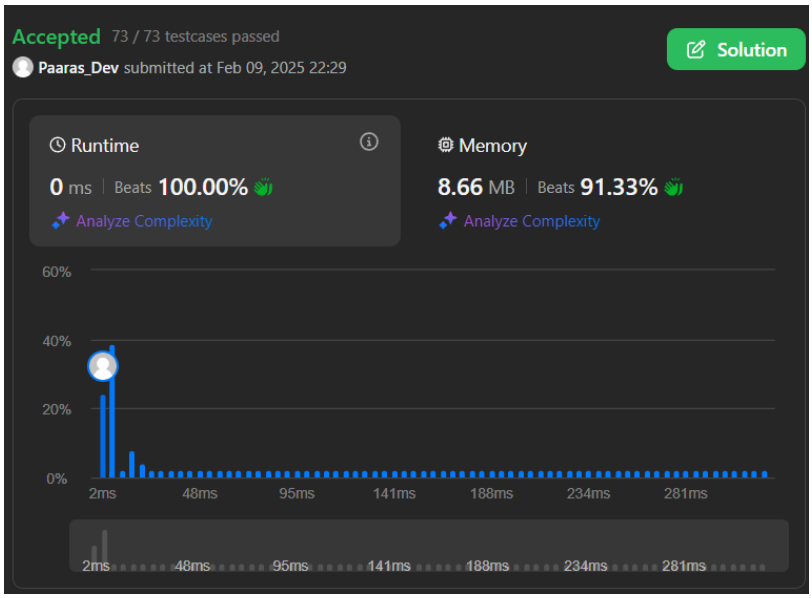
ASSIGNMENT -1 (ADVANCED PROGRAMMING)

1. **Problem 1:** Longest Nice Substring
2. **Implementation/Code:**

```
class Solution {
public:
    string longestNiceSubstring(string s) {
        if (s.length() < 2)
            return "";

        for (int i = 0; i < s.length(); i++) {
            char ch = s[i];
            if (s.find(tolower(ch)) != string::npos && s.find(toupper(ch)) !=
string::npos) {
                continue;
            }
            string left = longestNiceSubstring(s.substr(0, i));
            string right = longestNiceSubstring(s.substr(i + 1));
            return left.length() >= right.length() ? left : right;
        }
        return s;
    }
};
```

Output:

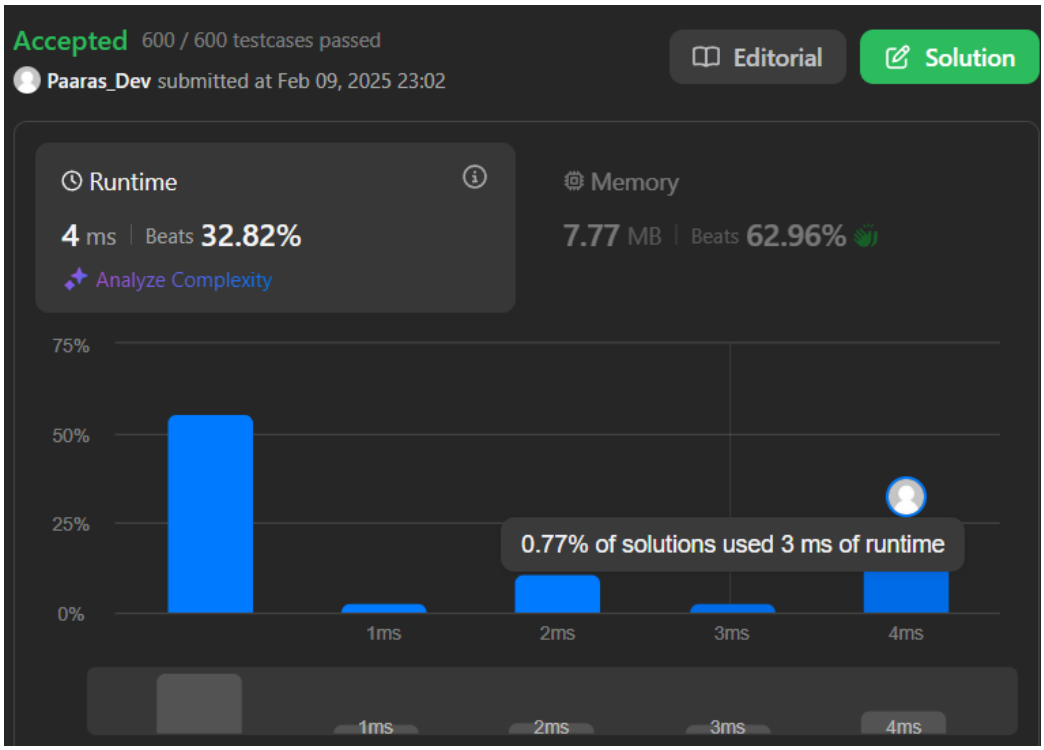


1. **Problem 2:** Reverse Bits

2. **Implementation/Code:**

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

Output:

