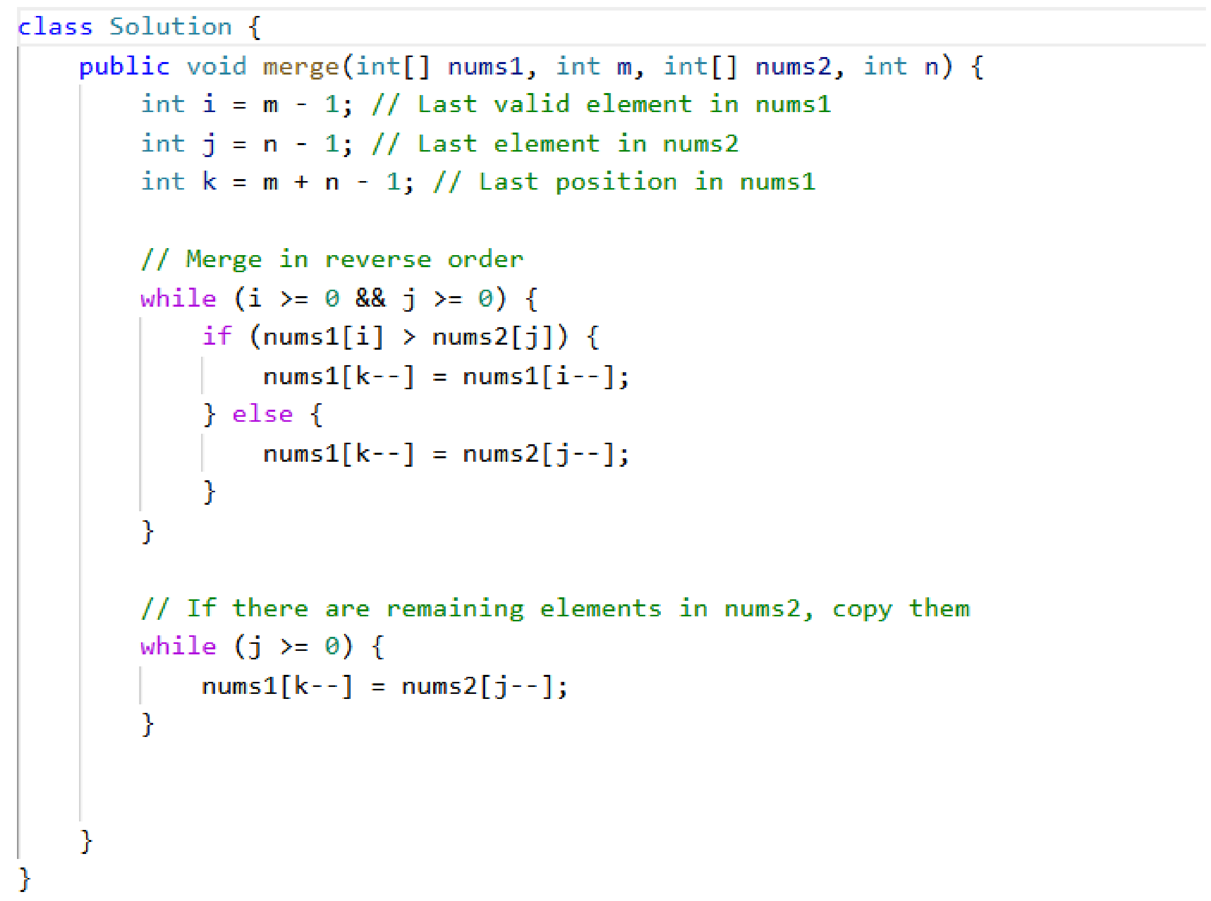
Assignment 5

Name : Muskan

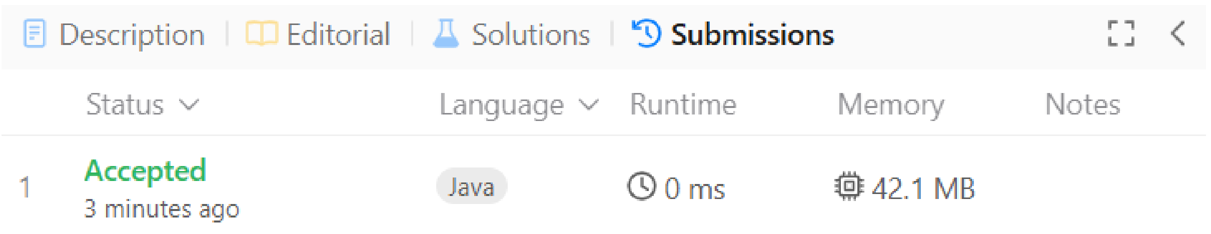
UID : 22BCS16260

Section : 608 B

1. Merge Sorted Array:

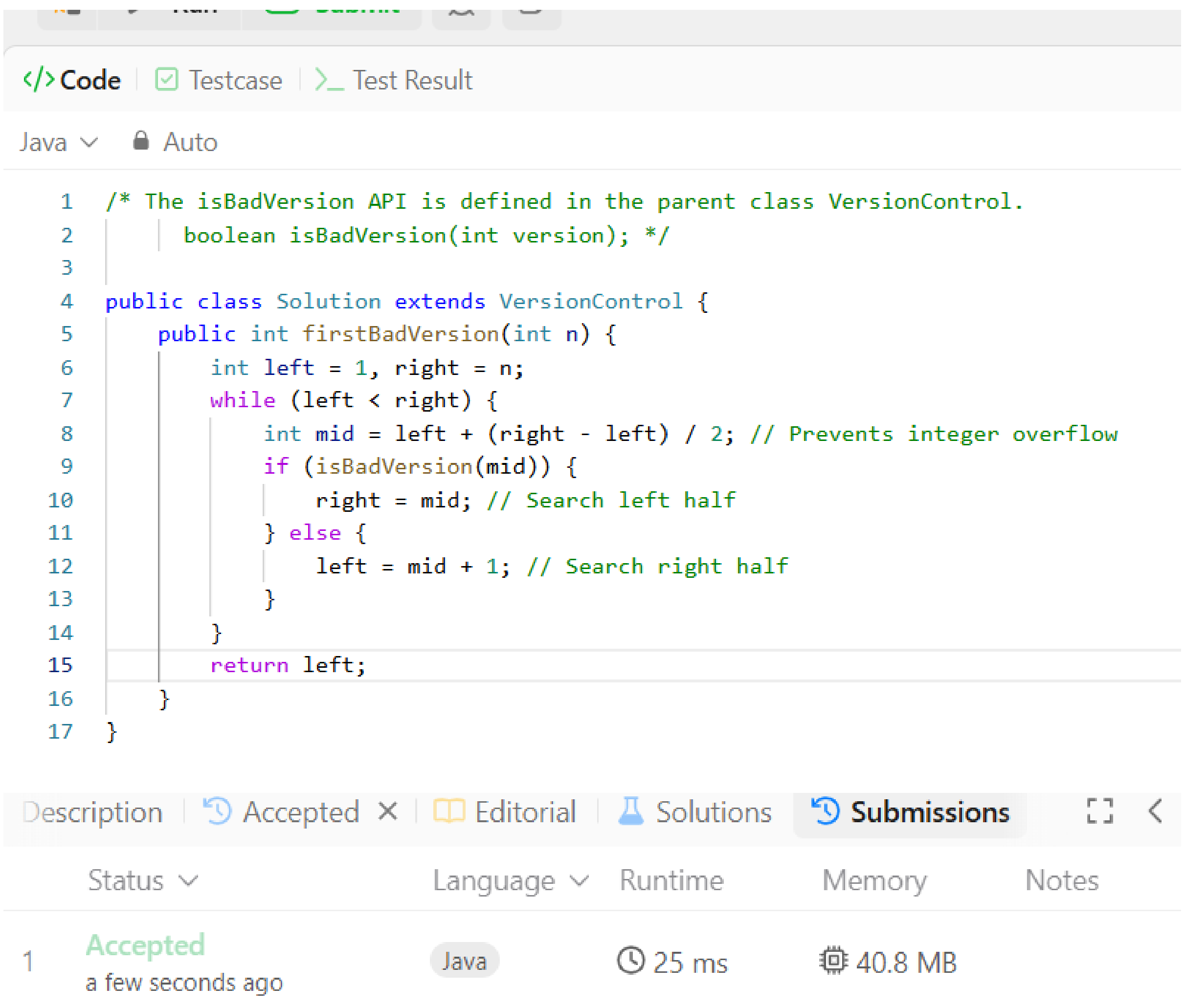


```
class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        int i = m - 1; // Last valid element in nums1
        int j = n - 1; // Last element in nums2
        int k = m + n - 1; // Last position in nums1

        // Merge in reverse order
        while (i >= 0 && j >= 0) {
            if (nums1[i] > nums2[j]) {
                nums1[k--] = nums1[i--];
            } else {
                nums1[k--] = nums2[j--];
            }
        }

        // If there are remaining elements in nums2, copy them
        while (j >= 0) {
            nums1[k--] = nums2[j--];
        }


    }
}
```



