4. Median of Two Sorted Arrays

Given two sorted arrays `nums1` and `nums2` of size `m` and `n` respectively, return **the median** of the two sorted arrays.

The overall run time complexity should be `O(log (m+n))`.

Example 1:

```
Input: nums1 = [1,3], nums2 = [2]
Output: 2.00000
Explanation: merged array = [1,2,3] and median is 2.
```

Example 2:

```
Input: nums1 = [1,2], nums2 = [3,4]
Output: 2.50000
Explanation: merged array = [1,2,3,4] and median is (2 + 3) / 2 = 2.5.
```

Constraints:

- `nums1.length == m`
- `nums2.length == n`
- `0 <= m <= 1000`
- `0 <= n <= 1000`
- `1 <= m + n <= 2000`
- -10^6 <= `nums1[i], nums2[i]` <= 10^6

```
class Solution:
    def findMedianSortedArrays(self, nums1: List[int], nums2: List[int]) -> float:
        ans = sorted(nums1+nums2)
        if len(ans) %2 ==1:
            return  ans[len(ans)//2]
        else:
            return (ans[(len(ans)//2) -1] + ans[len(ans)//2])/2
```

33. Search in Rotated Sorted Array

There is an integer array `nums` sorted in ascending order (with **distinct** values).

Prior to being passed to your function, `nums` is **possibly rotated** at an unknown pivot index `k` (`1 <= k < nums.length`) such that the resulting array is `[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]` (**0-indexed**). For example, `[0,1,2,4,5,6,7]` might be rotated at pivot index `3` and become `[4,5,6,7,0,1,2]`.

Given the array `nums` **after** the possible rotation and an integer `target`, return *the index of* `target` *if it is in* `nums`*, or* `-1` *if it is not in* `nums`.

You must write an algorithm with `O(log n)` runtime complexity.

Example 1:

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

Example 2:

```
Input: nums = [4,5,6,7,0,1,2], target = 3
Output: -1
```

Example 3:

```
Input: nums = [1], target = 0
Output: -1
```

Constraints:

- `1 <= nums.length <= 5000`
- -10^4 `<= nums[i] <=` 10^4
- All values of `nums` are **unique**.
- `nums` is an ascending array that is possibly rotated.
- -10^4 `<= target <=` 10^4

---

https://www.youtube.com/watch?v=U8XENwh8Oy8 (https://www.youtube.com/watch?v=U8XENwh8Oy8)

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        left,right  = 0 , len(nums)-1

        while left <= right:
            mid = (left+right)//2
            if nums[mid] == target:
                return mid
            # left sorted portion
            elif nums[left] <= nums[mid]:
                if target > nums[mid] or target < nums[left] :
                    left = mid+1
                else:
                    right = mid-1
            # Right sorted arr
            else:
                if target < nums[mid] or nums[right] < target:
                    right = mid - 1
                else:
                    left = mid + 1
        return -1
```

34. Find First and Last Position of Element in Sorted Array

Given an array of integers `nums` sorted in non-decreasing order, find the starting and ending position of a given `target` value.

If `target` is not found in the array, return `[-1, -1]`.

You must write an algorithm with `O(log n)` runtime complexity.

Example 1:

```
Input: nums = [5,7,7,8,8,10], target = 8
Output: [3,4]
```

Example 2:

```
Input: nums = [5,7,7,8,8,10], target = 6
Output: [-1,-1]
```

Example 3:

```
Input: nums = [], target = 0
Output: [-1,-1]
```

Constraints:

- `0 <= nums.length <= 10^5`
- `-10^9 <= nums[i] <= 10^9`
- `nums` is a non-decreasing array.
- `-10^9 <= target <= 10^9`

```
class Solution:
    def searchRange(self, nums: List[int], target: int) -> List[int]:
        first = last = -1
        i,j = 0,len(nums)-1
        while i <= j:
            mid  = (i+j)//2
            if nums[mid] == target:
                i = mid
                j = mid
                while i>=0 and nums[i] == target:
                    i-=1
                while j<= len(nums)-1 and nums[j] == target:
                    j+=1
                return i+1,j-1
            elif nums[mid] < target:
                i = mid+1
            else:
                j = mid-1
        return [-1,-1]
```

35. Search Insert Position

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You must write an algorithm with `O(log n)` runtime complexity.

Example 1:

```
Input: nums = [1,3,5,6], target = 5
Output: 2
```

Example 2:

```
Input: nums = [1,3,5,6], target = 2
Output: 1
```

Example 3:

```
Input: nums = [1,3,5,6], target = 7
Output: 4
```

Constraints:

- `1 <= nums.length <= 10`^{4}
- `-10`^{4} `<= nums[i] <= 10`^{4}
- `nums` contains **distinct** values sorted in **ascending** order.
- `-10`^{4} `<= target <= 10`^{4}

```
class Solution:
    def searchInsert(self, nums: List[int], target: int) -> int:
        i = 0
        j = len(nums)-1
        while i<=j:
            mid = (i+j)//2
            if nums[mid] == target:
                return mid
            elif nums[mid] < target:
                i = mid+1
            else:
                j = mid-1
        return i
```

74. Search a 2D Matrix

You are given an `m x n` integer matrix `matrix` with the following two properties:

- Each row is sorted in non-decreasing order.
- The first integer of each row is greater than the last integer of the previous row.

Given an integer `target`, return `true` *if* `target` *is in* `matrix` *or* `false` *otherwise*.

You must write a solution in `O(log(m * n))` time complexity.

Example 1:

1	3	5	7
10	11	16	20
23	30	34	60

```
Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3
Output: true
```

Example 2:

1	3	5	7
10	11	16	20
23	30	34	60

```
Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 13
Output: false
```

Constraints:

- `m == matrix.length`

- `n == matrix[i].length`
- `1 <= m, n <= 100`
- -10^4 `<= matrix[i][j], target <=` 10^4

```
class Solution:
    def searchMatrix(self, matrix: List[List[int]], target: int) -> bool:
        left = 0
        right = len(matrix)-1
        def find_target(arr):
            left,right = 0,len(arr)-1
            while left <= right:
                mid = (left+right)//2
                if arr[mid]== target:return True
                elif arr[mid] < target:left = mid+1
                else:right = mid-1
            return False
        while left <= right:
            mid = (left+right)//2
            if matrix[mid][0] <= target and target <= matrix[mid][-1]:
                if find_target(matrix[mid]):return True
                return False
            elif matrix[mid][0]> target:right = mid-1
            elif matrix[mid][-1]< target:left = mid+1
        return False
```

81. Search in Rotated Sorted Array II

There is an integer array `nums` sorted in non-decreasing order (not necessarily with **distinct** values).

Before being passed to your function, `nums` is **rotated** at an unknown pivot index `k` (`0 <= k < nums.length`) such that the resulting array is `[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]` (**0-indexed**). For example, `[0,1,2,4,4,4,5,6,6,7]` might be rotated at pivot index `5` and become `[4,5,6,6,7,0,1,2,4,4]`.

Given the array `nums` **after** the rotation and an integer `target`, return `true` *if* `target` *is in* `nums`*, or* `false` *if it is not in* `nums`.

You must decrease the overall operation steps as much as possible.

Example 1:

```
Input: nums = [2,5,6,0,0,1,2], target = 0
Output: true
```

Example 2:

```
Input: nums = [2,5,6,0,0,1,2], target = 3
Output: false
```

Constraints:

- `1 <= nums.length <= 5000`
- -10^4 `<= nums[i] <=` 10^4
- `nums` is guaranteed to be rotated at some pivot.
- -10^4 `<= target <=` 10^4

Follow up: This problem is similar to Search in Rotated Sorted Array (/problems/search-in-rotated-sorted-array/description/), but `nums` may contain **duplicates**. Would this affect the runtime complexity? How and why?