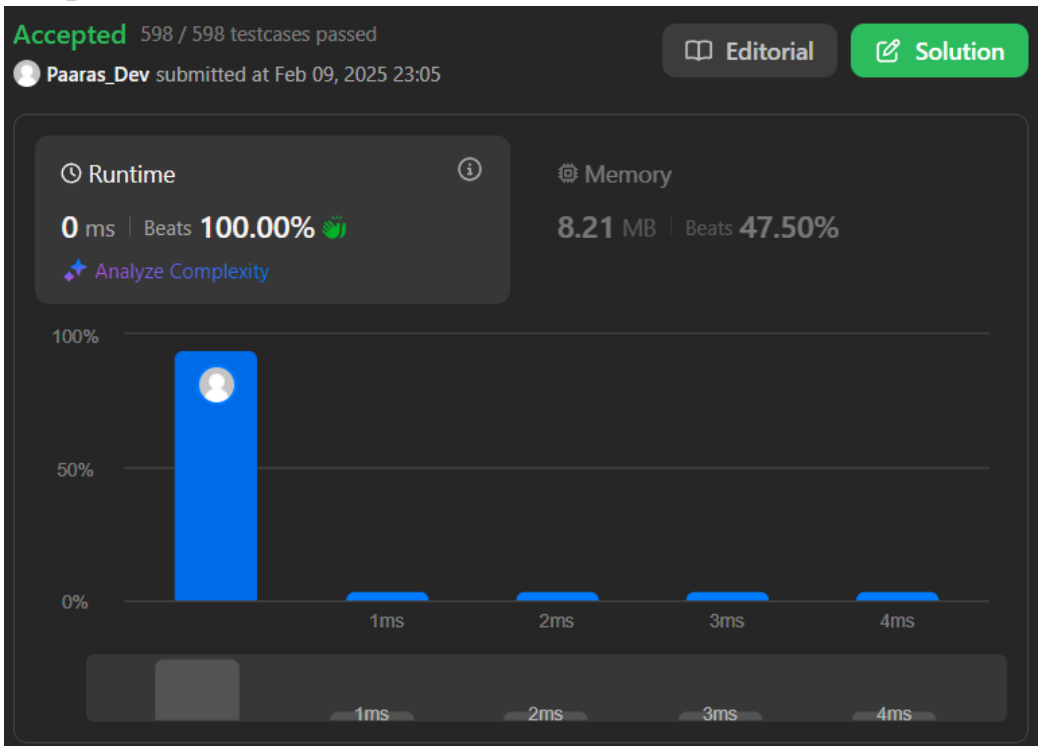


1. **Problem 3:** Number of 1 bits

2. **Implementation/code:**

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

Output:

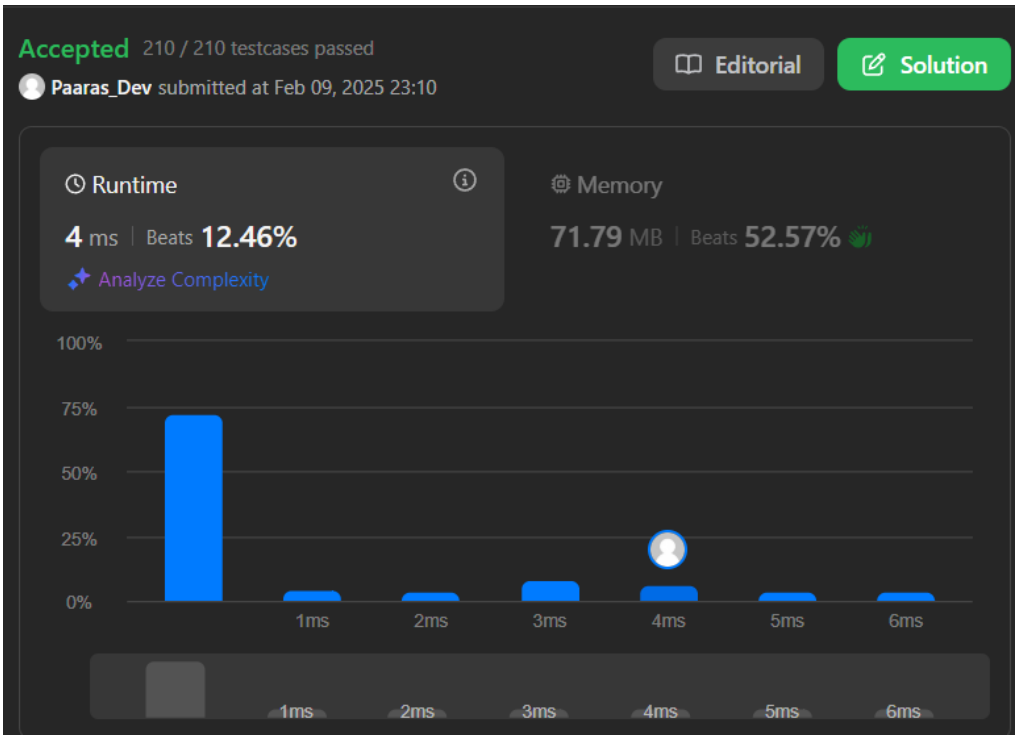


1. **Problem 4: Maximum Sub array**

2. **Implementation/code:**

```
int maxSubArray(vector<int>& nums) {
    int maxSum = nums[0], currentSum = 0;
    for (int num : nums) {
        currentSum = max(num, currentSum + num);
        maxSum = max(maxSum, currentSum);
    }
    return maxSum;
}
```

Output:



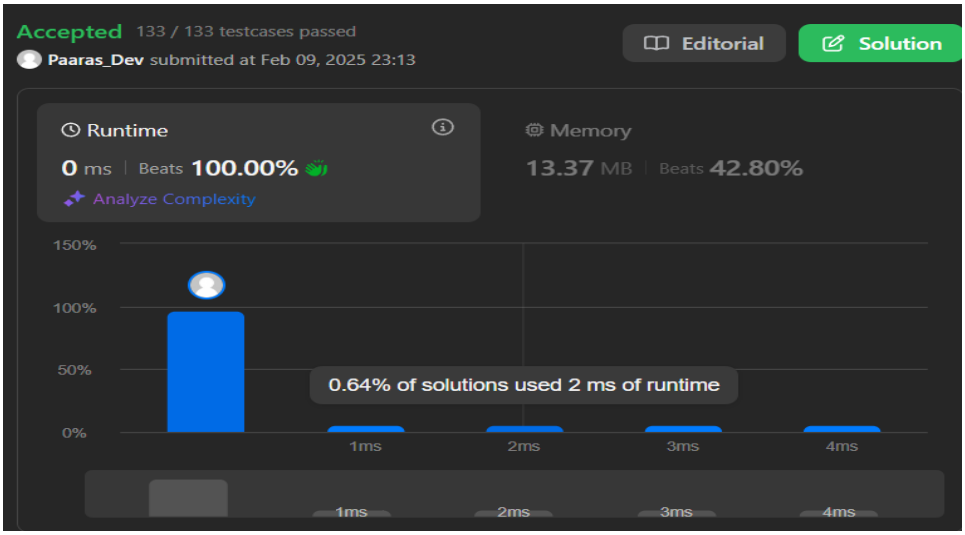
1. **Problem 5: Search a 2D Matrix II**

2. **Implementation/Code:**

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

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

};
```

Output:



1. **Problem 6: Super Pow**

2. **Implementation/Code:**

```
class Solution {
public:

const int MOD = 1337;
```

```
int pow(int a, int b) {
    int res = 1;
    a %= MOD;
    for (int i = 0; i < b; i++) {
        res = (res * a) % MOD;
    }
    return res;
}

