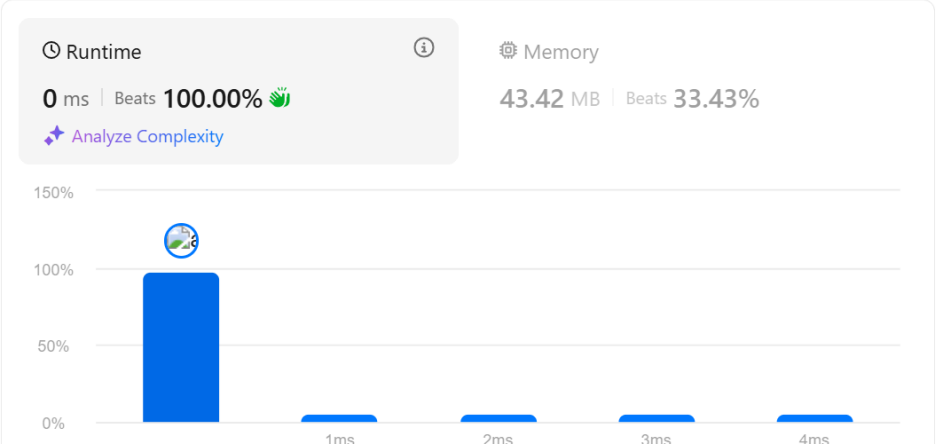
Name: Vanshika **UID: 22BCS15478**

Assignment-6

1. **Convert Sorted Array to Binary Search Tree**

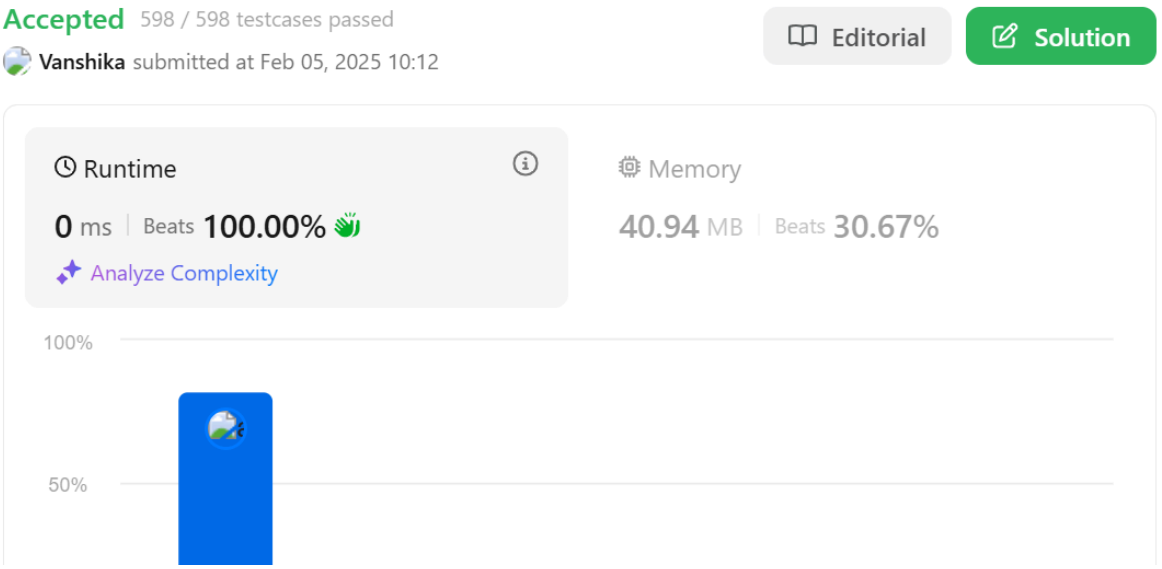
```
class Solution {
    public TreeNode sortedArrayToBST(int[] nums) {
        return helper(nums, 0, nums.length - 1);
    }

    private TreeNode helper(int[] nums, int left, int right) {
        if (left > right) return null;
        int mid = (left + right) / 2;
        TreeNode root = new TreeNode(nums[mid]);
        root.left = helper(nums, left, mid - 1);
        root.right = helper(nums, mid + 1, right);
        return root;
    }
}
```



