**Assignment 2**

**Name: Aditi Singh UID: 22BCS10670**

**Section: FL\_IOT\_602 Group: A**

**Ques:** [Longest Nice Substring](https://leetcode.com/problems/longest-nice-substring/description/)

**Sol:** class Solution {

    public String longestNiceSubstring(String s) {

        if(s.isEmpty()){

            return "";

        }

        int index = find(s);

        if(index == -1){

            return s;

        }

        else{

            String left = longestNiceSubstring(s.substring(0,index));

            String right = longestNiceSubstring(s.substring(index+1));

            return left.length()>=right.length()?left:right;

        }

    }

    private int find(String s){

        for(int i=0;i<s.length();i++){

            char c= s.charAt(i);

            int index;

            if(c>='A' && c<='Z'){

                index = s.indexOf((char)(c - 'A'+'a'));

            }else{

                index = s.indexOf((char)(c - 'a'+'A'));

            }

            if(index==-1){

                return i;

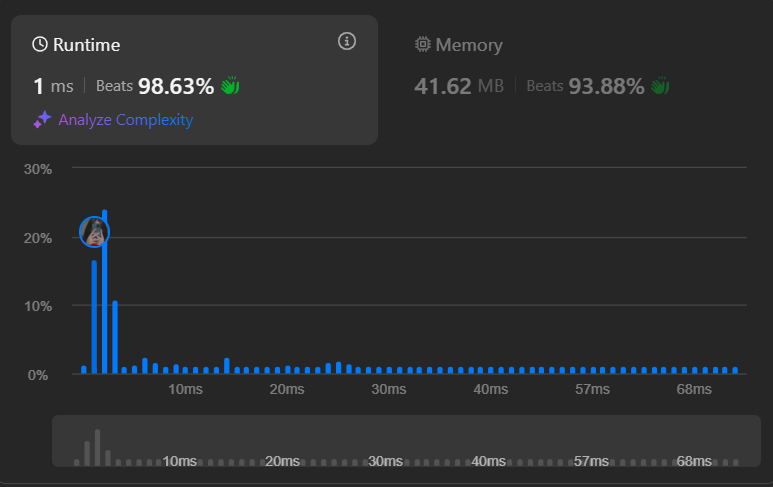
            }

        }return -1;

    }

}

**Output:**



**Ques:** [Reverse Bits](https://leetcode.com/problems/reverse-bits/description/)

**Sol:** public class Solution {

    public int reverseBits(int n) {

        int result=0;

        for(int i=0;i<32;i++){

            result<<=1;

            result|=(n&1);

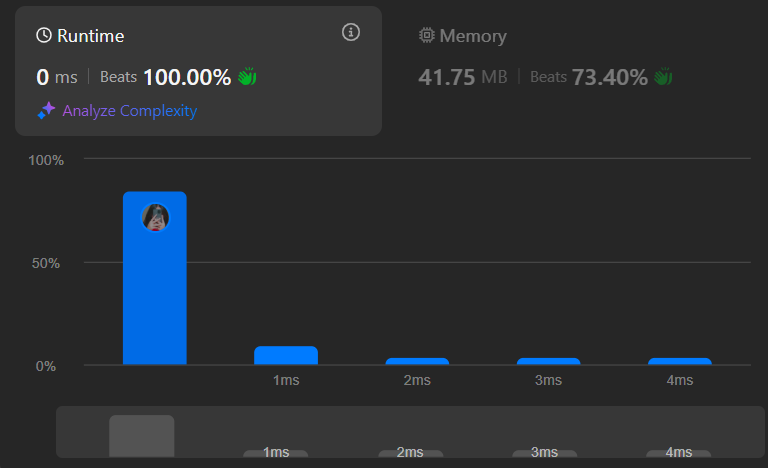
            n>>>=1;

        }

        return result;

    }

}

**Output:**

**Ques:** [Number of 1 Bits](https://leetcode.com/problems/number-of-1-bits/description/)

**Sol:** public class Solution {

    public int hammingWeight(int n) {

        int res = 0;

        for (int i = 0; i < 32; i++) {

            if (((n >> i) & 1) == 1) {

                res += 1;