int superPow(int a, vector<int>& b) {
    int res = 1;
    for (int i = b.size() - 1; i >= 0; i--) {
        res = (res * pow(a, b[i])) % MOD;
        a = pow(a, 10);
    }
    return res;
}
};
```

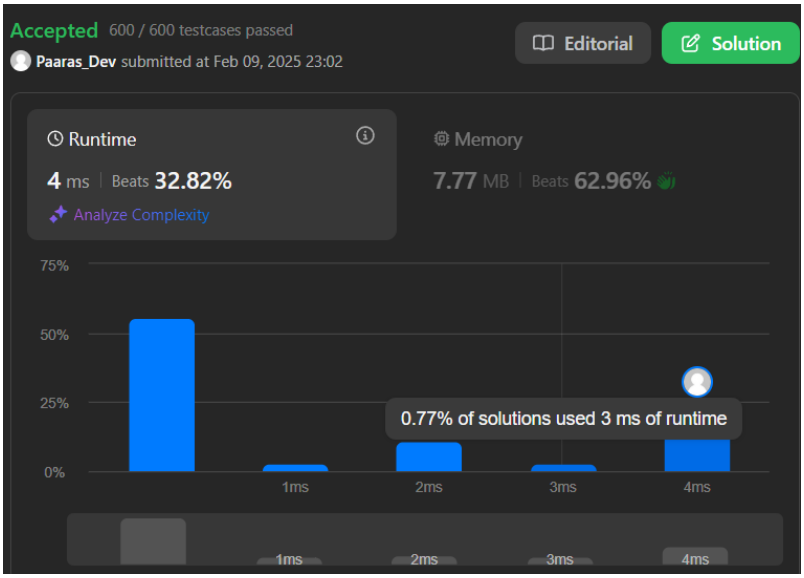
Output:

1. **Problem 7: Beautiful Array**

2. **Implementation/code:**

```
import java.util.*;
public class Solution {
    public int[] beautifulArray(int N) {
        List<Integer> result = new ArrayList<>();
        result.add(1);
        while (result.size() < N) {
            List<Integer> temp = new ArrayList<>();
            for (int num : result) {
                if (num * 2 - 1 <= N) temp.add(num * 2 - 1);          }
            for (int num : result) {
                if (num * 2 <= N) temp.add(num * 2);          }
            result = temp;       }
        int[] arr = new int[result.size()];
        for (int i = 0; i < result.size(); i++) {
            arr[i] = result.get(i);       }
        return arr;    }}
```

3. **Output:**

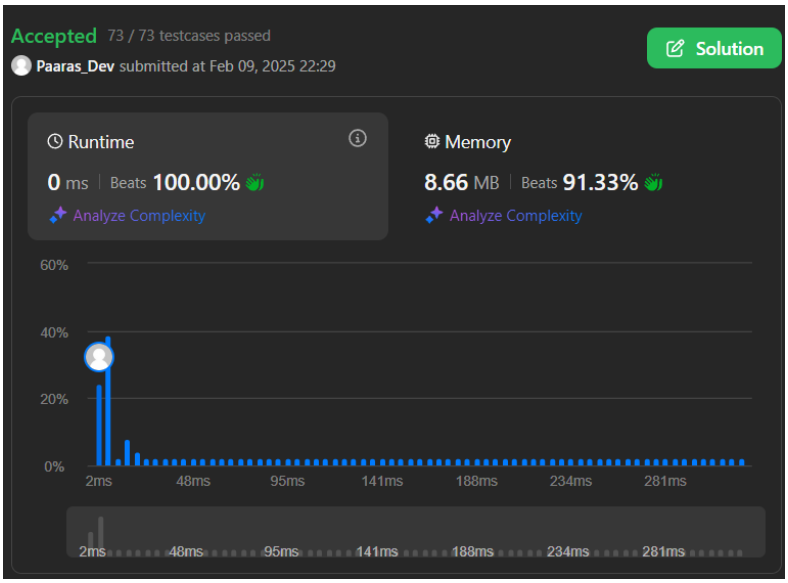


1. **Problem 8: The Skyline Problem.**
2. **Implementation/code:**

```
import java.util.*;

class Solution {
    public List<List<Integer>> getSkyline(int[][] buildings) {
        return divideAndConquer(buildings, 0, buildings.length - 1);    }
    private List<List<Integer>> divideAndConquer(int[][] buildings, int left,
int right) {
        if (left > right) return new ArrayList<>();
        if (left == right) {
            List<List<Integer>> result = new ArrayList<>();
            result.add(Arrays.asList(buildings[left][0], buildings[left][2]));
            result.add(Arrays.asList(buildings[left][1], 0));
            return result;        }
        int mid = left + (right - left) / 2;
        List<List<Integer>> leftSkyline = divideAndConquer(buildings, left,
mid);
        List<List<Integer>> rightSkyline = divideAndConquer(buildings, mid
+ 1, right);
        return mergeSkylines(leftSkyline, rightSkyline);    }
    private List<List<Integer>> mergeSkylines(List<List<Integer>> left,
List<List<Integer>> right) {
        List<List<Integer>> result = new ArrayList<>();
        int h1 = 0, h2 = 0, i = 0, j = 0;

        while (i < left.size() && j < right.size()) {
            List<Integer> point1 = left.get(i);
            List<Integer> point2 = right.get(j);            int x;
            if (point1.get(0) < point2.get(0)) {
                x = point1.get(0);
                h1 = point1.get(1);
                i++;
            } else if (point1.get(0) > point2.get(0)) {
```

```
                x = point2.get(0);
                h2 = point2.get(1);
                j++;
            } else {
                x = point1.get(0);
                h1 = point1.get(1);
                h2 = point2.get(1);
                i++;
                j++;          }
            int maxHeight = Math.max(h1, h2);
            if (result.isEmpty() || result.get(result.size() - 1).get(1) != maxHeight)
{               result.add(Arrays.asList(x, maxHeight));          }   }
        while (i < left.size()) result.add(left.get(i++));
        while (j < right.size()) result.add(right.get(j++));

