the number of pages in the `i-th` book and an integer `m` representing the number of students.
Allocate books to students such that:

- Each student gets at least one book.
- Each book is allocated to exactly one student.
- Books are allocated in **contiguous order**.
- The **maximum number of pages assigned to a student is minimized**.

Return the minimum possible maximum pages a student has to read.

Example Question Context

This is a **binary search on answer space** problem — as the maximum number of pages per student increases, the number of students required decreases.
We need to find the smallest maximum number of pages such that all books can be allocated to `m` students.

Approach / तरीका (English + Hinglish)

English

We binary search over the possible maximum pages per student in range `[max(arr), sum(arr)]`:

- At each step, assume `mid` as the maximum pages.
- Use `countStudents()` to count how many students are needed if no student reads more than `mid` pages.
- If required students > `m`, increase `mid` (`low = mid+1`).
- Otherwise, try a smaller `mid` (`high = mid-1`).

When the loop ends, `low` holds the minimum possible maximum pages.

Hinglish

Hum maximum pages per student par binary search karte hain (`[max(arr), sum(arr)]`):

- Har step pe `mid` ko assume karke check karte hain ki `m` students me distribute ho sakte hain ya nahi.
- `countStudents()` se students count karte hain.
- Agar students > `m`, to zyada pages allow karte hain (`low = mid+1`), warna kam karte hain (`high = mid-1`).

Loop ke baad `low` me minimum maximum pages hota hai.

Patterns / Concepts

Binary Search on Answer
Greedy Check
Search Space Reduction

Time Complexity

O(n * log(sum(arr) - max(arr))) — binary search + linear check each step.

Space Complexity

O(1) — constant space.

Example

Input:
`arr = [25,46,28,49,24]`, `m = 4`

Output:
`46` — minimum of the maximum pages allocated per student is 46.

Edge Cases

Edge Case 1:
Input: `arr = [10,20,30,40]`, `m = 2`
Output: `60`

Edge Case 2:
Input: `arr = [5,10,15]`, `m = 4`
Output: `-1` (not enough books)

Code with Comments

```
# Count number of students needed if no student reads more than 'pages' pages
def countStudents(arr, pages):
    student = 1
    pageCount = 0
    for num in arr:
        if pageCount + num <= pages:
            pageCount += num
        else:
            student += 1
```

```
            pageCount = num
    return student

# Main function: book allocation
def booksAllocation(arr, m, n):
    if m > n:
        return -1  # not enough books

    low = max(arr)  # each student must at least read one biggest book
    high = sum(arr)

    while low <= high:
        mid = (low + high) // 2
        students = countStudents(arr, mid)
        if students > m:
            low = mid + 1  # need to allow more pages
        else:
            high = mid - 1  # try to minimize max pages

    return low

# Example call
arr = [25, 46, 28, 49, 24]
m = 4
n = len(arr)
print(booksAllocation(arr, m, n))  # Output: 46
```

Split Array Largest Sum — Binary Search on Answer

Problem Statement

You are given an array `nums` of non-negative integers and an integer `k`.
Split the array into `k` or fewer non-empty contiguous subarrays such that the largest sum among these subarrays is minimized.
Return this minimum largest sum.

Example Question Context

This is a **binary search on answer space** problem — as the maximum allowed subarray sum increases, the number of subarrays required decreases.
We aim to find the smallest maximum sum such that we can split the array into at most `k` parts.

Approach / तरीका (English + Hinglish)

English

We binary search over possible maximum sums in the range `[max(nums), sum(nums)]`:

- For each `mid`, use `calculateSum()` to count how many subarrays are needed if no subarray has sum > `mid`.

- If more than `k` subarrays are needed, we need to allow larger sums (`low = mid+1`).
- Else, try to minimize the sum (`high = mid-1`).

When the loop ends, `low` holds the minimum largest sum.

