Name: Semit Tirkey

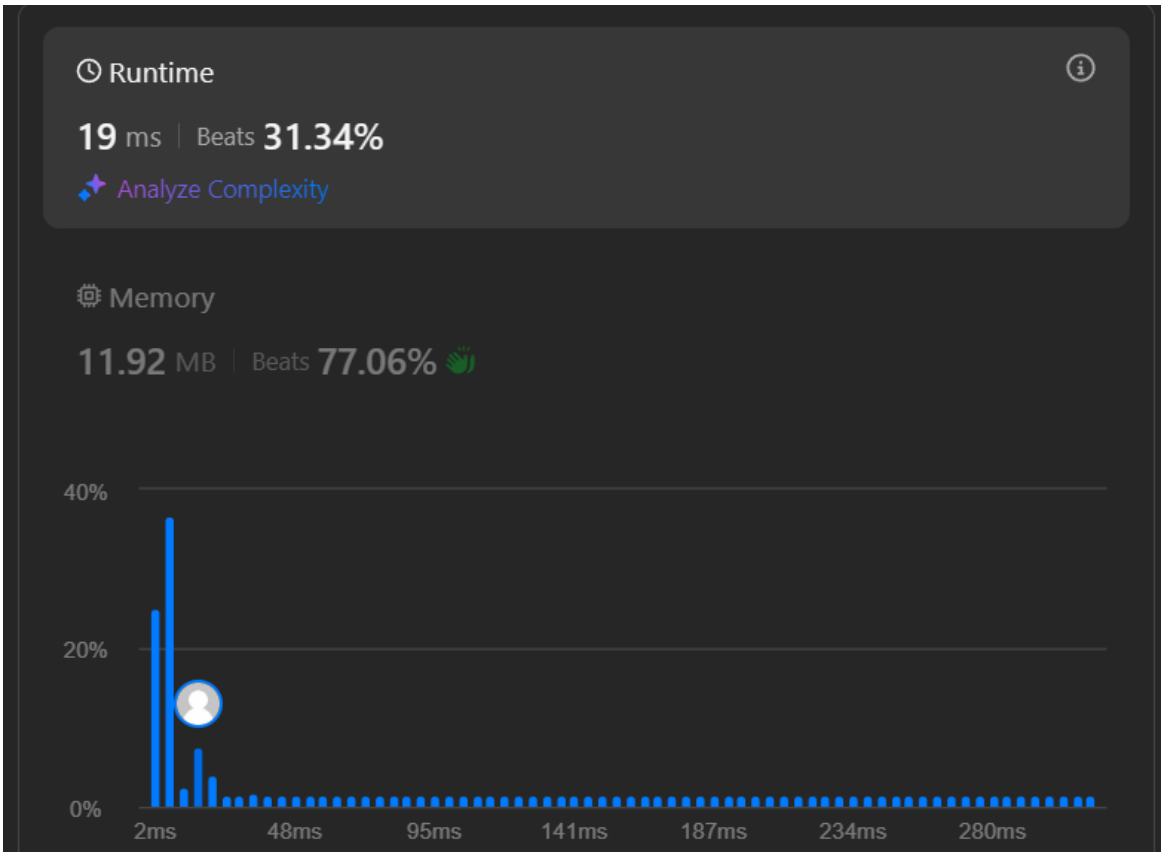
UID: 22BCS13024

Assignment – 2 Solutions:-

1. **Longest Nice Substring:**

```
class Solution {
public:
    string longestNiceSubstring(string s) {
        int n = s.size();
        int k = -1, mx = 0;
        for (int i = 0; i < n; ++i) {
            unordered_set<char> ss;
            for (int j = i; j < n; ++j) {
                ss.insert(s[j]);
                bool ok = true;
                for (auto& a : ss) {
                    char b = a ^ 32;
                    if (!(ss.count(a) && ss.count(b))) {
                        ok = false;
                        break;
                    }
                }
                if (ok && mx < j - i + 1) {
                    mx = j - i + 1;
                    k = i;
                }
            }
        }
        return k == -1 ? "" : s.substr(k, mx);
    }
};
```

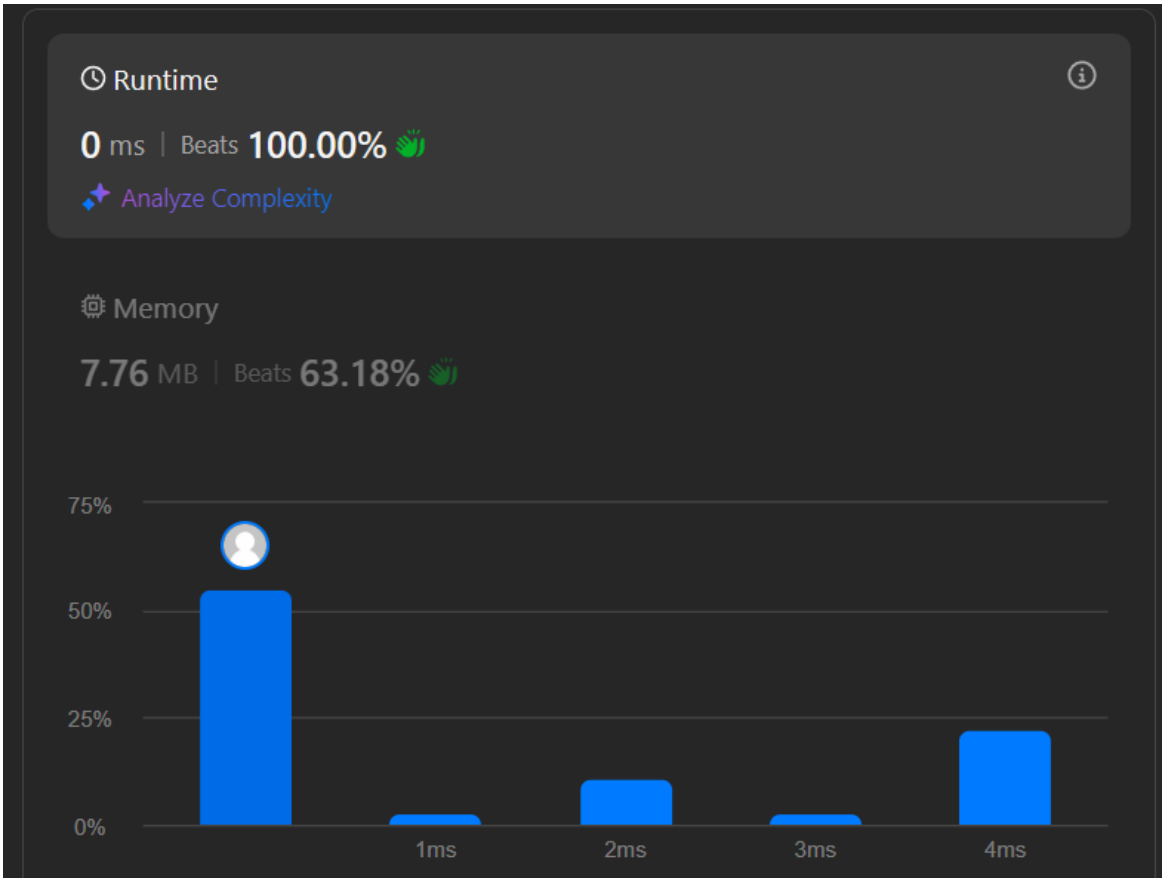
Result:-



2. **Reverse Bits:**

```
class Solution {
public:
    uint32_t reverseBits(uint32_t n) {
        uint32_t ans = 0;
        for (int i = 0; i < 32 && n; ++i) {
            ans |= (n & 1) << (31 - i);
            n >>= 1;
        }
        return ans;
    }
};
```