        return result;
    }
}
```

3. **Output:**



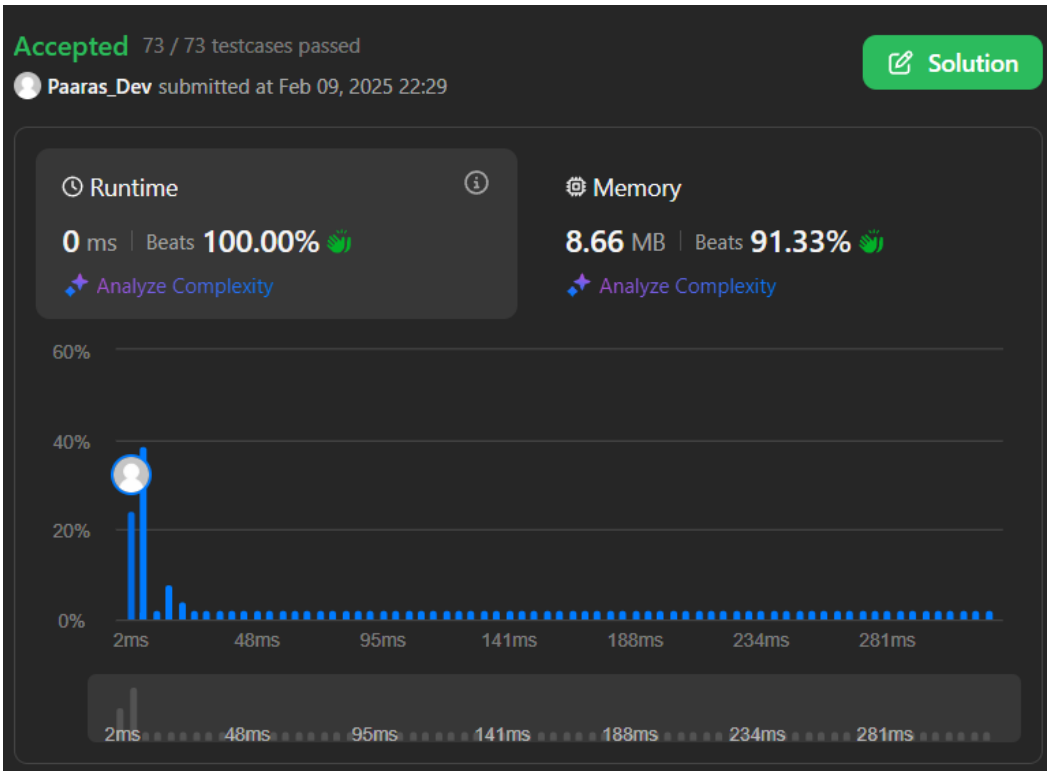
1. **Problem 9: Reverse Pairs**

2. **Implementation/code:**

```
public class Solution {
    public int reversePairs(int[] nums) {
        return mergeSort(nums, 0, nums.length - 1);
    }

    private int mergeSort(int[] nums, int left, int right) {
        if (left >= right) return 0;

        int mid = left + (right - left) / 2;
        int count = mergeSort(nums, left, mid) + mergeSort(nums, mid + 1,
right);

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

        merge(nums, left, mid, right);
        return count;
    }

    private void merge(int[] nums, int left, int mid, int right) {
        int[] temp = new int[right - left + 1];
        int i = left, j = mid + 1, k = 0;

        while (i <= mid && j <= right) {
            if (nums[i] <= nums[j]) {
                temp[k++] = nums[i++];
```

```
            } else {
                temp[k++] = nums[j++];
            }
        }

        while (i <= mid) temp[k++] = nums[i++];
        while (j <= right) temp[k++] = nums[j++];

        System.arraycopy(temp, 0, nums, left, temp.length);
    }
}
```

3. **Output:**

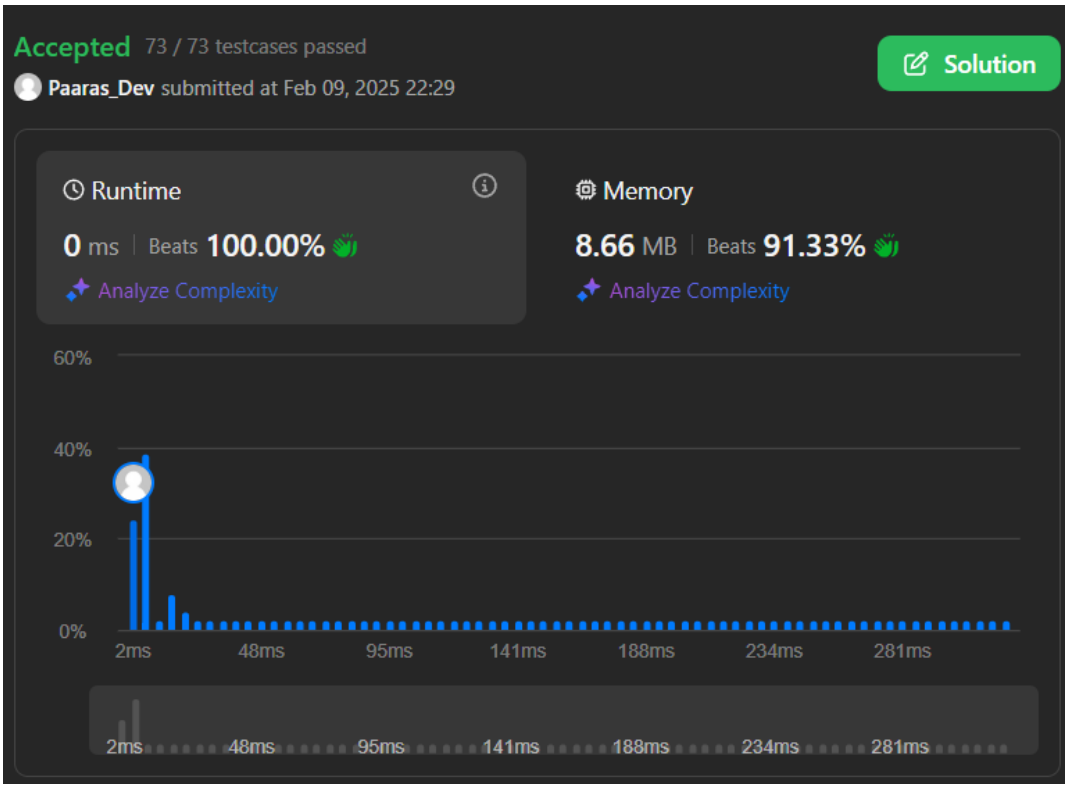


1. **Problem 10: Longest Increasing SubSequence**
2. **Implementation/Code:**

```
class Solution {
    int N = 100001;
    int[] seg = new int[2 * N];

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

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

```
    public int lengthOfLIS(int[] A, int k) {
        int result = 0;
        for (int i = 0; i < A.length; ++i) {
            int l = Math.max(0, A[i] - k);
            int r = A[i];
            int current = query(l, r) + 1;
            result = Math.max(result, current);
            update(A[i], current);
        }
        return result;
    }
}
```

