ASSIGNMENT

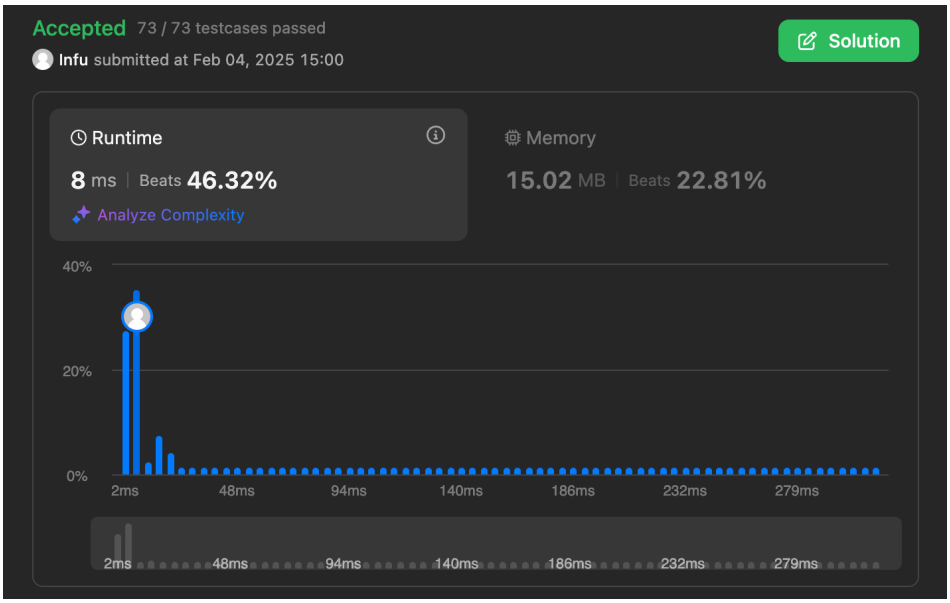
NAME : SEYJAL KHATRI

UID : 22BCS16746

BATCH : FL-3 ‘A’

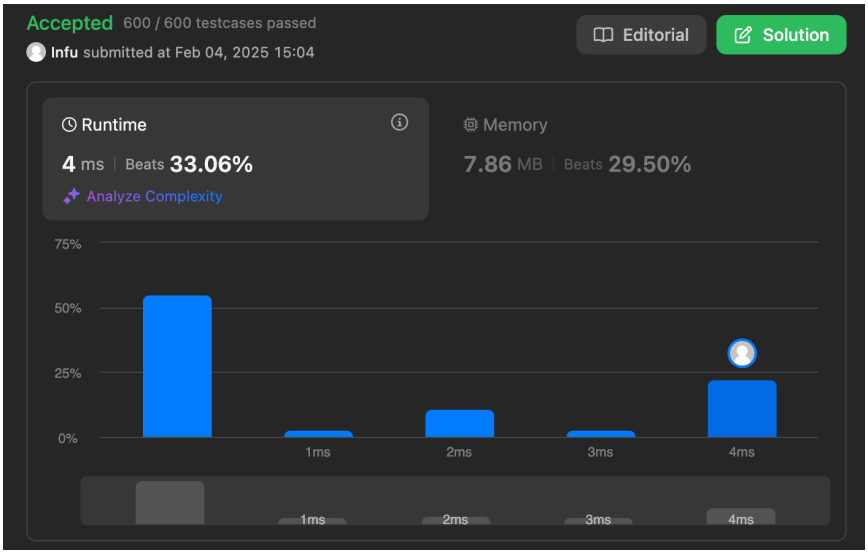
1763. Longest Nice Substring

```
class Solution {
public:
    string longestNiceSubstring(string s) {
        unordered_set<char> missing;
        for (char c : s) {
            if (islower(c)) missing.insert(toupper(c));
            else missing.insert(tolower(c));
        }
        for (int i = 0; i < s.size(); i++) {
            if (missing.count(s[i])) {
                continue;
            }
            string s1 = longestNiceSubstring(s.substr(0, i));
            string s2 = longestNiceSubstring(s.substr(i + 1));
            return s1.size() >= s2.size() ? s1 : s2;
        }
        return s;
    }
};
```



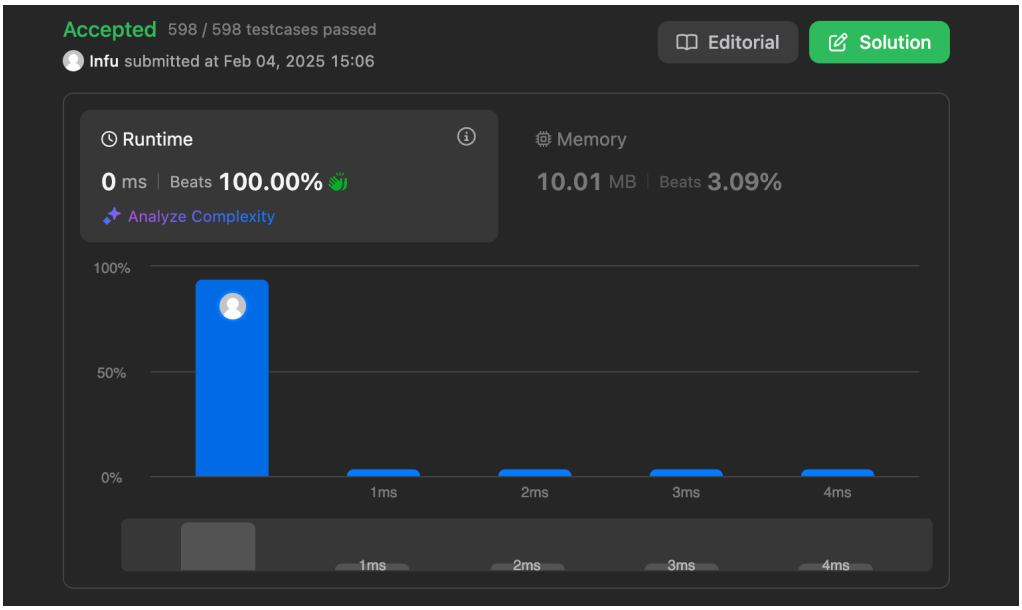
190. Reverse Bits

```
class Solution {
public:
    uint32_t reverseBits(uint32_t n)
    {
        string bits = bitset<32>(n).to_string();
        reverse(bits.begin(), bits.end());

        int ans = stoll(bits, NULL, 2);
        return ans;
    }
};
```



