

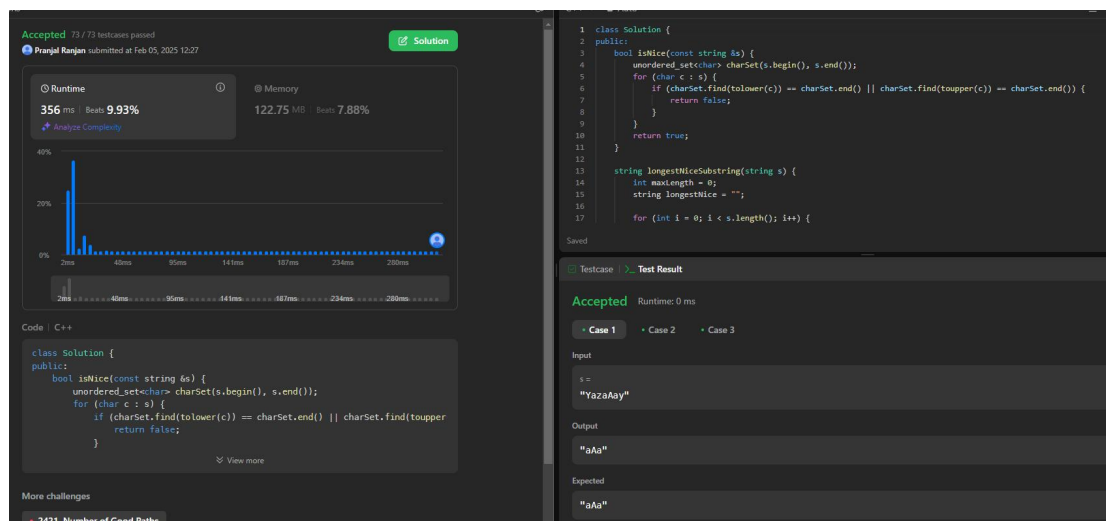
LEETCODE PROFILE - <https://leetcode.com/u/pranjalranjan899/>

1763. Longest Nice Substring

```
class Solution {
public:
    bool isNice(const string &s) {
        unordered_set<char> charSet(s.begin(), s.end());
        for (char c : s) {
            if (charSet.find(tolower(c)) == charSet.end() || charSet.find(toupper(c)) == charSet.end()) {
                return false;
            }
        }
        return true;
    }

    string longestNiceSubstring(string s) {
        int maxLength = 0;
        string longestNice = "";

        for (int i = 0; i < s.length(); i++) {
            for (int j = i + 1; j <= s.length(); j++) {
                string substring = s.substr(i, j - i);
                if (isNice(substring) && substring.length() > maxLength) {
                    maxLength = substring.length();
                    longestNice = substring;
                }
            }
        }
        return longestNice;
    }
};
```



