

1. 191. Number of 1 bits:

CODE:

```
class Solution {  
public:  
    int hammingWeight(uint32_t n) {  
        int res = 0;  
        for (int i = 0; i < 32; i++) {  
            if ((n >> i) & 1) {  
                res += 1;  
            }  
        }  
        return res;  
    }  
};
```

OUTPUT:

The screenshot shows a code editor interface with a dark theme. On the left, there's a sidebar with tabs for 'Description', 'Accepted', 'Editorial', 'Solutions', and 'Submissions'. The 'Accepted' tab is active, showing '598 / 598 testcases passed' and 'AST...' submitted at Feb 18, 2025 21:41. Below this, there's a 'Runtime' section showing '0 ms | Beats 100.00%' and a 'Memory' section showing '8.40 MB | Beats 11.88%'. On the right, the code editor shows the C++ code for the 'hammingWeight' function. The code is as follows:

```
1 class Solution {  
2 public:  
3     int hammingWeight(uint32_t n) {  
4         int res = 0;  
5         for (int i = 0; i < 32; i++) {  
6             if ((n >> i) & 1) {  
7                 res += 1;  
8             }  
9         }  
10        return res;  
11    }  
12 }  
13
```

At the bottom right of the code editor, there are 'Run' and 'Submit' buttons. Below the code editor, there's a 'Testcase' section and a 'Test Result' section.

2. 240. Search a 2D Matrix II:

CODE:

```
class Solution {
```

public:

```
bool searchMatrix(vector<vector<int>>& matrix, int target) {
```

```
    int cols = matrix[0].size() - 1;
```

```
    int n = matrix.size() - 1;
```

```
    int rows = 0;
```

```
    while(rows <= n && cols >= 0){
```

```
        int toCompare = matrix[rows][cols];
```

```
        if(toCompare > target){
```

```
            cols--;
```

```
        }else if(toCompare < target){
```

```
            rows++;
```

```
        }else{
```

```
            return true;
```

```
        }
```

```
    }
```

```
    return false;
```

```
}
```

```
};
```

OUTPUT:

The screenshot displays a code editor interface with a dark theme. On the left, a sidebar shows submission details: 'Accepted 130 / 130 testcases passed', 'AST...' submitted at Feb 18, 2025 21:42, and buttons for 'Editorial' and 'Solution'. Below this, performance metrics are shown: 'Runtime 39 ms | Beats 95.40%' and 'Memory 18.81 MB | Beats 37.15%'. A small bar chart at the bottom left indicates the user's performance relative to others. The main editor area shows the C++ code for the 'searchMatrix' function, which is identical to the code provided in the previous blocks. The code is line-numbered from 7 to 21. At the bottom right, there are 'Run' and 'Submit' buttons. Below the code editor, there are sections for 'Testcase' and 'Test Result'.

3. 53.Max Subarray:

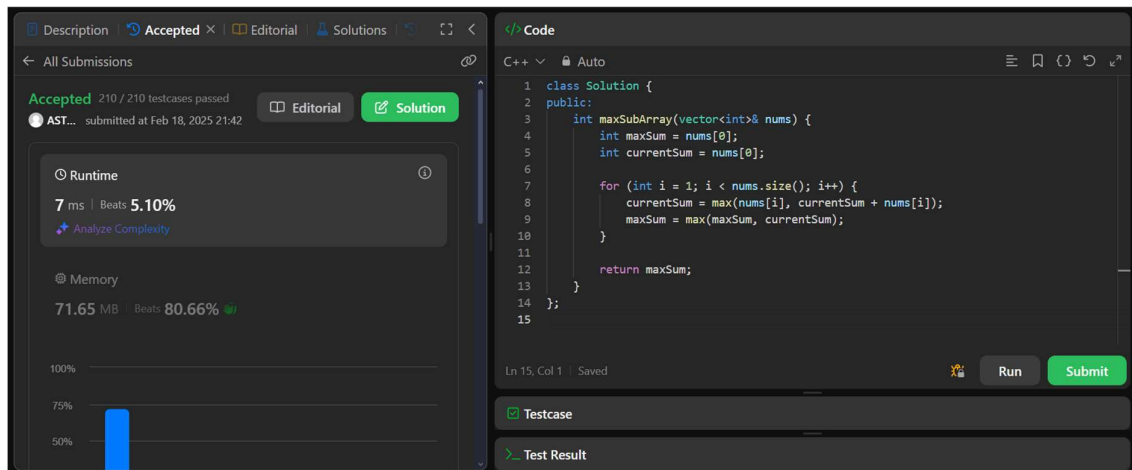
CODE:

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int maxSum = nums[0];
        int currentSum = nums[0];

        for (int i = 1; i < nums.size(); i++) {
            currentSum = max(nums[i], currentSum + nums[i]);
            maxSum = max(maxSum, currentSum);
        }

        return maxSum;
    }
};
```

OUTPUT:



The screenshot displays a code editor interface for the 'Max Subarray' problem. The left sidebar shows the submission status as 'Accepted' with 210/210 testcases passed. The runtime is 7 ms, beating 5.10% of solutions, and the memory usage is 71.65 MB, beating 80.66%. The right pane shows the C++ code for the solution, which is the same as the code provided in the previous block. The code is written in C++ and uses a single loop to calculate the maximum subarray sum using Kadane's algorithm. The code is saved and ready to be run or submitted.

4. 56. Merge Intervals:

CODE:

```

class Solution {
public:
    vector<vector<int>> merge(vector<vector<int>>& intervals) {

        sort(intervals.begin(), intervals.end());

        int k = 0;

        for (int i = 1; i < intervals.size(); i++) {

            if (intervals[k][1] >= intervals[i][0]) {

                intervals[k][1] = max(intervals[k][1], intervals[i][1]);

            } else {

                k++;

                intervals[k] = intervals[i];

            }

        }

        intervals.resize(k + 1);

        return intervals;

    }
};

```

OUTPUT

The screenshot displays a code editor interface with the following components:

- Left Sidebar (Submission Details):**
 - Tab: **Accepted** (171 / 171 testcases passed)
 - User: **AST...** (submitted at Feb 18, 2025 21:43)
 - Buttons: **Editorial**, **Solution**
 - Runtime:** 4 ms | Beats 70.28%
 - Memory:** 23.35 MB | Beats 98.38%
 - Graph: A bar chart showing performance comparison against other submissions.
- Main Editor (Code):**

```

6
7
8     for (int i = 1; i < intervals.size(); i++) {
9         if (intervals[k][1] >= intervals[i][0]) {
10            intervals[k][1] = max(intervals[k][1], intervals[i][1]);
11        } else {
12            k++;
13            intervals[k] = intervals[i];
14        }
15    }
16    intervals.resize(k + 1);
17    return intervals;
18
19 }
20

```
- Bottom Right:**
 - Ln 20, Col 1 | Saved
 - Buttons: **Run**, **Submit**
 - Testcase: **Testcase**
 - Test Result: **Test Result**

5.493. [Reverse Pairs](#)

CODE:

```
class Solution {
public:
    int mergeAndCount(vector<int>& nums, int left, int mid, int right) {
        int count = 0, j = mid + 1;

        for (int i = left; i <= mid; i++) {
            while (j <= right && nums[i] > 2LL * nums[j]) {
                j++;
            }
            count += (j - (mid + 1));
        }

        vector<int> temp;
        int i = left, k = mid + 1;
        while (i <= mid && k <= right) {
            if (nums[i] <= nums[k]) {
                temp.push_back(nums[i++]);
            } else {
                temp.push_back(nums[k++]);
            }
        }
        while (i <= mid) temp.push_back(nums[i++]);
        while (k <= right) temp.push_back(nums[k++]);

        for (int i = left; i <= right; i++) {
            nums[i] = temp[i - left];
        }

        return count;
    }
};
```

```

    }

    int mergeSortAndCount(vector<int>& nums, int left, int right) {
        if (left >= right) return 0;

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

        int count = mergeSortAndCount(nums, left, mid) +
                    mergeSortAndCount(nums, mid + 1, right) +
                    mergeAndCount(nums, left, mid, right);

        return count;
    }

    int reversePairs(vector<int>& nums) {
        return mergeSortAndCount(nums, 0, nums.size() - 1);
    }
};

```

OUTPUT:

The screenshot displays a code editor interface for a C++ solution. On the left, a sidebar provides submission statistics: 'Accepted 140 / 140 testcases passed', 'AST...' submitted at Feb 18, 2025 21:46, Runtime 596 ms (Beats 6.79%), and Memory 243.52 MB (Beats 15.22%). The main editor area shows the C++ code for the 'Reverse Pairs' problem, including the `mergeSortAndCount`, `mergeAndCount`, and `reversePairs` functions. The code is written in C++ and uses a divide-and-conquer approach to count reverse pairs. The bottom right of the editor has 'Run' and 'Submit' buttons.

6. 88. Merge Sorted Array

CODE:

```
class Solution {
```

```

public:

    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {

        for(int i=0; i<n; i++){

            nums1[m+i] = nums2[i];

        }

        sort(nums1.begin(), nums1.end());

    }

};

```

OUTPUT:

The screenshot shows the LeetCode submission interface for the 'Merge Sorted Array' problem. The submission is 'Accepted' with 59/59 testcases passed. The runtime is 0 ms, beating 100.00% of solutions. The memory usage is 12.34 MB, beating 39.44% of solutions. The code is a C++ class Solution with a merge function that copies elements from nums2 into nums1 and then sorts the result.