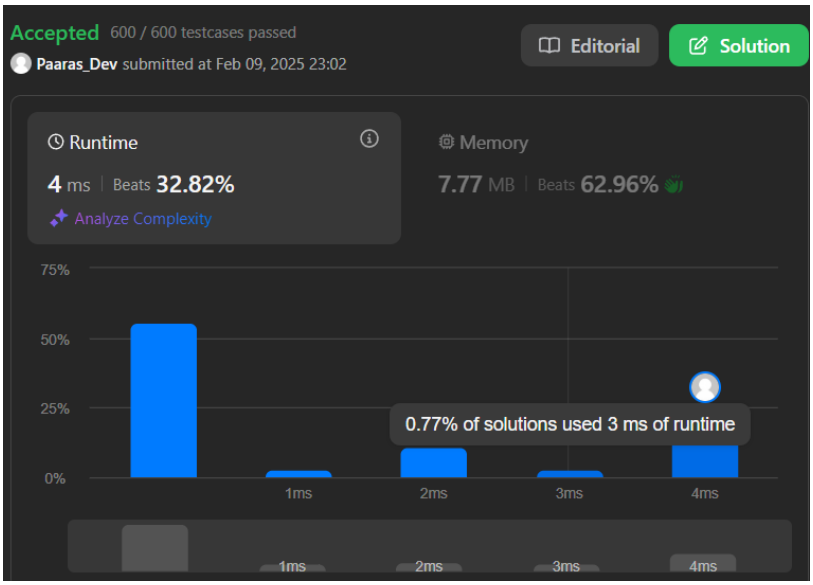
3. **Output:**



1. **Problem 11: Merge Sorted Array**
2. **Implementation/Code:**

```
public class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        int p1 = m - 1, p2 = n - 1, p = m + n - 1;

        while (p1 >= 0 && p2 >= 0) {
            if (nums1[p1] > nums2[p2]) {
                nums1[p] = nums1[p1];
                p1--;
            } else {
                nums1[p] = nums2[p2];
                p2--;
            }
            p--;
        }
        while (p2 >= 0) {
            nums1[p] = nums2[p2];
            p2--;
            p--;        }    }}
```

3. **Output:**

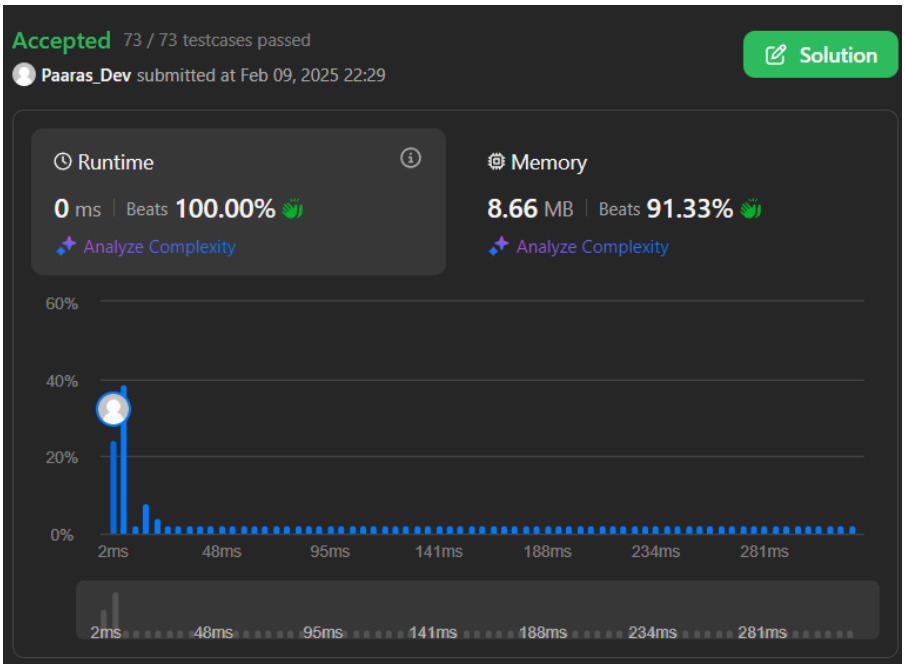


1. **Problem 12: First Bad Version**

2. **Code:**

```
public class Solution extends VersionControl {
    public int firstBadVersion(int n) {
        int left = 1, right = n;

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

            if (isBadVersion(mid)) {
                right = mid;
            } else {
                left = mid + 1;
            }       }
        return left;    }}
```

3. **Output:**

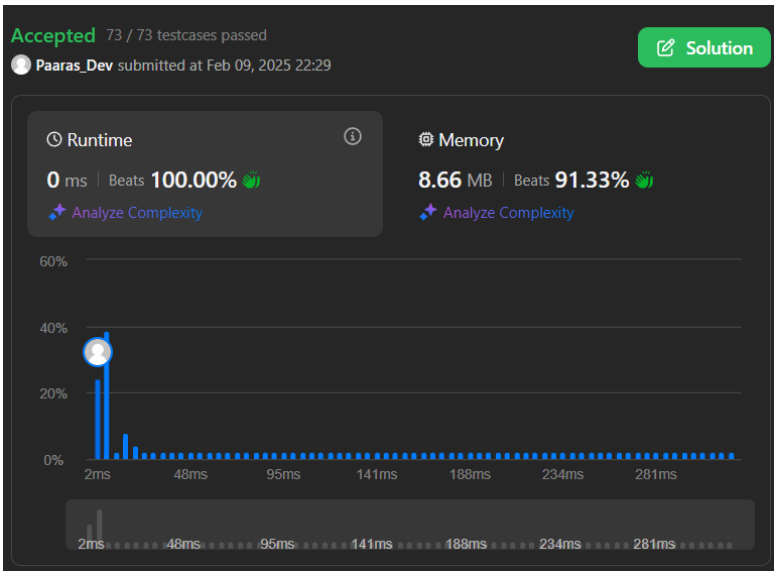


1. **Problem 13: Sort Colors**

2. **Code:**

```
class Solution {
    public void sortColors(int[] nums) {
        int low = 0, mid = 0, high = nums.length - 1;
        while (mid <= high) {
            if (nums[mid] == 0) {
                swap(nums, low++, mid++);
            } else if (nums[mid] == 1) {
                mid++;
            } else {
                swap(nums, mid, high--);
            }       }    }
    private void swap(int[] nums, int i, int j) {
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
    }
}
```