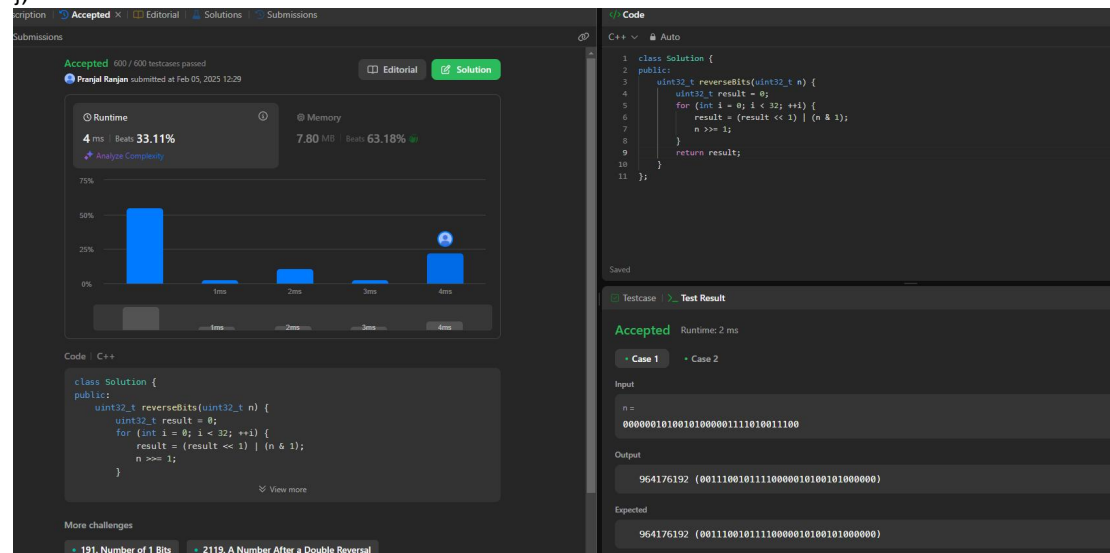
190. Reverse Bits

```
class Solution {
```

```

public:
uint32_t reverseBits(uint32_t n) {
uint32_t result = 0;
for (int i = 0; i < 32; ++i) {
result = (result << 1) | (n & 1);
n >>= 1;
}
return result;
}
};

```

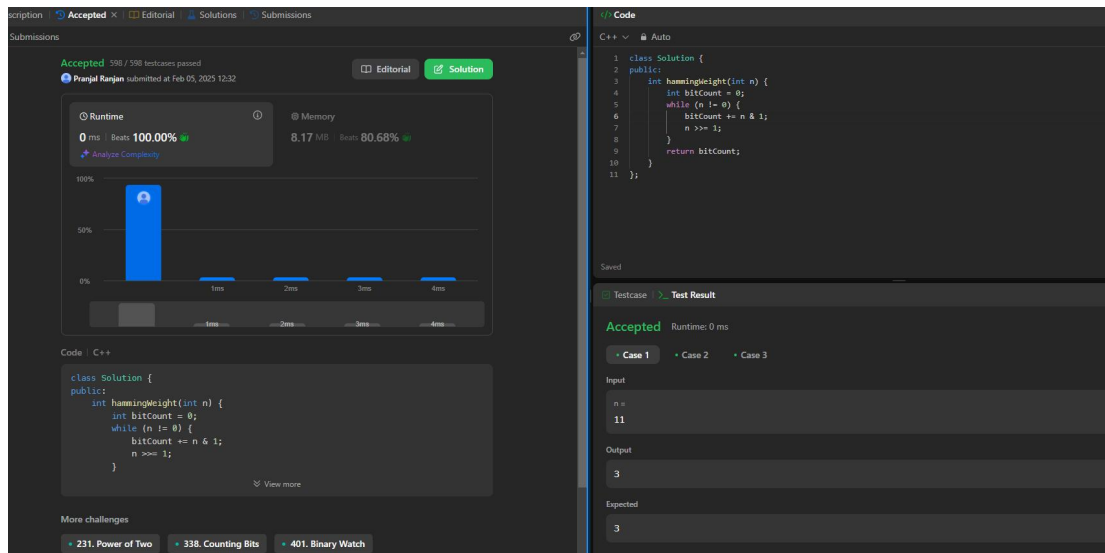


191. Number of 1 Bits

```

class Solution {
public:
int hammingWeight(int n) {
int bitCount = 0;
while (n != 0) {
bitCount += n & 1;
n >>= 1;
}
return bitCount;
}
};

```



53. Maximum Subarray

```

class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int C = nums[0];
        int G = nums[0];

