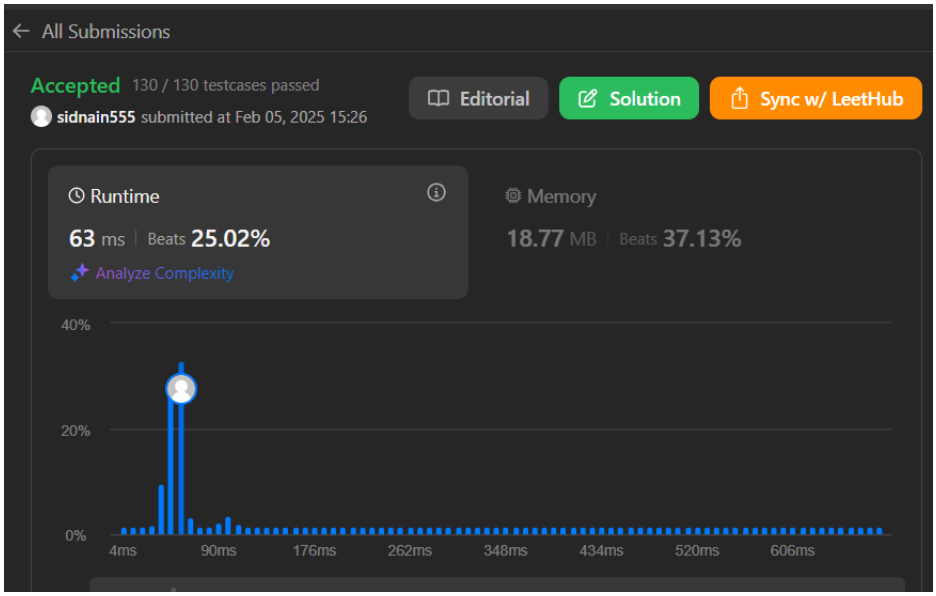
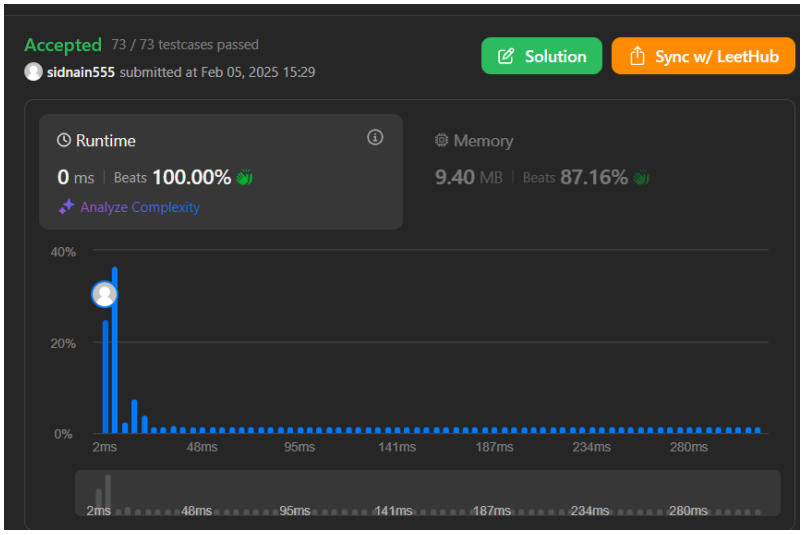
1. [Search a 2D Matrix II - LeetCode]




```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int row = matrix.size();
        int col = matrix[0].size();
        int rowIndex = 0;
        int colIndex = col-1;
        while(rowIndex<row&&colIndex >=0){
            int ele = matrix[rowIndex][colIndex];
            if(ele == target ){
                return 1;
            }
            if(ele < target){
                rowIndex++;
            }
            else{
                colIndex--;
            }
        }
```

```
        return 0;
    }
};
```

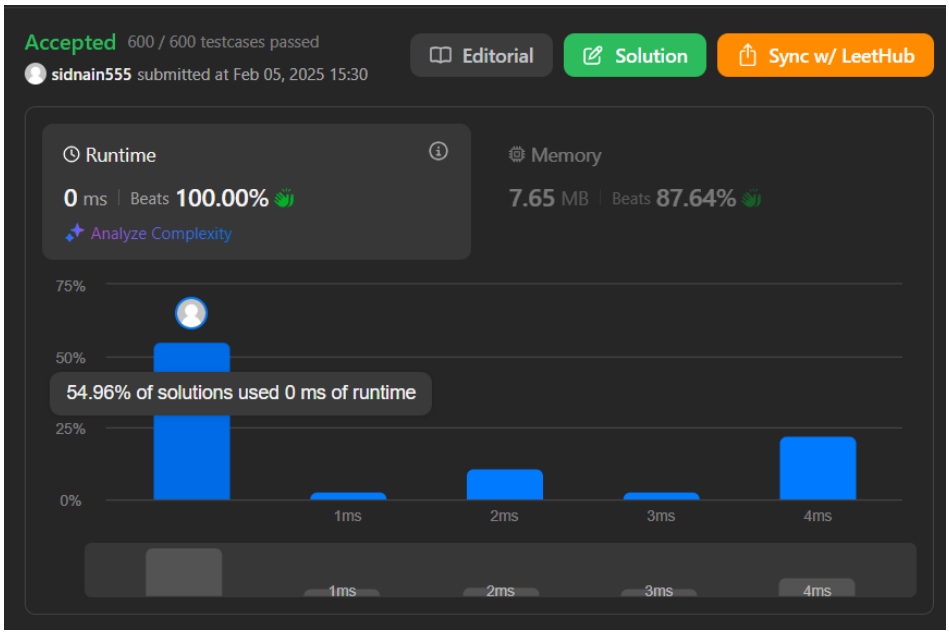
2. [Longest Nice Substring - LeetCode]




```
class Solution {
public:
    string longestNiceSubstring(string s, int start = 0, int end = -1) {
    if (end == -1)
        end = s.size();
    int cnt[26][2] = {}, j = start - 1;
    for (auto i = start; i < end; ++i)
        cnt[tolower(s[i]) - 'a'][(bool)islower(s[i])] = 1;
    string res;
    for (auto i = start; i <= end; ++i) {
        int ch = i == end ? -1 : tolower(s[i]) - 'a';
        if (ch == -1 || cnt[ch][0] + cnt[ch][1] == 1) {
            if (j == -1 && ch == -1)
                return s;
            auto res1 = longestNiceSubstring(s.substr(j + 1, i - j - 1));
            if (res1.size() > res.size())
```

```
            res = res1;
        j = i;
      }
    }
    return res;
}
};
```

