

```
public class StackUsingQueues {
    Queue<Integer> queue1 = new LinkedList<>();
    Queue<Integer> queue2 = new LinkedList<>();

    public void push(int x) {
        queue2.add(x);
        while (!queue1.isEmpty()) {
            queue2.add(queue1.remove());
        }

        Queue<Integer> temp = queue1;
        queue1 = queue2;
        queue2 = temp;
    }

    public int pop() {
        return queue1.isEmpty() ? -1 : queue1.remove();
    }

    public int top() {
        return queue1.isEmpty() ? -1 : queue1.peek();
    }

    public boolean isEmpty() {
        return queue1.isEmpty();
    }

    public static void main(String[] args) {
        StackUsingQueues stack = new StackUsingQueues();
        stack.push(1);
        stack.push(2);
        System.out.println(stack.top()); // Output: 2
        System.out.println(stack.pop()); // Output: 2
        System.out.println(stack.isEmpty()); // Output: false
    }
}
```

That completes the unique set of problems with Java code and explanations. Let me know if you need further modifications!



No man pick 20 best question that are most asked first explain them using example then write code



Memory updated

Here is a list of 20 most commonly asked array-based questions in coding interviews. I will explain each problem with an example first, then provide a solution in Java.

1. Find the Largest Sum Contiguous Subarray (Kadane's Algorithm)

Problem:

Find the maximum sum of a contiguous subarray within a one-dimensional array of numbers.

Example:

Input: `[-2, 1, -3, 4, -1, 2, 1, -5, 4]`

Output: `6`

Explanation: The subarray `[4, -1, 2, 1]` has the largest sum = 6.

Solution:

java

```
public class LargestSumSubarray {
    public static int maxSubArray(int[] nums) {
        int maxSoFar = nums[0], maxEndingHere = nums[0];
        for (int i = 1; i < nums.length; i++) {
            maxEndingHere = Math.max(nums[i], maxEndingHere + nums[i]);
            maxSoFar = Math.max(maxSoFar, maxEndingHere);
        }
        return maxSoFar;
    }

    public static void main(String[] args) {
        int[] nums = {-2, 1, -3, 4, -1, 2, 1, -5, 4};
        System.out.println(maxSubArray(nums)); // Output: 6
    }
}
```

2. Move All Zeros to End

Problem:

Move all zeroes in an array to the end while maintaining the relative order of the other elements.

Example:

Input: `[0, 1, 0, 3, 12]`

Output: `[1, 3, 12, 0, 0]`

Solution:

java

```
public class MoveZerosToEnd {
    public static void moveZeroes(int[] nums) {
        int index = 0;
        for (int i = 0; i < nums.length; i++) {
            if (nums[i] != 0) {
                nums[index++] = nums[i];
            }
        }
        while (index < nums.length) {
            nums[index++] = 0;
        }
    }

    public static void main(String[] args) {
        int[] nums = {0, 1, 0, 3, 12};
        moveZeroes(nums);
        for (int num : nums) {
            System.out.print(num + " "); // Output: 1 3 12 0 0
        }
    }
}
```

```
}
}
```

3. Find Missing Number in Array

Problem:

Given an array containing $n-1$ numbers from 1 to n , find the missing number.

Example:

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

Output: `6`

Solution:

```
java
```

```
public class MissingNumber {
    public static int findMissingNumber(int[] nums, int n) {
        int totalSum = n * (n + 1) / 2;
        int arraySum = 0;
        for (int num : nums) {
            arraySum += num;
        }
        return totalSum - arraySum;
    }

    public static void main(String[] args) {
        int[] nums = {3, 7, 1, 2, 8, 4, 5};
        int n = 8;
        System.out.println(findMissingNumber(nums, n)); // Output: 6
    }
}
```

4. Find Duplicate Number

Problem:

Given an array containing $n+1$ integers where each integer is between 1 and n , find the duplicate number.