2. **Number of 1 Bits**

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

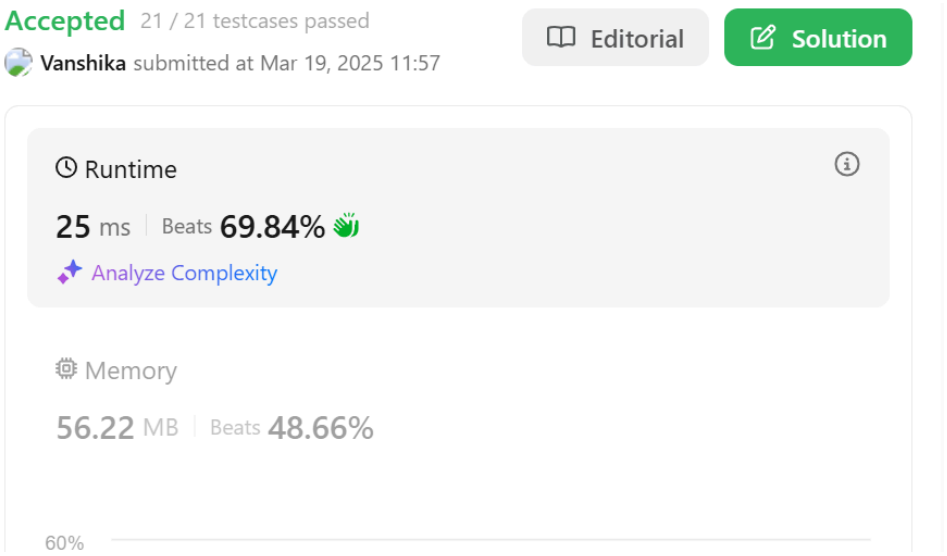
```
    }
}
```



3. Sort an Array

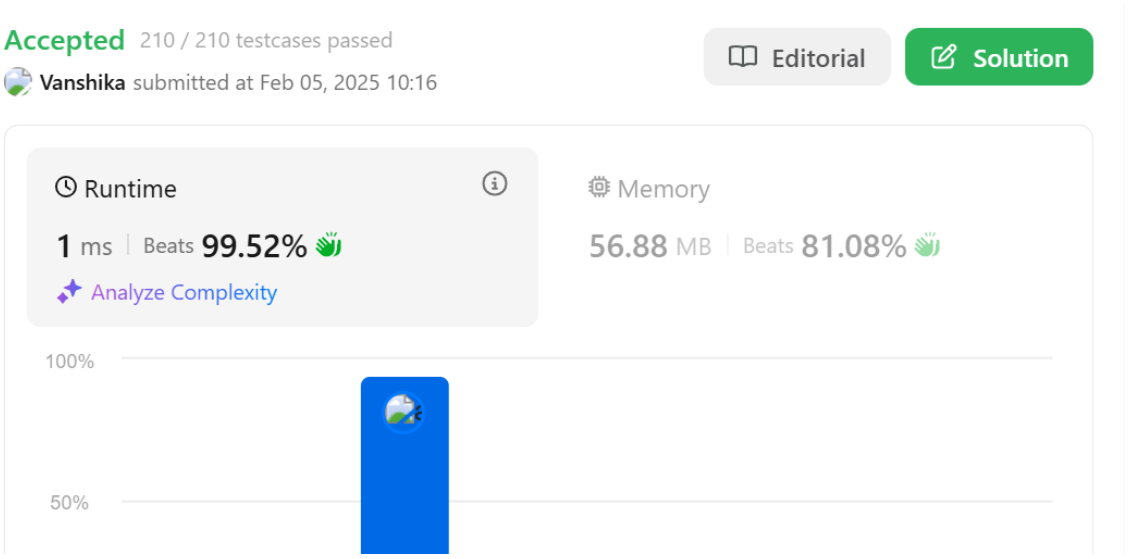
```
class Solution {
    public int[] sortArray(int[] nums) {
        mergeSort(nums, 0, nums.length - 1);
        return nums;
    }

    private void mergeSort(int[] array, int low, int high) {
        if (low >= high) {
            return;
        }
        int mid = low + (high - low) / 2;
        mergeSort(array, low, mid);
        mergeSort(array, mid + 1, high);
        merge(array, low, mid, high);
    }

    private void merge(int[] array, int low, int mid, int high) {
        int n1 = mid - low + 1;
        int n2 = high - mid;
        int[] leftPart = new int[n1];
```

```
        int[] rightPart = new int[n2];

        System.arraycopy(array, low, leftPart, 0, n1);
        System.arraycopy(array, mid + 1, rightPart, 0, n2);

        int p1 = 0, p2 = 0, writeInd = low;
        while (p1 < n1 && p2 < n2) {
            if (leftPart[p1] <= rightPart[p2]) {
                array[writeInd++] = leftPart[p1++];
            } else {
                array[writeInd++] = rightPart[p2++];
            }
        }

        while (p1 < n1) {
            array[writeInd++] = leftPart[p1++];
        }

        while (p2 < n2) {
            array[writeInd++] = rightPart[p2++];
        }
    }
}
```