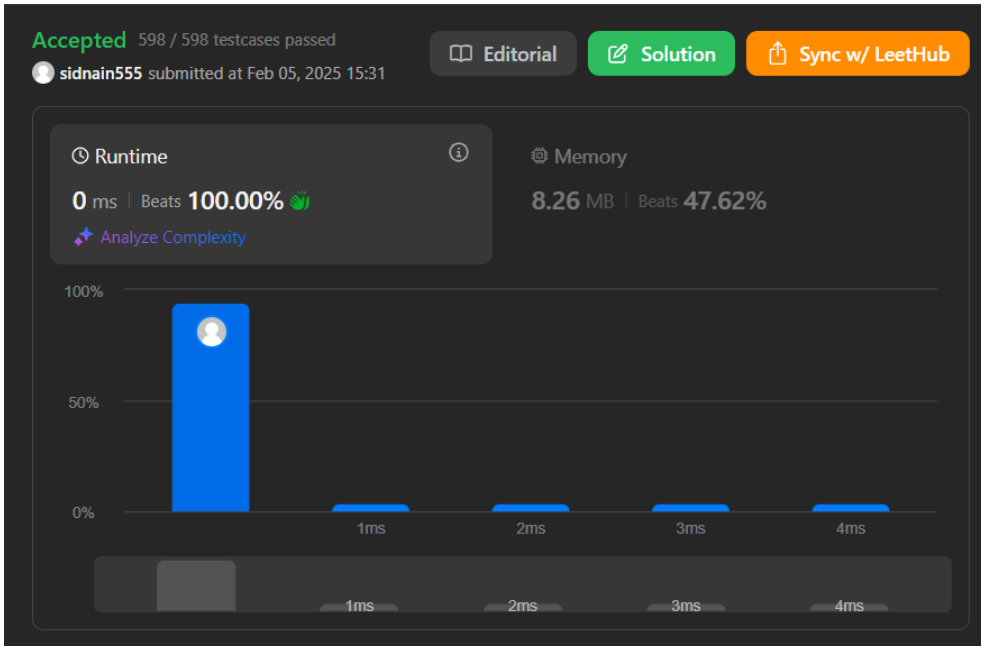
3. Reverse Bits - LeetCode




```
class Solution {
public:
    uint32_t reverseBits(uint32_t n) {
        n = ((n & 0xffff0000) >> 16) | ((n & 0x0000ffff) << 16);
        n = ((n & 0xff00ff00) >> 8) | ((n & 0x00ff00ff) << 8);
        n = ((n & 0xf0f0f0f0) >> 4) | ((n & 0x0f0f0f0f) << 4);
        n = ((n & 0xcccccccc) >> 2) | ((n & 0x33333333) << 2);
        n = ((n & 0xaaaaaaaa) >> 1) | ((n & 0x55555555) << 1);

        return n;
    }
```

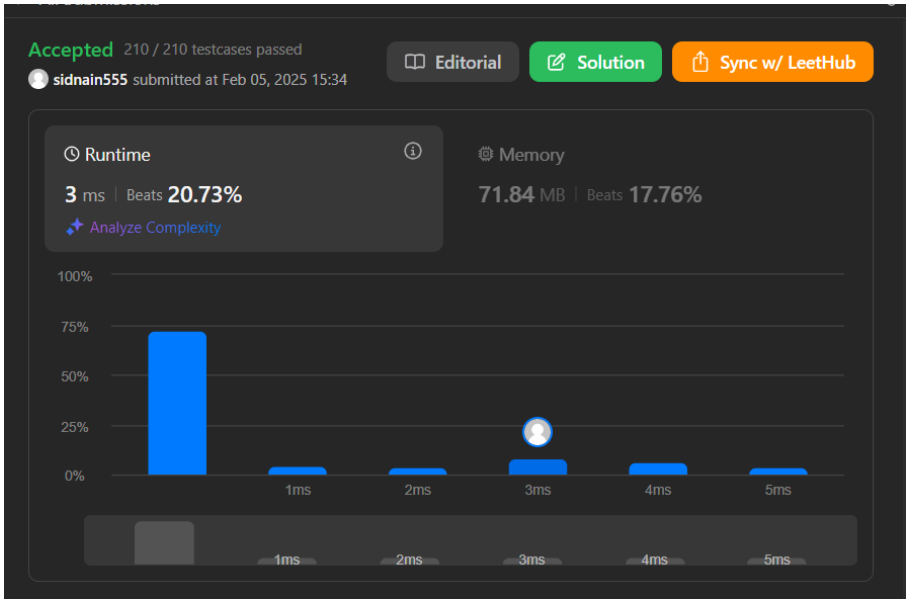
};