Hinglish

Hum possible maximum sum par binary search karte hain (`[max(nums), sum(nums)]`):

- Har `mid` par `calculateSum()` se count karte hain ki kitne subarrays banenge agar kisi ka sum `mid` se zyada na ho.
- Agar subarrays > `k`, to zyada sum allow karte hain (`low = mid+1`), warna kam karte hain (`high = mid-1`).

Loop ke baad `low` me minimum largest sum hota hai.

Patterns / Concepts

Binary Search on Answer
Greedy Check
Search Space Reduction

Time Complexity

O(n * log(sum(nums) - max(nums))) — binary search + linear scan each check.

Space Complexity

O(1) — constant space.

Example

Input:
`nums = [7,2,5,10,8]`, `k = 2`

Output:
`18` — split as `[7,2,5]` and `[10,8]` with max sum 18.

Edge Cases

Edge Case 1:
Input: `nums = [1,2,3,4,5]`, `k = 2`
Output: `9`

Edge Case 2:
Input: `nums = [1,4,4]`, `k = 3`
Output: `4`

Code with Comments

```python
from typing import List

class Solution:
    def splitArray(self, nums: List[int], k: int) -> int:
        # Helper: count subarrays needed if no subarray > mid
        def calculateSum(nums, mid):
            subArray = 1
            numberSum = 0
            for num in nums:
                if numberSum + num <= mid:
                    numberSum += num
                else:
                    subArray += 1
                    numberSum = num
            return subArray

        low = max(nums)  # min possible max sum
        high = sum(nums)  # max possible max sum

        while low <= high:
            mid = (low + high) // 2
            if calculateSum(nums, mid) > k:
                low = mid + 1  # need larger sums
            else:
                high = mid - 1  # try to minimize

        return low
```

Painter's Partition Problem — Binary Search on Answer

Problem Statement

You are given an array `boards` where `boards[i]` represents the length of the `i-th` board, and an integer `k` representing the number of painters.
You need to paint all the boards such that:

- Each board is painted by exactly one painter.
- Each painter paints **contiguous** boards.
- The goal is to minimize the maximum time taken by any painter.

Return this minimum possible maximum time.

Example Question Context

This is a **binary search on answer space** problem — as the maximum time per painter increases, fewer painters are needed.
We aim to find the smallest maximum time such that all boards can be painted by `k` painters.

Approach / तरीका (English + Hinglish)

English

We binary search over the maximum time per painter in range `[max(boards), sum(boards)]`:

- For each `mid`, use `countPainters()` to check how many painters are required if no painter paints more than `mid` time.
- If more than `k` painters are needed, increase `mid` (`low = mid+1`).
- Else, try a smaller `mid` (`high = mid-1`).

Finally, `low` holds the minimum maximum time.

Hinglish

Hum maximum time per painter par binary search karte hain (`[max(boards), sum(boards)]`):

- Har `mid` pe `countPainters()` se check karte hain ki kitne painters chahiye.
- Agar painters > `k`, to zyada time allow karte hain (`low = mid+1`), warna kam karte hain (`high = mid-1`).

Loop ke baad `low` me answer hota hai.

Patterns / Concepts

Binary Search on Answer
Greedy Check
Search Space Reduction

Time Complexity

O(n * log(sum(boards) - max(boards))) — binary search + linear scan each step.

Space Complexity

O(1) — constant space.

Example

Input:
`boards = [10,20,30,40]`, `k = 2`

Output:
`60` — split as `[10,20,30]` and `[40]` with max time 60.