```

```

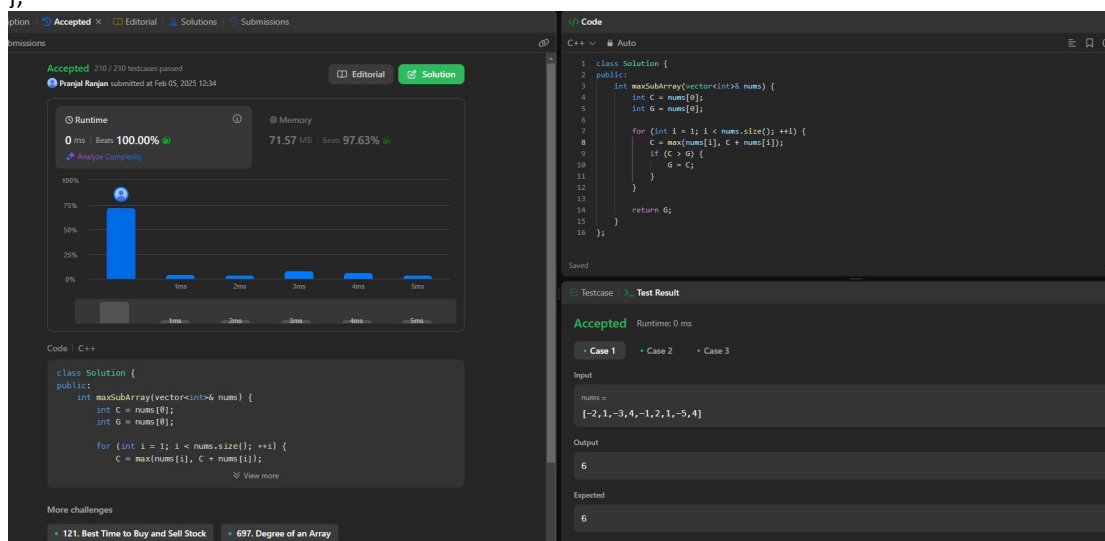
        for (int i = 1; i < nums.size(); ++i) {
            C = max(nums[i], C + nums[i]);
            if (C > G) {
                G = C;
            }
        }

```

```

        return G;
    }
};

```



240. Search a 2D Matrix II

```

class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {

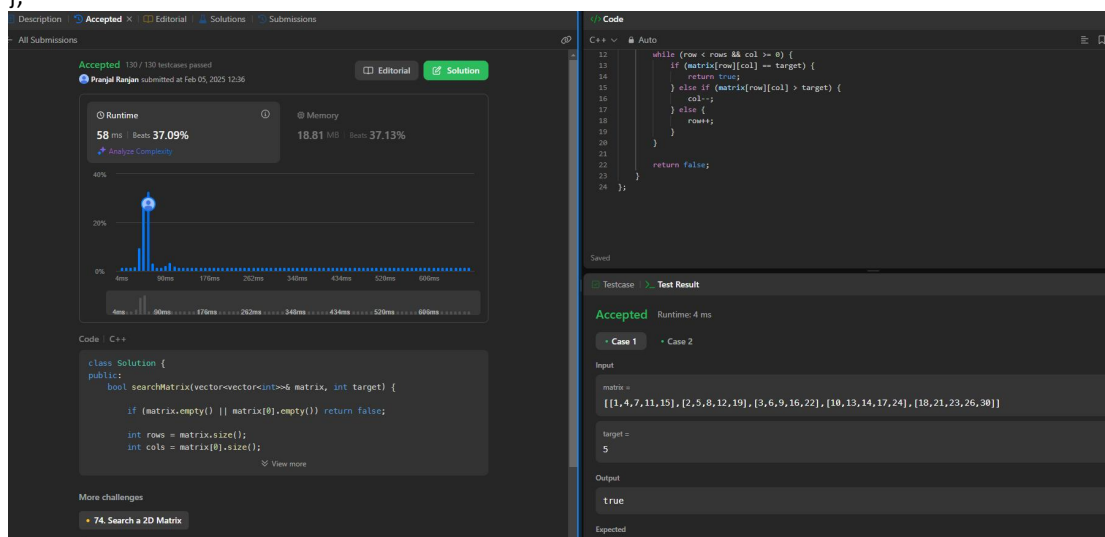
        if (matrix.empty() || matrix[0].empty()) return false;

        int rows = matrix.size();
        int cols = matrix[0].size();
        int row = 0;
        int col = cols - 1;

        while (row < rows && col >= 0) {
            if (matrix[row][col] == target) {
                return true;
            } else if (matrix[row][col] > target) {
                col--;
            } else {
                row++;
            }
        }

        return false;
    }
};