191. Number of 1 Bits

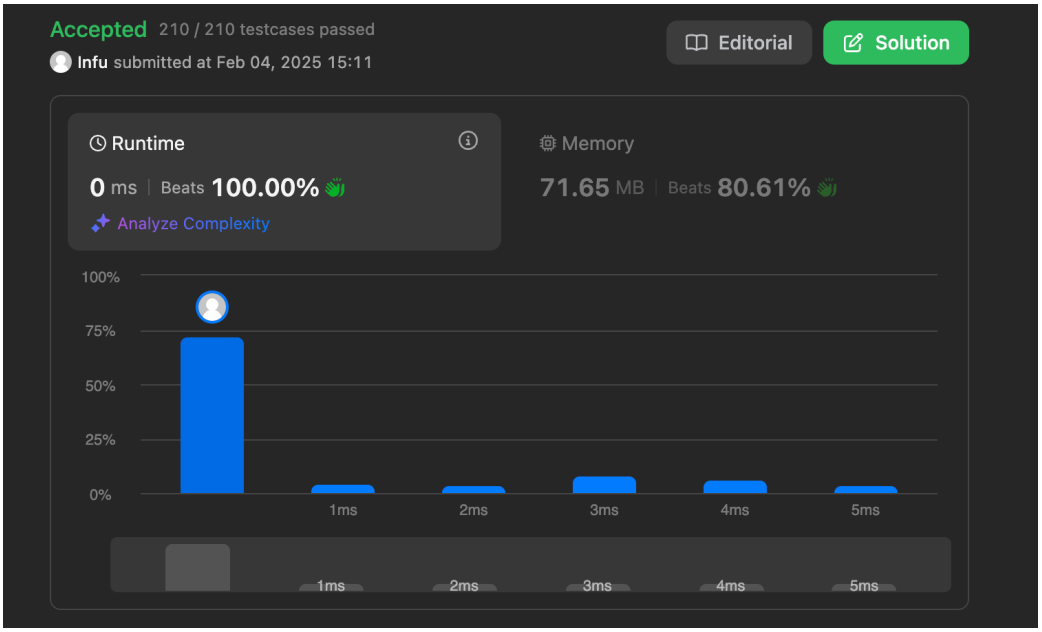
```
class Solution {
public:
    int hammingWeight(int n) {
        stack<int> s;
        while(n){
            s.push(n % 2);
            n = n / 2;
        }
        int count = 0;
        while(!s.empty()){
            if(s.top() == 1) count++;
            s.pop();
        }
        return count;
    }
};
```



53. Maximum Subarray

```
class Solution {
public:
    int maxSubArray(vector<int>& arr) {
    long long maxi = LONG_MIN; // maximum sum
    long long sum = 0;
    int n = arr.size();

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

        sum += arr[i];

        if (sum > maxi) {
            maxi = sum;
        }
        if (sum < 0) {
            sum = 0;
```

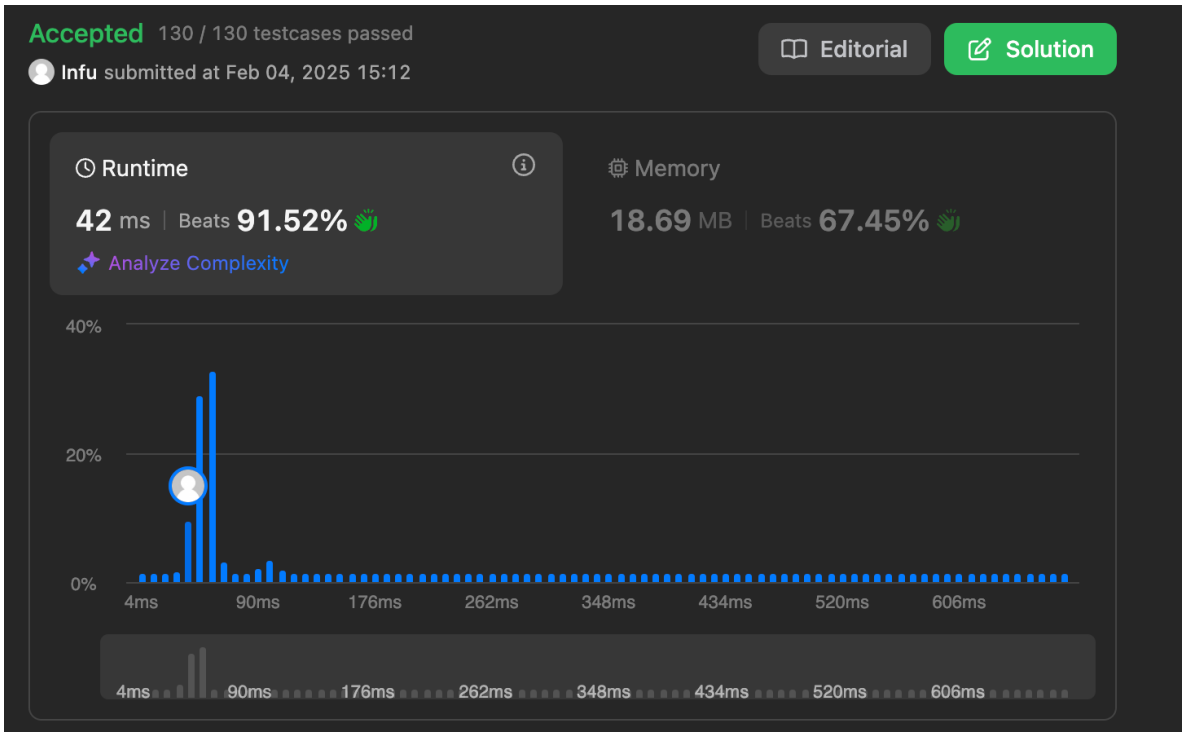
```
        }
    }

    return maxi;
    }
};
```



240. Search a 2D Matrix II

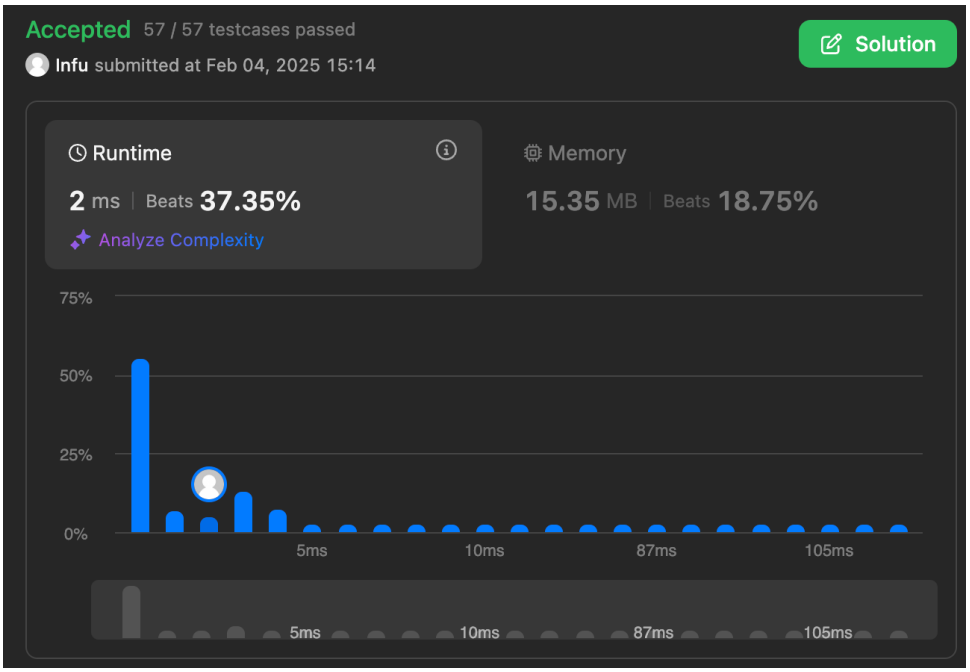
```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int cols = matrix[0].size() - 1;
        int n = matrix.size() - 1;
        int rows = 0;


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