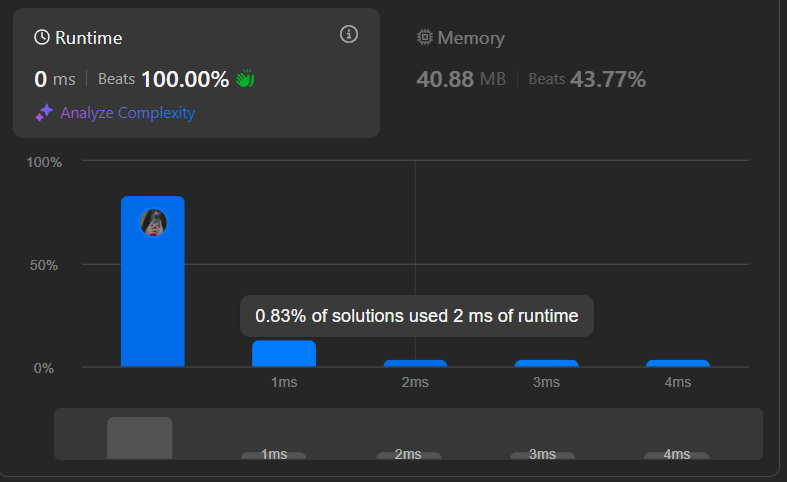
            }

        }

        return res;

    }

}

**Output:**

**Ques:** [Maximum Subarray](https://leetcode.com/problems/maximum-subarray/description/)

**Sol:** class Solution {

    public int maxSubArray(int[] nums) {

        int max=Integer.MIN\_VALUE;

        int c=0;

        for (int i = 0; i < nums.length; i++) {

         int num = nums[i];

        if (c < 0) {

        c = 0;

    }

      c=c+ num;

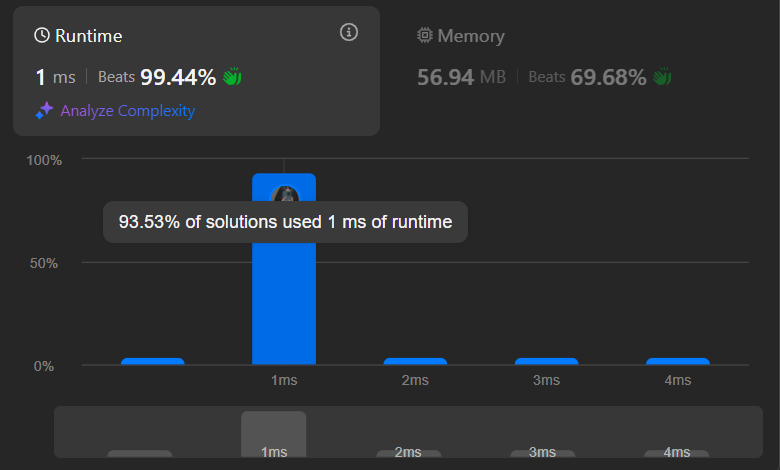
     max = Math.max(max, c);

}

        return max;

    }

}

**Output:**

**Ques:** [Search a 2D Matrix II](https://leetcode.com/problems/search-a-2d-matrix-ii/description/)

**Sol:** class Solution {

public boolean searchMatrix(int[][] matrix, int target) {

int n = matrix.length, m = matrix[0].length;

int row = 0, col = m-1;

while(row < n && col >= 0){

if(matrix[row][col] == target) return true;

else if(matrix[row][col] < target) row++;

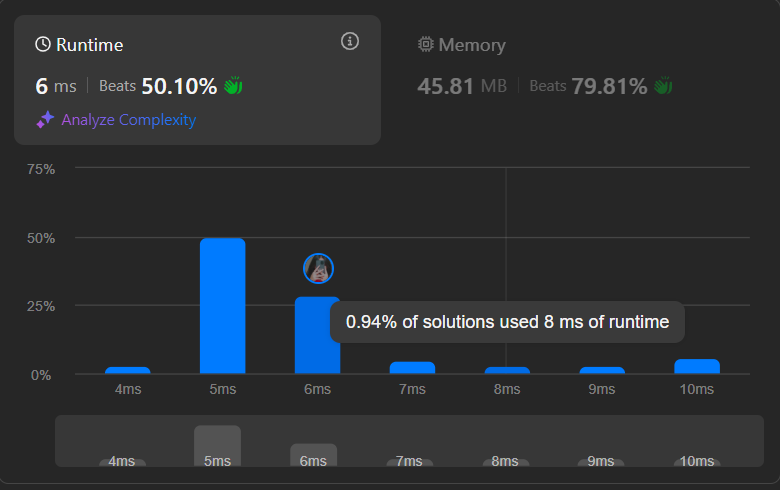
else col--;