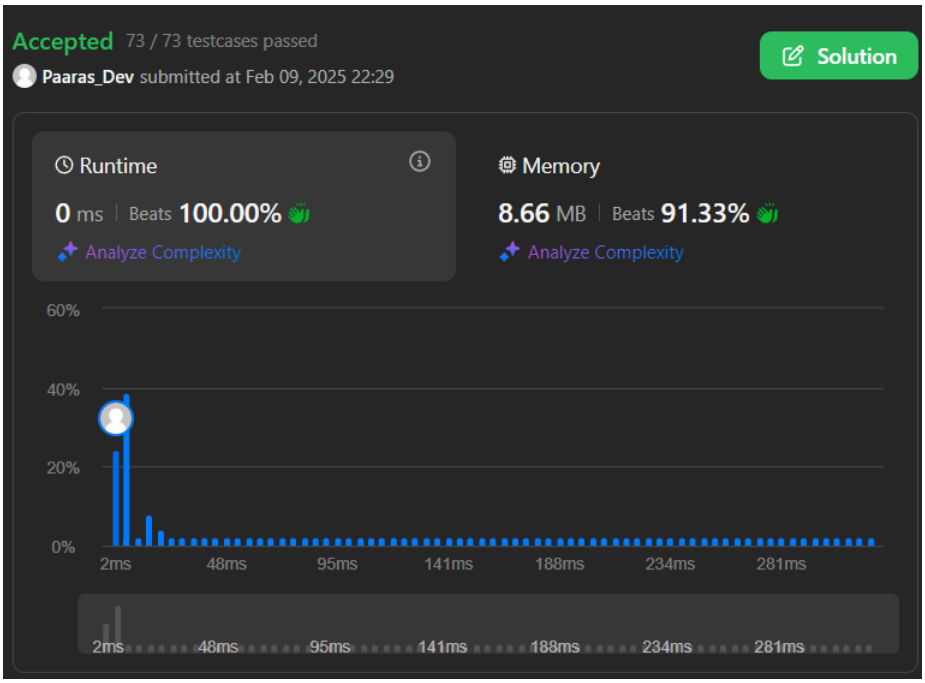
3. **Output:**



1. **Problem 14: Top K frequent Elements**
2. **Code:**

```
import java.util.*;

class Solution {
    public int[] topKFrequent(int[] nums, int k) {
        Map<Integer, Integer> freqMap = new HashMap<>();
        for (int num : nums) freqMap.put(num, freqMap.getOrDefault(num, 0)
+ 1);

        List<Integer> list = new ArrayList<>(freqMap.keySet());
        list.sort((a, b) -> freqMap.get(b) - freqMap.get(a));

        int[] result = new int[k];
        for (int i = 0; i < k; i++) result[i] = list.get(i);
        return result;
    }
}
```

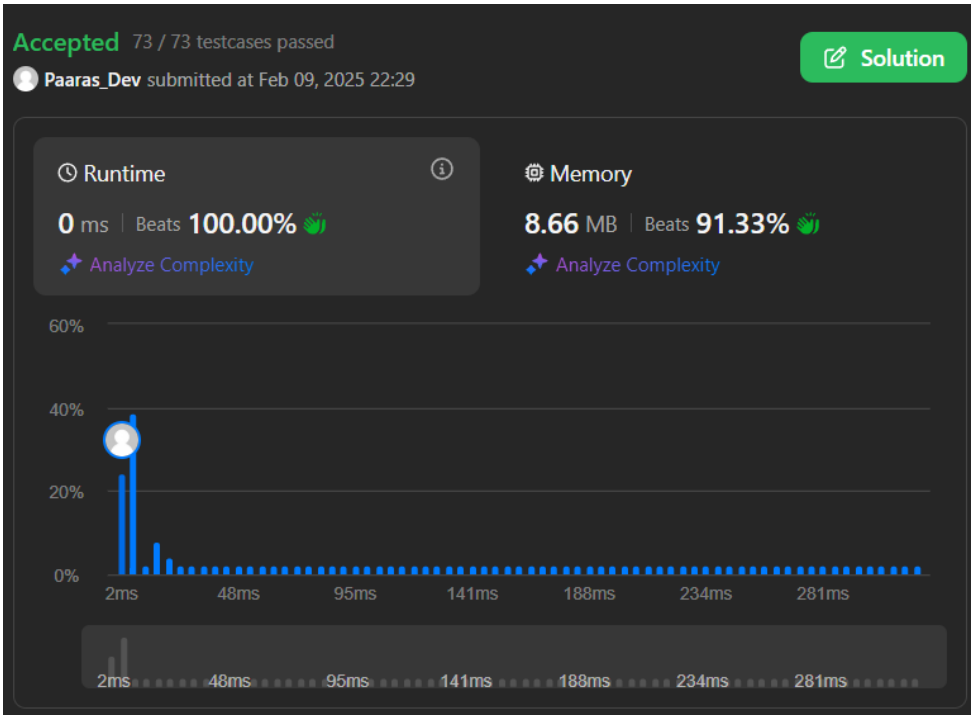
3. **Output:**



1. **Problem 15:** Kth Largest Element in an Array
2. **Code:**

```
class Solution {
    public int findKthLargest(int[] nums, int k) {
        int n = nums.length;
        int[] b = new int[n];
        int j = 0;

        for (int i = n - 1; i >= 0; i--) {
            b[j] = nums[i];
            j++;
        }

        Arrays.sort(b);
        return b[n - k];
    }
}
```