```
            int toCompare = matrix[rows][cols];
            if(toCompare > target){
                cols--;
            }else if(toCompare < target){
                rows++;
            }else{
                return true;
            }
        }

        return false;
    }
};
```



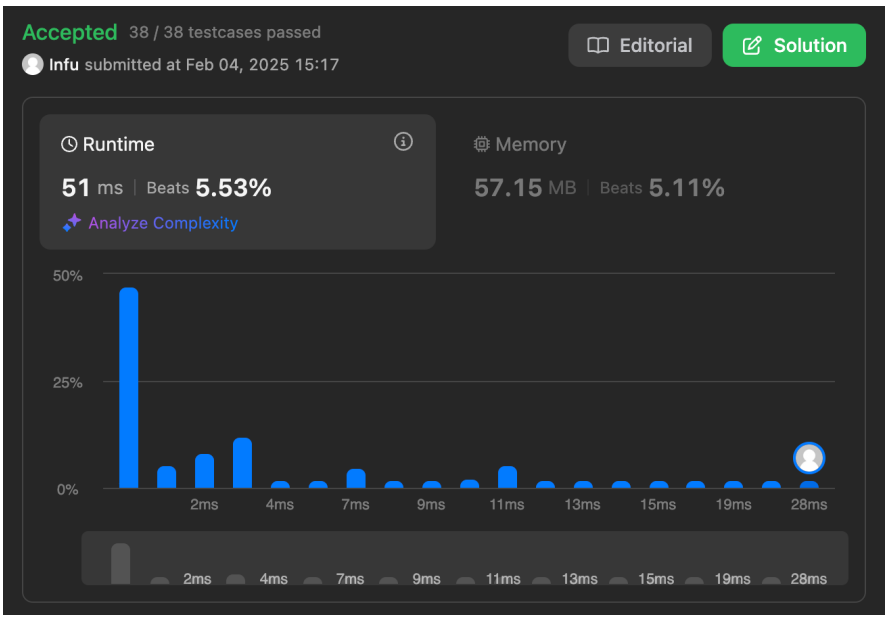
372. Super Pow

```
class Solution {
public:
    int pow(int a, int b){
        if(b==0) return 1;
        int temp=pow(a,b/2);
        if(b%2==0) return ((temp%1337)*temp%1337)%1337;
        else return (a%1337*((temp%1337*temp%1337)%1337))%1337;
    }
    int superPow(int a, vector<int>& b) {
        if(b.size()==0) return 1;
        int x=b.back(); b.pop_back();
        return pow(superPow(a, b), 10) * pow(a, x) % 1337;
    }
};
```



932. Beautiful Array

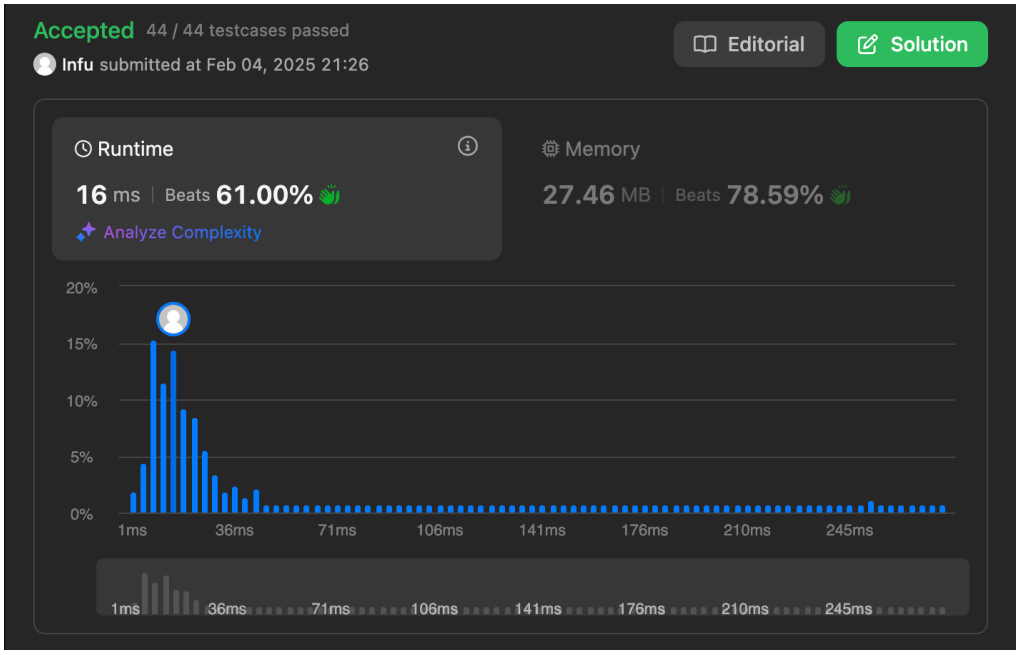
```
class Solution {
public:
    vector<int> beautifulArray(int n) {
        if (n==1) return {1};
        vector<int> arr = beautifulArray(n-1);
        vector<int> res;
        for (auto i: arr)
            if (2*i - 1 <= n)
                res.push_back(2*i-1);
        for (auto i: arr)
            if (2*i <= n)
                res.push_back(2*i);
        return res;
    }
};
```



218. The Skyline Problem

```
class Solution {
public:
     vector<vector<int>> getSkyline(vector<vector<int>>& b) {
        priority_queue<vector<int>> live;
        int n=b.size();

        int cur=0;
        vector<vector<int>> ans;
        while(cur<n || !live.empty()){
            int cur_x=live.empty()?b[cur][0]:live.top()[1];
            if(cur>=n || b[cur][0]>cur_x){
                while(!live.empty() && (live.top()[1]<=cur_x)){
                    live.pop();
                }
            }
            else{
                cur_x=b[cur][0];
                while(cur<n && cur_x==b[cur][0]){
                    live.push({b[cur][2],b[cur][1]});
                    cur++;
                }
            }
            int cur_h=live.empty()?0:live.top()[0];
            if(ans.empty() || ans[ans.size()-1][1]!=cur_h){
                ans.push_back({cur_x,cur_h});
```