```
class Solution:
    def search(self, nums: List[int], target: int) -> bool:
        left, right = 0, len(nums) - 1

        while left <= right:
            mid = (left + right) // 2

            if nums[mid] == target:
                return True

            if nums[mid] == nums[left]:
                left += 1
                continue

            if nums[left] <= nums[mid]:
                if nums[left] <= target < nums[mid]:
                    right = mid - 1
                else:
                    left = mid + 1
            else:
                if nums[mid] < target <= nums[right]:
                    left = mid + 1
                else:
                    right = mid - 1

        return False
```

153. Find Minimum in Rotated Sorted Array

Suppose an array of length `n` sorted in ascending order is **rotated** between `1` and `n` times. For example, the array `nums = [0,1,2,4,5,6,7]` might become:

- `[4,5,6,7,0,1,2]` if it was rotated `4` times.
- `[0,1,2,4,5,6,7]` if it was rotated `7` times.

Notice that **rotating** an array `[a[0], a[1], a[2], ..., a[n-1]]` 1 time results in the array `[a[n-1], a[0], a[1], a[2], ..., a[n-2]]`.

Given the sorted rotated array `nums` of **unique** elements, return *the minimum element of this array*.

You must write an algorithm that runs in `O(log n) time`.

Example 1:

```
Input: nums = [3,4,5,1,2]
Output: 1
Explanation: The original array was [1,2,3,4,5] rotated 3 times.
```

Example 2:

```
Input: nums = [4,5,6,7,0,1,2]
Output: 0
Explanation: The original array was [0,1,2,4,5,6,7] and it was rotated 4 times.
```

Example 3:

```
Input: nums = [11,13,15,17]
Output: 11
Explanation: The original array was [11,13,15,17] and it was rotated 4 times.
```

Constraints:

- `n == nums.length`
- `1 <= n <= 5000`
- `-5000 <= nums[i] <= 5000`
- All the integers of `nums` are **unique**.
- `nums` is sorted and rotated between `1` and `n` times.

```
class Solution:
    def findMin(self, nums: List[int]) -> int:
        left,right = 0,len(nums)-1
        while left < right:
            mid = (left+right)//2
            if nums[mid] > nums[right] :
                left = mid +1
            else:
                right = mid
        return nums[left]
```

162. Find Peak Element

A peak element is an element that is strictly greater than its neighbors.

Given a **0-indexed** integer array `nums`, find a peak element, and return its index. If the array contains multiple peaks, return the index to **any of the peaks**.

You may imagine that `nums[-1] = nums[n] = -∞`. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array.

You must write an algorithm that runs in `O(log n)` time.

Example 1:

```
Input: nums = [1,2,3,1]
Output: 2
Explanation: 3 is a peak element and your function should return the index number 2.
```

Example 2:

```
Input: nums = [1,2,1,3,5,6,4]
Output: 5
Explanation: Your function can return either index number 1 where the peak element is
```

Constraints:

- `1 <= nums.length <= 1000`
- -2^{31} <= nums[i] <= 2^{31} - 1
- `nums[i] != nums[i + 1]` for all valid `i`.

```
class Solution:
    def findPeakElement(self, nums: List[int]) -> int:
        peak = 0
        nums.append(-float("inf"))
        n = len(nums)
        if n < 3 :
            if n==1:return 0
            if nums[0] < nums[1] : return 1
            else:return 0
        for i in range(1,n-1):
            if nums[i-1] < nums[i] > nums[i+1]:
                return i
        return peak
```

540. Single Element in a Sorted Array

You are given a sorted array consisting of only integers where every element appears exactly twice, except for one element which appears exactly once.

Return *the single element that appears only once*.

Your solution must run in `O(log n)` time and `O(1)` space.

Example 1:

```
Input: nums = [1,1,2,3,3,4,4,8,8]
Output: 2
```

Example 2:

```
Input: nums = [3,3,7,7,10,11,11]
Output: 10
```

Constraints:

- `1 <= nums.length <=` 10^5
- `0 <= nums[i] <=` 10^5

```
class Solution:
    def singleNonDuplicate(self, nums: List[int]) -> int:
        n = len(nums)
        left,right = 0,n-1

        while left<right:
            mid = (left+right)//2
            if mid%2==1:
                mid-=1
            if nums[mid] == nums[mid+1]:
                left = mid+2
            else:
                right = mid
        return nums[left]
```

704. Binary Search

Given an array of integers `nums` which is sorted in ascending order, and an integer `target`, write a function to search `target` in `nums`. If `target` exists, then return its index. Otherwise, return `-1`.

You must write an algorithm with `O(log n)` runtime complexity.

Example 1:

```
Input: nums = [-1,0,3,5,9,12], target = 9
Output: 4
Explanation: 9 exists in nums and its index is 4
```

Example 2:

```
Input: nums = [-1,0,3,5,9,12], target = 2
Output: -1
Explanation: 2 does not exist in nums so return -1
```

Constraints:

- `1 <= nums.length <=` 10^4
- -10^4 `< nums[i], target <` 10^4
- All the integers in `nums` are **unique**.
- `nums` is sorted in ascending order.

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        n = len(nums)
        i =0
        j = n-1
        while i <= j:
            mid = (i+j)//2
            if nums[mid] == target:
                return mid
            elif nums[mid] < target:
                i = mid+1
            elif nums[mid] > target:
                j = mid - 1
        return -1
```

875. Koko Eating Bananas

Koko loves to eat bananas. There are `n` piles of bananas, the `i`th pile has `piles[i]` bananas. The guards have gone and will come back in `h` hours.

Koko can decide her bananas-per-hour eating speed of `k`. Each hour, she chooses some pile of bananas and eats `k` bananas from that pile. If the pile has less than `k` bananas, she eats all of them instead and will not eat any more bananas during this hour.

Koko likes to eat slowly but still wants to finish eating all the bananas before the guards return.

Return *the minimum integer* `k` *such that she can eat all the bananas within* `h` *hours*.

Example 1:

```
Input: piles = [3,6,7,11], h = 8
Output: 4
```

Example 2:

```
Input: piles = [30,11,23,4,20], h = 5
Output: 30
```

Example 3:

```
Input: piles = [30,11,23,4,20], h = 6
Output: 23
```

Constraints:

- `1 <= piles.length <=` 10^4
- `piles.length <= h <=` 10^9
- `1 <= piles[i] <=` 10^9

```
class Solution:
    import math
    def minEatingSpeed(self, piles: List[int], h: int) -> int:
        left,right  = 1,max(piles)
        def find_hour(piles,mid):
            total =0
            for i in range(len(piles)):
                total += math.ceil(piles[i]/mid)
            return total
        while left <= right:
            mid = (left+right)//2
            totalhours = find_hour(piles,mid)
            if totalhours <= h:
                right = mid-1
            else:
                left = mid+1
        return left
```

1011. Capacity To Ship Packages Within D Days

A conveyor belt has packages that must be shipped from one port to another within `days` days.

The `i`th package on the conveyor belt has a weight of `weights[i]`. Each day, we load the ship with packages on the conveyor belt (in the order given by `weights`). We may not load more weight than the maximum weight capacity of the ship.

Return the least weight capacity of the ship that will result in all the packages on the conveyor belt being shipped within `days` days.

Example 1:

```
Input: weights = [1,2,3,4,5,6,7,8,9,10], days = 5
Output: 15
Explanation: A ship capacity of 15 is the minimum to ship all the packages in 5 days
1st day: 1, 2, 3, 4, 5
2nd day: 6, 7
3rd day: 8
4th day: 9
5th day: 10

Note that the cargo must be shipped in the order given, so using a ship of capacity 1
```

Example 2:

```
Input: weights = [3,2,2,4,1,4], days = 3
Output: 6
Explanation: A ship capacity of 6 is the minimum to ship all the packages in 3 days l
1st day: 3, 2
2nd day: 2, 4
3rd day: 1, 4
```

Example 3:

```
Input: weights = [1,2,3,1,1], days = 4
Output: 3
Explanation:
1st day: 1
2nd day: 2
3rd day: 3
4th day: 1, 1
```

Constraints:

- `1 <= days <= weights.length <= 5 * 10`^{4}
- `1 <= weights[i] <= 500`

```
class Solution:
    def shipWithinDays(self, weights: List[int], days: int) -> int:
        left,right = max(weights),sum(weights)
        def find_days(arr,weight):
            s = 0
            d=1
            for i in arr:
                if s+i>weight:
                    s = i
                    d+=1
                else:
                    s+=i
            return d

        while left <= right:
            mid = (left+right)//2
            day = find_days(weights,mid)
            if day <= days:
                right = mid-1
            else:
                left = mid+1
        return left
```

1283. Find the Smallest Divisor Given a Threshold

Given an array of integers `nums` and an integer `threshold`, we will choose a positive integer `divisor`, divide all the array by it, and sum the division's result. Find the **smallest** `divisor` such that the result mentioned above is less than or equal to `threshold`.