```



932. Beautiful Array

```

class Solution {
public:
    vector<int> beautifulArray(int n) {
        vector<int> result = {1};

        while (result.size() < n) {
            vector<int> temp;
            for (int num : result) {
                if (num * 2 - 1 <= n) temp.push_back(num * 2 - 1);
            }
            for (int num : result) {
                if (num * 2 <= n) temp.push_back(num * 2);
            }
        }
    }
};

```

```

result = temp;
}

return result;
}
};

```

The screenshot shows a code editor with two panes. The left pane displays the 'Accepted' status of a solution, with a runtime of 0 ms and memory usage of 10.13 MB. It also shows a bar chart of test cases. The right pane shows the C++ code for the 'beautifulArray' function, which uses a while loop to build the result array by pushing back elements from two input arrays.

88. Merge Sorted Array

```

class Solution {
public:
void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
int last = m + n - 1; // Last index of nums1
while (m > 0 && n > 0) {
if (nums1[m - 1] > nums2[n - 1]) {
nums1[last] = nums1[m - 1];
m--;
} else {
nums1[last] = nums2[n - 1];
n--;
}
last--;
}
while (n > 0) {
nums1[last] = nums2[n - 1];
n--;
last--;
}
}
};

```

The screenshot shows the LeetCode interface for problem 88, "Merge Sorted Array". The problem description states: "You are given two integer arrays `nums1` and `nums2`, sorted in non-decreasing order, and two integers `m` and `n`, representing the number of elements in `nums1` and `nums2` respectively. Merge `nums1` and `nums2` into a single array sorted in non-decreasing order. The final sorted array should not be returned by the function, but instead be stored inside the array `nums1`. To accommodate this, `nums1` has a length of `m + n`, where the first `m` elements denote the elements that should be merged, and the last `n` elements are set to 0 and should be ignored. `nums2` has a length of `n`."

Example 1:
Input: `nums1 = [1,2,3,0,0,0]`, `m = 3`, `nums2 = [2,5,6]`, `n = 3`
Output: `[1,2,2,3,5,6]`
Explanation: The arrays we are merging are `[1,2,3]` and `[2,5,6]`. The result of the merge is `[1,2,2,3,5,6]` with the underlined elements coming from `nums1`.

Example 2:
Input: `nums1 = [1]`, `m = 1`, `nums2 = []`, `n = 0`
Output: `[1]`
Explanation: The arrays we are merging are `[1]` and `[]`. The result of the merge is `[1]`.

Example 3:
Input: `nums1 = [0]`, `m = 0`, `nums2 = [1]`, `n = 1`
Output: `[1]`
Explanation: The arrays we are merging are `[]` and `[1]`. Note that because `m = 0`, there are no elements in `nums1`. The 0 is only there to ensure the merge result can fit in `nums1`.

Constraints:

The code editor on the right shows a C++ solution:

```
1 class Solution {
2 public:
3     void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
4         int last = m + n - 1; // Last index of nums1
5         while (n > 0 && m > 0) {
6             if (nums1[m - 1] > nums2[n - 1]) {
7                 nums1[last] = nums1[m - 1];
8                 m--;
9             } else {
10                nums1[last] = nums2[n - 1];
11                n--;
12            }
13            last--;
14        }
15        while (n > 0) {
16            nums1[last] = nums2[n - 1];
17            n--;
18        }
19    }
20 }
```

The test result section shows "Accepted" with a runtime of 0 ms. The input for Case 1 is:

```
nums1 = [1,2,3,0,0,0]
m = 3
nums2 = [2,5,6]
n = 3
```