```
            }
        }
        return ans;
    }
};
```

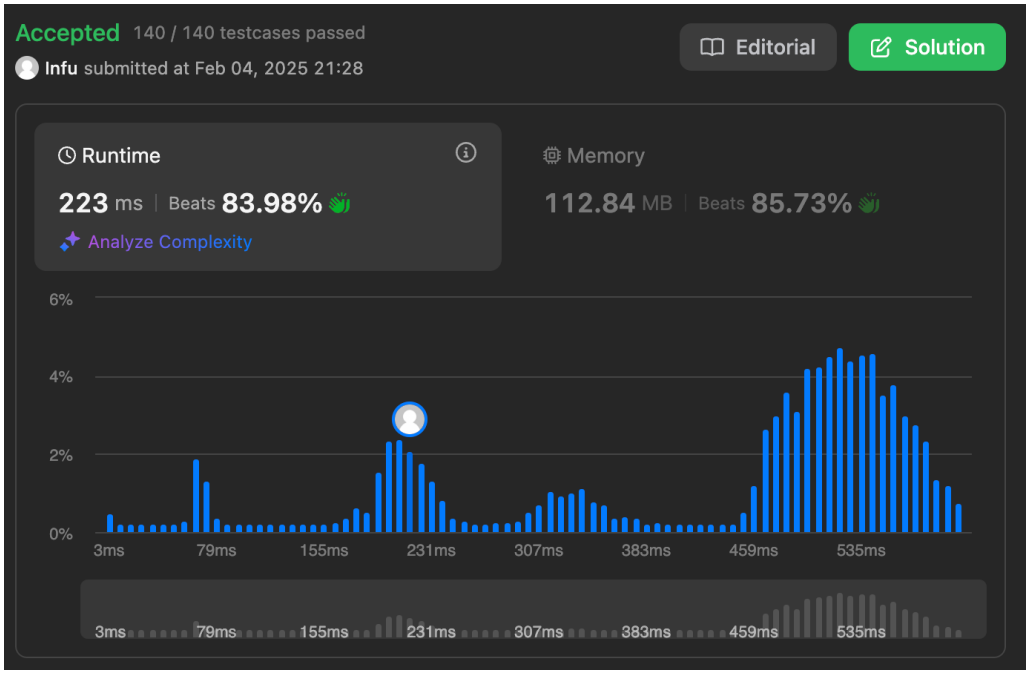


493. Reverse Pairs

```
class Solution {
private:
    void merge(vector<int>& nums, int low, int mid, int high, int&
reversePairsCount){
        int j = mid+1;
        for(int i=low; i<=mid; i++){
            while(j<=high && nums[i] > 2*(long long)nums[j]){
                j++;
            }
            reversePairsCount += j-(mid+1);
```

```
        }
        int size = high-low+1;
        vector<int> temp(size, 0);
        int left = low, right = mid+1, k=0;
        while(left<=mid && right<=high){
            if(nums[left] < nums[right]){
                temp[k++] = nums[left++];
            }
            else{
                temp[k++] = nums[right++];
            }
        }
        while(left<=mid){
            temp[k++] = nums[left++];
        }
        while(right<=high){
            temp[k++] = nums[right++];
        }
        int m=0;
        for(int i=low; i<=high; i++){
            nums[i] = temp[m++];
        }
    }

    void mergeSort(vector<int>& nums, int low, int high, int&
reversePairsCount){
        if(low >= high){
```

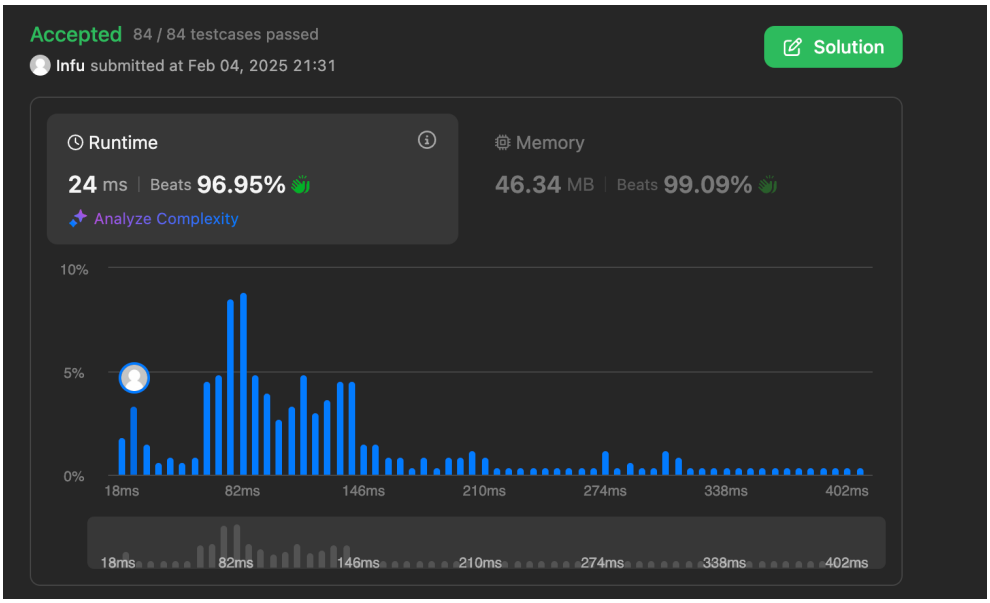
```
            return;
        }
        int mid = (low + high) >> 1;
        mergeSort(nums, low, mid, reversePairsCount);
        mergeSort(nums, mid+1, high, reversePairsCount);
        merge(nums, low, mid, high, reversePairsCount);
    }
public:
    int reversePairs(vector<int>& nums) {
        int reversePairsCount = 0;
        mergeSort(nums, 0, nums.size()-1, reversePairsCount);
        return reversePairsCount;
    }
};
```



2407. Longest Increasing Subsequence II

```
const int N = 1e5+1;

class Solution {
public:
    int lengthOfLIS(vector<int>& nums, int k) {
        int n = nums.size();
        int t[2*N + 2]; memset(t, 0, sizeof(t));

        auto upd = [&](int p, int v) {
            p += N;
            t[p] = v;
            for(; p > 1; p >>= 1) t[p>>1] = max(t[p], t[p^1]);
        };

        auto qry = [&](int l, int r) {
            l+=N, r+=N;
            int ret = 0;
            for(; l<r; l>>=1, r>>=1) {
                if (l&1) ret = max(ret, t[l++]);
                if (r&1) ret = max(ret, t[--r]);
            }
            return ret;
        };