Example:

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

Output: `3`

Solution:

```
java
```

```
public class FindDuplicate {
    public static int findDuplicate(int[] nums) {
        int slow = nums[0];
        int fast = nums[0];

        do {
            slow = nums[slow];
            fast = nums[nums[fast]];
        } while (slow != fast);

        fast = nums[0];
        while (slow != fast) {
            slow = nums[slow];
            fast = nums[fast];
        }

        return slow;
    }

    public static void main(String[] args) {
        int[] nums = {3, 1, 3, 4, 2};
        System.out.println(findDuplicate(nums)); // Output: 3
    }
}
```

5. Two Sum Problem

Problem:

Given an array of integers, return indices of the two numbers that add up to a specific target.

Example:

Input: `nums = [2, 7, 11, 15], target = 9`

Output: `[0, 1]`

Explanation: `nums[0] + nums[1] = 9`

Solution:

```
java
```

```
import java.util.HashMap;

public class TwoSum {
    public static int[] twoSum(int[] nums, int target) {
        HashMap<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
            int complement = target - nums[i];
            if (map.containsKey(complement)) {
                return new int[]{map.get(complement), i};
            }
            map.put(nums[i], i);
        }
        return new int[]{};
    }

    public static void main(String[] args) {
        int[] nums = {2, 7, 11, 15};
        int target = 9;
    }
}
```

```
int[] result = twoSum(nums, target);
System.out.println(result[0] + " " + result[1]); // Output: 0 1
}
}
```

6. Best Time to Buy and Sell Stock

Problem:

Given an array where each element represents the stock price on a given day, find the maximum profit you can achieve by buying on one day and selling on another.

Example:

Input: `[7, 1, 5, 3, 6, 4]`

Output: `5`

Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6 - 1 = 5.

Solution:

java

```
public class BestTimeToBuySellStock {
    public static int maxProfit(int[] prices) {
        int minPrice = Integer.MAX_VALUE;
        int maxProfit = 0;

        for (int price : prices) {
            if (price < minPrice) {
                minPrice = price;
            } else if (price - minPrice > maxProfit) {
                maxProfit = price - minPrice;
            }
        }

        return maxProfit;
    }

    public static void main(String[] args) {
        int[] prices = {7, 1, 5, 3, 6, 4};
        System.out.println(maxProfit(prices)); // Output: 5
    }
}
```

I'll continue the next set of problems. Let me know if you'd like adjustments so far!



continue its perfect



Great! Here are the next set of array-based interview questions:

7. Find All Pairs With a Given Sum

Problem:

Given an array of integers, find all unique pairs of elements that add up to a given sum.

Example:

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

Output: `[(1, 5), (7, -1)]`

Solution:

java

```
import java.util.HashSet;

public class PairWithGivenSum {
    public static void findPairs(int[] nums, int target) {
        HashSet<Integer> set = new HashSet<>();
        for (int num : nums) {
            int complement = target - num;
            if (set.contains(complement)) {
                System.out.println("(" + complement + ", " + num + ")");
            }
            set.add(num);
        }
    }

    public static void main(String[] args) {
        int[] nums = {1, 5, 7, -1, 5};
        int target = 6;
        findPairs(nums, target); // Output: (1, 5) (7, -1)
    }
}
```

8. Rotate Array

Problem:

Rotate an array of `n` elements to the right by `k` steps.

Example:

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

Output: `[5, 6, 7, 1, 2, 3, 4]`

Solution:

java

```
public class RotateArray {
    public static void rotate(int[] nums, int k) {
        k = k % nums.length;
        reverse(nums, 0, nums.length - 1);
    }
}
```

```

        reverse(nums, 0, k - 1);
        reverse(nums, k, nums.length - 1);
    }

    public static void reverse(int[] nums, int start, int end) {
        while (start < end) {
            int temp = nums[start];
            nums[start] = nums[end];
            nums[end] = temp;
            start++;
            end--;
        }
    }

    public static void main(String[] args) {
        int[] nums = {1, 2, 3, 4, 5, 6, 7};
        int k = 3;
        rotate(nums, k);
        for (int num : nums) {
            System.out.print(num + " "); // Output: 5 6 7 1 2 3 4
        }
    }
}

```

9. Find the Intersection of Two Arrays

Problem:

Given two arrays, find their intersection.