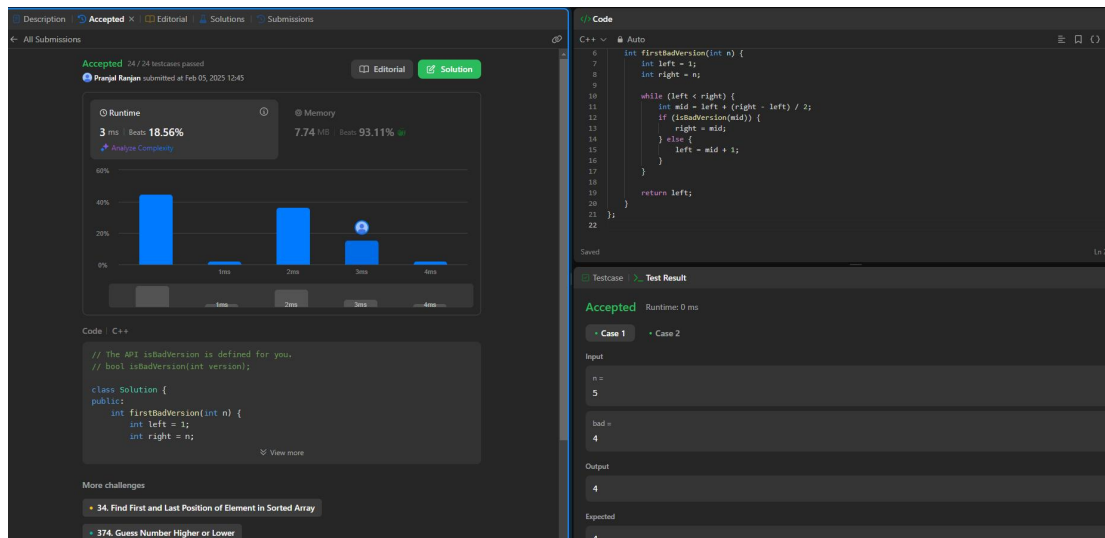
278. First Bad Version

```
// The API isBadVersion is defined for you.
// bool isBadVersion(int version);
```

```
class Solution {
public:
    int firstBadVersion(int n) {
        int left = 1;
        int right = n;

        while (left < right) {
            int mid = left + (right - left) / 2;
            if (isBadVersion(mid)) {
                right = mid;
            } else {
                left = mid + 1;
            }
        }

        return left;
    }
};
```



75. Sort Colors

```

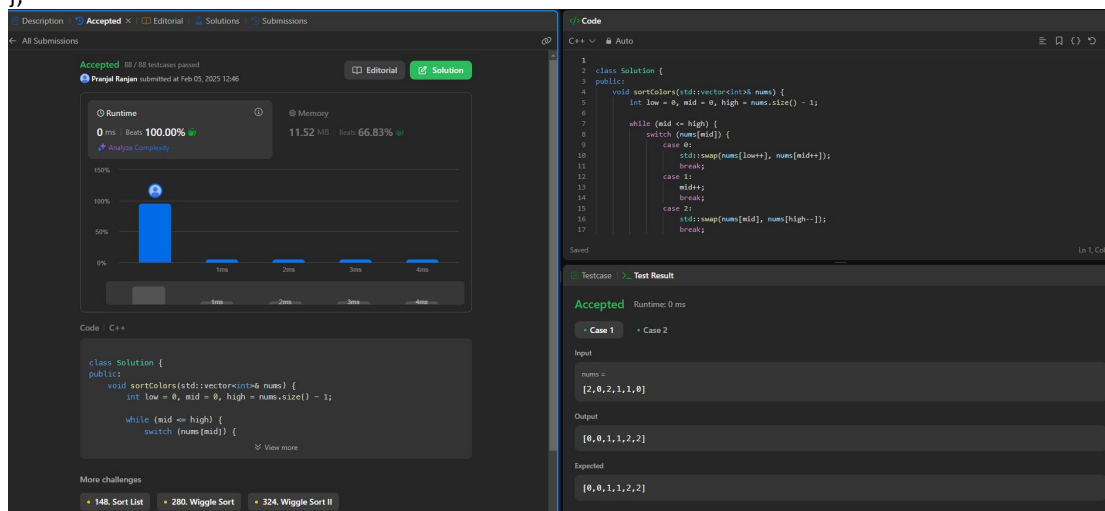
class Solution {
public:
    void sortColors(vector<int>& nums) {
        int low = 0, mid = 0, high = nums.size() - 1;