4. Number of 1 Bits - LeetCode



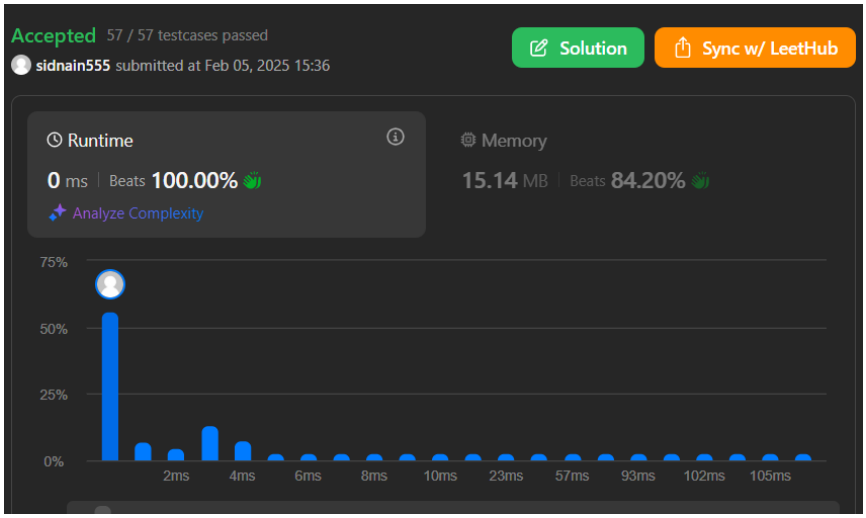

```
class Solution {
public:
    int hammingWeight(int n) {
        int res = 0;
        for (int i = 0; i < 32; i++) {
            if ((n >> i) & 1) {
                res += 1;
            }
        }
        return res;
    }
};
```

5. [Maximum Subarray - LeetCode]




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

        for (int n : nums) {
            if (total < 0) {
                total = 0;
            }

            total += n;
            res = max(res, total);
        }

        return res;
    }
};
```

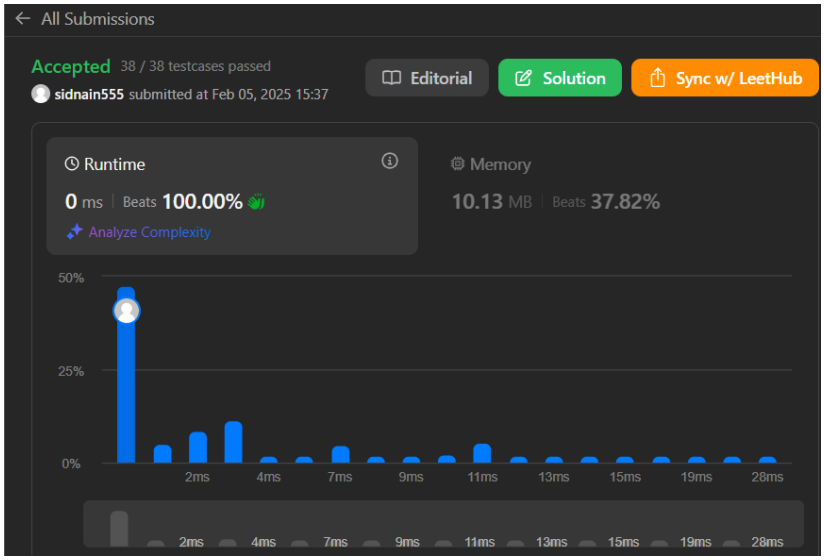
6. [Super Pow - LeetCode]




```
class Solution {
private:
    int solve(int base, int power, int mod) {
        int ans = 1;
        while (power > 0) {
            if (power & 1) {
                ans = (ans * base) % mod;
            }
            base = (base * base) % mod;
            power >>= 1;
        }
        return ans;
    }

public:
    int superPow(int a, vector<int>& b) {
        a%=1337;
        int n = b.size();
        int m = 1140;
        int expi = 0;
        for(int i : b){
```

```
            expi = (expi*10+i)%m;
        }
        if (expi == 0) {
            expi = m;
        }
        return solve(a,expi,1337);
    }
};
```

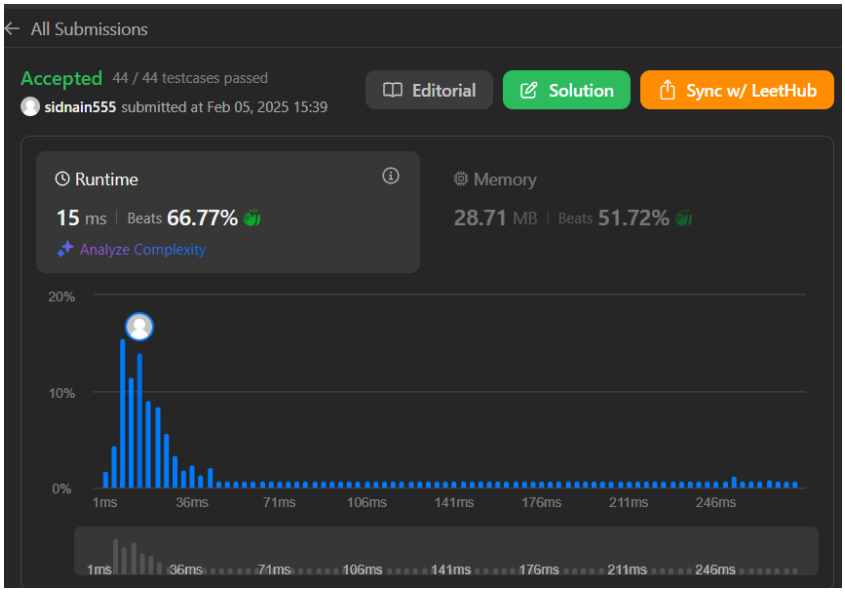
7. [Beautiful Array - LeetCode]