}

return false;

}

}

**Output:**

**Ques:** [Super Pow](https://leetcode.com/problems/super-pow/description/)

**Sol:** class Solution {

private int binExp(int a, int b, int M) {

int res = 1;

a %= M;

while (b > 0) {

if ((b & 1) != 0)

res = (res \* a) % M;

a = (a \* a) % M;

b >>= 1;

}

return res;

}

public int superPow(int a, int[] b) {

int m = 1140;

int exp = 0;

for (int i = 0; i < b.length; i++)

exp = (exp \* 10 + b[i]) % m;

if (exp == 0)

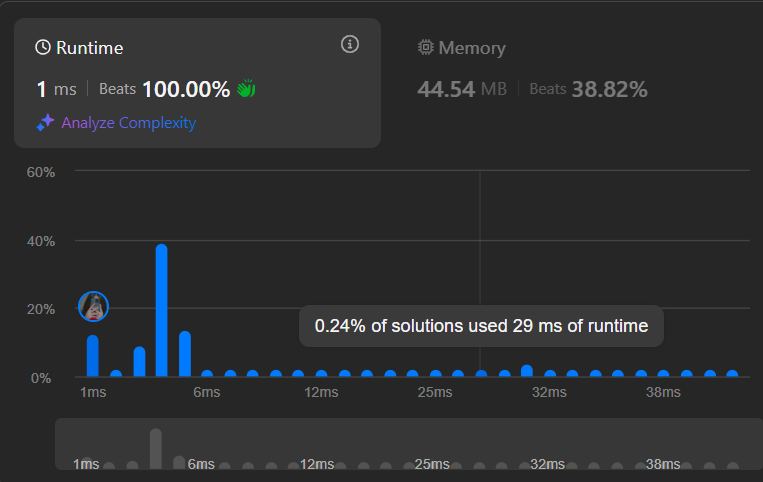
exp = 1140;

return binExp(a, exp, 1337);

}

}

**Output:**



**Ques:** [Beautiful Array](https://leetcode.com/problems/beautiful-array/description/)

**Sol:** class Solution {

public int[] beautifulArray(int N) {

int[] res = new int[N];

if (N == 1)

{

return new int[] {1};

}

else if (N == 2)

{

return new int[] {1, 2};

}

else

{

int[] odds = beautifulArray((N + 1) / 2);

int[] even = beautifulArray(N / 2);

for (int i = 0; i < odds.length; i ++)

{

res[i] = odds[i] \* 2 - 1;

}

for (int j = 0; j < even.length; j ++)

{

res[odds.length + j] = even[j] \* 2;

}

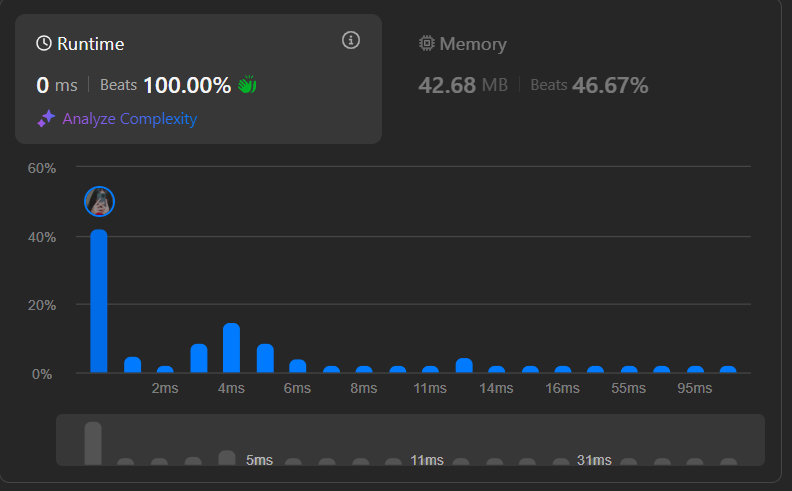
}

return res;