	Status	Language	Runtime	Memory	Notes
1	Accepted 3 minutes ago	Java	0 ms	42.1 MB	

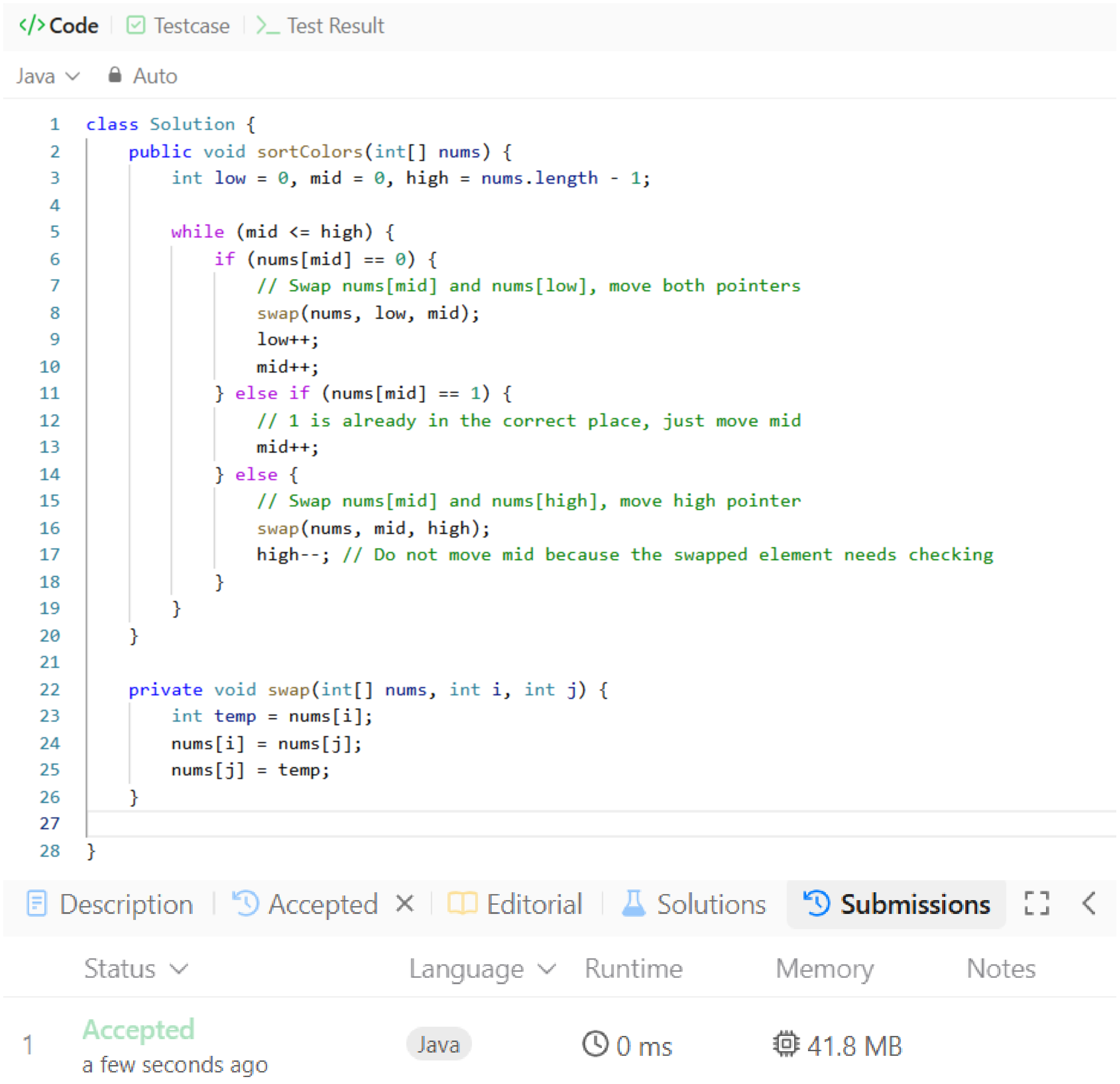
2. First Bad Version:



```java
/* The isBadVersion API is defined in the parent class VersionControl.
      boolean isBadVersion(int version); */

public class Solution extends VersionControl {
    public int firstBadVersion(int n) {
        int left = 1, right = n;
        while (left < right) {
            int mid = left + (right - left) / 2; // Prevents integer overflow
            if (isBadVersion(mid)) {
                right = mid; // Search left half
            } else {
                left = mid + 1; // Search right half
            }
        }
        return left;
    }
}
```