```

```

        while (mid <= high) {
            switch (nums[mid]) {
                case 0:
                    std::swap(nums[low++], nums[mid++]);
                    break;
                case 1:
                    mid++;
                    break;
                case 2:
                    std::swap(nums[mid], nums[high--]);
                    break;
            }
        }
    }
};

```



162. Find Peak Element

PRANJAL RANJAN ,22BCS13706

```

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

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

        return left;
    }
};

```

The screenshot shows a C++ IDE with a solution for finding a peak element. The code is accepted, and the test result shows the input [1, 2, 3, 1] with the output 2.

Runtime: 0 ms, Beats 100.00%, Memory: 12.52 MB, Beats 29.97%

Test Result: Accepted, Runtime: 0 ms

Case 1: Input: nums = [1, 2, 3, 1], Output: 2, Expected: 2

56. Merge Intervals

```

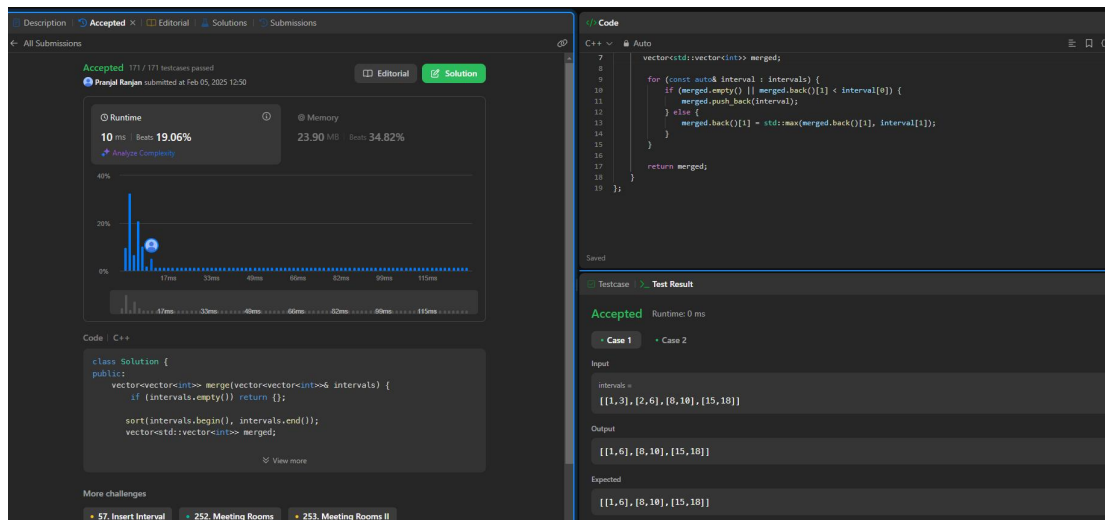
class Solution {
public:
    vector<vector<int>>> merge(vector<vector<int>>& intervals) {
        if (intervals.empty()) return {};

        sort(intervals.begin(), intervals.end());
        vector<std::vector<int>>> merged;

        for (const auto& interval : intervals) {
            if (merged.empty() || merged.back()[1] < interval[0]) {
                merged.push_back(interval);
            } else {
                merged.back()[1] = std::max(merged.back()[1], interval[1]);
            }
        }

        return merged;
    }
};