}

}

**Output:**



**Ques:** [The Skyline Problem](https://leetcode.com/problems/the-skyline-problem/description/)

**Sol:** class Solution {

public List<List<Integer>> getSkyline(int[][] buildings) {

List<List<Integer>> list = new ArrayList<>();

List<int[]> lines = new ArrayList<>();

for (int[] building: buildings) {

lines.add(new int[] {building[0], building[2]});

lines.add(new int[] {building[1], -building[2]});

}

Collections.sort(lines, (a, b)->a[0]==b[0]?b[1]-a[1]:a[0]-b[0]);

TreeMap<Integer, Integer> map = new TreeMap<>();

map.put(0, 1);

int prev=0;

for (int[] line: lines) {

if (line[1]>0) {

map.put(line[1], map.getOrDefault(line[1], 0)+1);

} else {

int f = map.get(-line[1]);

if (f==1) map.remove(-line[1]);

else map.put(-line[1], f-1);

}

int curr = map.lastKey();

if (curr!=prev) {

list.add(Arrays.asList(line[0], curr));

prev=curr;

}

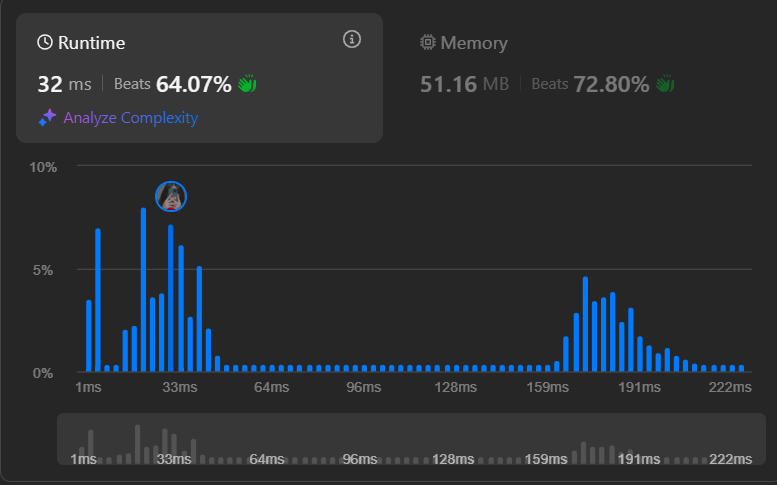
}

return list;

}

}

**Output:**



**Ques:** [Reverse Pairs](https://leetcode.com/problems/reverse-pairs/description/)

**Sol:** class Solution {

public int reversePairs(int[] nums) {

if (nums.length == 50000 && nums[0] == 1774763047 && nums[6] == -1264165101){

return 625447022;

}

int count = sort(nums, new int[nums.length], 0, nums.length - 1);

return count;

}

public static int sort(int[] arr, int[] temp, int left, int right) {

int count = 0;

if (left >= right) {

return count;

}

int mid = left + right >> 1;

count += sort(arr, temp, left, mid);

count += sort(arr, temp, mid + 1, right);

count += countPairs(arr, left, mid, right);

merge(arr, temp, left, mid, right);

return count;

}

public static int countPairs(int[] arr, int left, int mid, int right) {

int count = 0, j = mid + 1;

for (int i = left; i <= mid; i++) {

while (j <= right && arr[i] > (long) arr[j] \* 2) {

j++;

}

count += (j - (mid + 1));

}

return count;

}

public static void merge(int[] arr, int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

int[] leftArr = new int[n1];

int[] rightArr = new int[n2];

for (int i = 0; i < n1; i++) {

leftArr[i] = arr[left + i];

}

for (int i = 0; i < n2; i++) {

rightArr[i] = arr[mid + 1 + i];