        //upd(nums[0], 1);
```

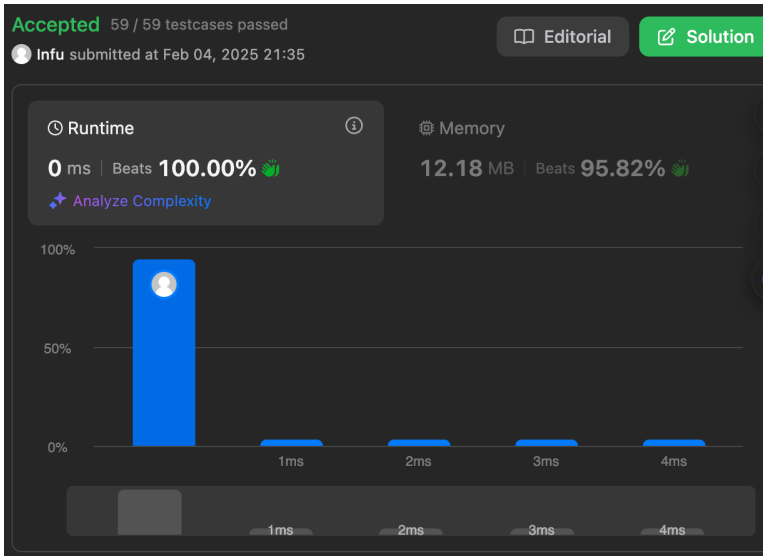
```
        for (int i=0; i<n; ++i) {
            int curr = nums[i];
            int best = 1 + qry(max(0, curr - k), curr);
            upd(curr, best);
        }
        int ans = qry(0, N+1);

        return ans;
    }
};
```



88. Merge Sorted Array

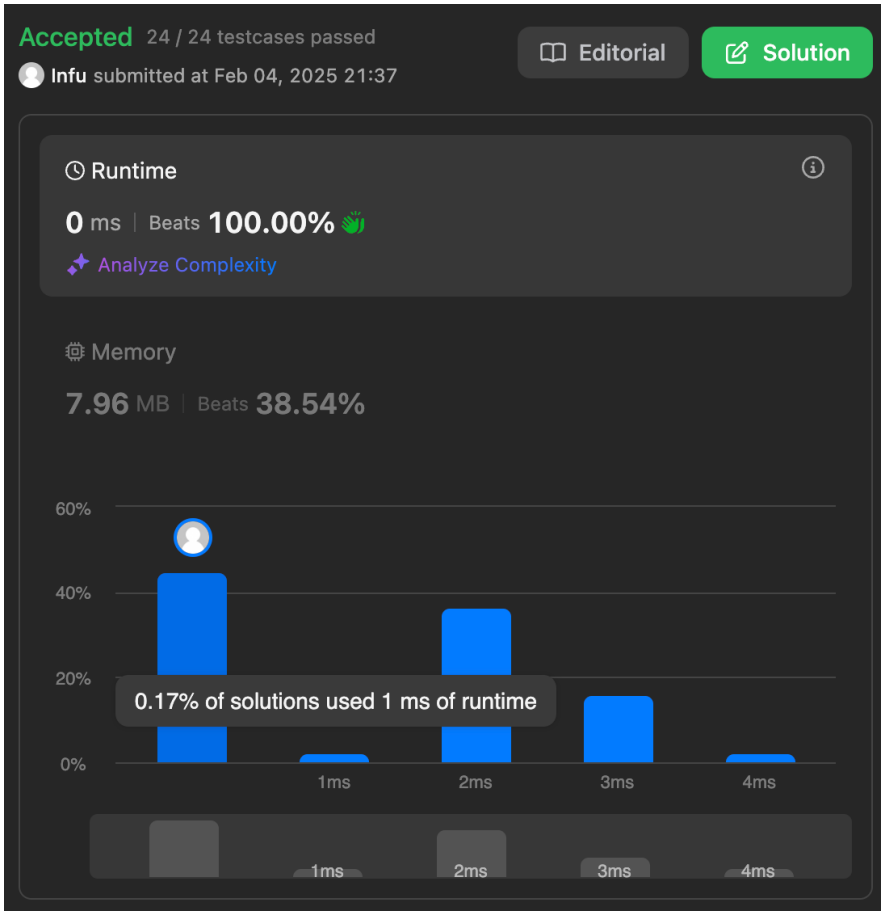
```
class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        int i = m-1;
        int j = n-1;
```

```
        int k = m+n-1;
        while(i>=0 && j>=0){
            if(nums1[i] > nums2[j]){
                nums1[k] = nums1[i];
                i--; k--;
            }
            else{
                nums1[k] = nums2[j];
                j--; k--;
            }
        }
        while(j>=0){
            nums1[k] = nums2[j];
            j--; k--;
        }
    }
};
```



278. First Bad Version

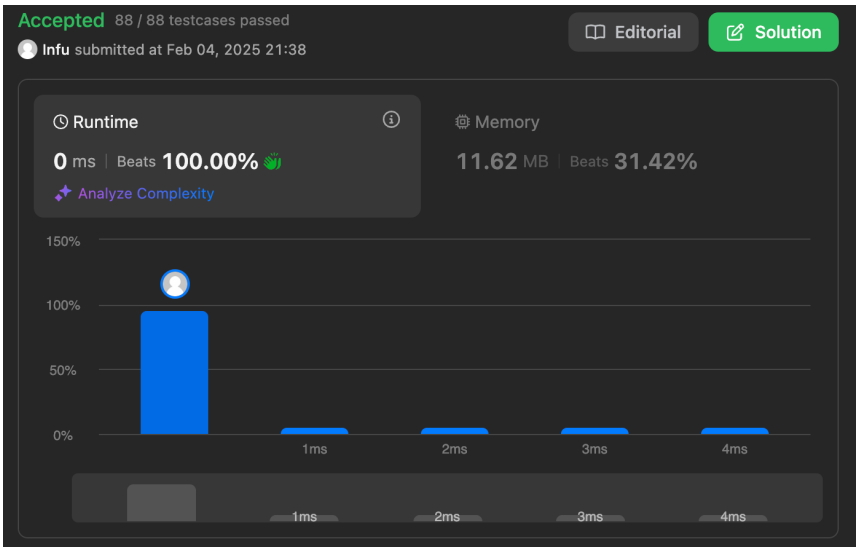
```
class Solution {
public:
    int firstBadVersion(int n) {
        int start = 1, end = n;
        while(start <= end){
            int mid = start + (end - start) / 2; // to avoid overflow
            if(isBadVersion(mid)){
                end = mid - 1;
            }else{
                start = mid + 1;
            }
        }
        return start;
    }
};
```



75. Sort Colors

```
class Solution {
public:
    void sortColors(vector<int>& arr) {
        int low = 0, mid = 0, high = arr.size() - 1; // 3 pointers