```

33. Search in Rotated Sorted Array

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

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

The screenshot displays the LeetCode submission page for the problem "Search a 2D Matrix II". The top section shows the problem description and the submission status: "Accepted" with 196/196 test cases passed. Below this, a runtime graph shows the performance of the solution. The code editor on the right contains the following C++ code:

```

class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        if (matrix.empty() || matrix[0].empty()) return false;

        int rows = matrix.size();
        int cols = matrix[0].size();
        int row = 0;
        int col = cols - 1;

        while (row < rows && col >= 0) {
            if (matrix[row][col] == target) {
                return true;
            } else if (matrix[row][col] > target) {
                col--;
            } else {
                row++;
            }
        }

        return false;
    }
};

```

The test results section shows the solution is accepted with a runtime of 0 ms and a memory usage of 15.00 MB. The input for the test case is:

```

matrix = [[4,5,6,7,8,9,1,2]]
target = 0

```

The output is 4.

240. Search a 2D Matrix II

```

class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {

```

```

        if (matrix.empty() || matrix[0].empty()) return false;

```

```

        int rows = matrix.size();
        int cols = matrix[0].size();
        int row = 0;
        int col = cols - 1;

```

```

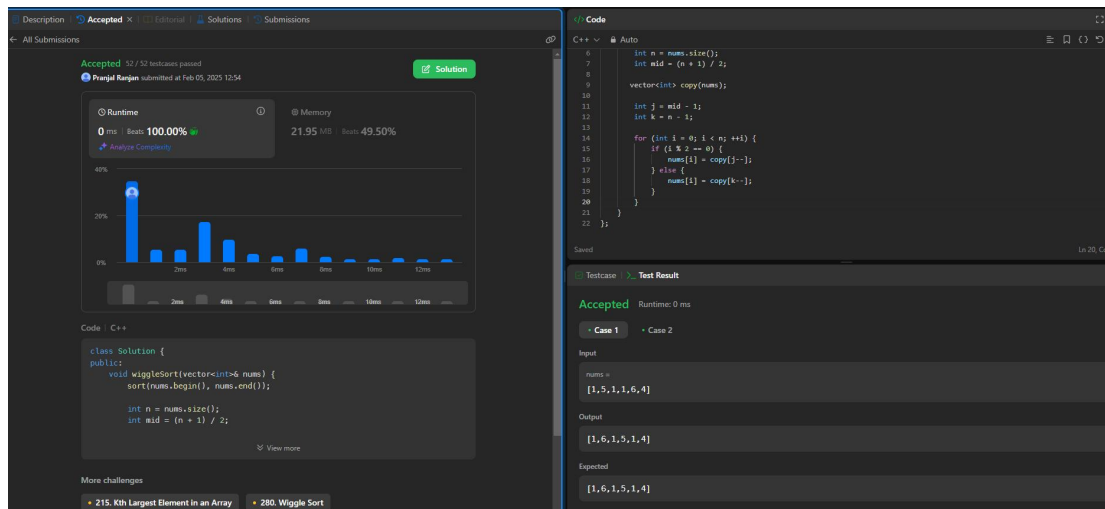
        while (row < rows && col >= 0) {
            if (matrix[row][col] == target) {
                return true;
            } else if (matrix[row][col] > target) {
                col--;
            } else {
                row++;
            }
        }

```

```

        return false;
    }
};