Edge Cases

Edge Case 1:
Input: `boards = [5,5,5,5]`, `k = 2`
Output: `10`

Edge Case 2:
Input: `boards = [10,20,30]`, `k = 1`

Output: 60

Code with Comments

```
# Count number of painters needed if no painter paints more than 'time'
def countPainters(boards, time):
    painters = 1
    boardsPainter = 0
    for board in boards:
        if boardsPainter + board <= time:
            boardsPainter += board
        else:
            painters += 1
            boardsPainter = board
    return painters

# Main function: Painter's Partition
def findLargestMinDistance(boards, k):
    low = max(boards)  # each painter must paint at least the largest board
    high = sum(boards)  # one painter paints all

    while low <= high:
        mid = (low + high) // 2
        painters = countPainters(boards, mid)
        if painters > k:
            low = mid + 1  # need more time per painter
        else:
            high = mid - 1  # try to minimize

    return low

# Example call
boards = [10, 20, 30, 40]
k = 2
ans = findLargestMinDistance(boards, k)
print("The answer is:", ans)  # Output: 60
```

Median of Two Sorted Arrays — Binary Search

Problem Statement

You are given two sorted arrays a and b.
Find the median of the two sorted arrays combined, in **O(log(min(n1, n2)))** time.

Example Question Context

This is a classic **binary search on partition point** problem.
We need to partition both arrays such that the left half contains exactly half of the elements and the largest element in the left half ≤ smallest element in the right half.

Approach / तरीका (English + Hinglish)

English

We binary search on the smaller array to find a valid partition:

- At each step, partition array `a` at index `mid1` and array `b` at `left - mid1`.
- Compare `l1, r1` (left/right of `a`) and `l2, r2` (left/right of `b`).
- If `l1 ≤ r2` and `l2 ≤ r1`, a valid partition is found:
  - If total number of elements is even: median is average of max(left) and min(right).
  - If odd: median is max of left.
- If `l1 > r2`, move partition in `a` left (`high = mid1 -1`), else right (`low = mid1+1`).

Hinglish

Hum chhoti array par binary search karke partition point nikalte hain:

- Har step par `a` ko `mid1` pe aur `b` ko `left-mid1` pe todte hain.
- `l1 ≤ r2` aur `l2 ≤ r1` ho to valid partition mil gaya:
  - Agar elements even ho to median = (max(left) + min(right))/2
  - Agar odd ho to median = max(left)
- Agar `l1 > r2` ho to left me jao (`high = mid1-1`), warna right me jao (`low = mid1+1`).

Patterns / Concepts

Binary Search on Partition
Divide and Conquer
Median Property

Time Complexity

O(log(min(n1, n2)))

Space Complexity

O(1) — constant extra space.

Example

Input:
`a = [1,4,7,10,12]`, `b = [2,3,6,15]`

Output:
`6.0`

Combined array: `[1,2,3,4,6,7,10,12,15]`
Median is `6`.

Edge Cases

Edge Case 1:
Input: `a = []`, `b = [1,2,3]`
Output: `2.0`

Edge Case 2:
Input: `a = [1]`, `b = [2,3,4]`
Output: `2.5`

Code with Comments

```
import sys

def medianOfSortedArray(a, b):
    n1 = len(a)
    n2 = len(b)

    # Ensure a is smaller
    if n1 > n2:
        return medianOfSortedArray(b, a)

    n = n1 + n2
    left = (n + 1) // 2
    low = 0
    high = n1

    while low <= high:
        mid1 = (low + high) // 2
        mid2 = left - mid1

        l1 = -sys.maxsize - 1 if mid1 == 0 else a[mid1 - 1]
        l2 = -sys.maxsize - 1 if mid2 == 0 else b[mid2 - 1]
        r1 = sys.maxsize if mid1 == n1 else a[mid1]
        r2 = sys.maxsize if mid2 == n2 else b[mid2]

        # Valid partition found
        if l1 <= r2 and l2 <= r1:
            if n % 2 == 0:
                return (max(l1, l2) + min(r1, r2)) / 2
            else:
                return max(l1, l2)
        elif l1 > r2:
            high = mid1 - 1
        else:
            low = mid1 + 1
    return 0

# Example call
a = [1, 4, 7, 10, 12]
b = [2, 3, 6, 15]
print("The median of two sorted arrays is {:.1f}".format(medianOfSortedArray(a,
b)))  # Output: 6.0
```

k-th Element of Two Sorted Arrays — Binary Search

Problem Statement

You are given two sorted arrays `a` and `b` of sizes `m` and `n`, and an integer `k`.
Find the **k-th smallest element** in the combined sorted array of `a` and `b`, in **O(log(min(m, n)))** time.