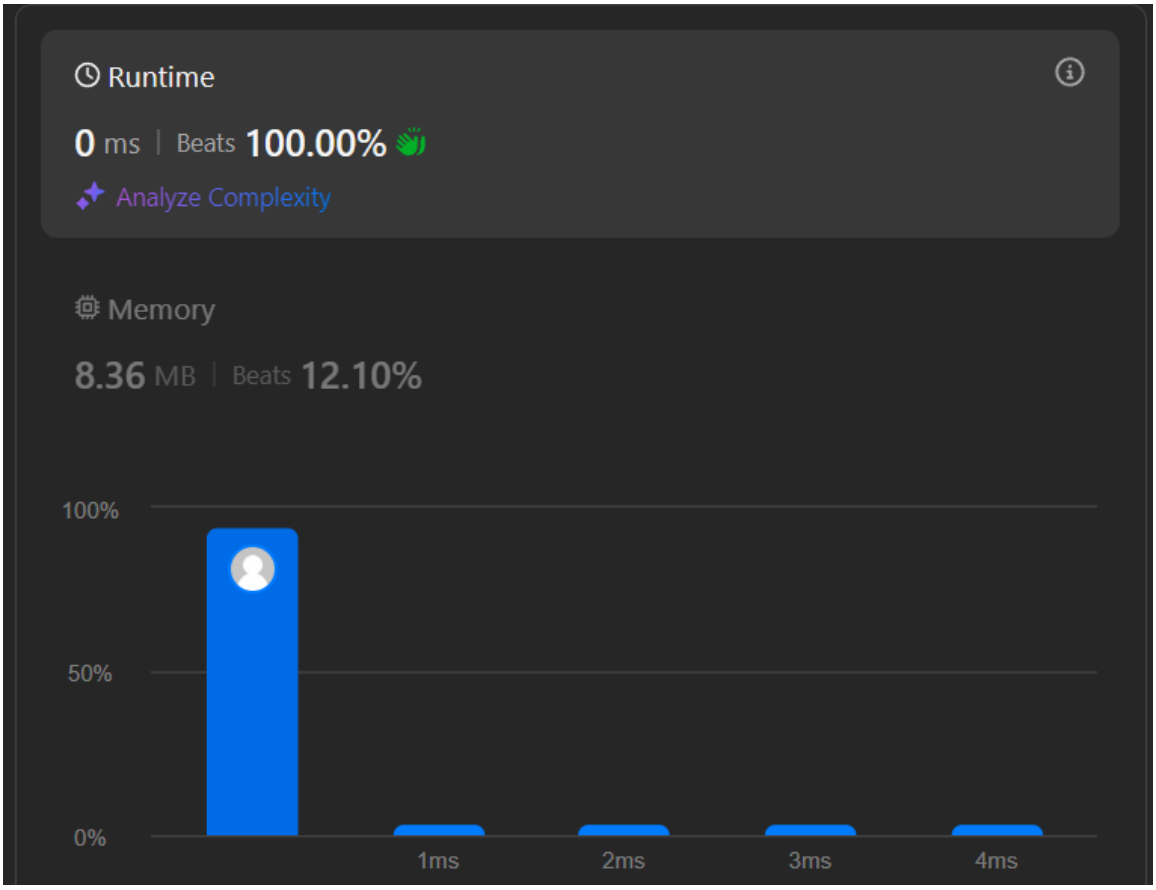
Result:



3. **Number of 1 Bits:**

```
class Solution {
public:
    int hammingWeight(int n) {
        int count=0;
        while(n!=0){
            if(n%2!=0) count++;
            n=n>>1;
        }
        return count;
    }
};
```

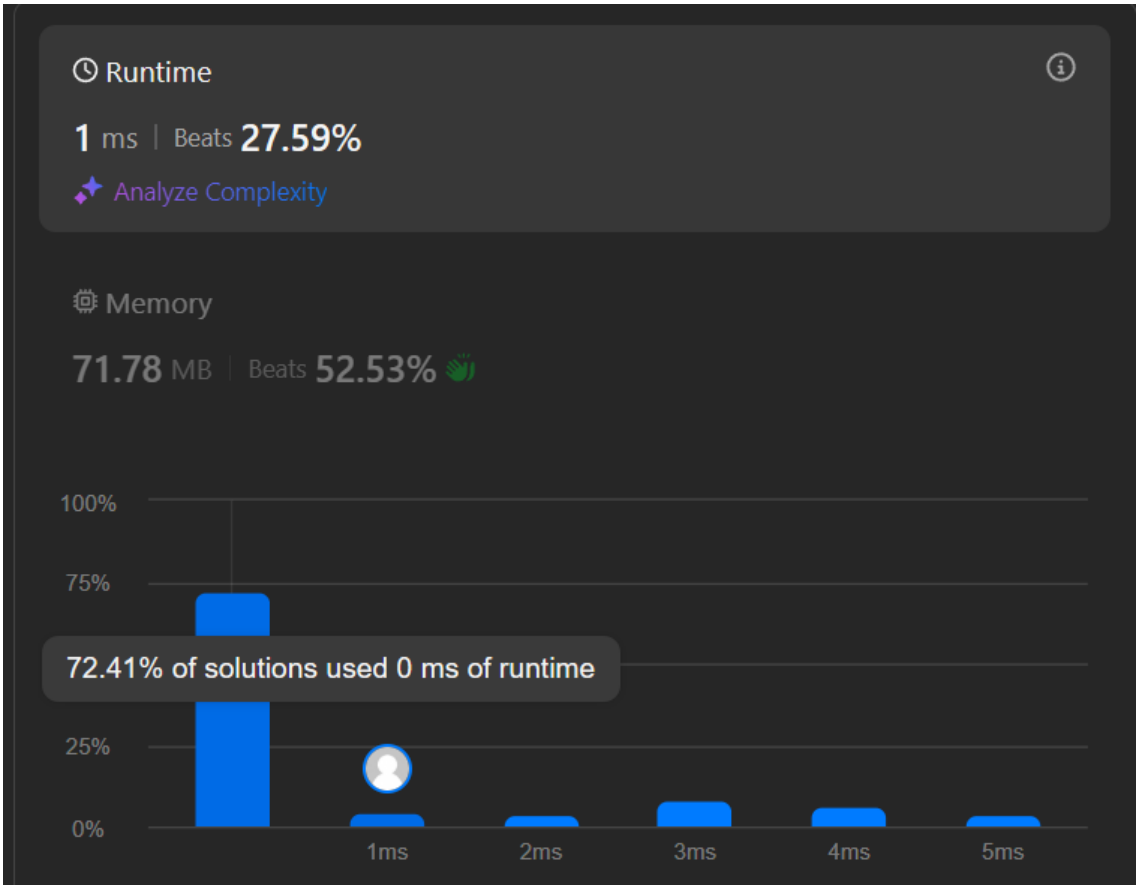
Result:



4. **Maximum Subarray:**

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int ans = nums[0], f = nums[0];
        for (int i = 1; i < nums.size(); ++i) {
            f = max(f, 0) + nums[i];
            ans = max(ans, f);
        }
        return ans;
    }
};
```

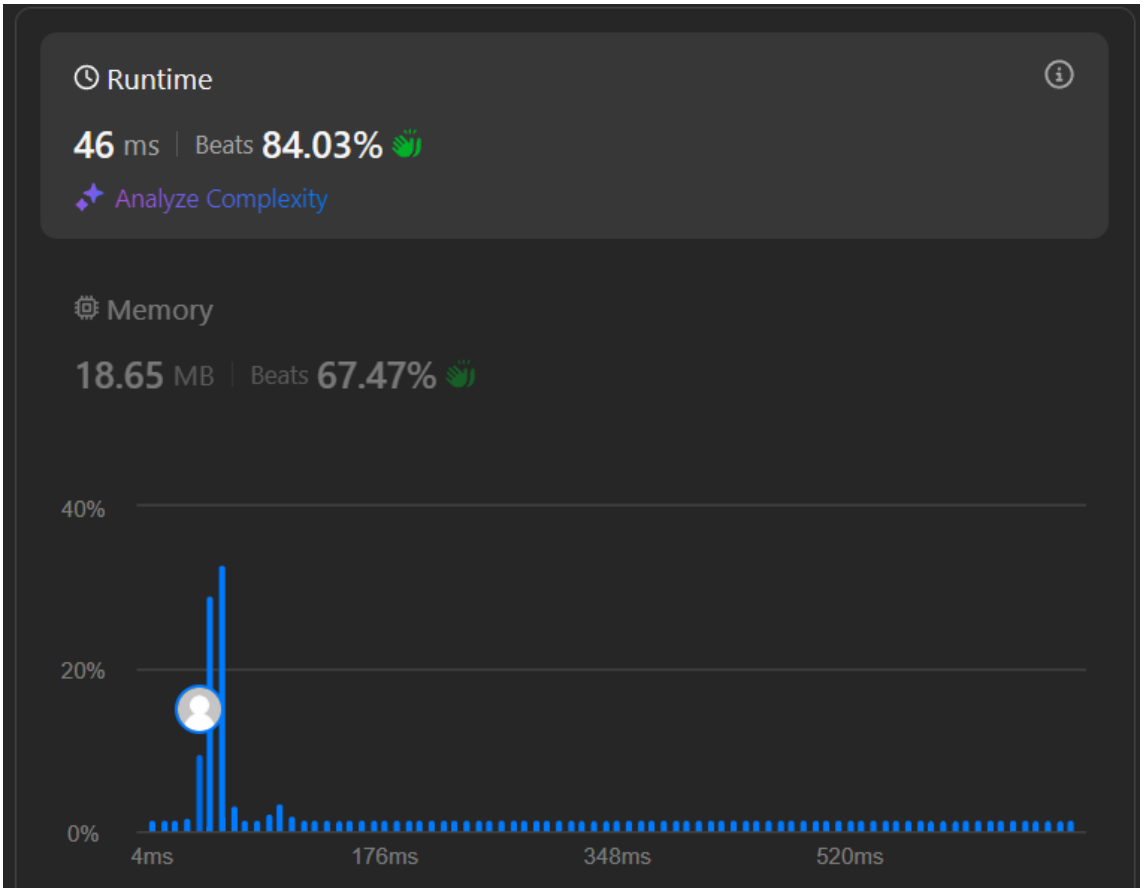
Result:



5. **Search a 2D Matrix II:**

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int i=0; int j=matrix[0].size()-1,a;
        while(i<matrix.size() && j>=0){
            a=matrix[i][j];
            if(a==target) return true;
            else if(a>target) j--;
            else i++;
        }
        return false;
    }
};
```