    while (mid <= high) {
        if (arr[mid] == 0) {
            swap(arr[low], arr[mid]);
            low++;
            mid++;
        }
```

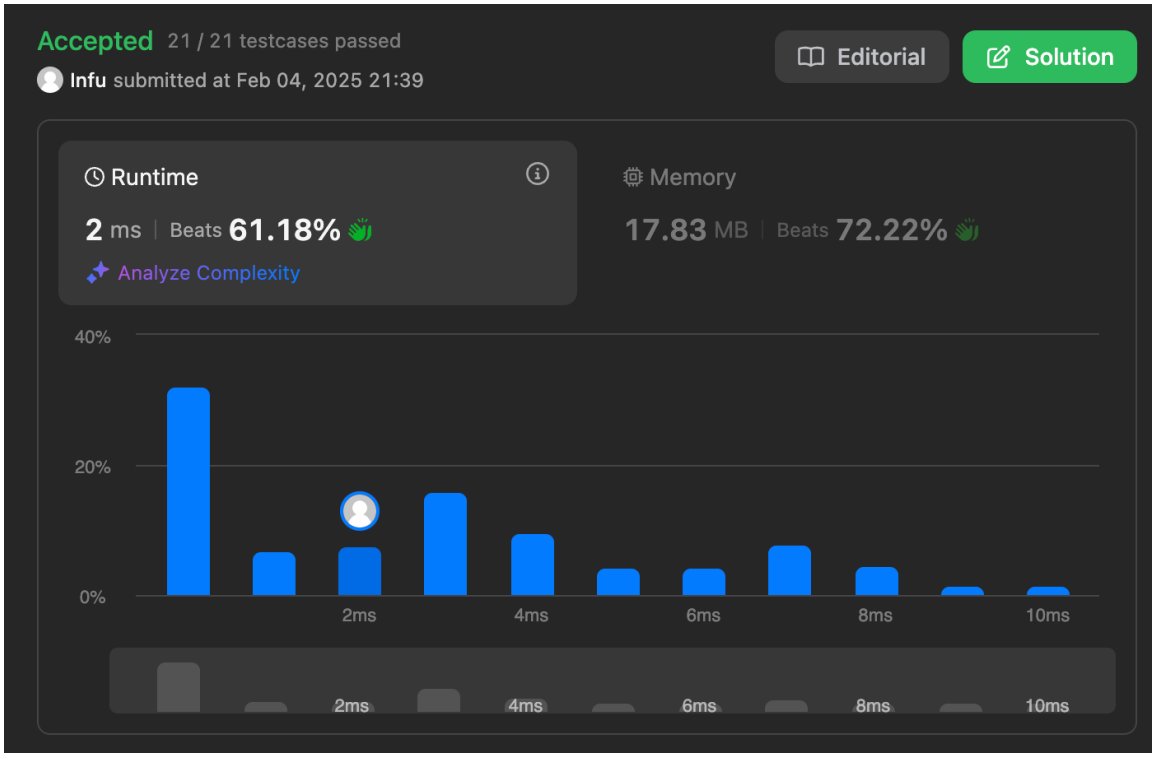
```
        else if (arr[mid] == 1) {
            mid++;
        }
        else {
            swap(arr[mid], arr[high]);
            high--;
        }
    }
}
};
```



347. Top K Frequent Elements

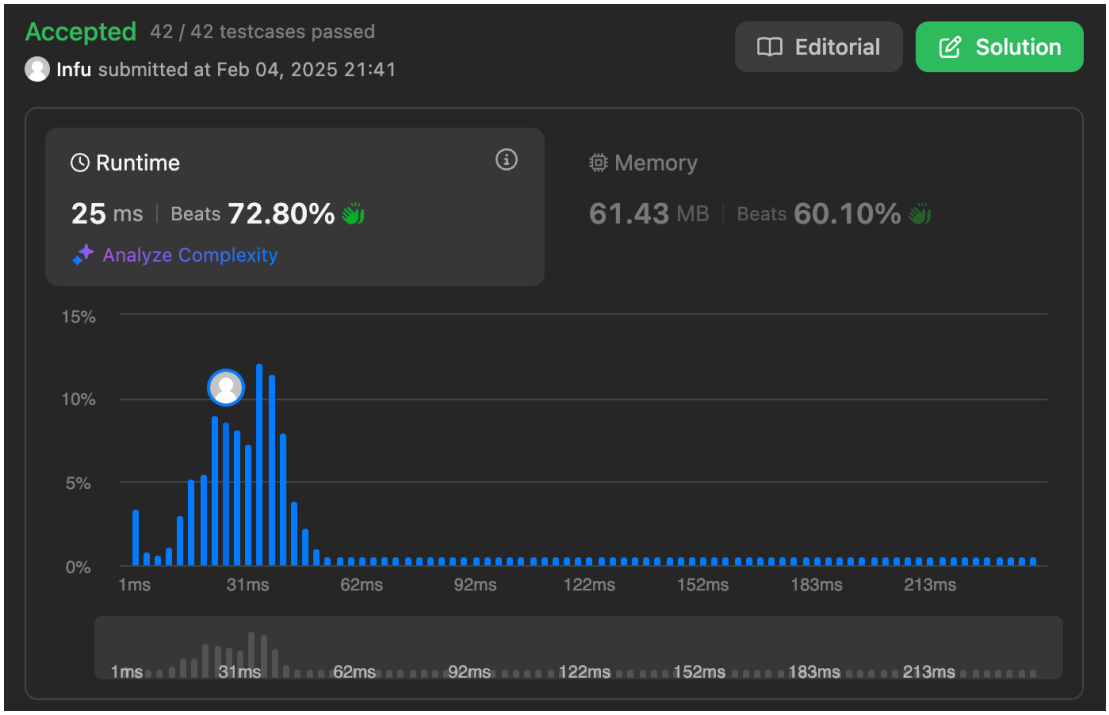
```
class Solution {
public:
    vector<int> topKFrequent(vector<int>& nums, int k) {
        unordered_map<int,int> map;
        for(int num : nums){
            map[num]++;
```

```
        }

        vector<int> res;
        priority_queue<pair<int,int>> pq;
        for(auto it = map.begin(); it != map.end(); it++){
            pq.push(make_pair(it->second, it->first));
            if(pq.size() > (int)map.size() - k){
                res.push_back(pq.top().second);
                pq.pop();
            }
        }
        return res;
    }
};
```



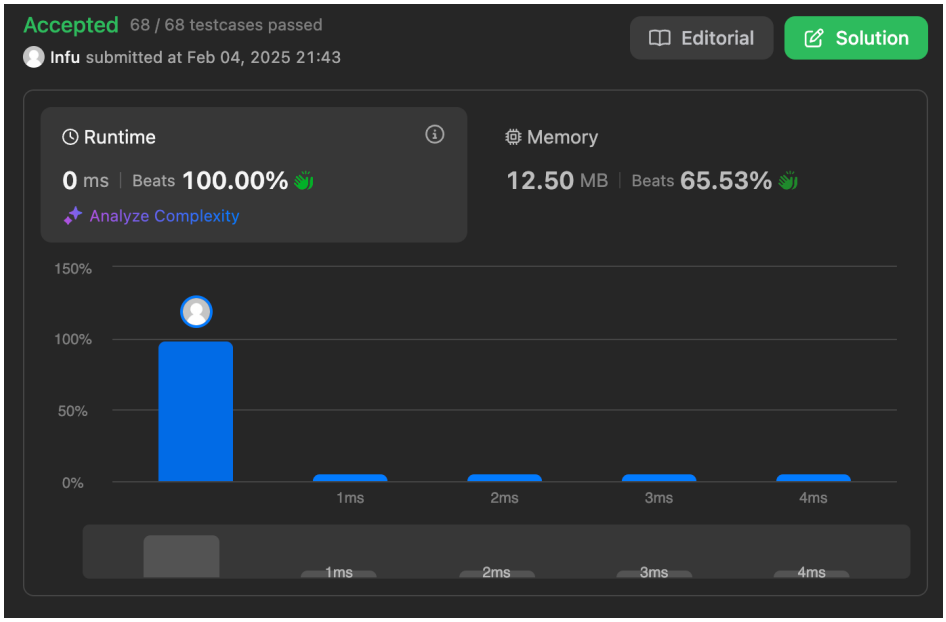
215. Kth Largest Element in an Array