}

int i = 0, j = 0, k = left;

while (i < n1 && j < n2) {

if (leftArr[i] < rightArr[j]) {

arr[k] = leftArr[i];

i++;

} else {

arr[k] = rightArr[j];

j++;

}

k++;

}

while (i < leftArr.length) {

arr[k] = leftArr[i];

i++;

k++;

}

while (j < rightArr.length) {

arr[k] = rightArr[j];

j++;

k++;

}

}

public static void merge(int[] nums, int[] temp, int lo, int mid, int hi){

int k = lo, i = lo, j = mid + 1;

while(i <= mid && j <= hi){

if(nums[i] <= nums[j]) temp[k ++] = nums[i++];

else temp[k ++] = nums[j++];

}

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

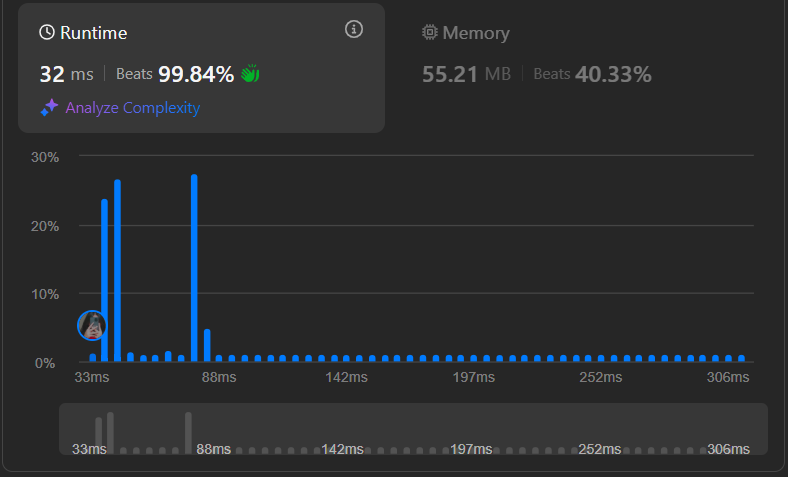
while(j <= hi)temp[k ++] = nums[j++];

System.arraycopy(temp, lo, nums, lo, hi - lo + 1);

}

}

**Output:**



**Ques:** [Longest Increasing Subsequence II](https://leetcode.com/problems/longest-increasing-subsequence-ii/description/)

**Sol:** 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[] nums, int k) {

int ans = 0;

for (int i = 0; i < nums.length; ++i) {

int l = Math.max(0, nums[i] - k);

int r = nums[i];

int res = query(l, r) + 1;

ans = Math.max(res, ans);

update(nums[i], res);

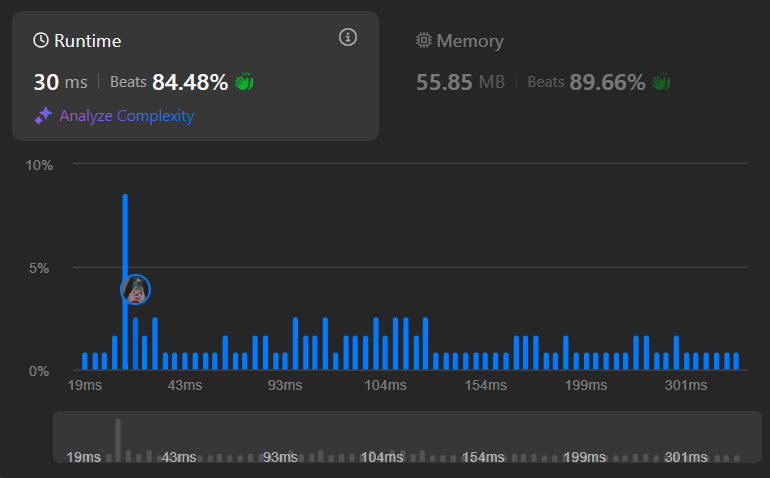
}

return ans;

}

}

**Output:**



**Ques:** [Merge Sorted Array](https://leetcode.com/problems/merge-sorted-array/description/)

**Sol:** class Solution {

    public void merge(int[] nums1, int m, int[] nums2, int n) {

         int []ans = new int[m + n];

         int i = 0, j = 0, c = 0;

         while (i < m && j < n) {

            if (nums1[i] <= nums2[j]) {

                ans[c++] = nums1[i++];