Example:

Input: `nums1 = [1, 2, 2, 1], nums2 = [2, 2]`

Output: `[2]`

Solution:

java

```

import java.util.HashSet;

public class ArrayIntersection {
    public static int[] intersection(int[] nums1, int[] nums2) {
        HashSet<Integer> set1 = new HashSet<>();
        for (int num : nums1) {
            set1.add(num);
        }

        HashSet<Integer> intersectionSet = new HashSet<>();
        for (int num : nums2) {
            if (set1.contains(num)) {
                intersectionSet.add(num);
            }
        }

        int[] result = new int[intersectionSet.size()];
        int index = 0;
        for (int num : intersectionSet) {
            result[index++] = num;
        }

        return result;
    }
}

```

```
public static void main(String[] args) {
    int[] nums1 = {1, 2, 2, 1};
    int[] nums2 = {2, 2};
    int[] result = intersection(nums1, nums2);
    for (int num : result) {
        System.out.print(num + " "); // Output: 2
    }
}
```

10. Find the Majority Element

Problem:

Find the element that appears more than $n/2$ times in an array of size n .

Example:

Input: `[3, 2, 3]`

Output: `3`

Solution:

java

```
public class MajorityElement {
    public static int findMajorityElement(int[] nums) {
        int count = 0, candidate = 0;

        for (int num : nums) {
            if (count == 0) {
                candidate = num;
            }
            count += (num == candidate) ? 1 : -1;
        }

        return candidate;
    }

    public static void main(String[] args) {
        int[] nums = {3, 2, 3};
        System.out.println(findMajorityElement(nums)); // Output: 3
    }
}
```

11. Find the First Missing Positive

Problem:

Given an unsorted array, find the smallest missing positive integer.

Example:

Input: `[1, 2, 0]`

Output: `3`

Solution:

java

```
public class FirstMissingPositive {
    public static int firstMissingPositive(int[] nums) {
        int n = nums.length;
        for (int i = 0; i < n; i++) {
            while (nums[i] > 0 && nums[i] <= n && nums[nums[i] - 1] != nums[i]) {
                int temp = nums[nums[i] - 1];
                nums[nums[i] - 1] = nums[i];
                nums[i] = temp;
            }
        }

        for (int i = 0; i < n; i++) {
            if (nums[i] != i + 1) {
                return i + 1;
            }
        }

        return n + 1;
    }

    public static void main(String[] args) {
        int[] nums = {1, 2, 0};
        System.out.println(firstMissingPositive(nums)); // Output: 3
    }
}
```

12. Find Kth Largest Element in an Array

Problem:

Find the `k`th largest element in an array.

Example:

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

Output: `5`

Solution:

java

```
import java.util.PriorityQueue;

public class KthLargestElement {
    public static int findKthLargest(int[] nums, int k) {
        PriorityQueue<Integer> minHeap = new PriorityQueue<>();
        for (int num : nums) {
            minHeap.add(num);
            if (minHeap.size() > k) {
                minHeap.poll();
            }
        }
    }
}
```

```

        return minHeap.peek();
    }

    public static void main(String[] args) {
        int[] nums = {3, 2, 1, 5, 6, 4};
        int k = 2;
        System.out.println(findKthLargest(nums, k)); // Output: 5
    }
}

```

13. Container With Most Water

Problem:

Given an array `height` representing the heights of vertical lines, find two lines that, together with the x-axis, form a container with the most water.

Example:

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

Output: `49`

Solution:

java

```

public class ContainerWithMostWater {
    public static int maxArea(int[] height) {
        int left = 0, right = height.length - 1;
        int maxArea = 0;

        while (left < right) {
            int currentArea = Math.min(height[left], height[right]) * (right - left);
            maxArea = Math.max(maxArea, currentArea);

            if (height[left] < height[right]) {
                left++;
            } else {
                right--;
            }
        }

        return maxArea;
    }

    public static void main(String[] args) {
        int[] height = {1, 8, 6, 2, 5, 4, 8, 3, 7};
        System.out.println(maxArea(height)); // Output: 49
    }
}

```

I'll continue with the rest of the questions. Let me know if you'd like any further changes!



i said 20 you gave 13 only write 7 more



Here are the remaining 7 questions to complete the list of 20 array-based interview questions:

14. Find the Longest Consecutive Sequence

Problem:

Given an unsorted array of integers, find the length of the longest consecutive elements sequence.