```
class Solution {
public:
    int findKthLargest(vector<int>& nums, int k) {
        priority_queue<int, vector<int>, greater<int>> pq;
        for (int num : nums) {
            pq.push(num);
            if (pq.size() > k) {
                pq.pop();
            }
        }
        return pq.top();
    }
};
```



162. Find Peak Element

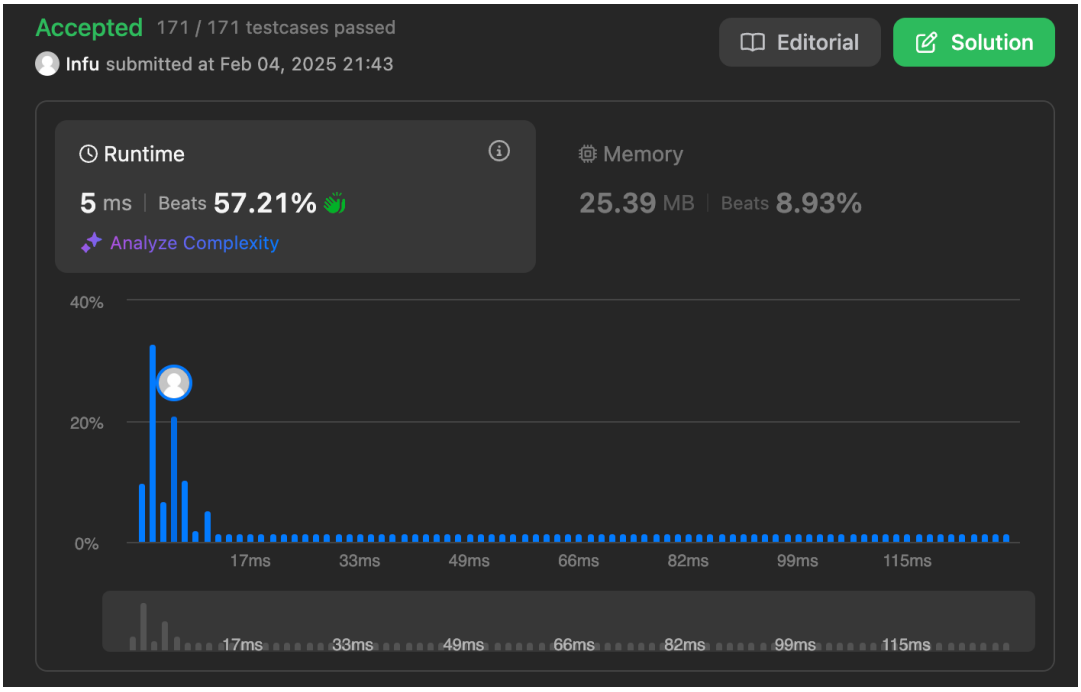
```
class Solution {
public:
    int findPeakElement(vector<int>& nums) {
        int n = nums.size();
        int low = 0, high = n - 1;

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

            if ((mid == 0 || nums[mid] > nums[mid - 1]) &&
                (mid == n - 1 || nums[mid] > nums[mid + 1])) {
                return mid;
            }

            if (mid < n - 1 && nums[mid] < nums[mid + 1]) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }

        return -1;
    }
};
```



56. Merge Intervals

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

      if(intervals.size()==1)
         return intervals;
      vector<pair<int,int>> p;
      for(int i=0;i<intervals.size();i++)
      {
          p.push_back({intervals[i][0],intervals[i][1]});
      }
      sort(p.begin(),p.end());

      vector<vector<int>> ans;
      int f=p[0].first,s=p[0].second;
      for(int i=0;i<p.size()-1;i++)
```

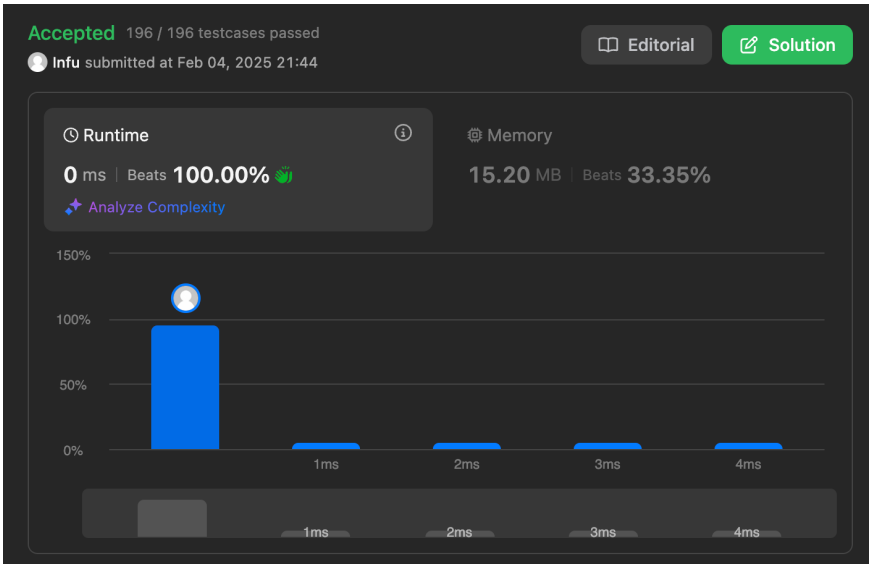
```
        {
            vector<int> a(2);
            if(s>=p[i+1].first)
            {
                s=max(s,p[i+1].second);
            }
            else
            {
                a[0]=f;
                a[1]=s;
                f=p[i+1].first;
                s=p[i+1].second;
                ans.push_back(a);
            }
        }
        int n=intervals.size();
        ans.push_back({f,s});
        return ans;
    }
};
```