Example Question Context

This is a **binary search on partition point** problem, similar to finding median.
We find a partition such that exactly `k` elements are on the left half of the merged array.

Approach / तरीका (English + Hinglish)

English

We binary search on the smaller array to find a valid partition:

- Partition `a` at index `mid1` and `b` at index `k-mid1`.
- Calculate `l1, r1` (left/right of `a`) and `l2, r2` (left/right of `b`).
- If `l1 ≤ r2` and `l2 ≤ r1`, valid partition is found. Return `max(l1, l2)`.
- If `l1 > r2`, move partition in `a` left (`high = mid1-1`), else right (`low = mid1+1`).

Hinglish

Hum chhoti array par binary search karke partition point nikalte hain:

- `a` ko `mid1` aur `b` ko `k-mid1` pe todte hain.
- `l1 ≤ r2` aur `l2 ≤ r1` ho to valid partition mil gaya: answer `max(l1,l2)`.
- Agar `l1 > r2` ho to left me jao (`high = mid1-1`), warna right me jao (`low = mid1+1`).

Patterns / Concepts

Binary Search on Partition
Divide and Conquer
k-th Order Statistic

Time Complexity

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

Space Complexity

O(1) — constant space.

Example

Input:
`a = [2,3,6,7,9]`, `b = [1,4,8,10]`, `k = 5`

Output:
`6`

Merged array: `[1,2,3,4,6,7,8,9,10]`
5-th element is `6`.

Edge Cases

Edge Case 1:
Input: `a = []`, `b = [1,2,3,4,5]`, `k = 3`
Output: `3`

Edge Case 2:
Input: `a = [1]`, `b = [2,3,4]`, `k = 2`
Output: `2`

Code with Comments

```
def kthElement(a, b, m, n, k):
    # Ensure a is the smaller array
    if m > n:
        return kthElement(b, a, n, m, k)

    left = k

    low = max(0, k - n)
    high = min(k, m)

    while low <= high:
        mid1 = (low + high) // 2
        mid2 = left - mid1

        # left and right elements around the partitions
        l1 = float('-inf') if mid1 == 0 else a[mid1 - 1]
        l2 = float('-inf') if mid2 == 0 else b[mid2 - 1]
        r1 = float('inf') if mid1 == m else a[mid1]
        r2 = float('inf') if mid2 == n else b[mid2]

        if l1 <= r2 and l2 <= r1:
            return max(l1, l2)
        elif l1 > r2:
            high = mid1 - 1
        else:
            low = mid1 + 1

    return 0  # dummy

# Example call
a = [2, 3, 6, 7, 9]
b = [1, 4, 8, 10]
```

```
print("The k-th element of two sorted arrays is:", kthElement(a, b, len(a),
len(b), 5))  # Output: 6
```

Row with Maximum 1's in a Binary Matrix — Binary Search

Problem Statement

You are given a binary matrix of size `n x m` (each row sorted).
Find the **index of the row with the maximum number of 1's**.
If multiple rows have the same number of 1's, return the smallest index. If no 1's are present, return `-1`.

Example Question Context

Each row is sorted in non-decreasing order (`0`s followed by `1`s).
We can use binary search to efficiently find the first occurrence of `1` in each row, and compute the count of `1`s.

Approach / तरीका (English + Hinglish)

English

We iterate through each row:

- Use `lowerBound()` (binary search) to find the first index of `1` in the row.
- Count of `1`s = `m - lowerBound()`.
- Keep track of the row with the highest count of `1`s.

Hinglish

Hum har row par iterate karte hain:

- Binary search (`lowerBound()`) se pehla `1` ka index nikalte hain.
- Count of `1`s = `m - lowerBound()`.
- Sabse zyada `1`s wali row ka index track karte hain.

