**Assignment- 8**

**Student Name: Avin Kumar UID: 22BCS10600 Section/Group: IOT-614 B Semester: 6th**

**Subject Name: Advanced Programming Subject Code:** **22CSP-351**

**Lab-2**

[**1710. Maximum Units on a Truck**](https://leetcode.com/problems/maximum-units-on-a-truck/):

**Code:**

class Solution {

public int maximumUnits(int[][] B, int T) {

Arrays.sort(B, (a,b) -> b[1] - a[1]);

int ans = 0;

for (int[] b : B) {

int count = Math.min(b[0], T);

ans += count \* b[1];

T -= count;

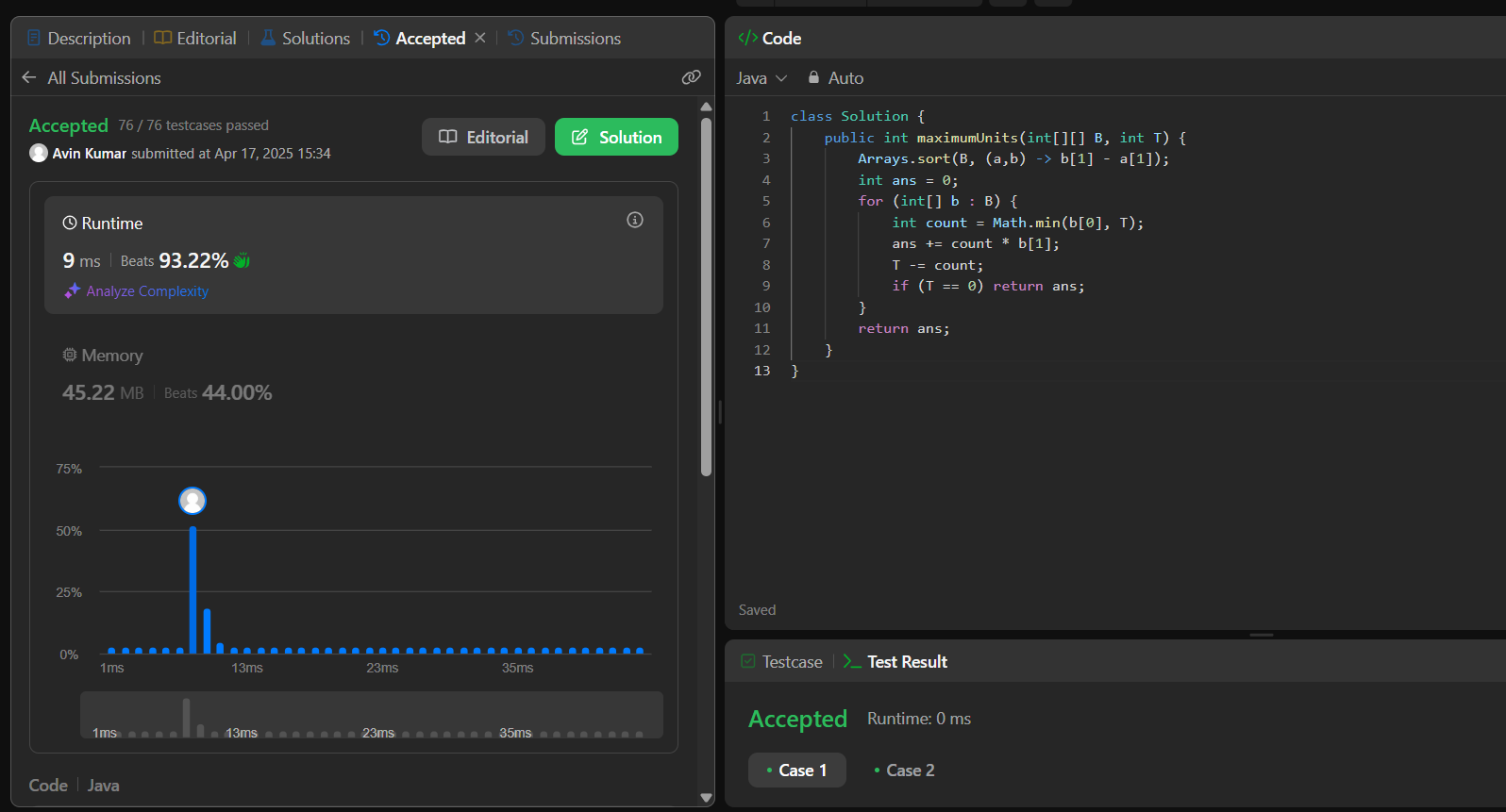
if (T == 0) return ans;

}

return ans;

}

}



[**1827. Minimum Operations to Make the Array Increasing**](https://leetcode.com/problems/minimum-operations-to-make-the-array-increasing/)**:**

**Code:**

class Solution {

public static int minOperations(int[] nums) {

int count = 0;

for (int i = 0; i < nums.length-1; i++) {

if(nums[i] >= nums[i+1]) {

int diff = nums[i] - nums[i + 1] + 1;

count += diff;

nums[i+1] += diff;