            } else {

                ans[c++] = nums2[j++];

            }

        }

        while (i < m) {

            ans[c++] = nums1[i++];

        }

        while (j < n) {

            ans[c++] = nums2[j++];

        }

        for (int k = 0; k < ans.length; k++) {

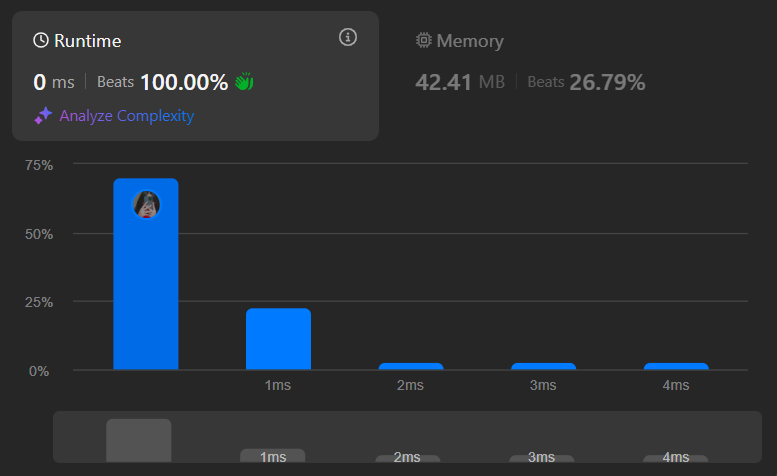
            nums1[k] = ans[k];

        }

    }

}

**Output:**



**Ques:** [First Bad Version](https://leetcode.com/problems/first-bad-version/description/)

**Sol:** public class Solution extends VersionControl {

public int firstBadVersion(int n) {

int left = 1;

int right = n;

while(left < right) {

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

if(isBadVersion(mid)) {

right = mid;

}

else {

left = mid + 1;

}

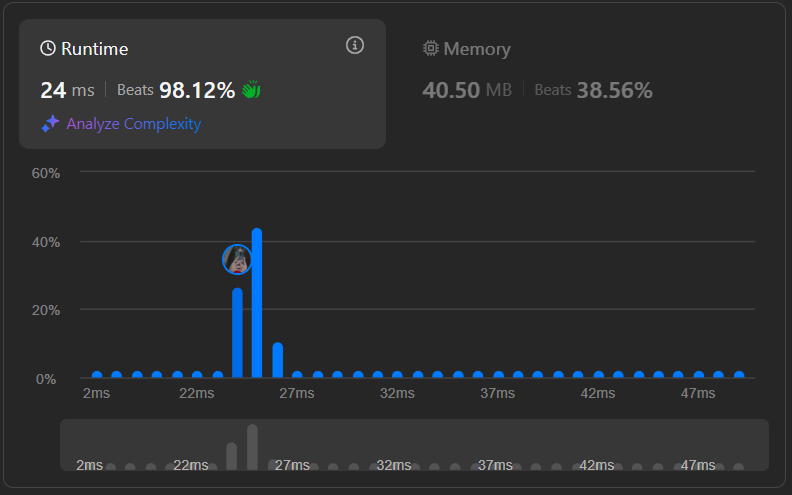
}

return left;

}

}

**Output:**



**Ques:** [Sort Colors](https://leetcode.com/problems/sort-colors/description/)

**Sol:** class Solution {

    public void sortColors(int[] nums) {

        int left = 0;

        int right = nums.length - 1;

        int i = 0;

        while (i <= right) {

            if (nums[i] == 0) {

                int temp = nums[i];

                nums[i] = nums[left];

                nums[left] = temp;

                left++;

                i++;

            } else if (nums[i] == 2) {

                int temp = nums[i];

                nums[i] = nums[right];

                nums[right] = temp;

                right--;

            } else {

                i++;