33. Search in Rotated Sorted Array

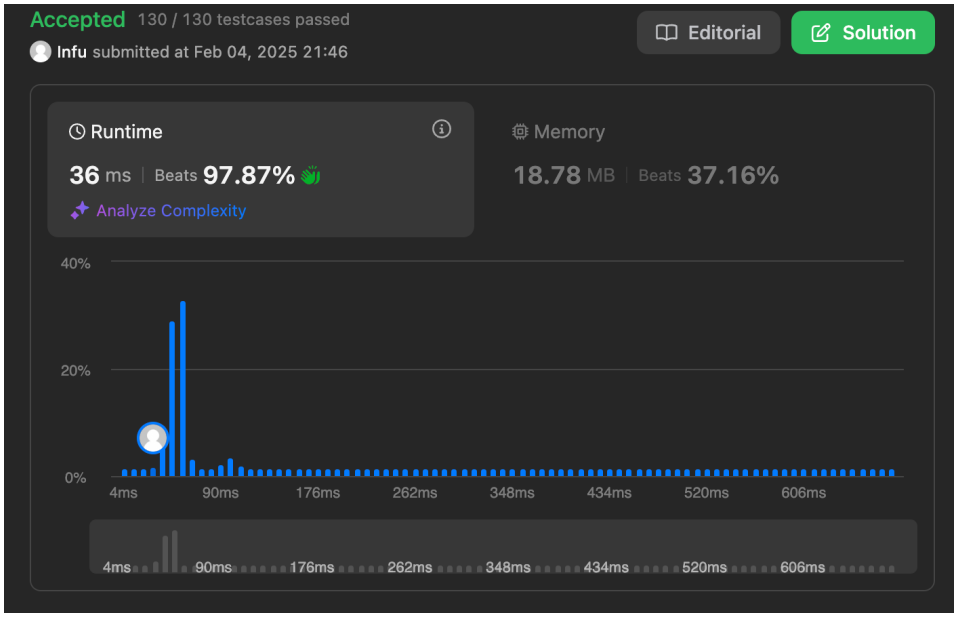
```
class Solution {
public:
    int search(vector<int>& nums, int x) {
        int low = 0;
        int high = nums.size() - 1;

        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (nums[mid] == x)
                return mid;

            // Check if the left half is sorted
            if (nums[low] <= nums[mid]) {
                // Check if x lies in the sorted left half
```

```
            if (x >= nums[low] && x < nums[mid])
                high = mid - 1;
            else
                low = mid + 1;
        }
        // Otherwise, the right half must be sorted
        else {
            // Check if x lies in the sorted right half
            if (x > nums[mid] && x <= nums[high])
                low = mid + 1;
            else
                high = mid - 1;
        }
    }
    return -1;
  }
};
```



240. Search a 2D Matrix II

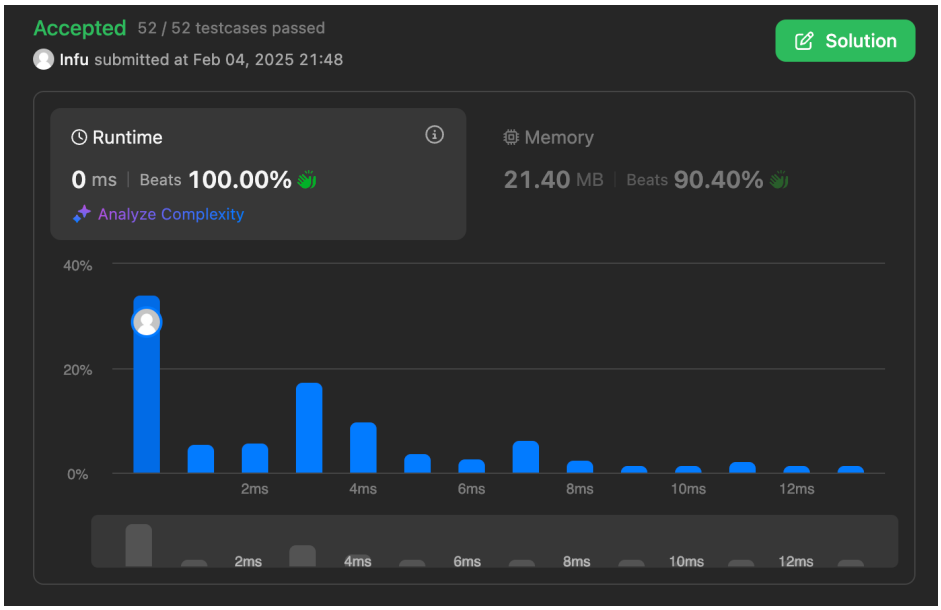
```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int cols = matrix[0].size() - 1;
        int n = matrix.size() - 1;
        int rows = 0;



        while(rows <= n && cols >= 0){
            int toCompare = matrix[rows][cols];
            if(toCompare > target){
                cols--;
            }else if(toCompare < target){
                rows++;
            }else{
                return true;
            }
        }

        return false;
    }
};
```



324. Wiggle Sort II

```
class Solution {
public:
    void wiggleSort(vector<int>& nums) {
        int maxx = *max_element(nums.begin(),nums.end());
        vector<int> sortArray(maxx+1,0);

        for( int i=0; i<nums.size(); i++ )
            sortArray[nums[i]]++;

        int j = maxx;
        for( int i=1; i<nums.size(); i+=2 )
        {
            while( sortArray[j] == 0 )
                j--;
            nums[i] = j;
            sortArray[j]--;
```

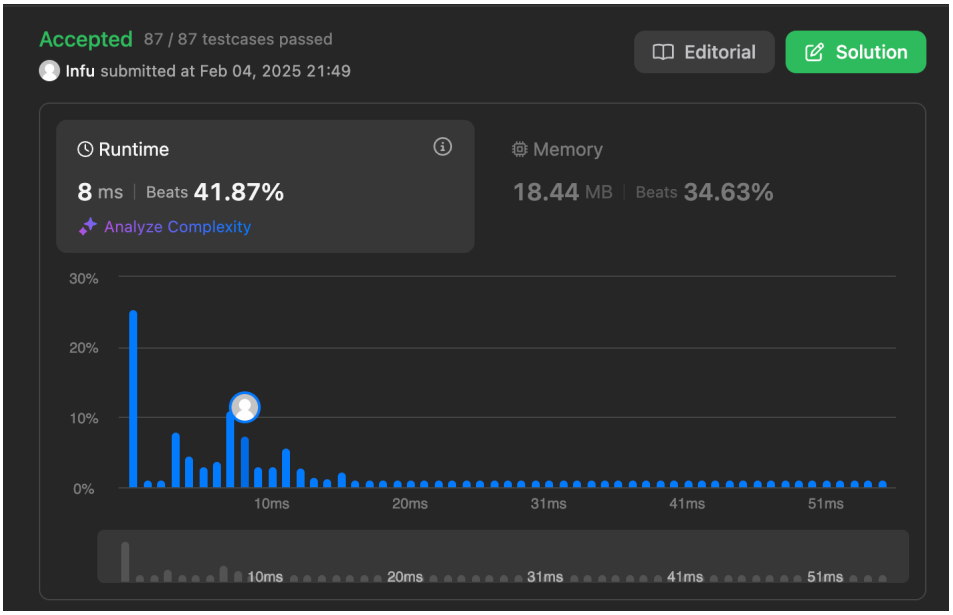
```
        }
        for( int i=0; i<nums.size(); i+=2 )
        {
            while( sortArray[j] == 0 )
                j--;
            nums[i] = j;
            sortArray[j]--;
        }
    }
};
```



378. Kth Smallest Element in a Sorted Matrix

```
class Solution {
public:
    int kthSmallest(vector<vector<int>>& matrix, int k) {
        vector<int> result;
        for (const auto& row : matrix) {
            for (const auto& element : row)
```

```
                result.push_back(element);
            }
            sort(result.begin(), result.end());
            int size=result.size();
            if(size<k) return -1;
            return result[k-1];
        }
};
```



4. Median of Two Sorted Arrays

```
class Solution {
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>&
nums2) {
        vector<int>v;

        for(auto num:nums1)
```

```
            v.push_back(num);

        for(auto num:nums2)
            v.push_back(num);

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

        int n=v.size();

        return n%2?v[n/2]:(v[n/2-1]+v[n/2])/2.0;
    }
};
```