	Status	Language	Runtime	Memory	Notes
1	Accepted a few seconds ago	Java	25 ms	40.8 MB	

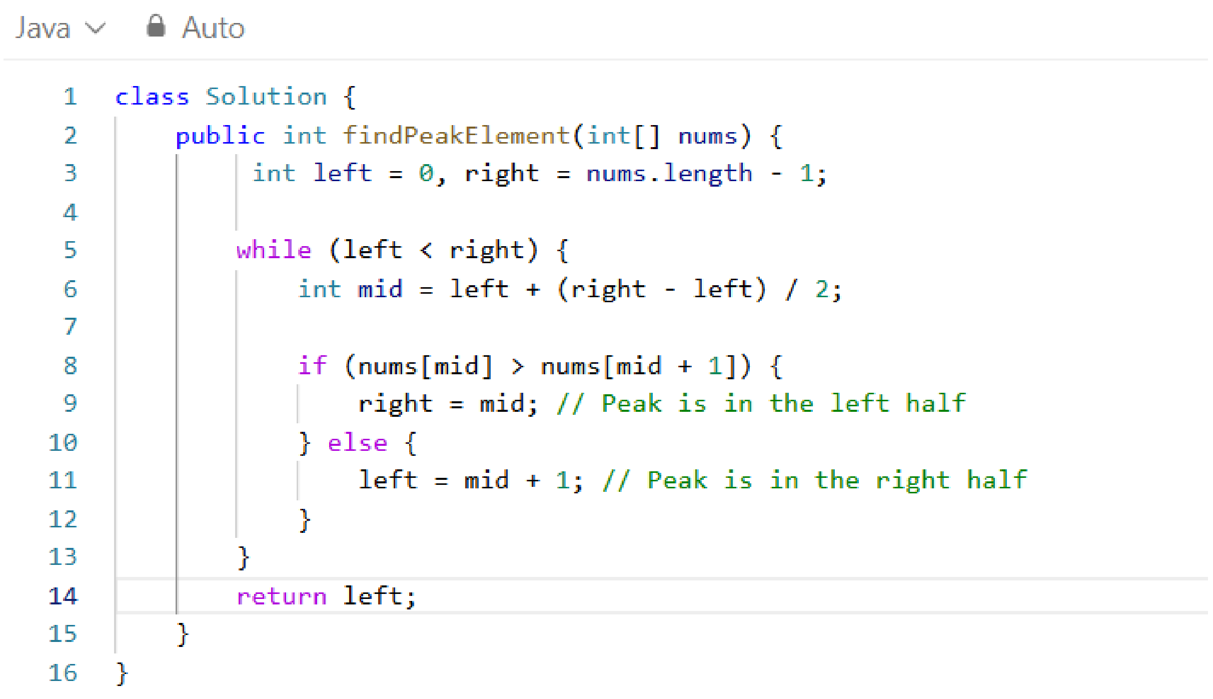
3. Sort Colors:



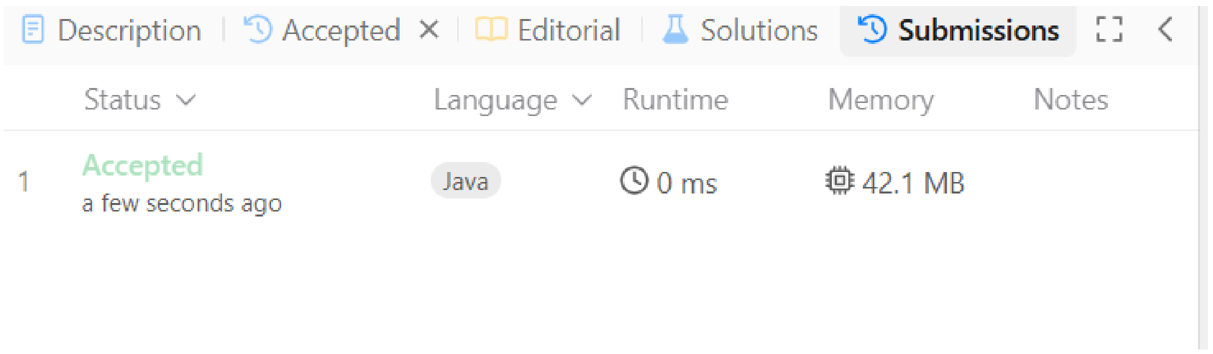
```java
class Solution {
    public void sortColors(int[] nums) {
        int low = 0, mid = 0, high = nums.length - 1;

        while (mid <= high) {
            if (nums[mid] == 0) {
                // Swap nums[mid] and nums[low], move both pointers
                swap(nums, low, mid);
                low++;
                mid++;
            } else if (nums[mid] == 1) {
                // 1 is already in the correct place, just move mid
                mid++;
            } else {
                // Swap nums[mid] and nums[high], move high pointer
                swap(nums, mid, high);
                high--; // Do not move mid because the swapped element needs checking
            }
        }
    }

    private void swap(int[] nums, int i, int j) {
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
    }

}
```

	Status	Language	Runtime	Memory	Notes
1	Accepted a few seconds ago	Java	0 ms	41.8 MB	

4. Find Peak Element:

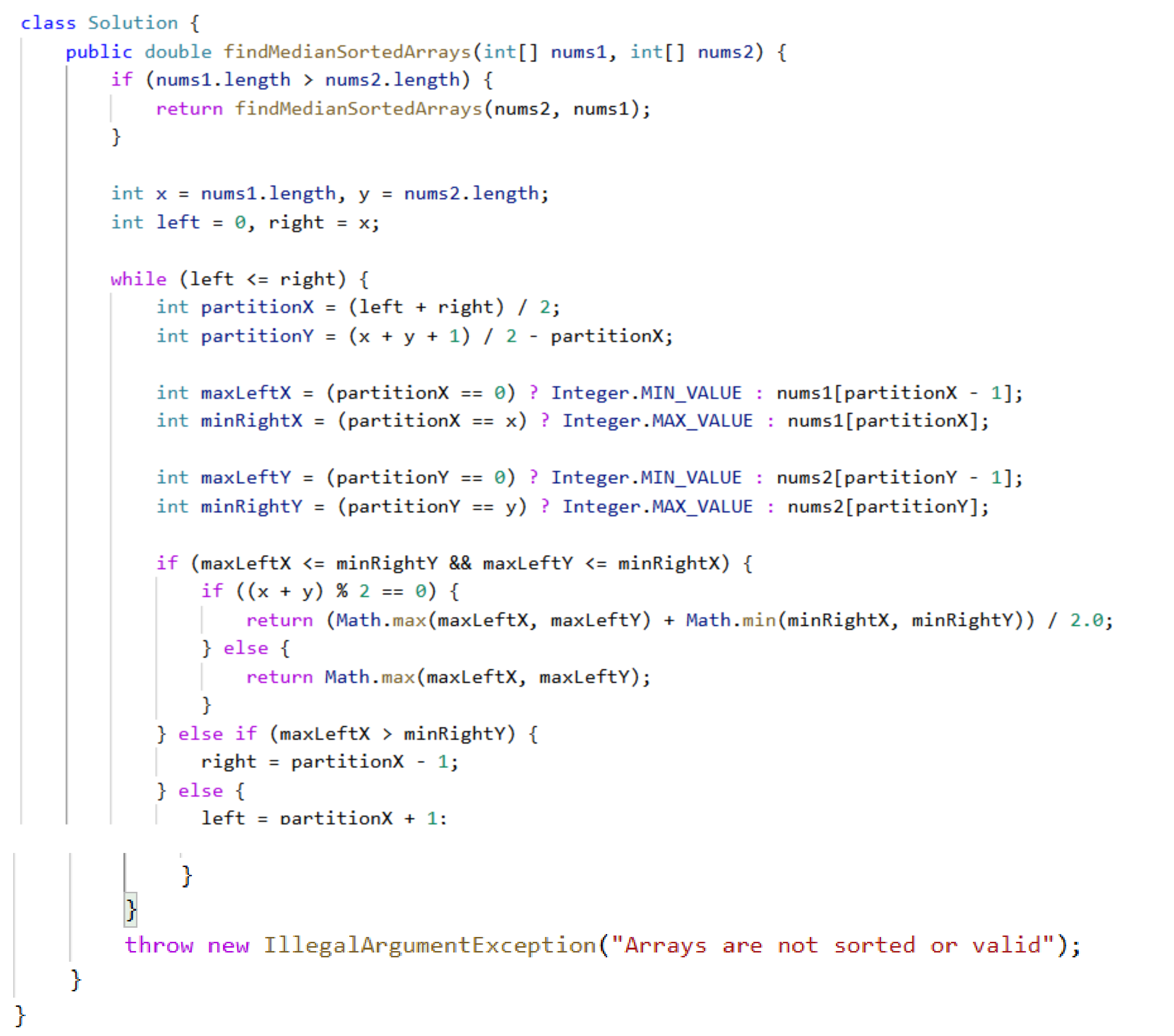


```java
class Solution {
    public int findPeakElement(int[] nums) {
        int left = 0, right = nums.length - 1;

        while (left < right) {
            int mid = left + (right - left) / 2;

            if (nums[mid] > nums[mid + 1]) {
                right = mid; // Peak is in the left half
            } else {
                left = mid + 1; // Peak is in the right half
            }
        }
        return left;
    }
}
```