Patterns / Concepts

Binary Search per Row
Matrix Traversal
Greedy Maximum Tracking

Time Complexity

O(n log m) — for each row, binary search.

Space Complexity

O(1) — constant extra space.

Example

Input:
`matrix = [[1,1,1],[0,0,1],[0,0,0]]`, `n = 3`, `m = 3`

Output:
`0` — first row has 3 ones.

Edge Cases

Edge Case 1:
Input: `matrix = [[0,0],[0,0]]`
Output: `-1` — no 1's.

Edge Case 2:
Input: `matrix = [[0,1],[0,1]]`
Output: `0` — smallest index row.

Code with Comments

```
# Binary search to find first index of 1 in sorted row
def lowerBound(arr, n, x):
    low = 0
    high = n - 1
    ans = n
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] >= x:
            ans = mid
            high = mid - 1
        else:
            low = mid + 1
    return ans

# Main function to find row with max 1s
def rowWithMax1s(matrix, n, m):
    cnt_max = 0
    index = -1
    for i in range(n):
        cnt_ones = m - lowerBound(matrix[i], m, 1)
        if cnt_ones > cnt_max:
            cnt_max = cnt_ones
            index = i
    return index

# Example call
matrix = [[1, 1, 1], [0, 0, 1], [0, 0, 0]]
n = 3
m = 3
print("The row with maximum no. of 1's is:", rowWithMax1s(matrix, n, m))  #
Output: 0
```

Search in a 2D Sorted Matrix — Optimized Search

Problem Statement

You are given a `n x m` matrix where:

- Each row is sorted in ascending order from left to right.
- Each column is sorted in ascending order from top to bottom.

Write a function to check if a given `target` exists in the matrix.

Example Question Context

We take advantage of the fact that:

- Moving left decreases value.
- Moving down increases value.

We start from the **top-right corner** and eliminate rows or columns based on comparison.

Approach / तरीका (English + Hinglish)

English

We start at `(0, m-1)` (top-right):

- If `matrix[row][col] == target`, return `True`.
- If `matrix[row][col] < target`, move down (`row += 1`).
- If `matrix[row][col] > target`, move left (`col -= 1`).

Hinglish

Hum `(0, m-1)` (top-right) se shuru karte hain:

- Agar `matrix[row][col] == target`, to `True` return karte hain.
- Agar `matrix[row][col] < target`, to neeche jao (`row +=1`).
- Agar `matrix[row][col] > target`, to left jao (`col -=1`).

Patterns / Concepts

Matrix Traversal
Search Space Reduction
Greedy Elimination

Time Complexity

O(n + m) — at most `n + m` steps.

Space Complexity

O(1) — constant space.

Example

Input:
`matrix = [[1,4,7,11,15], [2,5,8,12,19], [3,6,9,16,22], [10,13,14,17,24], [18,21,23,26,30]]`,
`target = 8`

Output:
`True`

Edge Cases

Edge Case 1:
Input: `target = 100` (greater than all)
Output: `False`

Edge Case 2:
Input: `target = 1` (smallest)
Output: `True`

Code with Comments

```
# Function to search for target in sorted matrix
def searchElement(matrix, target):
    n = len(matrix)
    m = len(matrix[0])
    row = 0
    col = m - 1

    # Start from top-right corner
    while row < n and col >= 0:
        if matrix[row][col] == target:
            return True
        elif matrix[row][col] < target:
            row += 1  # move down
        else:
            col -= 1  # move left
    return False

# Example call
matrix = [
    [1, 4, 7, 11, 15],
    [2, 5, 8, 12, 19],
    [3, 6, 9, 16, 22],
    [10, 13, 14, 17, 24],
    [18, 21, 23, 26, 30]
]
```

```
result = searchElement(matrix, 8)
print(result)  # Output: True
```

Search a 2D Matrix II — Optimized Search

Problem Statement

You are given a `n x m` matrix where:

- Each row is sorted in ascending order from left to right.
- Each column is sorted in ascending order from top to bottom.

