

Ques 1: Merge Sort Array

The screenshot displays a LeetCode submission interface for the problem "Merge Sort Array". The submission is accepted, with 59 out of 59 testcases passed. The user MR_ARYA17 submitted it on Feb 22, 2025 at 08:40. The performance metrics show a runtime of 0 ms (beats 100.00%) and a memory usage of 12.34 MB (beats 39.45%). A bar chart indicates the runtime performance across different time intervals. The code is written in C++ and implements a merge sort algorithm. The test result shows that the solution is accepted with a runtime of 0 ms.

Runtime: 0 ms | Beats: 100.00%
Memory: 12.34 MB | Beats: 39.45%

```
class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        int i = m - 1; // Last valid element index in nums1
        int j = n - 1; // Last element index in nums2
        int k = m + n - 1; // Last index of nums1 (including extra space)

        // Merge from the back to avoid overwriting elements in nums1
        while (i >= 0 && j >= 0) {
            if (nums1[i] > nums2[j]) {
                nums1[k--] = nums1[i--];
            } else {
                nums1[k--] = nums2[j--];
            }
        }

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

Ques 2: Find Peak Element

The screenshot displays a LeetCode submission interface for the problem "Find Peak Element". The submission is accepted, with 68 out of 68 testcases passed. The user MR_ARYA17 submitted it on Feb 22, 2025 at 08:44. The performance metrics show a runtime of 0 ms (beats 100.00%) and a memory usage of 12.57 MB (beats 31.00%). A bar chart indicates the runtime performance across different time intervals. The code is written in C++ and implements a binary search algorithm to find a peak element. The test result shows that the solution is accepted with a runtime of 0 ms.

Runtime: 0 ms | Beats: 100.00%
Memory: 12.57 MB | Beats: 31.00%

```
class Solution {
public:
    int findPeakElement(vector<int>& nums) {
        int left = 0, right = nums.size() - 1;

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

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

        return left; // or return right, both will be the same
    }
};
```

Ques 3: Sort Colors

The screenshot shows a C++ solution for the 'Sort Colors' problem. The solution is accepted, with 89/89 testcases passed. The runtime is 0 ms, which is 100.00% better than the average, and the memory usage is 11.68 MB, which is 31.72% better than the average. The code uses a counting sort approach, iterating through the array and counting the occurrences of 0s, 1s, and 2s, then placing them back in sorted order.

Runtime: 0 ms | Beats 100.00%
Memory: 11.68 MB | Beats 31.72%

```
class Solution {
public:
    void sortColors(vector<int>& nums) {
        int low = 0, mid = 0, high = nums.size() - 1;

        while (mid <= high) {
            if (nums[mid] == 0) {
                swap(nums[low], nums[mid]);
                low++;
                mid++;
            }
            else if (nums[mid] == 1) {
                mid++;
            }
            else { // nums[mid] == 2
                swap(nums[mid], nums[high]);
                high--;
            }
        }
    }
};
```

Ques 4: Search a 2D matrix

The screenshot shows a C++ solution for the 'Search a 2D matrix' problem. The solution is accepted, with 130/130 testcases passed. The runtime is 51 ms, which is 69.68% better than the average, and the memory usage is 18.77 MB, which is 37.07% better than the average. The code uses a binary search approach, starting from the top-right corner and moving left or down based on the comparison with the target.

Runtime: 51 ms | Beats 69.68%
Memory: 18.77 MB | Beats 37.07%

```
bool searchMatrix(vector<vector<int>>& matrix, int target) {
    int m = matrix.size();
    int n = matrix[0].size();

    int row = 0, col = n - 1; // Start from the top-right corner

    while (row < m && col >= 0) {
        if (matrix[row][col] == target) {
            return true; // Found target
        }
        else if (matrix[row][col] > target) {
            col--; // Move left
        }
        else {
            row++; // Move down
        }
    }

    return false; // Target not found
}
```

Ques 5: Median of 2 Sorted Array

The screenshot displays a LeetCode submission for the "Median of Two Sorted Arrays" problem. The submission is marked as "Accepted" with 2096/2096 testcases passed. The user "MR_ARYA17" submitted it on Feb 22, 2025, at 09:05. The runtime is 0 ms, beating 100.00% of solutions, and the memory usage is 95.24 MB, beating 49.66%.

A bar chart shows the runtime distribution, with a peak at 3 ms. A tooltip indicates that 9.43% of solutions used 3 ms of runtime.

The C++ code is as follows:

```
class Solution {
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
        int m = nums1.size(), n = nums2.size();

        // Ensure nums1 is the smaller array
        if (m > n) return findMedianSortedArrays(nums2, nums1);

        int low = 0, high = m;
        int half = (m + n + 1) / 2;

        while (low <= high) {
            int mid1 = (low + high) / 2;
            int mid2 = half - mid1;

            int left1 = (mid1 == 0) ? INT_MIN : nums1[mid1 - 1];
            int left2 = (mid2 == 0) ? INT_MIN : nums2[mid2 - 1];
            int right1 = (mid1 == m) ? INT_MAX : nums1[mid1];
            int right2 = (mid2 == n) ? INT_MAX : nums2[mid2];

            if (left1 <= right2 && left2 <= right1) {
```

The test results show "Accepted" with a runtime of 0 ms. The input for Case 1 is "nums1 = ".