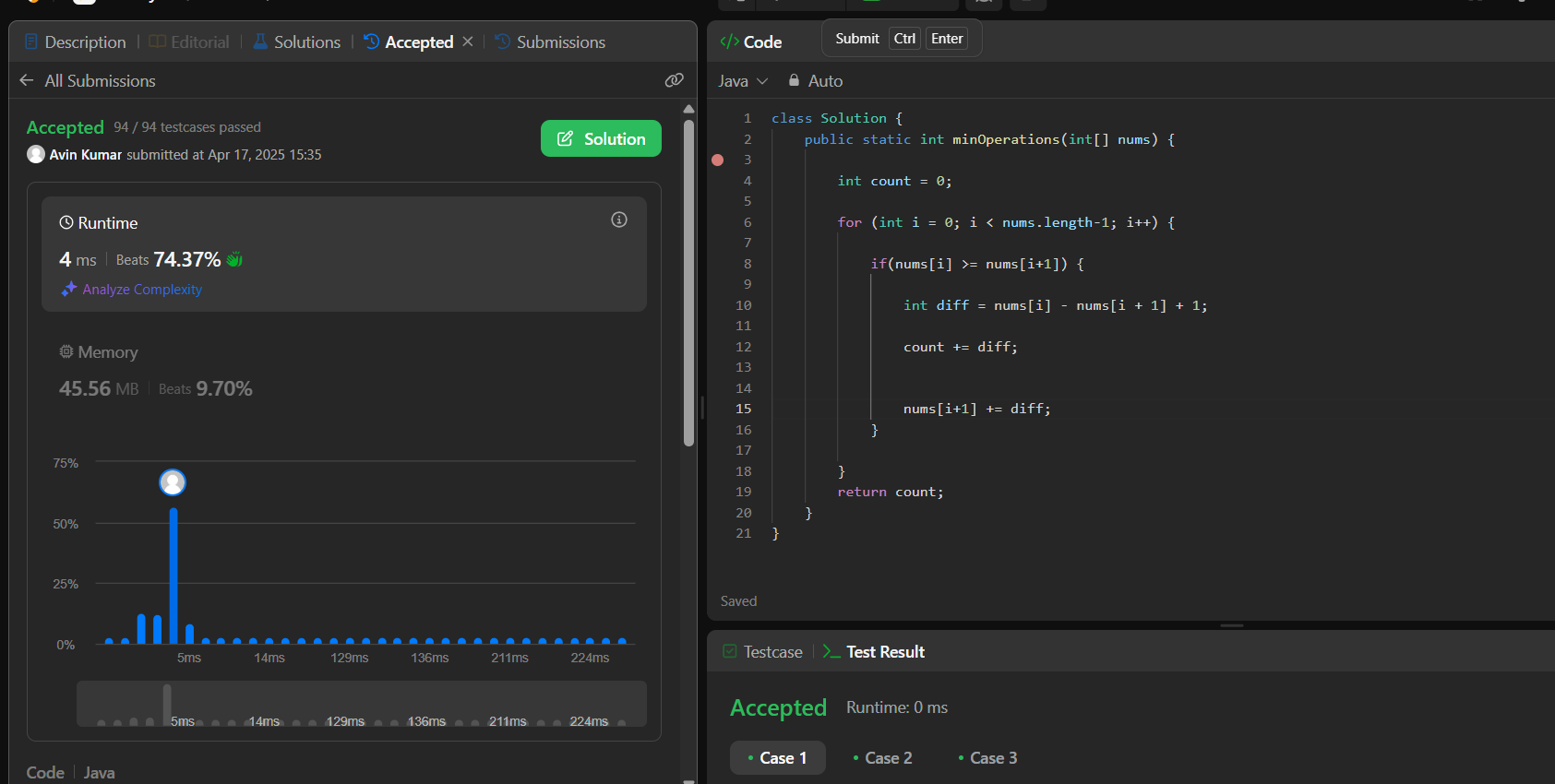
}

}

return count;

}

}



[**1962. Remove Stones to Minimize the Total**](https://leetcode.com/problems/remove-stones-to-minimize-the-total/)**:**

**Code:**

class Solution {

    public int minStoneSum(int[] piles, int k) {

        Queue <Integer> heap = new PriorityQueue (new Comparator <Integer> () {

           public int compare (Integer a, Integer b) {

               if (a < b)

                   return 1;

               else if (a >= b)

                   return -1;

               else

                   return 0;

           }

        });

        for (int val : piles)

            heap.offer (val);

        while (k-- > 0) {

            int stones = heap.poll ();

            stones -= (int) (Math.floor (stones / 2));

            heap.offer (stones);

        }

        int sum = 0;

