1 ----- GFG: Median in a row-wise sorted Matrix

Given a row wise sorted matrix of size **R*C** where R and C are always **odd**, find the median of the matrix.

Example 1:

```
Input:
R = 3, C = 3
M = [[1, 3, 5],
     [2, 6, 9],
     [3, 6, 9]]
Output: 5
Explanation: Sorting matrix elements gives
us {1,2,3,3,5,6,6,9,9}. Hence, 5 is median.
```

Example 2:

```
Input:
R = 3, C = 1
M = [[1], [2], [3]]
Output: 2
Explanation: Sorting matrix elements gives
us {1,2,3}. Hence, 2 is median.
```

Your Task:
You don't need to read input or print anything. Your task is to complete the function **median()** which takes the integers **R** and **C** along with the 2D **matrix** as input parameters and returns the **median** of the matrix.

Expected Time Complexity: O(32 * R * log(C))
Expected Auxiliary Space: O(1)

Constraints:
1 <= R, C <= 400
1 <= matrix[i][j] <= 2000

2 ----- Leetcode 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`

3 ----- Leetcode 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`

4 ----- Leetcode 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`

5 ----- GFG: K-th element of two sorted arrays

Given two sorted arrays **arr1** and **arr2** and an element **k**. The task is to find the element that would be at the k^{th} position of the combined sorted array.

Examples :

```
Input: k = 5, arr1[] = [2, 3, 6, 7, 9], arr2[] = [1, 4, 8, 10]
Output: 6
```

Explanation: The final combined sorted array would be - 1, 2, 3, 4, 6, 7, 8, 9, 10. The 5th element of this array is 6.

Input: k = 7, arr1[] = [100, 112, 256, 349, 770], arr2[] = [72, 86, 113, 119, 265, 445, 892]
Output: 256
Explanation: Combined sorted array is - 72, 86, 100, 112, 113, 119, 256, 265, 349, 445, 770, 892. 7th element of this array is 256

Expected Time Complexity: O(log(n) + log(m))
Expected Auxiliary Space: O(log (n))

Constraints:
1 <= k<= arr1.size()+arr2.size()
1 <= arr1.size(), arr2.size() <= 10^6
0 <= arr1[i], arr2[i] < 10^8

6 ----- GFG: Allocate Minimum Number of Pages from N books to M students

Given that there are **N books** and **M students**. Also given are the **number of pages in each book in ascending order**. The task is to assign books in such a way that the **maximum number of pages assigned to a student is minimum**, with the condition that every student is assigned to read some consecutive books. Print that minimum number of pages.

Examples:

Input: N = 4, pages[] = {12, 34, 67, 90}, M = 2
Output: 113
Explanation: There are 2 students. Books can be distributed in following combinations:

- {12} and {34, 67, 90} -> Max number of pages is allocated to student 2 with 34 + 67 + 90 = 191 pages
- {12, 34} and {67, 90} -> Max number of pages is allocated to student 2 with 67 + 90 = 157
- {12, 34, 67} and {90} -> Max number of pages is allocated to student 1 with 12 + 34 + 67 = 113 pages

The third combination has the minimum pages assigned to a student = 113.

Input: N = 3, pages[] = {15, 17, 20}, M = 2
Output: 32
Explanation: There are 2 students. Books can be distributed in following combinations:

- {15} and {17, 20} -> Max number of pages is allocated to student 2 with 17 + 20 = 37 pages

- {15, 17} and {20} -> Max number of pages is allocated to student 1 with 15 + 17 = 32 pages

The second combination has the minimum pages assigned to a student = 32.

7 ----- Leetcode 153: 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.
```

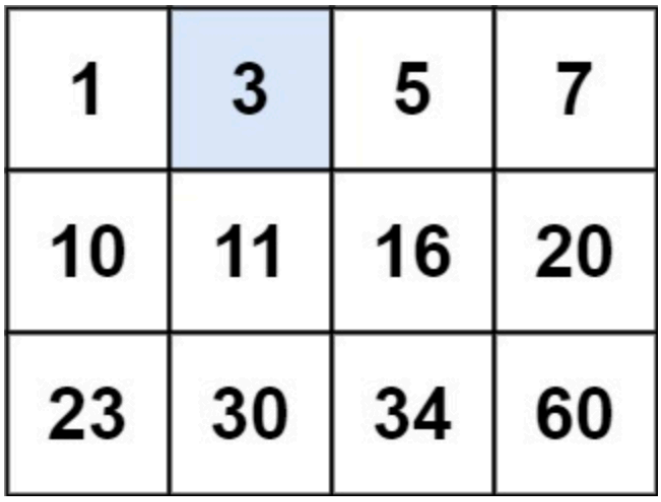
Constraints:

- `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.

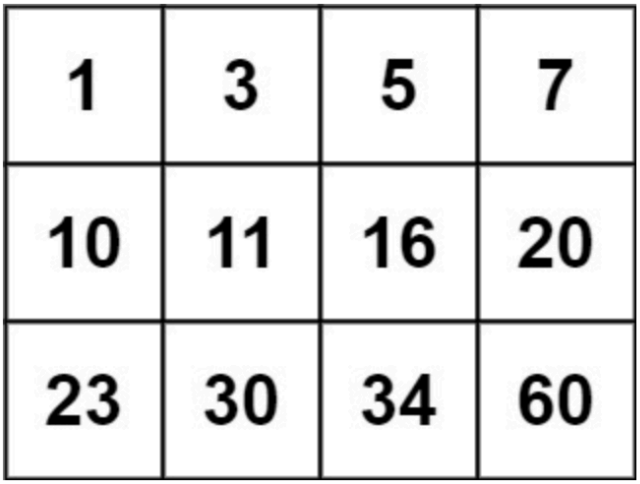
8 ----- Leetcode 74: Search a 2D Matrix

Example 1:




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

Example 2:




```
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

9 ----- Leetcode 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

10 ----- GFG: Aggressive Cows

You are given an **array** consisting of **n integers** which denote the position of a **stall**. You are also given an **integer k** which denotes the number of aggressive cows. You are given the task of **assigning stalls to k cows** such that the **minimum distance between any two of them is the maximum possible**.
The first line of input contains two space-separated integers **n** and **k**.
The second line contains **n** space-separated integers denoting the position of the stalls.

Example 1:

Input:
n=5
k=3
stalls = [1 2 4 8 9]
Output:
3
Explanation:
The first cow can be placed at stalls[0],
the second cow can be placed at stalls[2] and
the third cow can be placed at stalls[3].
The minimum distance between cows, in this case, is 3,
which also is the largest among all possible ways.

Example 2:

Input:
n=5
k=3
stalls = [10 1 2 7 5]
Output:
4
Explanation:
The first cow can be placed at stalls[0],
the second cow can be placed at stalls[1] and
the third cow can be placed at stalls[4].
The minimum distance between cows, in this case, is 4,
which also is the largest among all possible ways.

Your Task:

Complete the function int solve(), which takes integer n, k, and a vector stalls with n integers as input and returns the largest possible minimum distance between cows.

Expected Time Complexity: O(n*log(10^9)).
Expected Auxiliary Space: O(1).

Constraints:

2 <= n <= 10^5
2 <= k <= n
0 <= stalls[i] <= 10^9