```



324. Wiggle Sort II

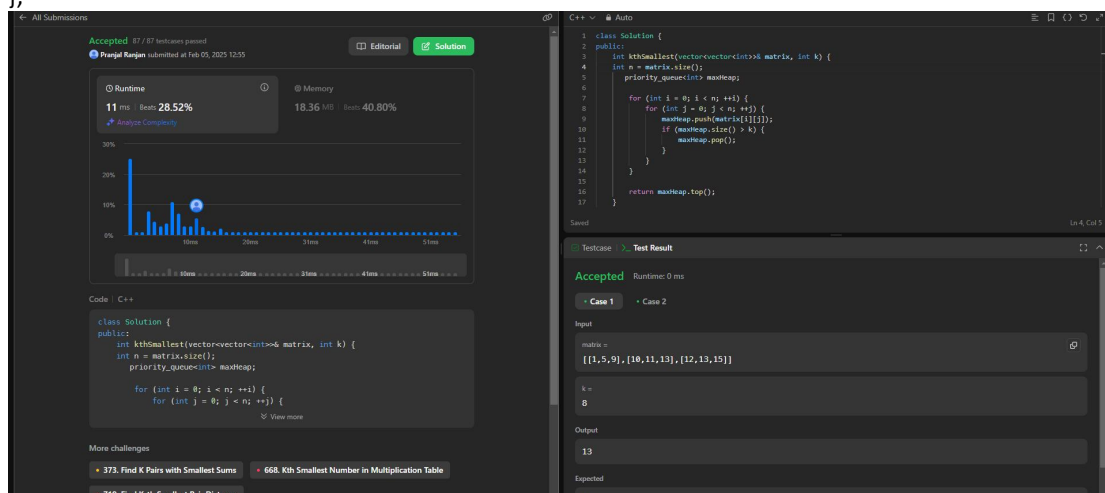
```
class Solution {
public:
    void wiggleSort(vector<int>& nums) {
        sort(nums.begin(), nums.end());;
```

```
        int n = nums.size();
        int mid = (n + 1) / 2;
```

```
        vector<int> copy(nums);
```

```
        int j = mid - 1;
        int k = n - 1;
```

```
        for (int i = 0; i < n; ++i) {
            if (i % 2 == 0) {
                nums[i] = copy[j--];
            } else {
                nums[i] = copy[k--];
            }
        }
    }
};
```



4. Median of Two Sorted Arrays

```
class Solution {
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
        if (nums1.size() > nums2.size()) {
            return findMedianSortedArrays(nums2, nums1); // Ensure nums1 is the smaller array
        }

        int m = nums1.size();
        int n = nums2.size();
        int imin = 0, imax = m, half_len = (m + n + 1) / 2;
        double max_of_left, min_of_right;

        while (imin <= imax) {
            int i = (imin + imax) / 2;
            int j = half_len - i;
            if (i < m && nums2[j - 1] > nums1[i]) {
                imin = i + 1; // i is too small
            } else if (i > 0 && nums1[i - 1] > nums2[j]) {
                imax = i - 1; // i is too large
            } else { // i is perfect
                if (i == 0) { max_of_left = nums2[j - 1]; }
                else if (j == 0) { max_of_left = nums1[i - 1]; }
                else { max_of_left = max(nums1[i - 1], nums2[j - 1]); }

                if ((m + n) % 2 == 1) { return max_of_left; }

                if (i == m) { min_of_right = nums2[j]; }
                else if (j == n) { min_of_right = nums1[i]; }
                else { min_of_right = min(nums1[i], nums2[j]); }

                return (max_of_left + min_of_right) / 2.0;
            }
        }

        return 0.0; // Should never be reached
    }
};
```

DescriptionEditorialSolutionsSubmissions

4. Median of Two Sorted Arrays

ReadTopicsCompanies

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`
- `-106 <= nums1[i], nums2[i] <= 106`

295K597420 Online

Solved

Code

```
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
// ...
else if (j == 0) { max_of_left = nums1[l - 1]; }
else { max_of_left = max(nums1[l - 1], nums2[j - 1]); }

if ((m + n) % 2 == 1) { return max_of_left; }

if (l == m) { min_of_right = nums2[j]; }
else if (j == n) { min_of_right = nums1[l]; }
else { min_of_right = min(nums1[l], nums2[j]); }

return (max_of_left + min_of_right) / 2.0;
}

return 0.0; // should never be reached
}
```

Testcase: > Test Result

Accepted Runtime: 0 ms

Case 1Case 2

Input

nums1 =

[1, 3]

nums2 =

[2]

Output

2.000000

Expected

2.000000