```
class Solution {
public:
  static bool comp(const int &a, const int &b){
    int mask = 1;
    while(true)
      if((a&mask) == (b&mask)) mask = mask<<1;
      else return (a&mask) > (b&mask);
  }

  vector<int> beautifulArray(int n) {
```

```
    vector<int> answer;
    while(n) answer.push_back(n--);

    sort(answer.begin(), answer.end(), comp);

    return answer;
  }
};
```

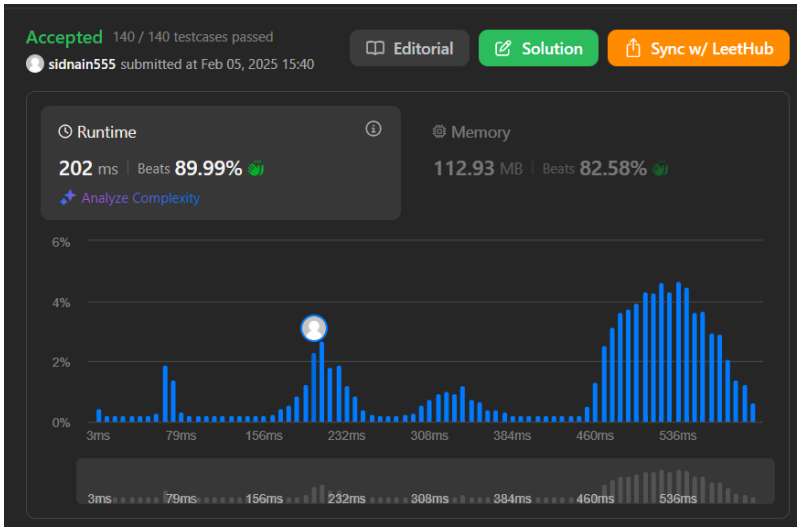
8. [The Skyline Problem - LeetCode]




```
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
        vector<vector<int>> ans;
        multiset<int> pq{0};

        vector<pair<int, int>> points;

        for(auto b: buildings){
            points.push_back({b[0], -b[2]});
```

```
            points.push_back({b[1], b[2]});
        }

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

        int ongoingHeight = 0;

        for(int i = 0; i < points.size(); i++){
            int currentPoint = points[i].first;
            int heightAtCurrentPoint = points[i].second;

            if(heightAtCurrentPoint < 0){
                pq.insert(-heightAtCurrentPoint);
            } else {
                pq.erase(pq.find(heightAtCurrentPoint));
            }

            auto pqTop = *pq.rbegin();
            if(ongoingHeight != pqTop){
                ongoingHeight = pqTop;
                ans.push_back({currentPoint, ongoingHeight});
            }
        }

        return ans;
    }
};
```

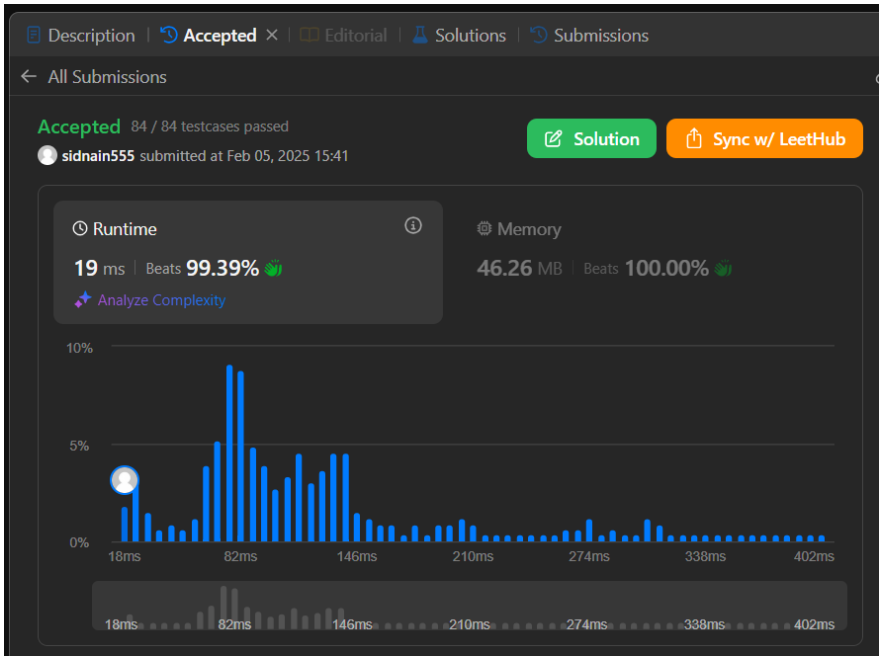
9. [Reverse Pairs - LeetCode]




```
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
        vector<vector<int>> ans;
        multiset<int> pq{0};

        vector<pair<int, int>> points;

        for(auto b: buildings){
            points.push_back({b[0], -b[2]});
            points.push_back({b[1], b[2]});
        }

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

        int ongoingHeight = 0;

        for(int i = 0; i < points.size(); i++){
            int currentPoint = points[i].first;
            int heightAtCurrentPoint = points[i].second;
```