Each result of the division is rounded to the nearest integer greater than or equal to that element. (For example: `7/3 = 3` and `10/2 = 5`).

The test cases are generated so that there will be an answer.

Example 1:

```
Input: nums = [1,2,5,9], threshold = 6
Output: 5
Explanation: We can get a sum to 17 (1+2+5+9) if the divisor is 1.
If the divisor is 4 we can get a sum of 7 (1+1+2+3) and if the divisor is 5 the sum w
```

Example 2:

```
Input: nums = [44,22,33,11,1], threshold = 5
Output: 44
```

Constraints:

- `1 <= nums.length <=` $5 * 10^4$
- `1 <= nums[i] <=` 10^6
- `nums.length <= threshold <=` 10^6

```
class Solution:
    import math
    def smallestDivisor(self, nums: List[int], threshold: int) -> int:
        left,right = 1,max(nums)
        def find_total(nums,num):
            s= 0
            for i in nums:
                s+= math.ceil(i/num)
            return s

        min_ans = right
        while left <= right:
            mid = (left+right)//2
            summ = find_total(nums,mid)
            if summ<=threshold:
                min_ans = min(min_ans,mid)
            if summ <= threshold:
                right = mid-1
            else:
                left = mid+1
        return min_ans
```

1482. Minimum Number of Days to Make m Bouquets

You are given an integer array `bloomDay` , an integer `m` and an integer `k` .

You want to make `m` bouquets. To make a bouquet, you need to use `k` **adjacent flowers** from the garden.

The garden consists of `n` flowers, the i^{th} flower will bloom in the `bloomDay[i]` and then can be used in **exactly one** bouquet.

Return *the minimum number of days you need to wait to be able to make* `m` *bouquets from the garden*. If it is impossible to make m bouquets return `-1`.

Example 1:

```
Input: bloomDay = [1,10,3,10,2], m = 3, k = 1
Output: 3
Explanation: Let us see what happened in the first three days. x means flower bloomed
We need 3 bouquets each should contain 1 flower.
After day 1: [x, _, _, _, _]   // we can only make one bouquet.
After day 2: [x, _, _, _, x]   // we can only make two bouquets.
After day 3: [x, _, x, _, x]   // we can make 3 bouquets. The answer is 3.
```

Example 2:

```
Input: bloomDay = [1,10,3,10,2], m = 3, k = 2
Output: -1
Explanation: We need 3 bouquets each has 2 flowers, that means we need 6 flowers. We
```

Example 3:

```
Input: bloomDay = [7,7,7,7,12,7,7], m = 2, k = 3
Output: 12
Explanation: We need 2 bouquets each should have 3 flowers.
Here is the garden after the 7 and 12 days:
After day 7: [x, x, x, x, _, x, x]
We can make one bouquet of the first three flowers that bloomed. We cannot make anoth
After day 12: [x, x, x, x, x, x, x]
It is obvious that we can make two bouquets in different ways.
```

Constraints:

- `bloomDay.length == n`
- `1 <= n <=` 10^5
- `1 <= bloomDay[i] <=` 10^9
- `1 <= m <=` 10^6
- `1 <= k <= n`

```
class Solution:
    def minDays(self, bloomDay: List[int], m: int, k: int) -> int:
        left,right = min(bloomDay),max(bloomDay)
        if len(bloomDay) < m*k:
            return -1
        def find_bouquet(arr,day):
            total = 0
            temp = 0
            for i in range(len(arr)):
                if arr[i] <=day:
                    temp+=1
                    if temp == k:
                        temp = 0
                        total+=1
                else:
                    temp = 0
            return total
        while left<=right:
            mid = (left+right)//2
            cor = find_bouquet(bloomDay,mid)
            if cor >= m:
                right = mid-1
            else:
                left = mid+1
        return left
```

1539. Kth Missing Positive Number

Given an array `arr` of positive integers sorted in a **strictly increasing order**, and an integer `k`.