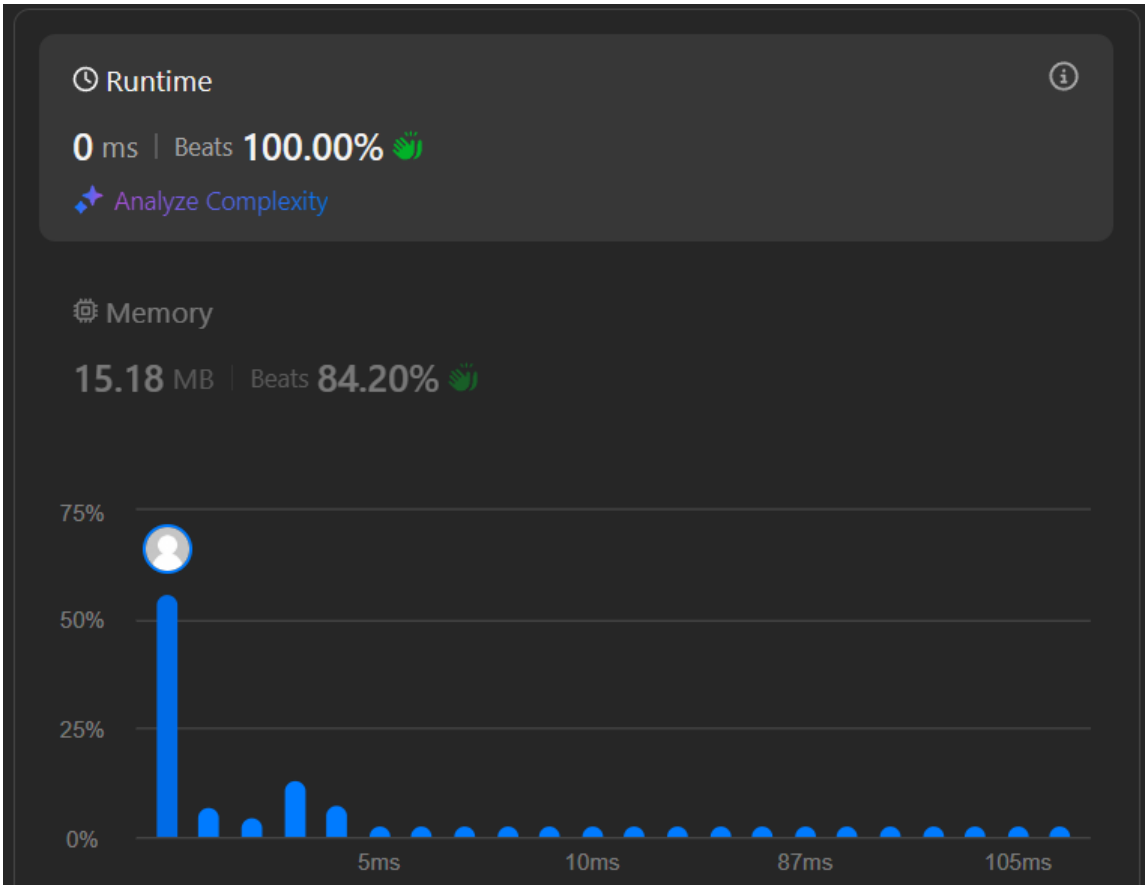
Result:



6. **Super Pow:**

```
class Solution {
    const int base = 1337;
    int powmod(int a, int k){
        a %= base;
        int result = 1;
        for (int i = 0; i < k; ++i)  result = (result * a) % base;
        return result;
    }

public:
    int superPow(int a, vector<int>& b) {
        if (b.empty()) return 1;
        int last_digit = b.back();
        b.pop_back();
        return powmod(superPow(a, b), 10) * powmod(a, last_digit) % base;
    }
};
```

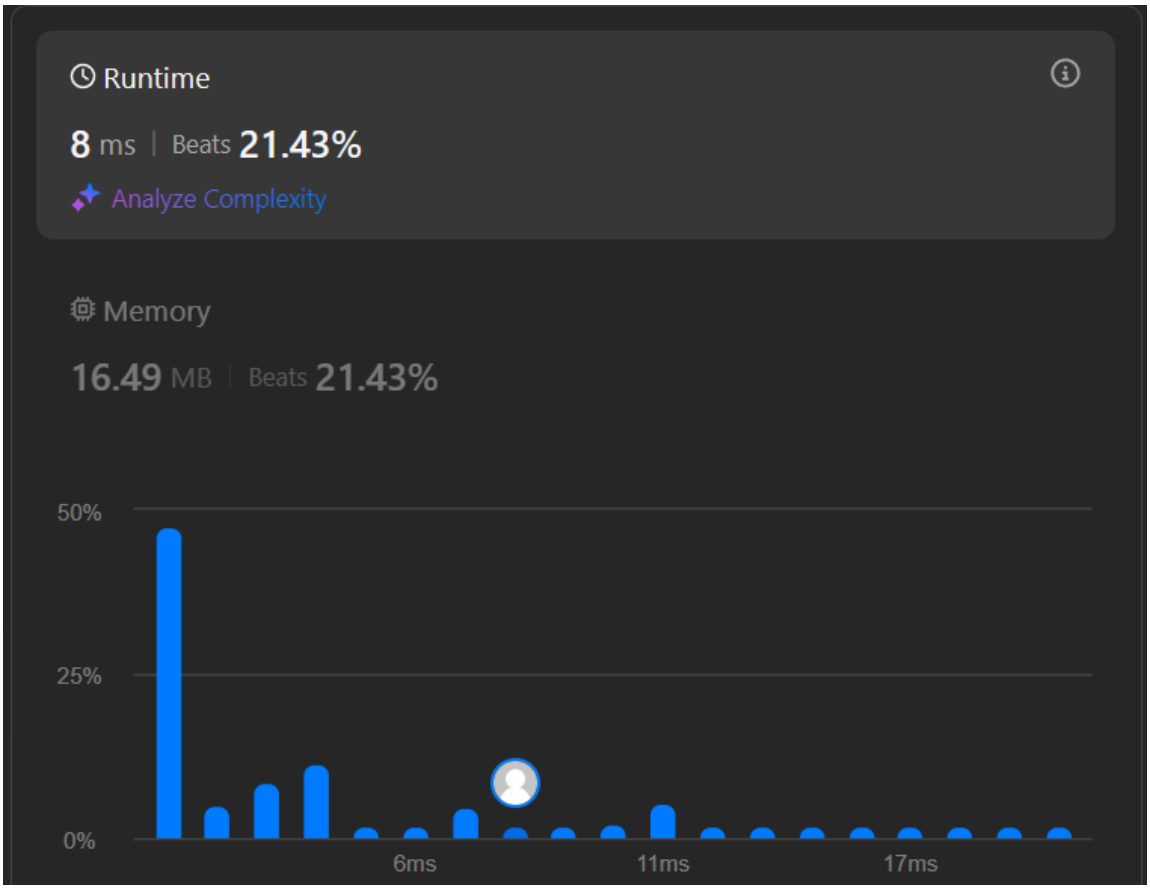
Result:



7. **Beautiful Array:**

```
class Solution {
public:
    vector<int> beautifulArray(int n) {
        if (n == 1) return {1};
        vector<int> left = beautifulArray((n + 1) >> 1);
        vector<int> right = beautifulArray(n >> 1);
        vector<int> ans(n);
        int i = 0;
        for (int& x : left) ans[i++] = x * 2 - 1;
        for (int& x : right) ans[i++] = x * 2;
        return ans;
    }
};
```

Result:

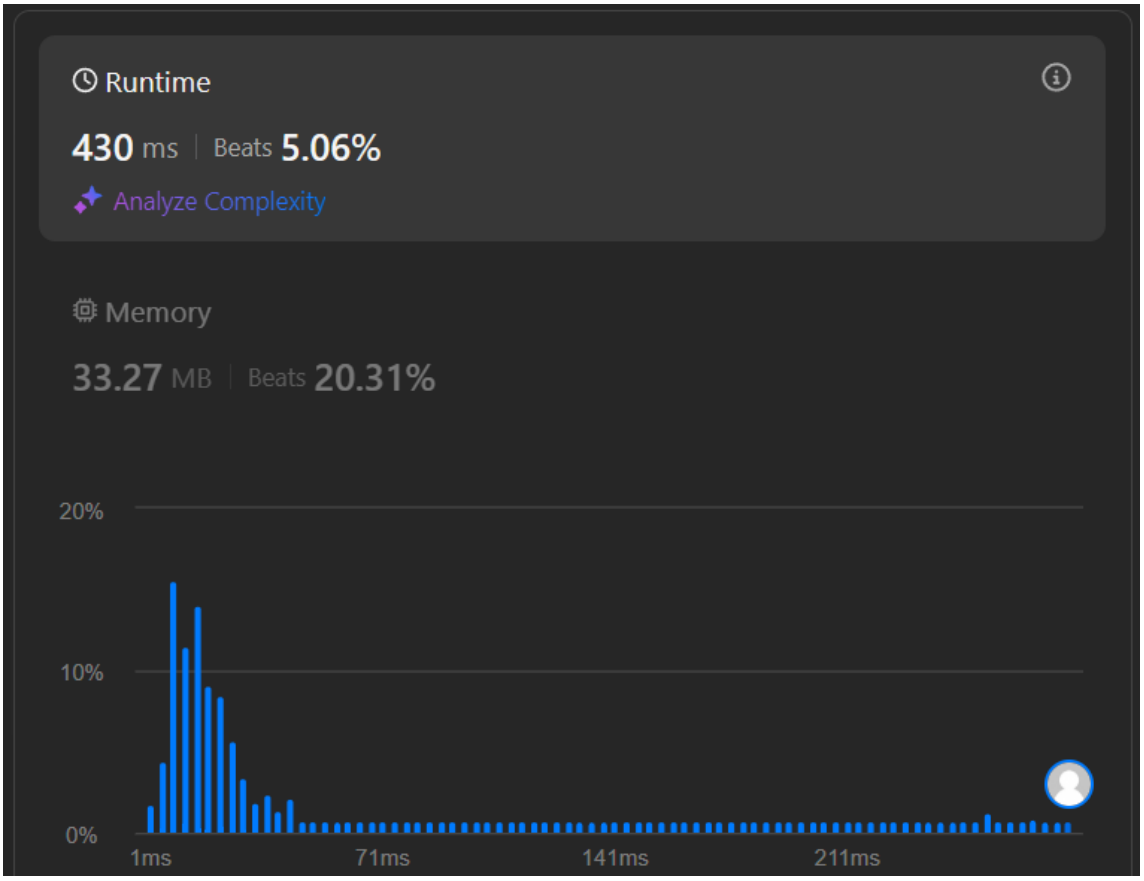


8. **The Skyline Problem:**

```
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
        set<int> poss;
        map<int, int> m;
        for (auto v : buildings) {
            poss.insert(v[0]);
            poss.insert(v[1]);
        }

        int i = 0;
        for (int pos : poss)
            m.insert(pair<int, int>(pos, i++));

        vector<int> highs(m.size(), 0);
```

```
        for (auto v : buildings) {
            const int b = m[v[0]], e = m[v[1]];
            for (int i = b; i < e; ++i)
                highs[i] = max(highs[i], v[2]);
        }

        vector<vector<int>> res;
        vector<int> mm(poss.begin(), poss.end());
        for (int i = 0; i < highs.size(); i++) {
            if (i+1 < highs.size() && highs[i] != highs[i + 1])
                res.push_back({mm[i], highs[i]});
            else {
                const int start = i;
                res.push_back({mm[start], highs[i]});
                while (i+1 < highs.size() && highs[i] == highs[i + 1])
                    ++i;
            }
        };
        return res;
    }
};
```

Result:

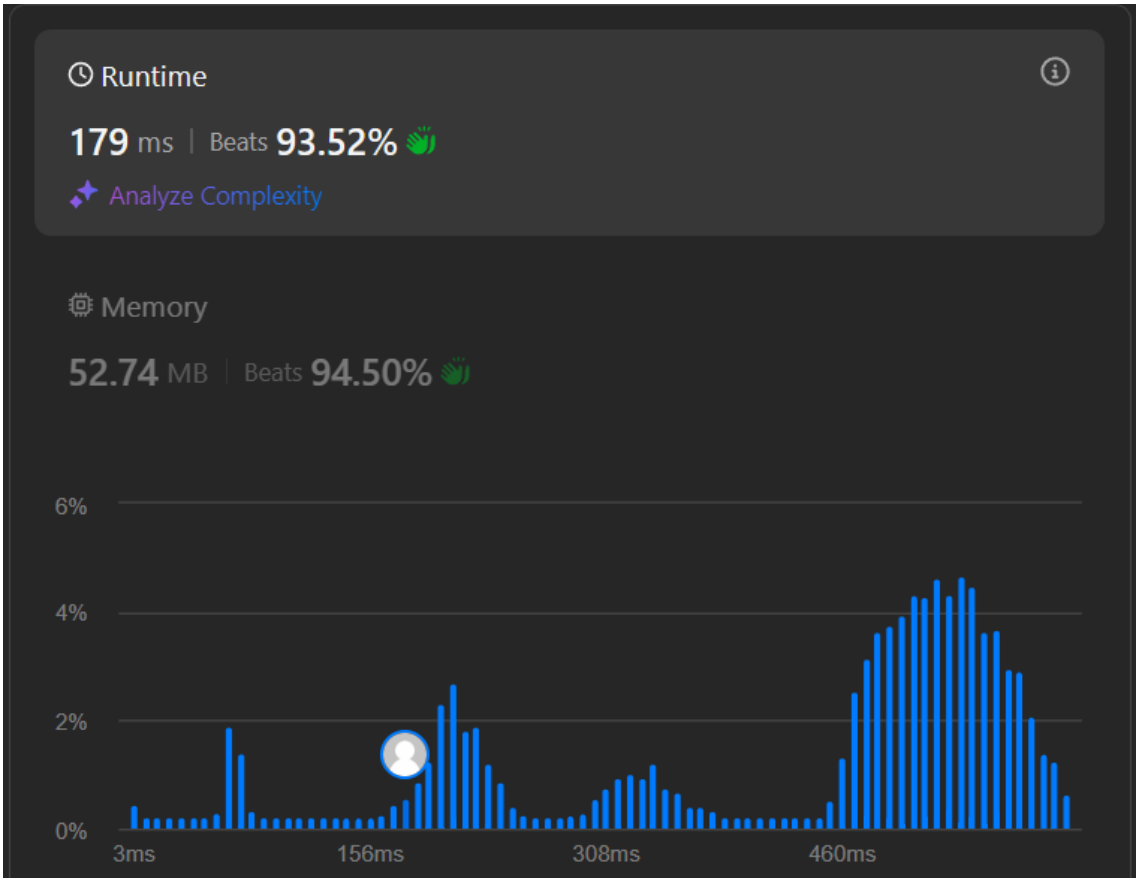


9. **Reverse Pairs:**

```
class Solution {
int mergeSort(int l, int r, vector<int>& nums){
    int t[r+1];
    if (l >= r) {
        return 0;
    }
    int mid = (l + r) >> 1;
    int ans = mergeSort(l, mid, nums) + mergeSort(mid + 1, r, nums);
    int i = l, j = mid + 1, k = 0;
    while (i <= mid && j <= r) {
        if (nums[i] <= nums[j] * 2LL) {
            ++i;
        } else {
            ans += mid - i + 1;
            ++j;
        }
    }
    i = l;
```

```
    j = mid + 1;
    while (i <= mid && j <= r) {
        if (nums[i] <= nums[j]) {
            t[k++] = nums[i++];
        } else {
            t[k++] = nums[j++];
        }
    }
    while (i <= mid) {
        t[k++] = nums[i++];
    }
    while (j <= r) {
        t[k++] = nums[j++];
    }
    for (i = l; i <= r; ++i) {
        nums[i] = t[i - l];
    }
    return ans;
};

public:
    int reversePairs(vector<int>& nums) {
        int n = nums.size();
        return mergeSort(0, n - 1, nums);
    }
};
```

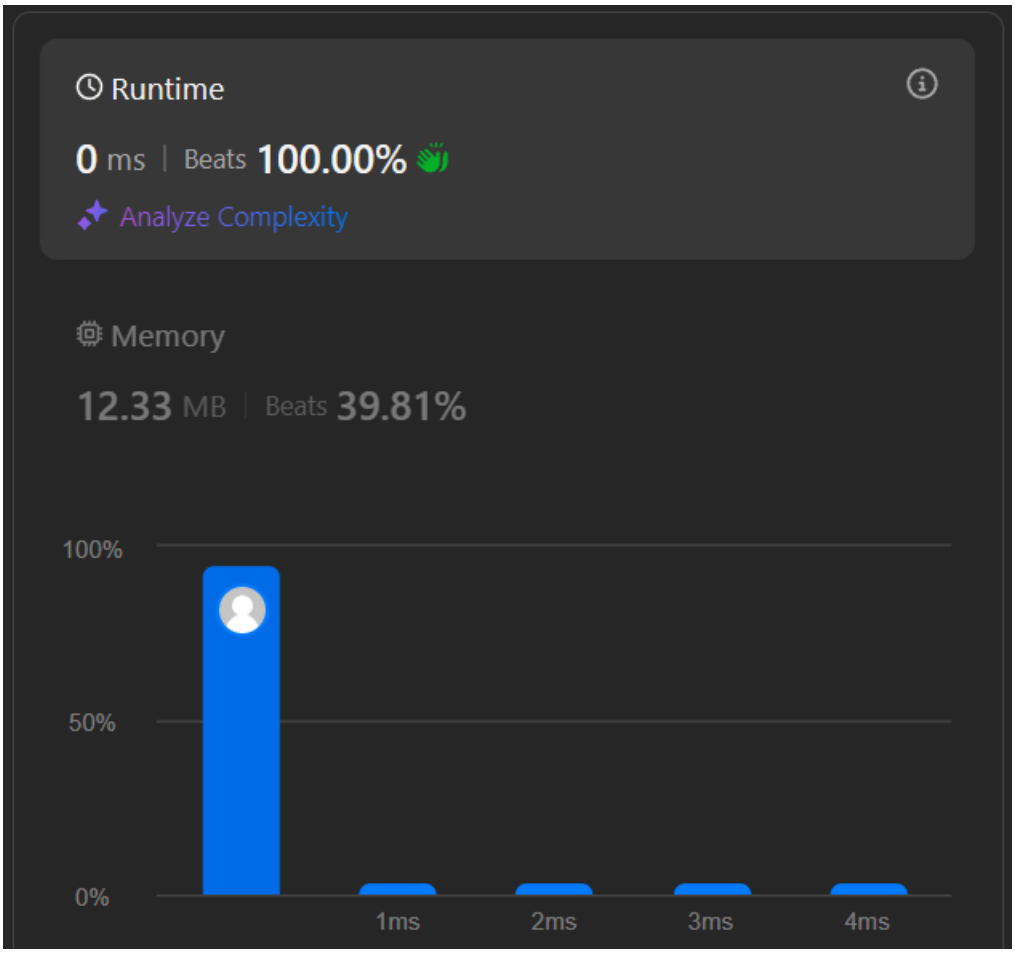
Result:



10. <u>Merge Sorted Array:</u>

```
class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        int j=0;
        for(int i=nums1.size()-1;i>=0;i--){
            if(nums1[i]!=0){
                j=i+1;
                break;
            }
        }
        for(int i=0;i<n;i++){
            nums1[j]=nums2[i];
            j++;
        }
        sort(nums1.begin(),nums1.end());
    }
};
```

Result:

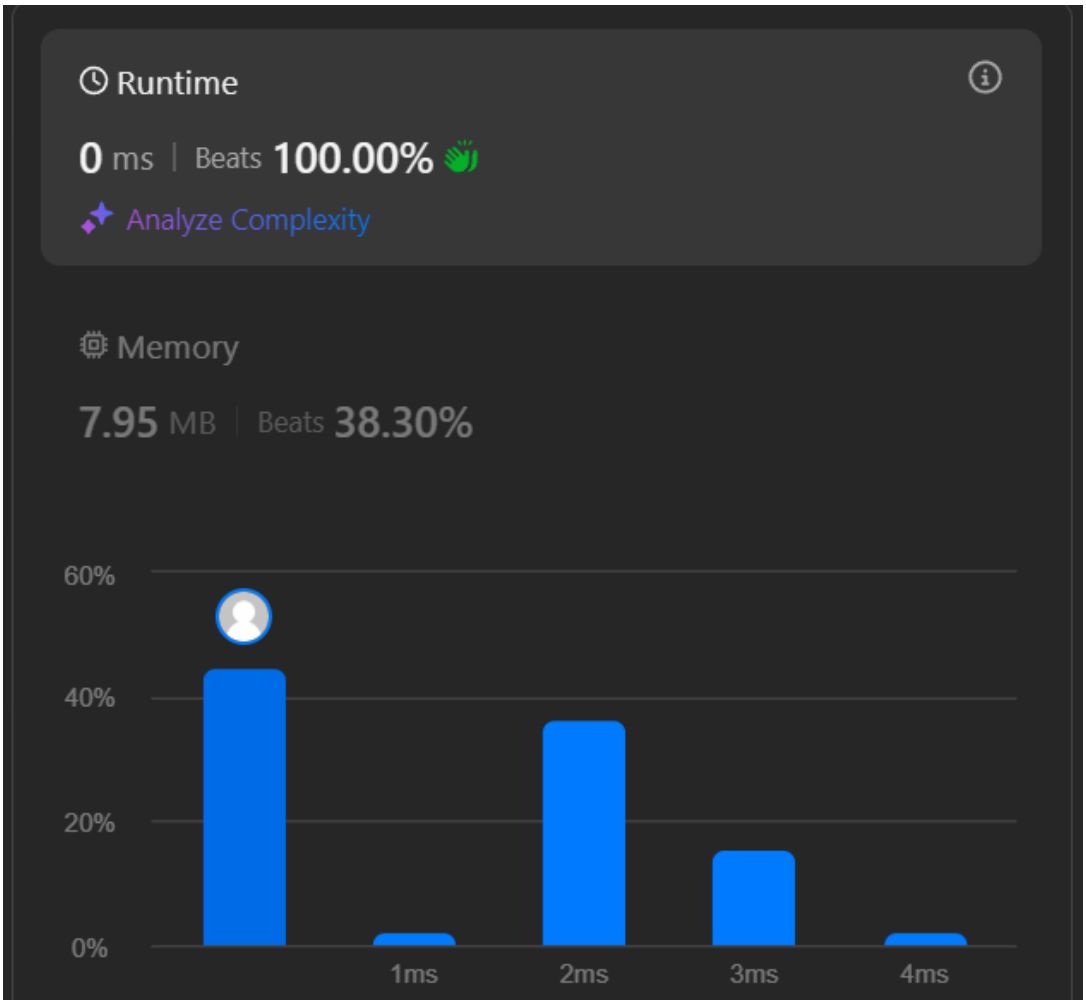


11. First Bad Version:

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

class Solution {
public:
    int firstBadVersion(int n) {
        int left = 1, right = n;
        while (left < right) {
            int mid = left + ((right - left) >> 1);
            if (isBadVersion(mid)) {
                right = mid;
            } else {
                left = mid + 1;
            }
```

```
        }
        return left;
    }
};
```

Result:

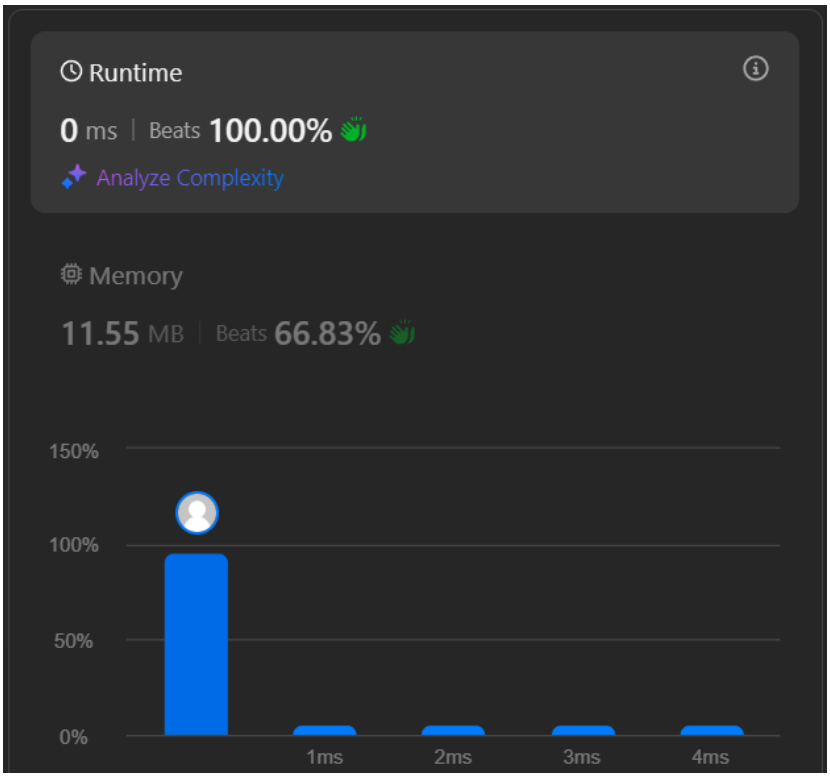


12. Sort Colors:

```
class Solution {
public:
    void sortColors(vector<int>& nums) {
        vector<int> total_count(3,0);
        int j=0;
        for(int i=0;i<nums.size();i++){
```