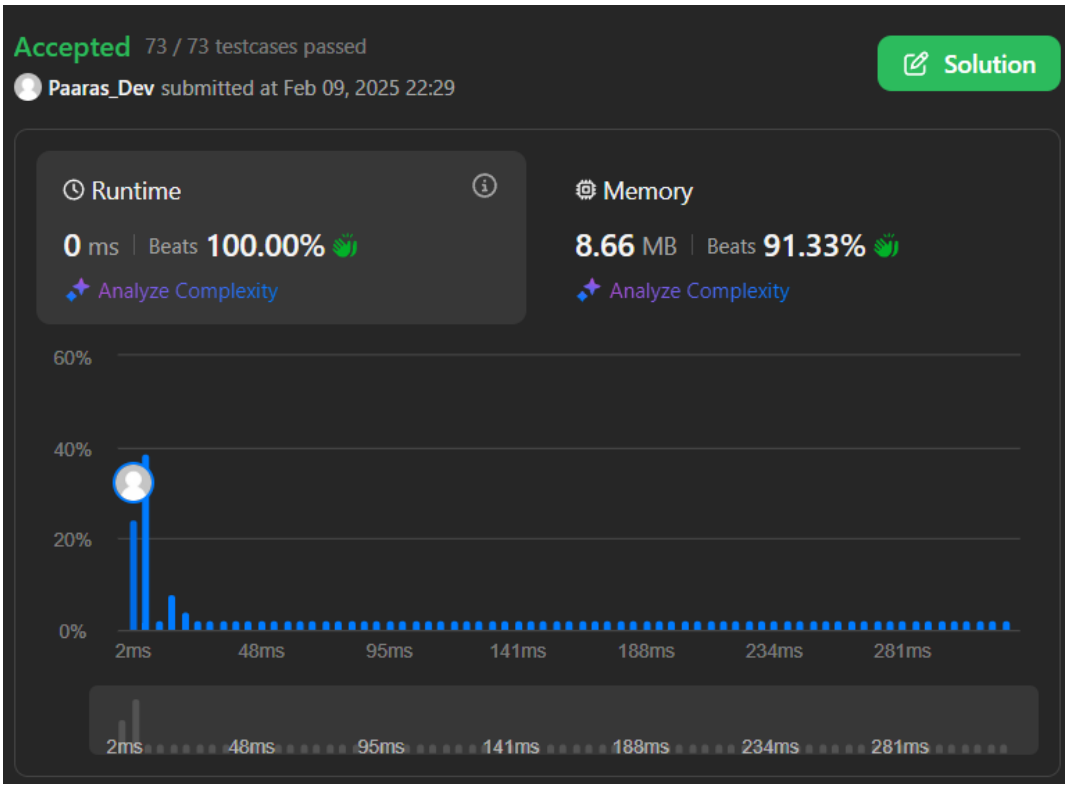
3. **Output:**



1. **Problem 16: Find Peak Element**
2. **Code:**

```
class Solution {
    public int findPeakElement(int[] nums) {
        int left = 0, right = nums.length - 1;
        while (left < right) {
            int mid = (left + right) / 2;
            if (nums[mid] > nums[mid + 1]) right = mid;
            else left = mid + 1;
        }
        return left;
    }
}
```

3. **Output:**

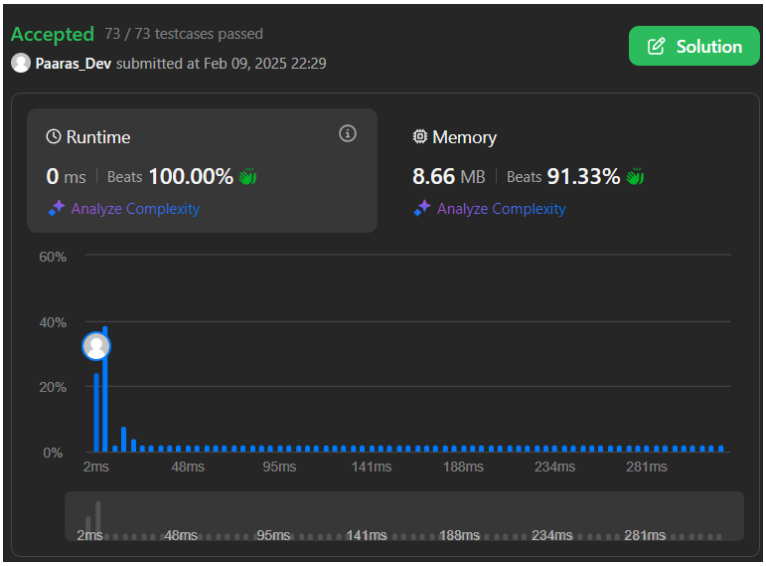


1. **Problem 17: Merge Intervals**

2. **Code:**

```
import java.util.*;
class Solution {
    public int[][] merge(int[][] intervals) {
        Arrays.sort(intervals, new Comparator<int[]>() {
            public int compare(int[] a, int[] b) {
                return a[0] - b[0];
            }
        });
        List<int[]> result = new ArrayList<>();
        for (int[] interval : intervals) {
            if (result.isEmpty() || result.get(result.size() - 1)[1] < interval[0]) {
                result.add(interval);
            } else {
                result.get(result.size() - 1)[1] = Math.max(result.get(result.size() -
1)[1], interval[1]);
            }       }
        return result.toArray(new int[result.size()][]);
    }}
```

3. **Output:**

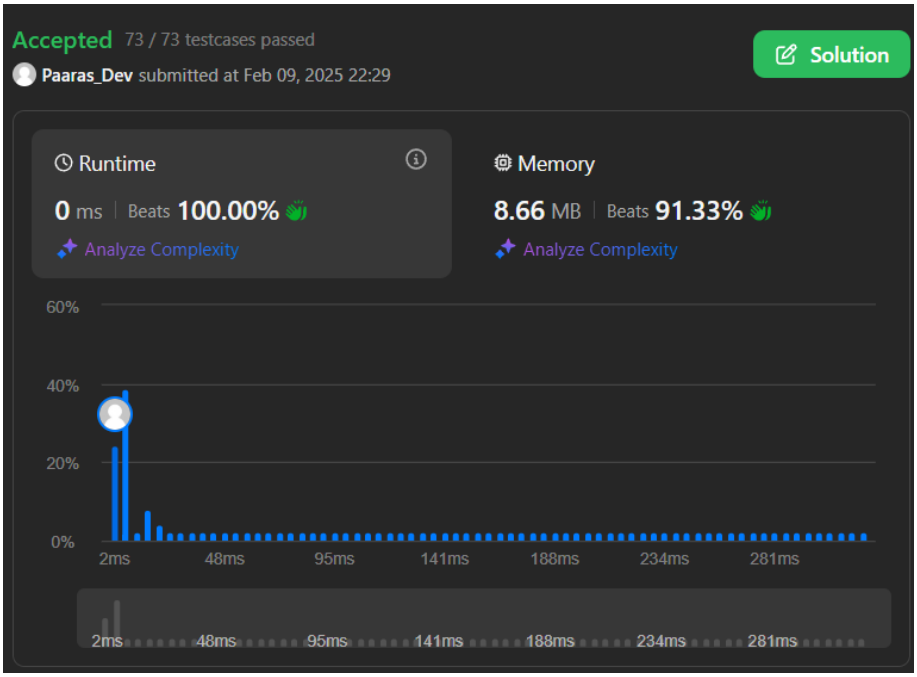


1. **Problem 18: Search in rotated Sorted Array**

2. **Code:**

```
class Solution {
    public int search(int[] nums, int target) {
        int left = 0, right = nums.length - 1;
        while (left <= right) {
            int mid = (left + right) / 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;   }}
```