Approach

English

We start at the **top-right corner** `(0, m-1)`:

- If `matrix[row][col] == target`, return `True`.
- If `matrix[row][col] < target`, move down (`row += 1`) — because all elements in current row to the left are smaller.
- If `matrix[row][col] > target`, move left (`col -= 1`) — because all elements below in current column are larger.

This eliminates one row or one column at each step.

Hinglish

Hum `(0, m-1)` (top-right) se start karte hain:

- Agar `matrix[row][col] == target`, to `True` return karte hain.
- Agar `matrix[row][col] < target`, to neeche jao (`row += 1`).
- Agar `matrix[row][col] > target`, to left jao (`col -= 1`).

Patterns / Concepts

Matrix Traversal
Search Space Reduction
Greedy Elimination

Time Complexity

O(n + m) — each step removes a row or column.

Space Complexity

O(1) — constant space.

Example

Input:
`matrix = [[1,4,7,11,15], [2,5,8,12,19], [3,6,9,16,22], [10,13,14,17,24], [18,21,23,26,30]]`,
`target = 5`

Output:
`True`

Edge Cases

Edge Case 1:
Input: `target = 100` (greater than all)
Output: `False`

Edge Case 2:
Input: `target = 1` (smallest)
Output: `True`

Code with Comments

```
from typing import List

class Solution:
    def searchMatrix(self, matrix: List[List[int]], target: int) -> bool:
        n = len(matrix)
        m = len(matrix[0])
        row = 0
        col = m - 1

        # Start from top-right corner
        while row < n and col >= 0:
            if matrix[row][col] == target:
                return True
            elif matrix[row][col] < target:
                row += 1  # move down
            else:
                col -= 1  # move left
        return False

# Example usage
matrix = [
    [1, 4, 7, 11, 15],
    [2, 5, 8, 12, 19],
    [3, 6, 9, 16, 22],
    [10, 13, 14, 17, 24],
    [18, 21, 23, 26, 30]
]
target = 5
print(Solution().searchMatrix(matrix, target))  # Output: True
```

Find Peak Element in a 2D Grid — Binary Search on Columns

Problem Statement

You are given a `n x m` matrix. A **peak element** is one which is strictly greater than its left and right neighbors (if they exist).
Return the position `[row, col]` of any peak element.

A peak is defined as:
`mat[row][col] > mat[row][col-1]` and `mat[row][col] > mat[row][col+1]`

Example Question Context

This is an extension of the 1D peak finding problem.
We apply binary search **on columns**, finding the maximum in the middle column, and then decide which half to search next.

Approach / तरीका (English + Hinglish)

English

We perform binary search on columns:

- In the middle column, find the row index with maximum value.
- Compare this value with its left and right neighbors.
  - If it's greater than both, it's a peak.
  - If left neighbor is greater, move to the left half.
  - If right neighbor is greater, move to the right half.

Hinglish

Hum columns par binary search karte hain:

- Middle column me max element ka row index nikalte hain.
- Us element ko left aur right neighbors se compare karte hain.
  - Agar dono se bada ho to ye peak hai.
  - Agar left bada ho to left half me search karo.
  - Agar right bada ho to right half me search karo.

Patterns / Concepts

Binary Search on 2D Matrix
Greedy Elimination of Half Search Space

Time Complexity

O(n log m) — for each column search, we scan one full column.

Space Complexity

O(1) — constant extra space.

Example

Input:
`mat = [[10,8,10,10], [14,13,12,11], [15,9,11,21], [16,17,19,20]]`

Output:
`[2,0]` or another valid peak position.

Edge Cases

Edge Case 1:
Input: Single row matrix
Output: Position of max element.

Edge Case 2:
Input: All elements equal
Output: Any position.