Return *the* k^{th} ***positive*** *integer that is* ***missing*** *from this array.*

Example 1:

```
Input: arr = [2,3,4,7,11], k = 5
Output: 9
Explanation: The missing positive integers are [1,5,6,8,9,10,12,13,...]. The 5th miss
```

Example 2:

```
Input: arr = [1,2,3,4], k = 2
Output: 6
Explanation: The missing positive integers are [5,6,7,...]. The 2nd missing positive
```

Constraints:

- `1 <= arr.length <= 1000`
- `1 <= arr[i] <= 1000`
- `1 <= k <= 1000`
- `arr[i] < arr[j]` for `1 <= i < j <= arr.length`

Follow up:

Could you solve this problem in less than O(n) complexity?

```
def findKthPositive(self, arr: List[int], k: int) -> int:
        left = 0
        right = len(arr)
        while left < right:
                mid = (left+right)//2
                if arr[mid]-1 -mid < k:
                        left = mid+1
                else:
                        right = mid
        return right+k
```

1901. Find a Peak Element II

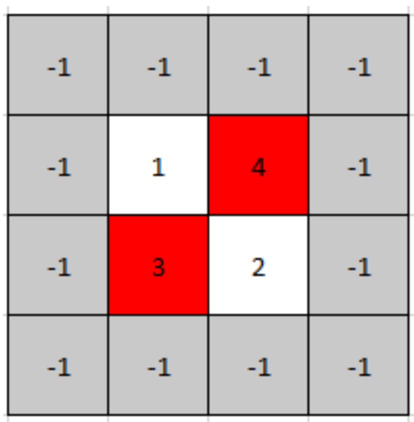
A **peak** element in a 2D grid is an element that is **strictly greater** than all of its **adjacent** neighbors to the left, right, top, and bottom.

Given a **0-indexed** `m x n` matrix `mat` where **no two adjacent cells are equal**, find **any** peak element `mat[i][j]` and return *the length 2 array* `[i,j]`.

You may assume that the entire matrix is surrounded by an **outer perimeter** with the value `-1` in each cell.

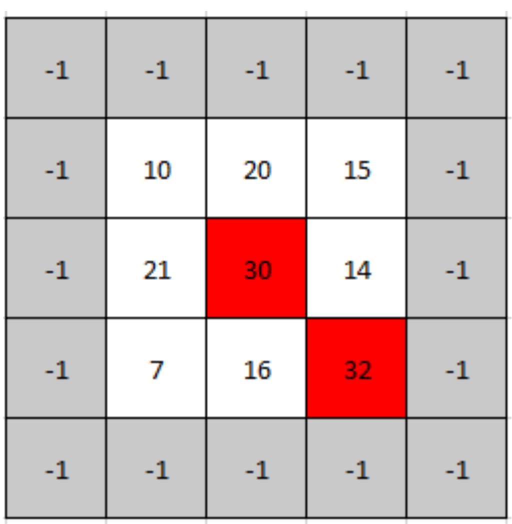
You must write an algorithm that runs in `O(m log(n))` or `O(n log(m))` time.

Example 1:




```
Input: mat = [[1,4],[3,2]]
Output: [0,1]
Explanation: Both 3 and 4 are peak elements so [1,0] and [0,1] are both acceptable ar
```

Example 2:




```
Input: mat = [[10,20,15],[21,30,14],[7,16,32]]
Output: [1,1]
Explanation: Both 30 and 32 are peak elements so [1,1] and [2,2] are both acceptable
```

Constraints:

- `m == mat.length`
- `n == mat[i].length`
- `1 <= m, n <= 500`
- `1 <= mat[i][j] <=` 10^5

- No two adjacent cells are equal.

```
class Solution:
    def findPeakGrid(self, mat: List[List[int]]) -> List[int]:
        def find_row(arr,col,low,high):
            r = -1
            value = -1
            for row in range(0,low):
                if arr[row][col] > value:
                    value = arr[row][col]
                    r = row
            return r

        left,right = 0,len(mat[0])-1
        n = len(mat)
        m = len(mat[0])
        while left <= right:
            mid = (left+right)//2
            row = find_row(mat,mid,n,m) # find the row and col with max_ele
            l = -1 if mid == 0 else mat[row][mid-1]
            r = -1 if mid == m-1 else mat[row][mid+1]
            if l < mat[row][mid] > r:
                return [row,mid]
            elif l > mat[row][mid]:
                right = mid -1
            else:
                left = mid+1
        return [-1,-1]
```