```

1 class Solution {
2 public:
3     void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
4         for(int i=0; i<n; i++){
5             nums1[m+i] = nums2[i];
6         }
7         sort(nums1.begin(), nums1.end());
8     }
9 };
10

```

7. 347. Top K Frequent Element:

CODE:

```

class Solution {
public:

    vector<int> topKFrequent(vector<int>& nums, int k) {

        int n = nums.size();

        unordered_map<int, int> map;

        vector<int> ans;

        for (int &x : nums) map[x]++;

        vector<vector<int>>> arr(n + 1);

        for (auto [a, b] : map) arr[b].push_back(a);

    }

};

```

```

        for (int i = n; i > 0; i--) {
            for (int &x : arr[i]) {
                if (ans.size() == k) return ans;
                ans.push_back(x);
            }
        }
        return ans;
    }
};

```

OUTPUT:

The screenshot shows a code editor interface with two main panels. The left panel displays the submission status as 'Accepted' with 21/21 testcases passed. It shows the runtime as 0 ms and memory as 20.04 MB, both with 100.00% efficiency. The right panel shows the C++ code for finding the kth largest element in an array using a map and a vector.

Runtime Performance:

- Status: Accepted
- Testcases: 21 / 21 passed
- Submitted: Feb 18, 2025 21:48
- Runtime: 0 ms | Beats 100.00%
- Memory: 20.04 MB | Beats 6.27%

Code:

```

5 unordered_map<int, int> map;
6 vector<int> ans;
7 for (int &x : nums) map[x]++;
8 vector<vector<int>> arr(n + 1);
9 for (auto [a, b] : map) arr[b].push_back(a);
10 for (int i = n; i > 0; i--) {
11     for (int &x : arr[i]) {
12         if (ans.size() == k) return ans;
13         ans.push_back(x);
14     }
15 }
16 return ans;
17 };
18 };
19

```

8. 215. Kth Largest Element in an Array:

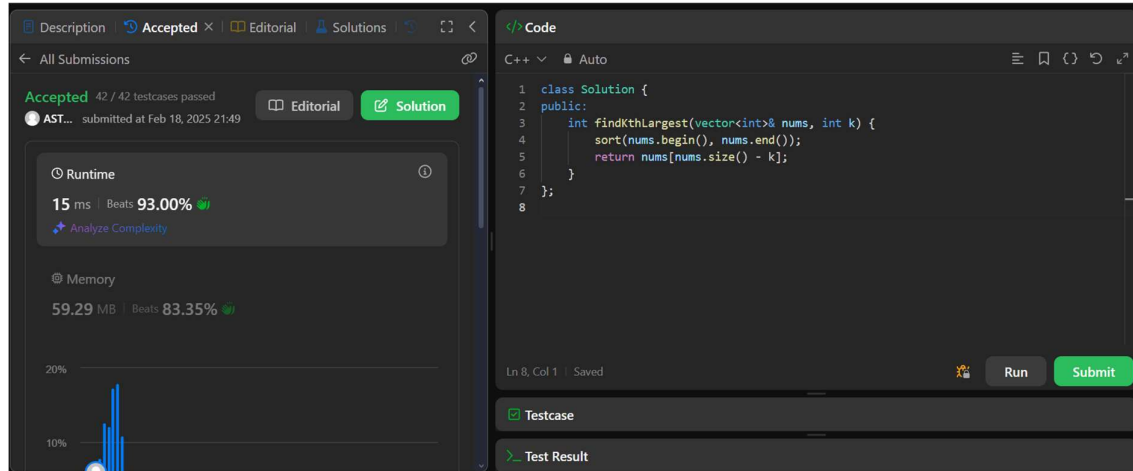
CODE:

```

class Solution {
public:
    int findKthLargest(vector<int>& nums, int k) {
        sort(nums.begin(), nums.end());
        return nums[nums.size() - k];
    }
};

```


OUTPUT:



The screenshot shows a code editor interface with a dark theme. On the left, a sidebar displays submission statistics: "Accepted 42 / 42 testcases passed", "AST..." submitted at Feb 18, 2025 21:49, and a "Solution" button. Below this, a "Runtime" section shows "15 ms" and "Beats 93.00%", and a "Memory" section shows "59.29 MB" and "Beats 83.35%". A small bar chart is visible at the bottom of the sidebar. The main editor area shows a C++ code snippet for a class named "Solution" with a method "findKthLargest". The code sorts the input array and returns the kth largest element. At the bottom right, there are "Run" and "Submit" buttons. Below the code editor, there are tabs for "Testcase" and "Test Result".

9. 33. Search in Rotated Sorted Array:

CODE:

```
class Solution {
public:
    int search(vector<int>& nums, int target) {
        int n = nums.size();

        int i = 0;
        while (i < n - 1 && nums[i] < nums[i + 1]) i++;

        int low, high;
        if (target >= nums[0]) low = 0, high = i;
        else low = i + 1, high = n - 1;

        while (low <= high) {
            int mid = (low + high) / 2;

            if (nums[mid] == target) return mid;
            else if (nums[mid] > target) high = mid - 1;
            else low = mid + 1;
        }
    }
};
```

```

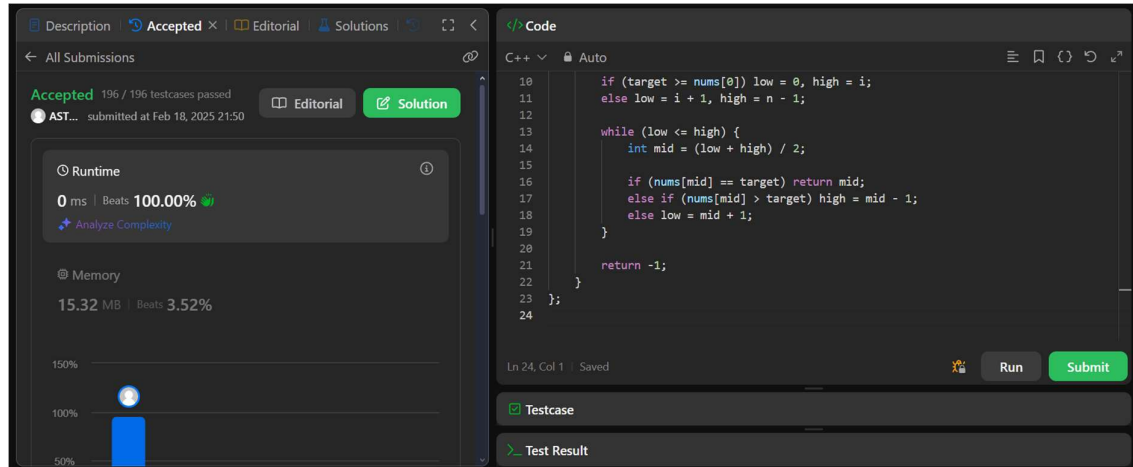
    }

    return -1;
}

};

```

OUTPUT:



10. [162. Find Peak Element:](#)

CODE:

```

class Solution {
public:
    int findPeakElement(vector<int>& nums) {
        if(nums.size() == 1) return 0;
        int l = 1;
        int h = nums.size() - 2;
        if(nums[0] > nums[1]) return 0;
        if(nums[nums.size() - 1] > nums[nums.size() - 2]) return nums.size() - 1;
        while(l <= h){
            int mid = (l+h)/2;

            if(nums[mid] > nums[mid+1] && nums[mid] > nums[mid-1]){

```

```

        return mid;
    }
    else if(nums[mid] < nums[mid+1]){
        l = mid + 1;
    } else {
        h = mid - 1;
    }
}
return -1;
}
};

```

OUTPUT:

The screenshot displays a code editor interface for a C++ solution. The left sidebar provides submission statistics: **Accepted** (68 / 68 testcases passed), **Runtime** (0 ms, 100.00% beats), and **Memory** (12.36 MB, 91.14% beats). The main editor shows the following C++ code:

```

10     int mid = (l+h)/2;
11
12     if(nums[mid] > nums[mid+1] && nums[mid] > nums[mid-1]){
13         return mid;
14     }
15     else if(nums[mid] < nums[mid+1]){
16         l = mid + 1;
17     } else {
18         h = mid - 1;
19     }
20 }
21 return -1;
22 }
23 };
24

```

The right sidebar includes a **Testcase** section and a **Test Result** section, along with **Run** and **Submit** buttons.