Code with Comments

```
from typing import List

# Helper to find row index of max element in a column
def findMaxIndex(mat, n, m, col):
    maxValue = -1
    index = -1
    for i in range(n):
        if mat[i][col] > maxValue:
            maxValue = mat[i][col]
            index = i
    return index

# Main function to find peak
def findPeakGrid(mat: List[List[int]]) -> List[int]:
    n = len(mat)
    m = len(mat[0])
    low = 0
    high = m - 1

    while low <= high:
        mid = (low + high) // 2
        maxRowIndex = findMaxIndex(mat, n, m, mid)

        left = mat[maxRowIndex][mid - 1] if mid > 0 else -1
        right = mat[maxRowIndex][mid + 1] if mid < m - 1 else -1

        if mat[maxRowIndex][mid] > left and mat[maxRowIndex][mid] > right:
```

```
            return [maxRowIndex, mid]
        elif mat[maxRowIndex][mid] < left:
            high = mid - 1
        else:
            low = mid + 1

    return [-1, -1]  # fallback if no peak found

# Example usage
mat = [
    [10,8,10,10],
    [14,13,12,11],
    [15,9,11,21],
    [16,17,19,20]
]
print(findPeakGrid(mat))  # Example Output: [2, 0]
```

Find Median in a Row-wise Sorted Matrix — Binary Search on Answer

Problem Statement

You are given a `m x n` matrix, where each row is sorted in ascending order.
Find the median of the matrix.

Example Question Context

Since the matrix is not fully sorted, but each row is sorted, we cannot flatten it and sort in `O(nm log(nm))`.
Instead, we use binary search on the value range.

Approach / तरीका (English + Hinglish)

English

We know:

- The smallest possible element is `min(matrix[i][0])`
- The largest possible element is `max(matrix[i][n-1])` We perform binary search on this range. For a candidate `mid`, count how many elements in the matrix are `<= mid`.
- If count ≤ required (half of total), move right.
- Else, move left.

Hinglish

Sabse chhota element har row ke first element me se minimum hoga.
Sabse bada element har row ke last element me se maximum hoga.
Hum unke beech value range par binary search karte hain. Mid leke dekhte hain kitne elements `<= mid` hain.

- Agar count ≤ required, to right me jao.

- Warna left me jao.

Patterns / Concepts

Binary Search on Answer
Matrix Row-wise Sorted Property
Upper Bound in Row

Time Complexity

O(32 * m * log n) — 32 for value range (since integers ≤ 2^{32}), and `log n` per row binary search.

Space Complexity

O(1) — constant extra space.

Example

Input:
`matrix = [[1,2,3,4,5], [8,9,11,12,13], [21,23,25,27,29]]`

Output:
`11`

Edge Cases

Edge Case 1:
Matrix of size `1x1` → Median is the only element.

Edge Case 2:
All elements are equal → Median is the same element.

Code with Comments

```
def upperBound(arr, x, n):
    low = 0
    high = n - 1
    ans = n
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] > x:
            ans = mid
            high = mid - 1
        else:
            low = mid + 1
    return ans

# Count how many elements ≤ x in matrix
def countSmallEqual(matrix, m, n, x):
    cnt = 0
    for i in range(m):
```

```
        cnt += upperBound(matrix[i], x, n)
    return cnt

# Main function to find median
def median(matrix, m, n):
    low = float('inf')
    high = float('-inf')
    for i in range(m):
        low = min(low, matrix[i][0])
        high = max(high, matrix[i][n - 1])

    req = (n * m) // 2
    while low <= high:
        mid = (low + high) // 2
        smallEqual = countSmallEqual(matrix, m, n, mid)
        if smallEqual <= req:
            low = mid + 1
        else:
            high = mid - 1
    return low

# Example usage
if __name__ == "__main__":
    matrix = [
        [1, 2, 3, 4, 5],
        [8, 9, 11, 12, 13],
        [21, 23, 25, 27, 29]
    ]
    m = len(matrix)
    n = len(matrix[0])
    ans = median(matrix, m, n)
    print("The median element is:", ans)  # Output: 11
```