```
            if(heightAtCurrentPoint < 0){
                pq.insert(-heightAtCurrentPoint);
            } else {
                pq.erase(pq.find(heightAtCurrentPoint));
            }

            auto pqTop = *pq.rbegin();
            if(ongoingHeight != pqTop){
                ongoingHeight = pqTop;
                ans.push_back({currentPoint, ongoingHeight});
            }
        }

        return ans;
    }
};
```

10. [Longest Increasing Subsequence II - LeetCode]

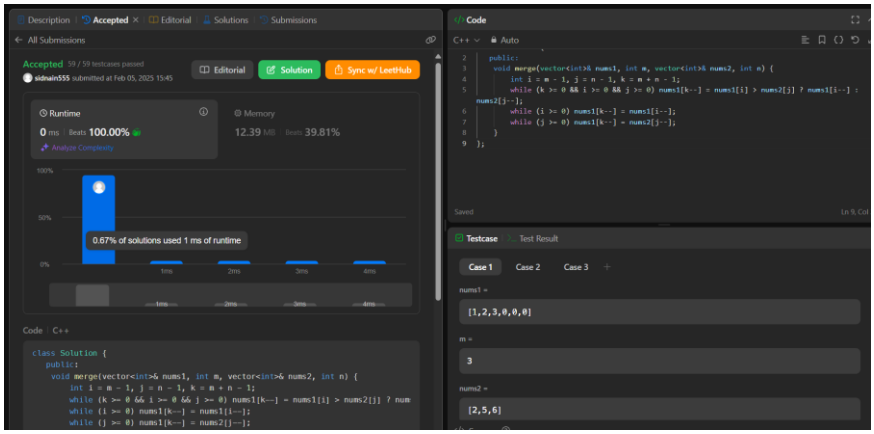



```
constexpr int N = 100001;

class Solution {
```

```
public:
    array<int, 2*N> seg{};

    void update(int pos, int val){
        pos += N;
        seg[pos] = val;

        while (pos > 1) {
            pos >>= 1;
            seg[pos] = max(seg[2*pos], seg[2*pos+1]);
        }
    }

    int query(int lo, int hi){
        lo += N;
        hi += N;
        int res = 0;

        while (lo < hi) {
            if (lo & 1) {
                res = max(res, seg[lo++]);
            }
            if (hi & 1) {
                res = max(res, seg[--hi]);
            }
            lo >>= 1;
            hi >>= 1;
        }
        return res;
    }
```

```
    int lengthOfLIS(vector<int>& A, int k) {
        int ans = 0;
        for (int i = 0; i < size(A); ++i){
            int l = max(0, A[i]-k);
            int r = A[i];
            int res = query(l, r) + 1; ans = max(res, ans);
            update(A[i], res);
        }

        return ans;
    }
};
```

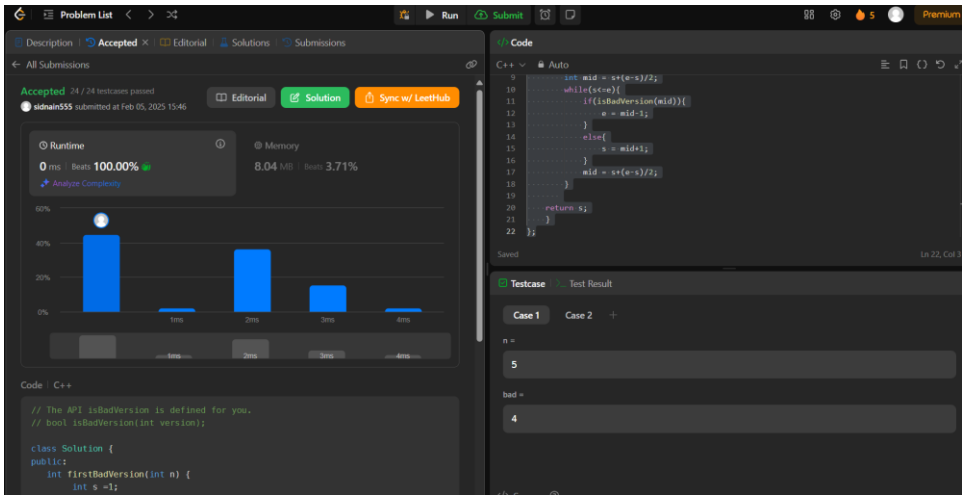
11. [Merge Sorted Array - LeetCode]




```
class Solution {
public:
    void merge(int* nums1, int nums1Size, int m, int* nums2, int nums2Size, int n) {
    int ptr1 = m - 1, ptr2 = n - 1;
    int indexPtr = nums1Size - 1;

    while (ptr2 >= 0) {
        if (ptr1 >= 0 && nums1[ptr1] > nums2[ptr2]) {
            nums1[indexPtr--] = nums1[ptr1--];
```

```
        } else {
            nums1[indexPtr--] = nums2[ptr2--];
        }
    }
}

};
```

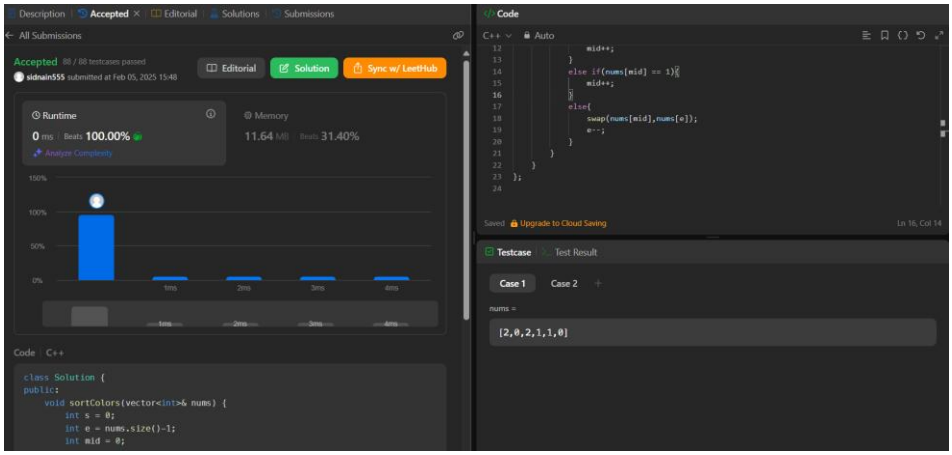
12. [First Bad Version - LeetCode]




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


class Solution {
public:
    int firstBadVersion(int n) {
        int s =1;
        int e = n;
        int mid = s+(e-s)/2;
        while(s<=e){
            if(isBadVersion(mid)){
                e = mid-1;
            }
```

```
            else{
                s = mid+1;
            }
            mid = s+(e-s)/2;
        }

    return s;
    }
};
```