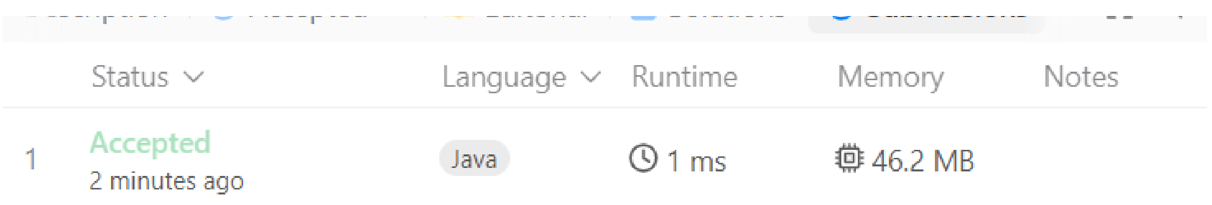


	Status	Language	Runtime	Memory	Notes
1	Accepted a few seconds ago	Java	0 ms	42.1 MB	

5. Median of Two Sorted Arrays:



```java
class Solution {
    public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        if (nums1.length > nums2.length) {
            return findMedianSortedArrays(nums2, nums1);
        }

        int x = nums1.length, y = nums2.length;
        int left = 0, right = x;

        while (left <= right) {
            int partitionX = (left + right) / 2;
            int partitionY = (x + y + 1) / 2 - partitionX;

            int maxLeftX = (partitionX == 0) ? Integer.MIN_VALUE : nums1[partitionX - 1];
            int minRightX = (partitionX == x) ? Integer.MAX_VALUE : nums1[partitionX];

            int maxLeftY = (partitionY == 0) ? Integer.MIN_VALUE : nums2[partitionY - 1];
            int minRightY = (partitionY == y) ? Integer.MAX_VALUE : nums2[partitionY];

            if (maxLeftX <= minRightY && maxLeftY <= minRightX) {
                if ((x + y) % 2 == 0) {
                    return (Math.max(maxLeftX, maxLeftY) + Math.min(minRightX, minRightY)) / 2.0;
                } else {
                    return Math.max(maxLeftX, maxLeftY);
                }
            } else if (maxLeftX > minRightY) {
                right = partitionX - 1;
            } else {
                left = partitionX + 1;
            }
        }
        throw new IllegalArgumentException("Arrays are not sorted or valid");
    }
}
```



	Status	Language	Runtime	Memory	Notes
1	Accepted 2 minutes ago	Java	1 ms	46.2 MB	