4. **Maximum Subarray**

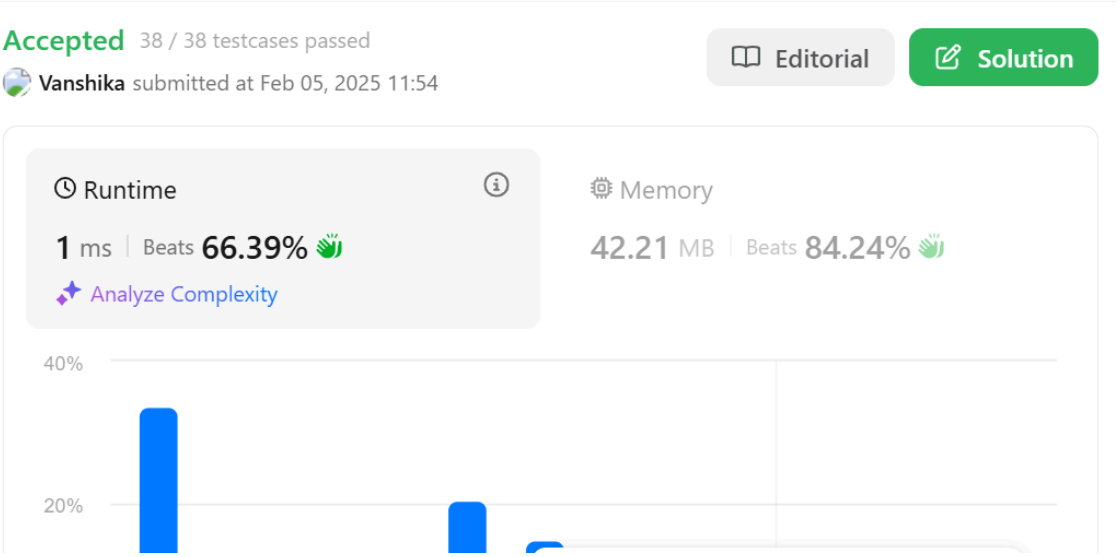
```
class Solution {
    public int maxSubArray(int[] nums) {
        int maxSum = nums[0], currentSum = nums[0];
        for (int i = 1; i < nums.length; i++) {
            currentSum = Math.max(nums[i], currentSum + nums[i]);
            maxSum = Math.max(maxSum, currentSum);
        }
        return maxSum;
    }
}
```



5. **Beautiful Array**

```
class Solution {

    public static void test(int start , int increment , ArrayList<Integer> ans , int n){
        if(start + increment > n){
            ans.add(start);
            return;
        }

        test(start , 2 * increment , ans , n);
        test(start + increment , 2 * increment , ans , n);
    }
```