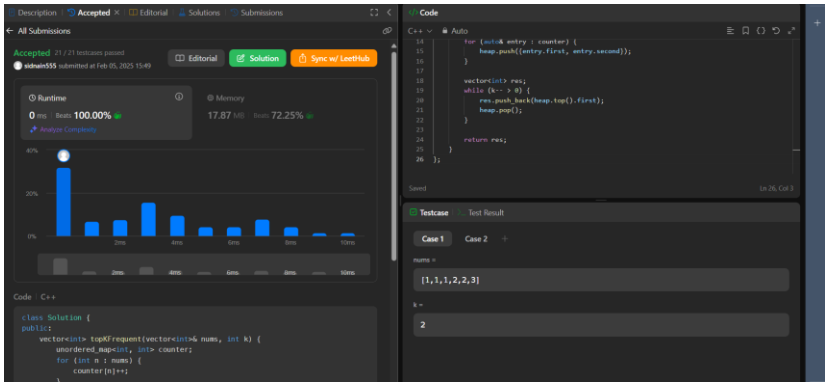
13. [Sort Colors - LeetCode]




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

        while(mid<=e){
            if(nums[mid] == 0){
                swap(nums[s],nums[mid]);
                s++;
                mid++;
```

```
        }
        else if(nums[mid] == 1){
            mid++;
        }
        else{
            swap(nums[mid],nums[e]);
            e--;
        }
    }
  }
};
```

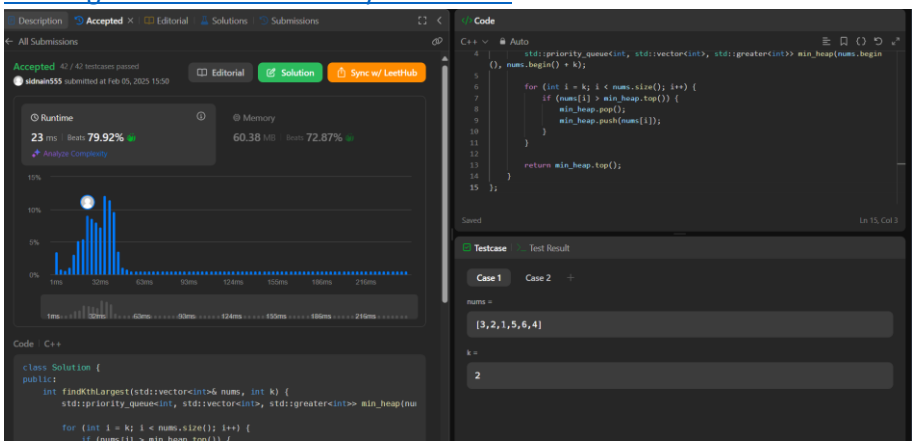
14. Top K Frequent Elements - LeetCode



```
class Solution {
public:
    vector<int> topKFrequent(vector<int>& nums, int k) {
        unordered_map<int, int> counter;
        for (int n : nums) {
            counter[n]++;
        }

        auto comp = [](pair<int, int>& a, pair<int, int>& b) {
            return a.second < b.second;
        };
```

```
        priority_queue<pair<int, int>, vector<pair<int, int>>, decltype(comp)> heap(comp);

        for (auto& entry : counter) {
            heap.push({entry.first, entry.second});
        }

        vector<int> res;
        while (k-- > 0) {
            res.push_back(heap.top().first);
            heap.pop();
        }

        return res;
    }
};
```

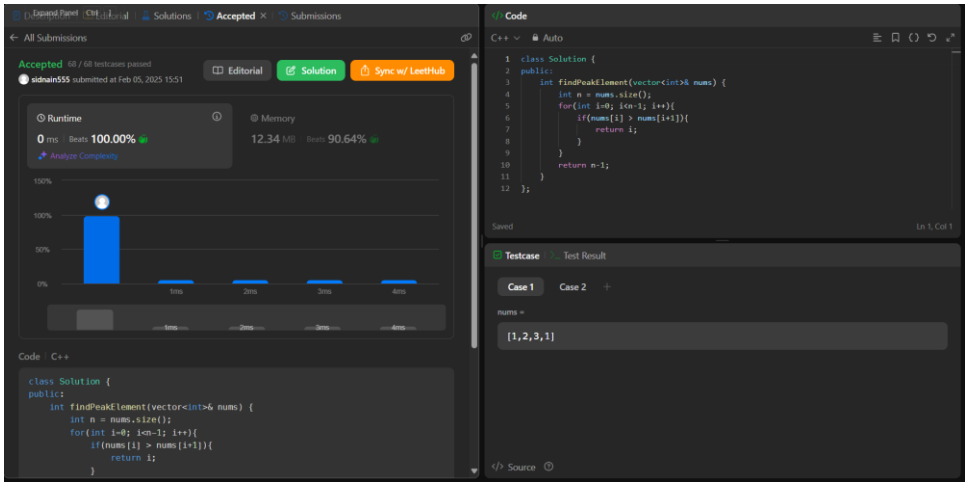
15. [Kth Largest Element in an Array - LeetCode]