3. **Output:**

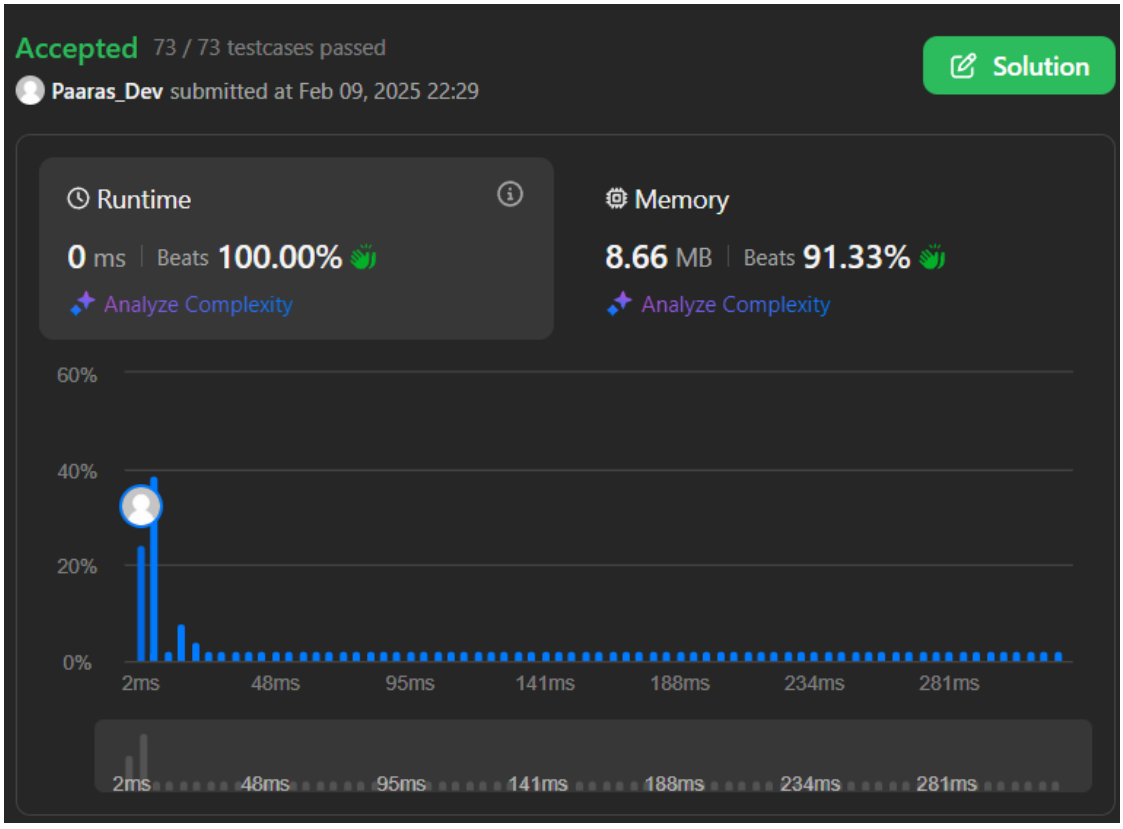


1. **Problem 19: Wiggle Sort II**

2. **Code:**

```
class Solution {
    public void wiggleSort(int[] nums) {
      int n=nums.length-1;
      int[] newarr=Arrays.copyOf(nums,nums.length);
      Arrays.sort(newarr);
       for(int i=1;i<nums.length;i+=2)
          nums[i]=newarr[n--];
       for(int i=0;i<nums.length;i+=2)
          nums[i]=newarr[n--];
    }
}
```

3. **Output:**

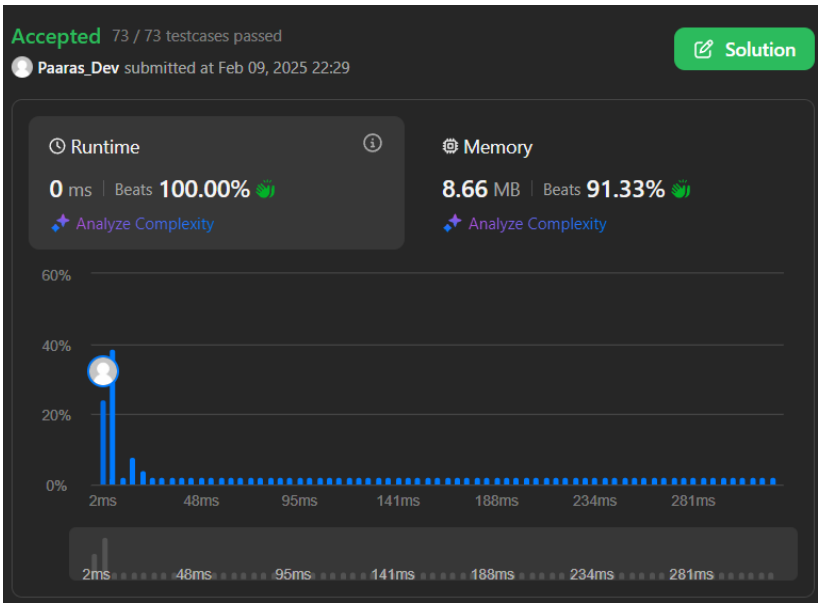


1. **Problem 20: Kth Smallest Element in a Sorted Matrix**

2. **Code:**

```
import java.util.*;

class Solution {
    public int kthSmallest(int[][] matrix, int k) {
        int n = matrix.length;
        int m = matrix[0].length;
        List<Integer> p = new ArrayList<>();

        for (int i = 0; i < n * m; i++) {
            p.add(matrix[i / m][i % m]);
        }

        Collections.sort(p);

        return p.get(k - 1);
    }
}
```

3. **Output:**



1. **Problem 21: Median of Two Sorted Arrays.**

2. **Code:**

```
import java.util.Arrays;
class Solution {
    public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        int n = nums1.length;
        int m = nums2.length;
        int[] merged = new int[n + m];
        int k = 0;
        for (int i = 0; i < n; i++) {  merged[k++] = nums1[i]; }
        for (int i = 0; i < m; i++) { merged[k++] = nums2[i];  }
        Arrays.sort(merged);
        int total = merged.length;
        if (total % 2 == 1) {
            return (double) merged[total / 2];
        } else {
            int middle1 = merged[total / 2 - 1];
            int middle2 = merged[total / 2];
            return ((double) middle1 + (double) middle2) / 2.0;  } }}
```

3. **Output:**