Example:

Input: `nums = [100, 4, 200, 1, 3, 2]`

Output: `4`

Explanation: The longest consecutive elements sequence is `[1, 2, 3, 4]`.

Solution:

```
java

import java.util.HashSet;

public class LongestConsecutiveSequence {
    public static int longestConsecutive(int[] nums) {
        HashSet<Integer> set = new HashSet<>();
        for (int num : nums) {
            set.add(num);
        }

        int longestStreak = 0;

        for (int num : set) {
            if (!set.contains(num - 1)) {
                int currentNum = num;
                int currentStreak = 1;

                while (set.contains(currentNum + 1)) {
                    currentNum += 1;
                    currentStreak += 1;
                }

                longestStreak = Math.max(longestStreak, currentStreak);
            }
        }

        return longestStreak;
    }

    public static void main(String[] args) {
        int[] nums = {100, 4, 200, 1, 3, 2};
        System.out.println(longestConsecutive(nums)); // Output: 4
    }
}
```

15. Find the Subarray With the Largest Sum (Kadane's Algorithm)

Problem:

Given an integer array, find the contiguous subarray (containing at least one number) that has the largest sum and return its sum.

Example:

Input: `nums = [-2,1,-3,4,-1,2,1,-5,4]`

Output: `6`

Explanation: `[4,-1,2,1]` has the largest sum = 6.

Solution:

```
java

public class MaximumSubarraySum {
    public static int maxSubArray(int[] nums) {
        int maxCurrent = nums[0];
        int maxGlobal = nums[0];

        for (int i = 1; i < nums.length; i++) {
            maxCurrent = Math.max(nums[i], maxCurrent + nums[i]);
            if (maxCurrent > maxGlobal) {
                maxGlobal = maxCurrent;
            }
        }

        return maxGlobal;
    }

    public static void main(String[] args) {
        int[] nums = {-2, 1, -3, 4, -1, 2, 1, -5, 4};
        System.out.println(maxSubArray(nums)); // Output: 6
    }
}
```

16. Find Product of Array Except Self

Problem:

Given an array, return an array such that each element at index `i` is the product of all the numbers in the array except `nums[i]`.

Example:

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

Output: `[24, 12, 8, 6]`

Solution:

```
java

public class ProductOfArrayExceptSelf {
    public static int[] productExceptSelf(int[] nums) {
        int[] result = new int[nums.length];

        int left = 1;
```

```

    for (int i = 0; i < nums.length; i++) {
        result[i] = left;
        left *= nums[i];
    }

    int right = 1;
    for (int i = nums.length - 1; i >= 0; i--) {
        result[i] *= right;
        right *= nums[i];
    }

    return result;
}

public static void main(String[] args) {
    int[] nums = {1, 2, 3, 4};
    int[] result = productExceptSelf(nums);
    for (int num : result) {
        System.out.print(num + " "); // Output: 24 12 8 6
    }
}

```

17. 3Sum Problem

Problem:

Given an array of integers, return all triplets `[nums[i], nums[j], nums[k]]` such that `i != j != k`, and `nums[i] + nums[j] + nums[k] = 0`.

Example:

Input: `nums = [-1, 0, 1, 2, -1, -4]`

Output: `[[-1, 0, 1], [-1, -1, 2]]`

Solution:

java

```

import java.util.*;

public class ThreeSum {
    public static List<List<Integer>> threeSum(int[] nums) {
        Arrays.sort(nums);
        List<List<Integer>> result = new ArrayList<>();

        for (int i = 0; i < nums.length - 2; i++) {
            if (i > 0 && nums[i] == nums[i - 1]) continue;

            int left = i + 1;
            int right = nums.length - 1;
            int target = -nums[i];

            while (left < right) {
                int sum = nums[left] + nums[right];
                if (sum == target) {
                    result.add(Arrays.asList(nums[i], nums[left], nums[right]));
                    while (left < right && nums[left] == nums[left + 1]) left++;
                    while (left < right && nums[right] == nums[right - 1]) right--;
                    left++;
                    right--;
                } else if (sum < target) {
                    left++;
                } else {
                    right--;
                }
            }
        }

        return result;
    }
}

```

```

        } else {
            right--;
        }
    }
}

return result;
}

public static void main(String[] args) {
    int[] nums = {-1, 0, 1, 2, -1, -4};
    List<List<Integer>> result = threeSum(nums);
    System.out.println(result); // Output: [[-1, -1, 2], [-1, 0, 1]]
}
}

```

18. Find Minimum in Rotated Sorted Array

Problem:

Suppose an array is sorted in ascending order and then rotated at an unknown pivot. Find the minimum element.