```
class Solution {
public:
    int findKthLargest(std::vector<int>& nums, int k) {
        std::priority_queue<int, std::vector<int>, std::greater<int>> min_heap(nums.begin(), nums.begin() + k);

        for (int i = k; i < nums.size(); i++) {
```

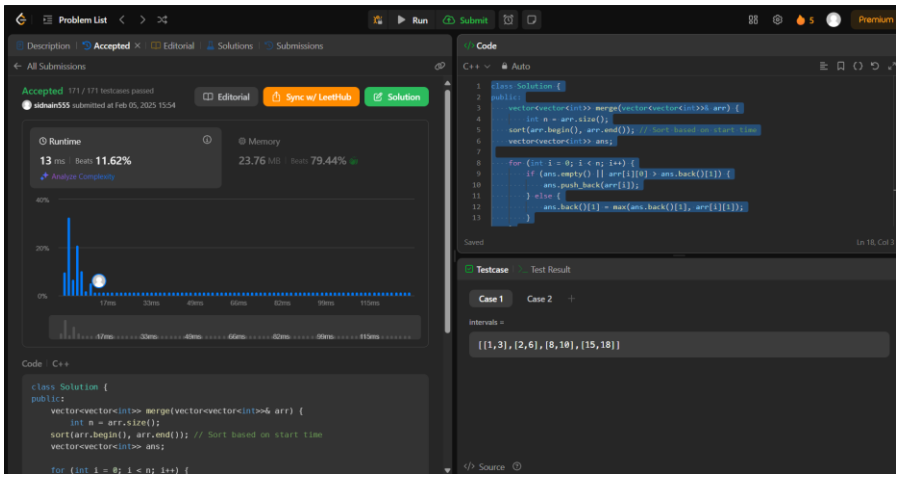
```
            if (nums[i] > min_heap.top()) {
                min_heap.pop();
                min_heap.push(nums[i]);
            }
        }

        return min_heap.top();
    }
};
```

16. [Find Peak Element - LeetCode]



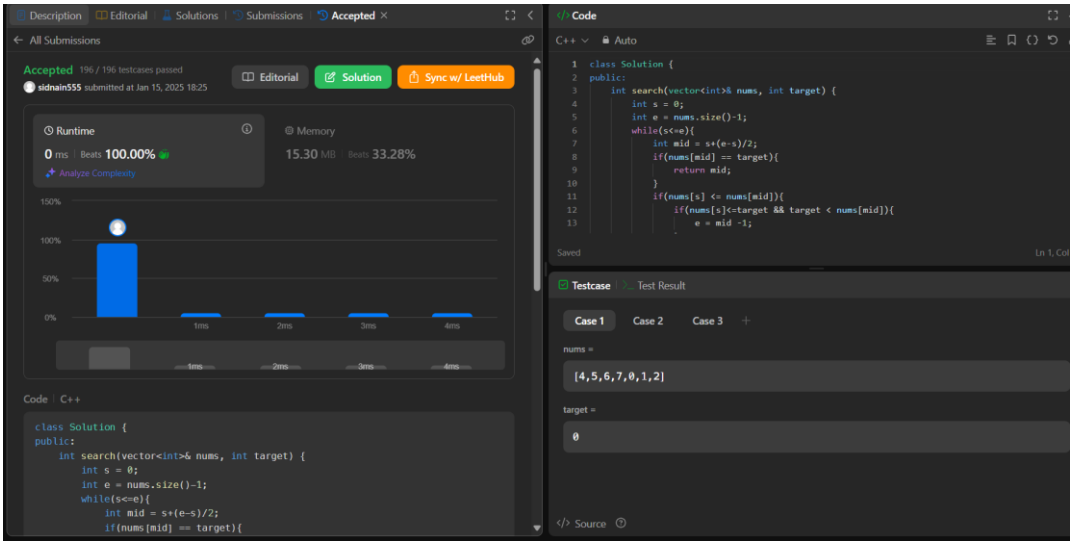

```
class Solution {
public:
    int findPeakElement(vector<int>& nums) {
        int n = nums.size();
        for(int i=0; i<n-1; i++){
            if(nums[i] > nums[i+1]){
                return i;
            }
        }
        return n-1;
    }
};
```

17. [Merge Intervals - LeetCode]




```
class Solution {
public:
    vector<vector<int>> merge(vector<vector<int>>& arr) {
        int n = arr.size();
    sort(arr.begin(), arr.end()); // Sort based on start time
    vector<vector<int>> ans;

    for (int i = 0; i < n; i++) {
        if (ans.empty() || arr[i][0] > ans.back()[1]) {
            ans.push_back(arr[i]);
        } else {
            ans.back()[1] = max(ans.back()[1], arr[i][1]);
        }
    }

    return ans;
    }
};
```

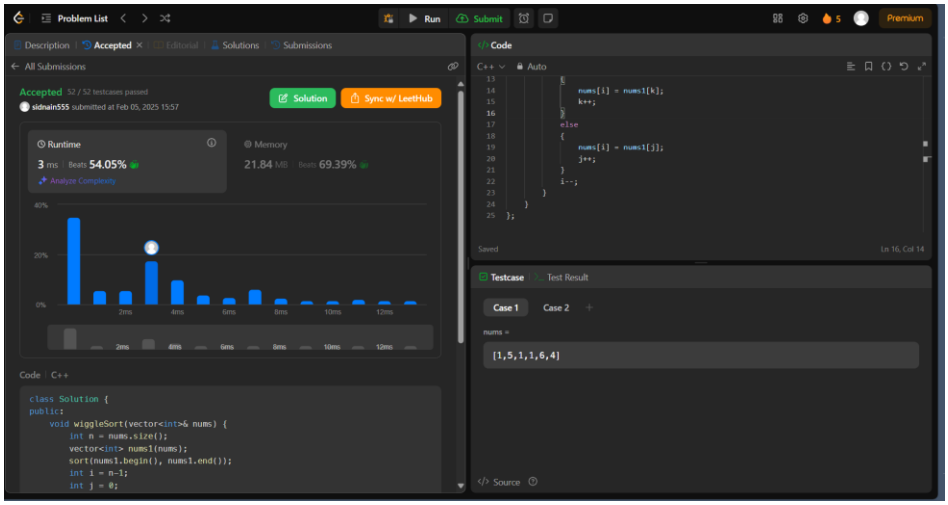
18. [Search in Rotated Sorted Array - LeetCode]