```
            total_count[nums[i]]++;
        }
        for(int i=0;i<3;i++){
            while(total_count[i]>0){
                nums[j]=i;
                j++;
                total_count[i]--;
            }
        }
    }
};
```

Result:

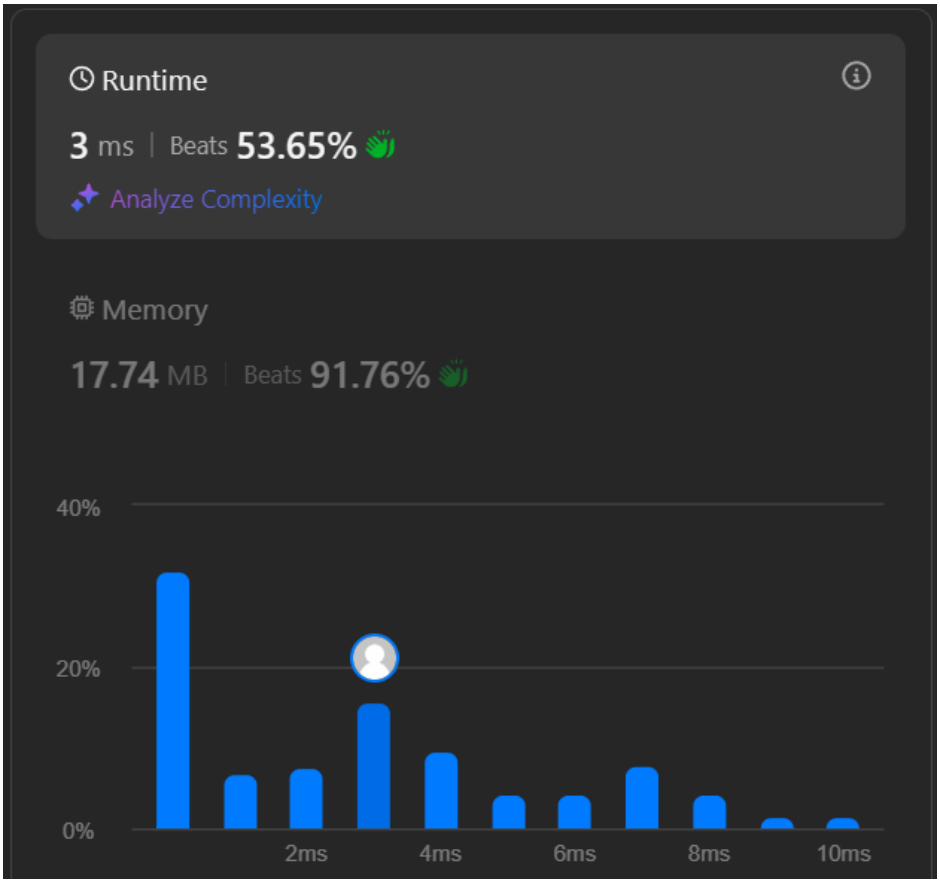


13. <u>Top K Frequent Elements in an Array:</u>

```
class Solution {
public:
    vector<int> topKFrequent(vector<int>& nums, int k) {
        unordered_map<int, int> cnt;
        using pii = pair<int, int>;
        for (int x : nums) {
            ++cnt[x];
```

```
        }
        priority_queue<pii, vector<pii>, greater<pii>> pq;
        for (auto& [x, c] : cnt) {
            pq.push({c, x});
            if (pq.size() > k) {
                pq.pop();
            }
        }
        vector<int> ans;
        while (!pq.empty()) {
            ans.push_back(pq.top().second);
            pq.pop();
        }
        return ans;
    }
};
```

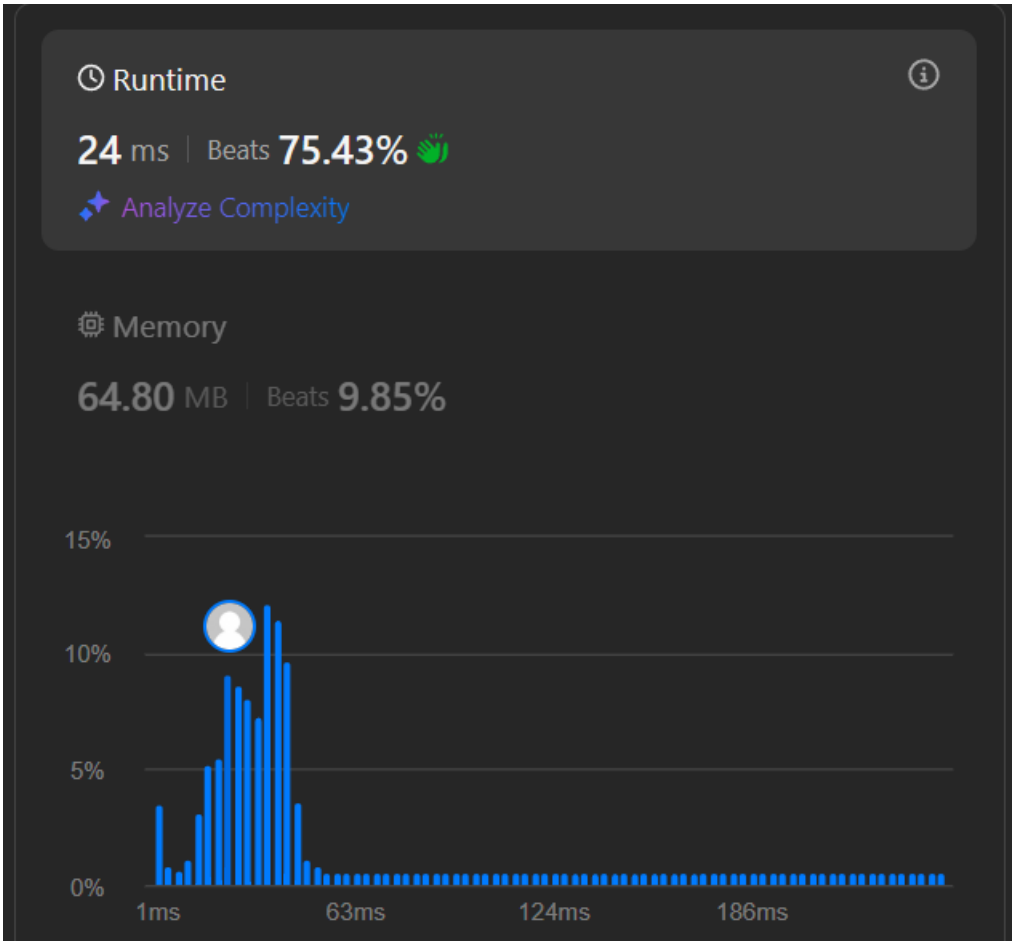
Result:



14. <u>Kth Largest Element in an Array:</u>

```
class Solution {
public:
    int findKthLargest(vector<int>& nums, int k) {
        priority_queue<int> pq;
        for(int i=0;i<nums.size();i++) pq.push(nums[i]);
        for(int i=1;i<k;i++) pq.pop();
        return pq.top();
    }
};
```

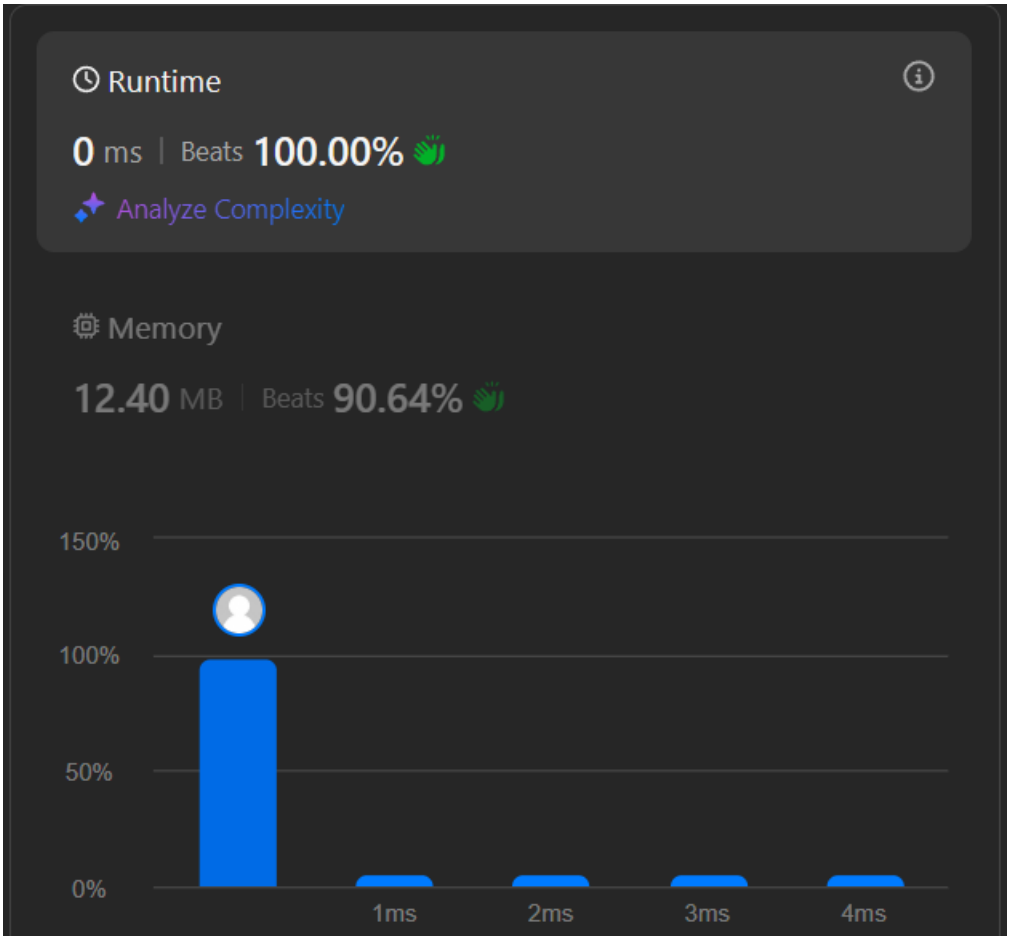
<u>Result:</u>



15. <u>Find Peak Element:</u>