Example:

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

Output: `1`

Solution:

java

```

public class FindMinInRotatedArray {
    public static int findMin(int[] nums) {
        int left = 0, right = nums.length - 1;

        while (left < right) {
            int mid = left + (right - left) / 2;
            if (nums[mid] > nums[right]) {
                left = mid + 1;
            } else {
                right = mid;
            }
        }

        return nums[left];
    }

    public static void main(String[] args) {
        int[] nums = {3, 4, 5, 1, 2};
        System.out.println(findMin(nums)); // Output: 1
    }
}

```

19. Find Duplicate Number

Problem:

Given an array of $n + 1$ integers where each integer is between 1 and n , find the duplicate number.

Example:

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

Output: `2`

Solution:

java

```
public class FindDuplicate {
    public static int findDuplicate(int[] nums) {
        int slow = nums[0];
        int fast = nums[0];

        do {
            slow = nums[slow];
            fast = nums[nums[fast]];
        } while (slow != fast);

        slow = nums[0];
        while (slow != fast) {
            slow = nums[slow];
            fast = nums[fast];
        }

        return slow;
    }

    public static void main(String[] args) {
        int[] nums = {1, 3, 4, 2, 2};
        System.out.println(findDuplicate(nums)); // Output: 2
    }
}
```

20. Search in Rotated Sorted Array

Problem:

Search for a target value in a rotated sorted array. Assume no duplicate exists.

Example:

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

Output: `4`

Solution:

java

```
public class SearchInRotatedArray {
    public static int search(int[] nums, int target) {
        int left = 0, right = nums.length - 1;
```

```

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

    if (nums[mid] == target) {
        return mid;
    }

    if (nums[left] <= nums[mid]) {
        if (target >= nums[left] && target < nums[mid]) {
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    } else {
        if (target > nums[mid] && target <= nums[right]) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
}

return -1;
}

public static void main(String[] args) {
    int[] nums = {4, 5, 6, 7, 0, 1, 2};
    int target = 0;
    System.out.println(search(nums, target));
}

```



add 5 more



Here are 5 more array-based interview questions to further enrich your list:

21. Maximum Product Subarray

Problem:

Given an integer array, find the contiguous subarray within an array (containing at least one number) which has the largest product.

Example:

Input: `nums = [2, 3, -2, 4]`

Output: `6`

Explanation: The subarray `[2, 3]` has the largest product.

Solution:

java

```

public class MaximumProductSubarray {
    public static int maxProduct(int[] nums) {
        int maxSoFar = nums[0];
    }
}

```



```

int minSoFar = nums[0];
int result = nums[0];

for (int i = 1; i < nums.length; i++) {
    if (nums[i] < 0) {
        int temp = maxSoFar;
        maxSoFar = minSoFar;
        minSoFar = temp;
    }

    maxSoFar = Math.max(nums[i], maxSoFar * nums[i]);
    minSoFar = Math.min(nums[i], minSoFar * nums[i]);

    result = Math.max(result, maxSoFar);
}

return result;
}

public static void main(String[] args) {
    int[] nums = {2, 3, -2, 4};
    System.out.println(maxProduct(nums)); // Output: 6
}

```

22. Check If Array Contains Duplicate

Problem:

Given an integer array, find if the array contains any duplicates.