```
class Solution {
public:
    int search(vector<int>& nums, int target) {
        int s = 0;
        int e = nums.size()-1;
        while(s<=e){
            int mid = s+(e-s)/2;
            if(nums[mid] == target){
                return mid;
            }
            if(nums[s] <= nums[mid]){
                if(nums[s]<=target && target < nums[mid]){
                    e = mid -1;
                }
                else{
                    s = mid+1;
                }
            }
            else{
                if (nums[mid] < target && target <= nums[e]) {
                    s = mid + 1;
```

```
            } else {
                e = mid - 1;
            }
        }
    }
    return -1;
  }
};
```

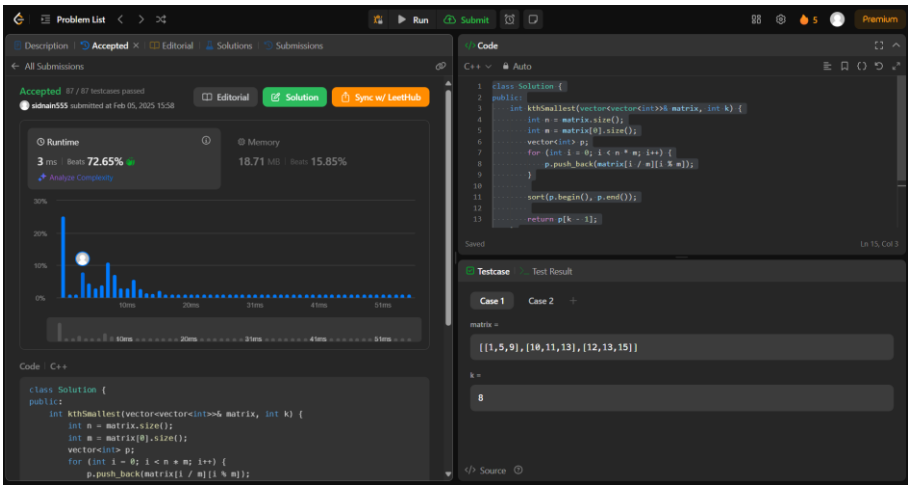
19. [Wiggle Sort II - LeetCode]




```
class Solution {
public:
    void wiggleSort(vector<int>& nums) {
        int n = nums.size();
        vector<int> nums1(nums);
        sort(nums1.begin(), nums1.end());
        int i = n-1;
        int j = 0;
        int k = i/2 + 1;
        while(i >= 0)
        {
            if(i % 2 == 1)
```

```
        {
            nums[i] = nums1[k];
            k++;
        }
        else
        {
            nums[i] = nums1[j];
            j++;
        }
        i--;
    }
  }
};
```

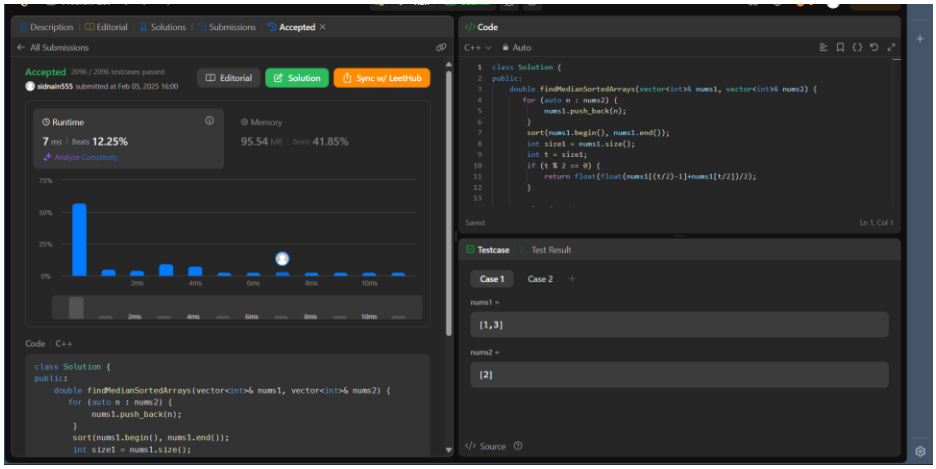
20. [Kth Smallest Element in a Sorted Matrix - LeetCode]




```
class Solution {
public:
    int kthSmallest(vector<vector<int>>& matrix, int k) {
        int n = matrix.size();
        int m = matrix[0].size();
        vector<int> p;
        for (int i = 0; i < n * m; i++) {
```

```
            p.push_back(matrix[i / m][i % m]);
        }

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

        return p[k - 1];
    }
};
```

21. [Median of Two Sorted Arrays - LeetCode]



```
class Solution {
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
        for (auto n : nums2) {
            nums1.push_back(n);
        }
        sort(nums1.begin(), nums1.end());
        int size1 = nums1.size();
        int t = size1;
        if (t % 2 == 0) {
            return float(float(nums1[(t/2)-1]+nums1[t/2])/2);
        }
```

```
        else { t=t/2;
        float ans=float(nums1[t]);
            return ans;

        }
    }
};
```