```
class Solution {
public:
    int findPeakElement(vector<int>& nums) {
        if(nums.size()==2 && nums[1]>nums[0]) return 1;
        for(int i=1;i<nums.size()-1;i++){
            if(nums[i]>nums[i+1] && nums[i]>nums[i-1]) return i;
            if(i==nums.size()-2 && nums[i+1]>nums[i]) return i+1;
        }
        return 0;
    }
};
```

<u>Result:</u>

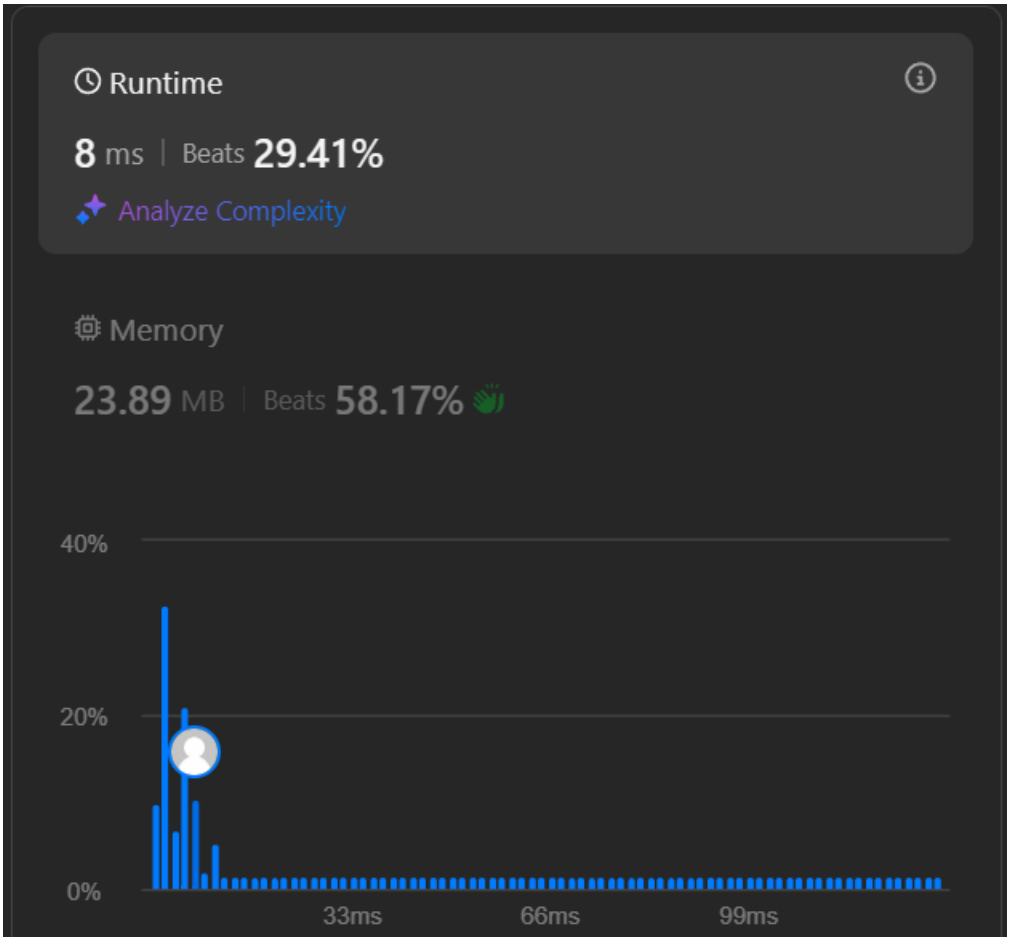


16. <u>Merge Intervals:</u>

```
class Solution {
public:
    vector<vector<int>> merge(vector<vector<int>>& intervals) {
        sort(intervals.begin(), intervals.end());
        int st = intervals[0][0], ed = intervals[0][1];
        vector<vector<int>> ans;
        for (int i = 1; i < intervals.size(); ++i) {
            if (ed < intervals[i][0]) {
                ans.push_back({st, ed});
                st = intervals[i][0];
                ed = intervals[i][1];
            } else {
                ed = max(ed, intervals[i][1]);
            }
        }
        ans.push_back({st, ed});
```

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

Result:

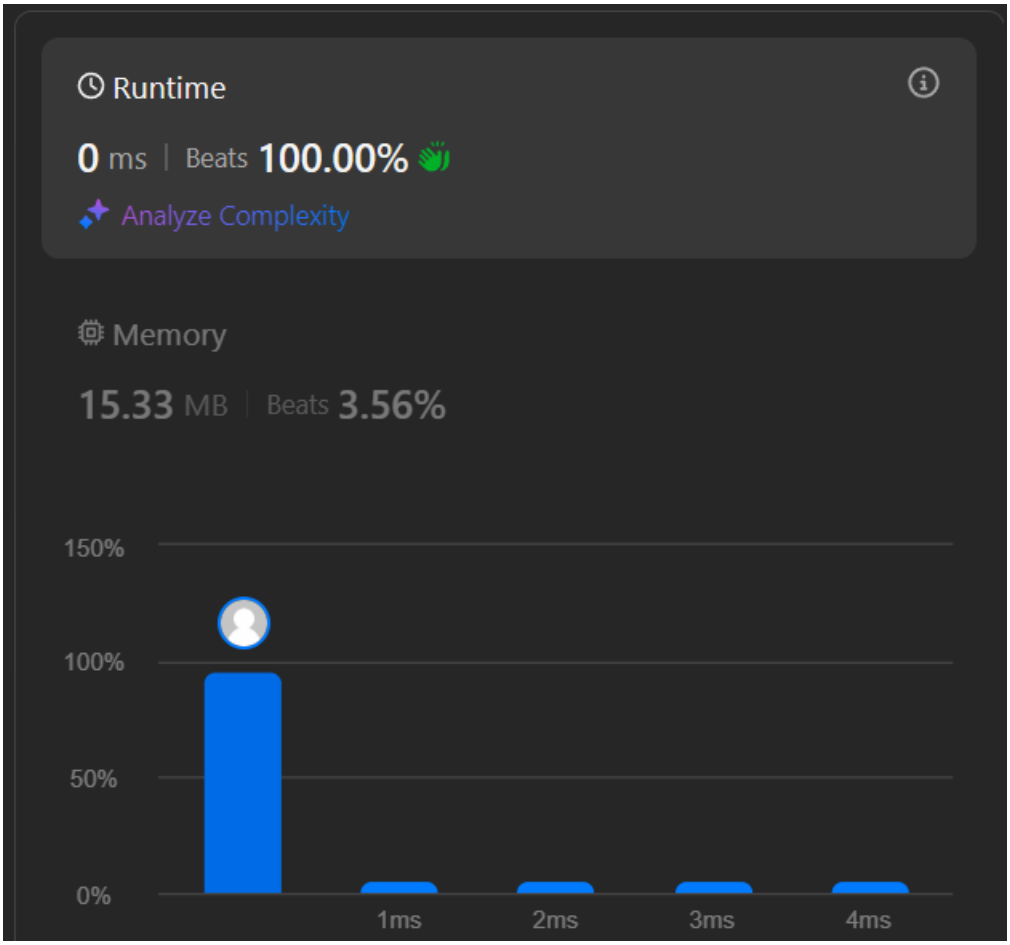


17. <u>Search in a Rotated Sorted Array:</u>

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

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

Result:

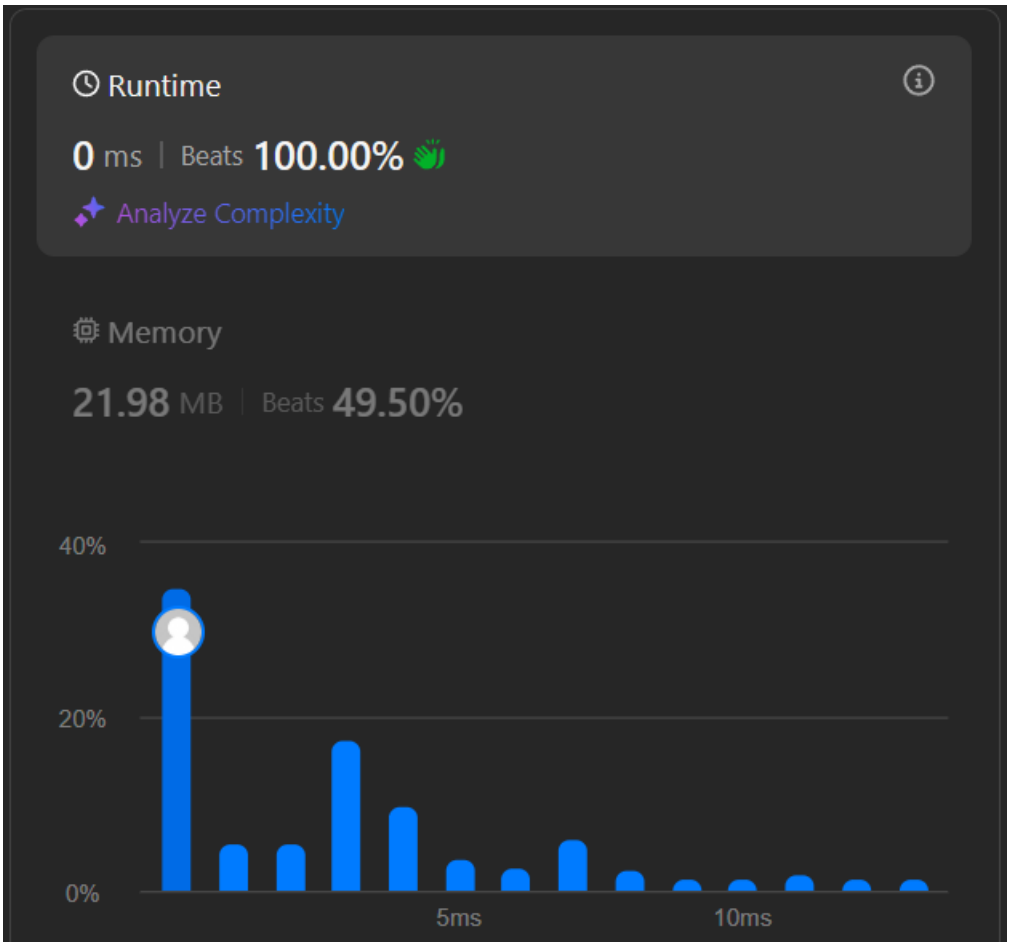


18. Wiggle Sort II:

```
class Solution {
public:
    void wiggleSort(vector<int>& nums) {
        vector<int> arr = nums;
```

```
        sort(arr.begin(), arr.end());
        int n = nums.size();
        int i = (n - 1) >> 1, j = n - 1;
        for (int k = 0; k < n; ++k) {
            if (k % 2 == 0)  nums[k] = arr[i--];
            else   nums[k] = arr[j--];
        }
    }
};
```

Result:

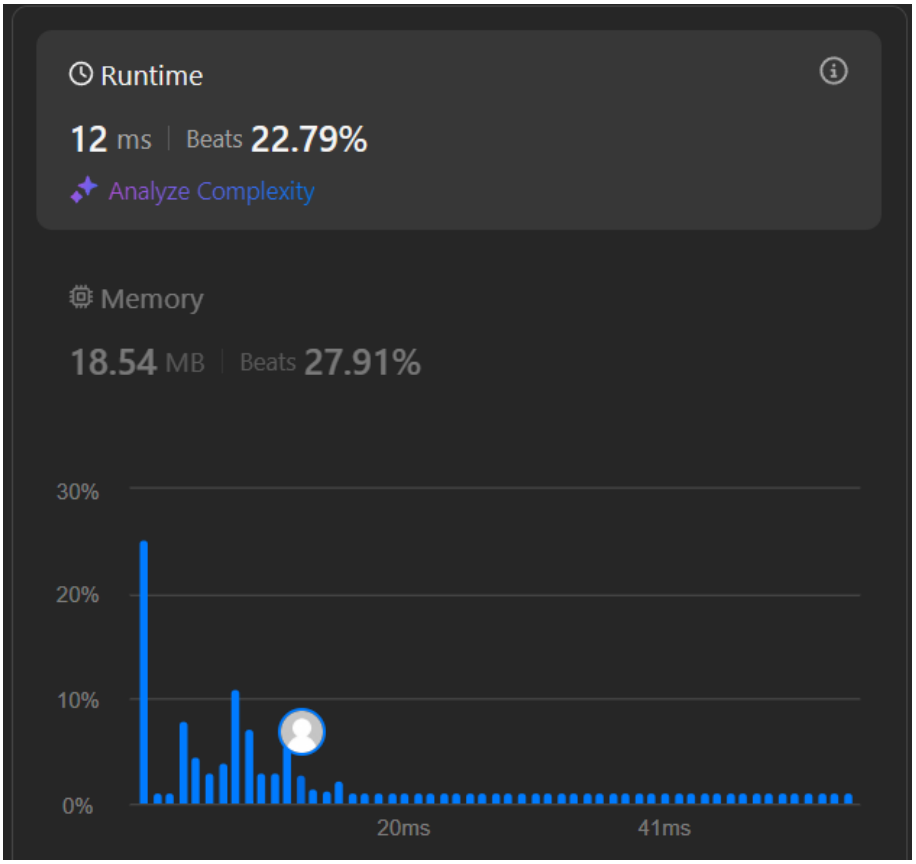


19. Kth Smallest Element in a Sorted Matrix:

```
class Solution {
public:
    int kthSmallest(vector<vector<int>>& matrix, int k) {
        int count=0;
        vector<int> values;
```

```
        for(int i=0;i<matrix.size();i++){
            for(int j=0;j<matrix[i].size();j++){
                values.push_back(matrix[i][j]);
            }
        }
        sort(values.begin(),values.end());
        return values[k-1];
        return 0;
    }
};
```

Output:

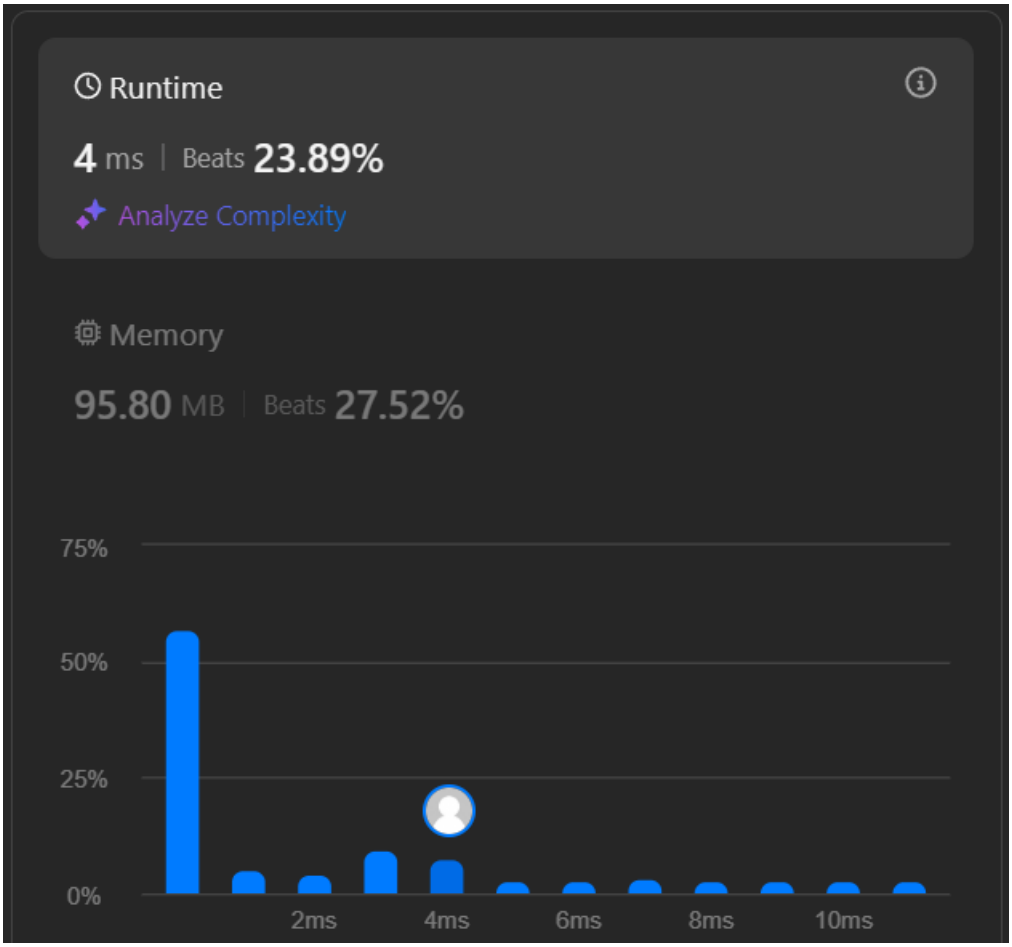


20. <u>Median of Two Sorted Arrays:</u>

```
class Solution {
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>&
nums2) {
        int i=0,j=0,k=0;
        double median;
```

```
        int n1=nums1.size();
        int n2=nums2.size();
        int n=n1+n2;
        vector<int> arr(n1+n2,0);
        while(k<(n1+n2)){
            if(i<n1 && j==n2){
                arr[k++]=(nums1[i++]);
            }
            else if(j<n2 && i==n1){
                arr[k++]=(nums2[j++]);
            }
            else{
                if(nums1[i]<=nums2[j]){
                    arr[k++]=nums1[i++];
                }
                else{
                    arr[k++]=(nums2[j++]);
                }
            }
        }
        if(n%2!=0) median=arr[(n/2)];
        else{
            cout<<double(arr[n/2])<<' '<<double(arr[n/2-1]);
            median=(double(arr[n/2])+double(arr[n/2-1]))/2;
        }
        return median;
    }
};
```

Output:


Runtime
4 ms | Beats 23.89%
Analyze Complexity
Memory
95.80 MB | Beats 27.52%
75%
50%
25%
0%
2ms
4ms
6ms
8ms
10ms