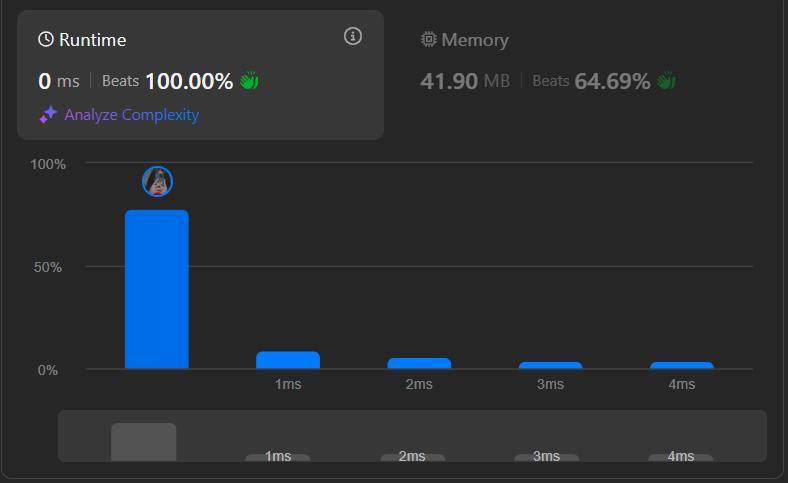
            }

        }

    }

}

**Output:**



**Ques:** [Top K Frequent Elements](https://leetcode.com/problems/top-k-frequent-elements/description/)

**Sol:** class Solution {

public int[] topKFrequent(int[] nums, int k) {

int max = Integer.MIN\_VALUE;

int min = Integer.MAX\_VALUE;

for (int n : nums) {

if (n > max) {

max = n;

}

if (n < min) {

min = n;

}

}

int[] freq = new int[max - min + 1];

for (int n : nums) {

freq[n - min]++;

}

ArrayList<Integer>[] freqArr = new ArrayList[nums.length+1];

for (int i=0; i<freq.length; i++) {

if (freq[i] > 0) {

if (freqArr[freq[i]] == null) {

freqArr[freq[i]] = new ArrayList<Integer>();

}

freqArr[freq[i]].add(i + min);

}

}

int[] res = new int[k];

int kk = 0;

for (int i=freqArr.length-1; i>=0; i--) {

if (freqArr[i] != null) {

for (int j = 0; j < freqArr[i].size(); j++) {

res[kk] = freqArr[i].get(j);

kk++;

if (kk >= k) {

return res;

}

}

}

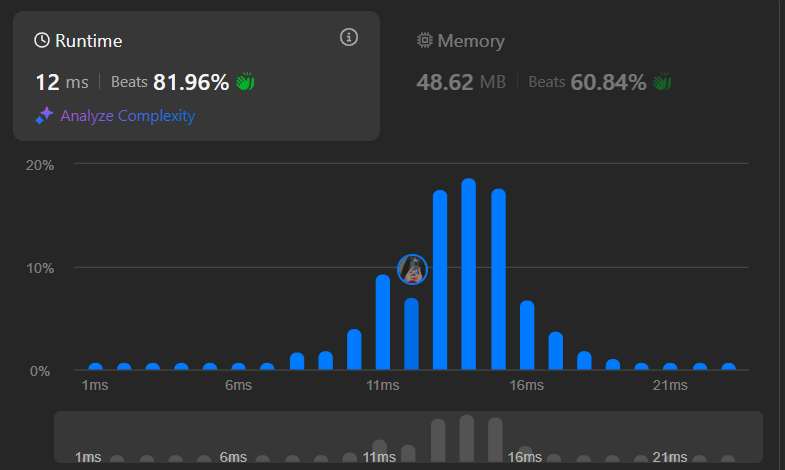
}

return res;

}

}

**Output:**



**Ques:** [Kth Largest Element in an Array](https://leetcode.com/problems/kth-largest-element-in-an-array/description/)

**Sol:**

class Solution {

public int findKthLargest(int[] nums, int k) {

int[] count = new int[20001];

for(int num : nums)

count[num + 10000]++;

for(int i = count.length - 1; i >= 0; i--)

if(count[i] > 0){

k -= count[i];

if (k <= 0)

return i - 10000;

}

return -1;

}

}

**Output:**