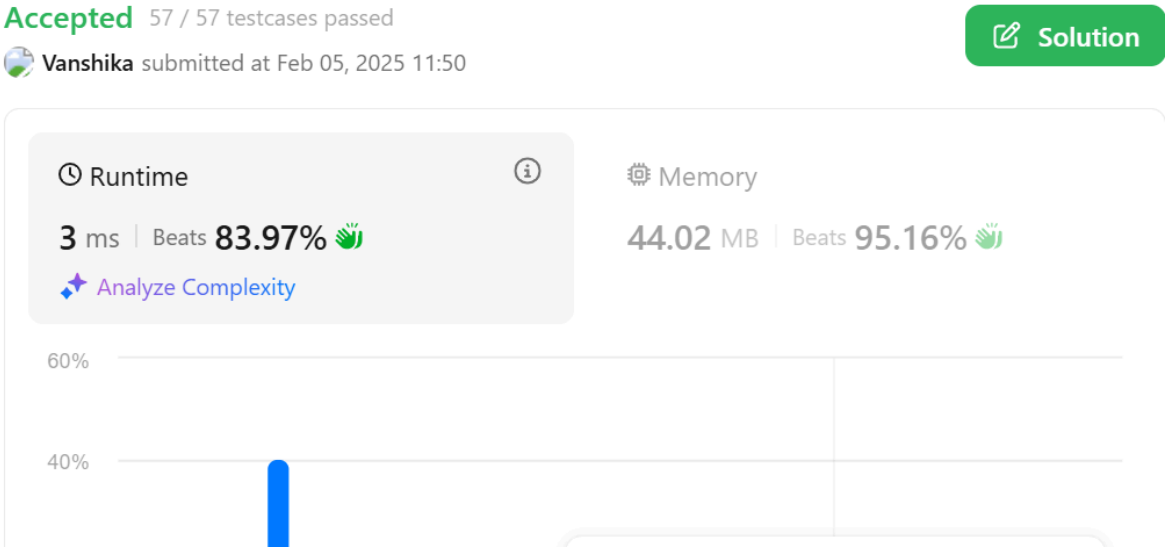
```
    public int[] beautifulArray(int n) {
        ArrayList<Integer> ans = new ArrayList<>();

        test(1,1,ans, n);

        int arr[] = new int[n];
        for(int i=0; i<n; i++){
            arr[i] = ans.get(i);
        }
        return arr;
    }
}
```



6. **Super Pow**

```
class Solution {
    private static final int MOD = 1337;

    public int superPow(int a, int[] b) {
        a %= MOD;
        return helper(a, b, b.length);
    }

    private int helper(int a, int[] b, int length) {
        if (length == 0) return 1;

        int lastDigit = b[length - 1];
```

```
        int remainingPow = helper(a, b, length - 1);

        return powerMod(remainingPow, 10) * powerMod(a, lastDigit) % MOD;
    }

    private int powerMod(int base, int exp) {
        int result = 1;
        while (exp > 0) {
            if (exp % 2 == 1) {
                result = result * base % MOD;
            }
            base = base * base % MOD;
            exp /= 2;
        }
        return result;
    }
}
```



7. **The Skyline Problem**

```
import java.util.AbstractList;
class Solution {
    public List<List<Integer>> getSkyline(int[][] buildings) {
        return new AbstractList<List<Integer>>() {

            private List<List<Integer>> resList;
```

```
private void onload() {
    resList = new ArrayList<>();

    List<int[]> heights = new ArrayList<>();
    for (int[] building : buildings) {
        heights.add(new int[] { building[0], -building[2] });
        heights.add(new int[] { building[1], building[2] });
    }

    // sort by X (ascending) if different, and sort by height (ascending) if X is
    // the same
    // left always comes before right
    Collections.sort(heights, (a, b) -> a[0] == b[0] ? a[1] - b[1] : a[0] - b[0]);

    PriorityQueue<Integer> pq = new PriorityQueue<>((a, b) -> b - a);
    pq.offer(0);

    int prev = 0;
    for (int[] height : heights) {
        if (height[1] < 0) { // left edge
            pq.offer(-height[1]);
            // even if there's multiple buildingd endign before current index, it will be
            // fine
            // becuase priority can take duplicate numbers
        } else {// right edge
            pq.remove(height[1]);
        }
        int cur = pq.peek();
        if (prev != cur) {// if height has changed
            resList.add(Arrays.asList(height[0], cur));
            prev = cur;
        }
    }
}

private void init() {
    if (null == resList) {
        onload();
    }
}
```

```
        @Override
        public List<Integer> get(int index) {
            init();
            return resList.get(index);
        }

        @Override
        public int size() {
            init();
            return resList.size();
        }

    };
  }
}
```