Example:

Input: `nums = [1, 2, 3, 1]`

Output: `true`

Solution:

java

```

import java.util.HashSet;

public class ContainsDuplicate {
    public static boolean containsDuplicate(int[] nums) {
        HashSet<Integer> set = new HashSet<>();
        for (int num : nums) {
            if (set.contains(num)) {
                return true;
            }
            set.add(num);
        }
        return false;
    }

    public static void main(String[] args) {
        int[] nums = {1, 2, 3, 1};
        System.out.println(containsDuplicate(nums)); // Output: true
    }
}

```

23. Two Sum II - Input Array Is Sorted

Problem:

Given a 1-indexed array of integers `numbers` that is already sorted in non-decreasing order, find two numbers such that they add up to a specific target number.

Example:

Input: `numbers = [2, 7, 11, 15], target = 9`

Output: `[1, 2]`

Solution:

```
java
```

```
public class TwoSumII {
    public static int[] twoSum(int[] numbers, int target) {
        int left = 0, right = numbers.length - 1;

        while (left < right) {
            int sum = numbers[left] + numbers[right];
            if (sum == target) {
                return new int[] {left + 1, right + 1};
            } else if (sum < target) {
                left++;
            } else {
                right--;
            }
        }

        return new int[] {}; // Return an empty array if no solution found
    }

    public static void main(String[] args) {
        int[] numbers = {2, 7, 11, 15};
        int target = 9;
        int[] result = twoSum(numbers, target);
        System.out.println "[" + result[0] + ", " + result[1] + "]"; // Output: [1, 2]
    }
}
```

24. Find All Anagrams in a String

Problem:

Given a string `s` and a non-empty string `p`, find all the start indices of `p`'s anagrams in `s`.

Example:

Input: `s = "cbaebabacd", p = "abc"`

Output: `[0, 6]`

Solution:

```
java
```

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

public class FindAllAnagrams {
    public static List<Integer> findAnagrams(String s, String p) {
        List<Integer> result = new ArrayList<>();
        if (s.length() < p.length()) return result;

        HashMap<Character, Integer> pCount = new HashMap<>();
        HashMap<Character, Integer> sCount = new HashMap<>();

        for (char c : p.toCharArray()) {
            pCount.put(c, pCount.getOrDefault(c, 0) + 1);
        }

        for (int i = 0; i < p.length(); i++) {
            sCount.put(s.charAt(i), sCount.getOrDefault(s.charAt(i), 0) + 1);
        }

        if (pCount.equals(sCount)) {
            result.add(0);
        }

        for (int i = p.length(); i < s.length(); i++) {
            sCount.put(s.charAt(i), sCount.getOrDefault(s.charAt(i), 0) + 1);
            sCount.put(s.charAt(i - p.length()), sCount.get(s.charAt(i - p.length())) -
1);
            if (sCount.get(s.charAt(i - p.length())) == 0) {
                sCount.remove(s.charAt(i - p.length()));
            }
            if (pCount.equals(sCount)) {
                result.add(i - p.length() + 1);
            }
        }

        return result;
    }

    public static void main(String[] args) {
        String s = "cbaebabacd";
        String p = "abc";
        List<Integer> result = findAnagrams(s, p);
        System.out.println(result); // Output: [0, 6]
    }
}
```

25. Find the Median of Two Sorted Arrays

Problem:

Given two sorted arrays, find the median of the two sorted arrays.

Example:

Input: `nums1 = [1, 3], nums2 = [2]`

Output: `2.0`

Solution:

java

```
public class MedianOfTwoSortedArrays {
    public static double findMedianSortedArrays(int[] nums1, int[] nums2) {
        int len1 = nums1.length;
        int len2 = nums2.length;
        int totalLen = len1 + len2;

        int left = 0, right = 0;
        int current = 0, prev = 0;

        for (int i = 0; i <= totalLen / 2; i++) {
            prev = current;

            if (left < len1 && (right >= len2 || nums1[left] < nums2[right])) {
                current = nums1[left++];
            } else {
                current = nums2[right++];
            }
        }

        return (totalLen % 2 == 0) ? (current + prev) / 2.0 : current;
    }

    public static void main(String[] args) {
        int[] nums1 = {1, 3};
        int[] nums2 = {2};
        System.out.println(findMedianSortedArrays(nums1, nums2)); // Output: 2.0
    }
}
```

These additional questions cover a variety of concepts related to arrays and are commonly asked in interviews. Let me know if you need any further assistance!

ChatGPT can make mistakes. Check important info.