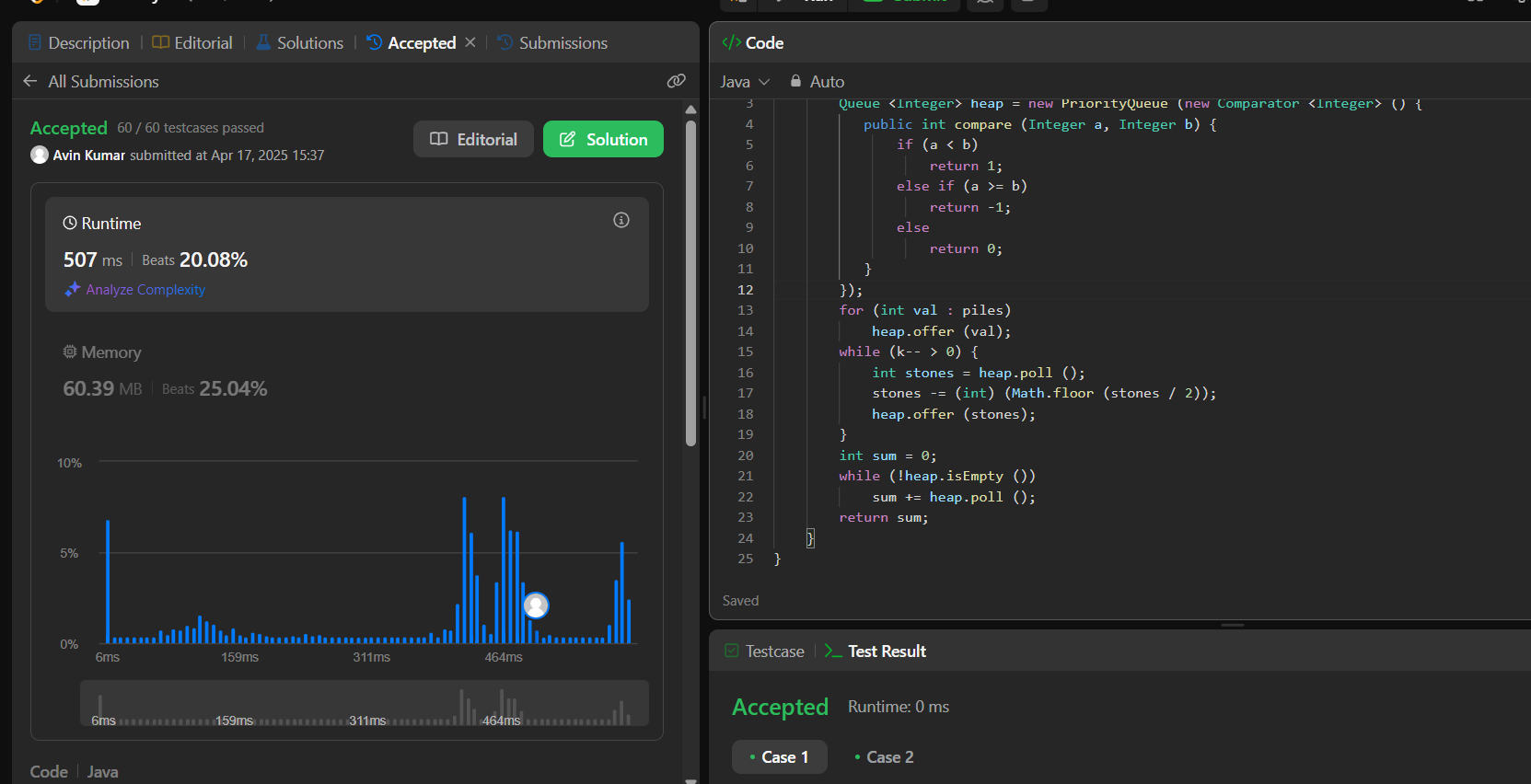
        while (!heap.isEmpty ())

            sum += heap.poll ();

        return sum;

    }

}



[**1717. Maximum Score From Removing Substrings**](https://leetcode.com/problems/maximum-score-from-removing-substrings/)**:**

**Code:**

class Solution {

public int maximumGain(String s, int x, int y) {

int n = s.length(), ans = 0;

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

if (s.charAt(i) != 'a' && s.charAt(i) != 'b') continue;

int j = i;

StringBuilder t = new StringBuilder();

while (j < n && (s.charAt(j) == 'a' || s.charAt(j) == 'b')) {

t.append(s.charAt(j));

j++;

}

ans += Math.max(processAB(t.toString(), x, y), processBA(t.toString(), x, y));

i = j - 1;

}

return ans;

}

public int processAB(String s, int x, int y) {

Stack<Character> stk = new Stack<>();

int ans = 0;

for (char c : s.toCharArray()) {

if (!stk.isEmpty() && stk.peek() == 'a' && c == 'b') {

stk.pop();

ans += x;

} else {

stk.push(c);

}

}

int a = 0, b = 0;

for (char c : stk) {

if (c == 'a') a++;

else b++;

}

return Math.min(a, b) \* y + ans;

}

public int processBA(String s, int x, int y) {

Stack<Character> stk = new Stack<>();

int ans = 0;

for (char c : s.toCharArray()) {

if (!stk.isEmpty() && stk.peek() == 'b' && c == 'a') {

stk.pop();

ans += y;

} else {

stk.push(c);

}

}

int a = 0, b = 0;

for (char c : stk) {

if (c == 'a') a++;

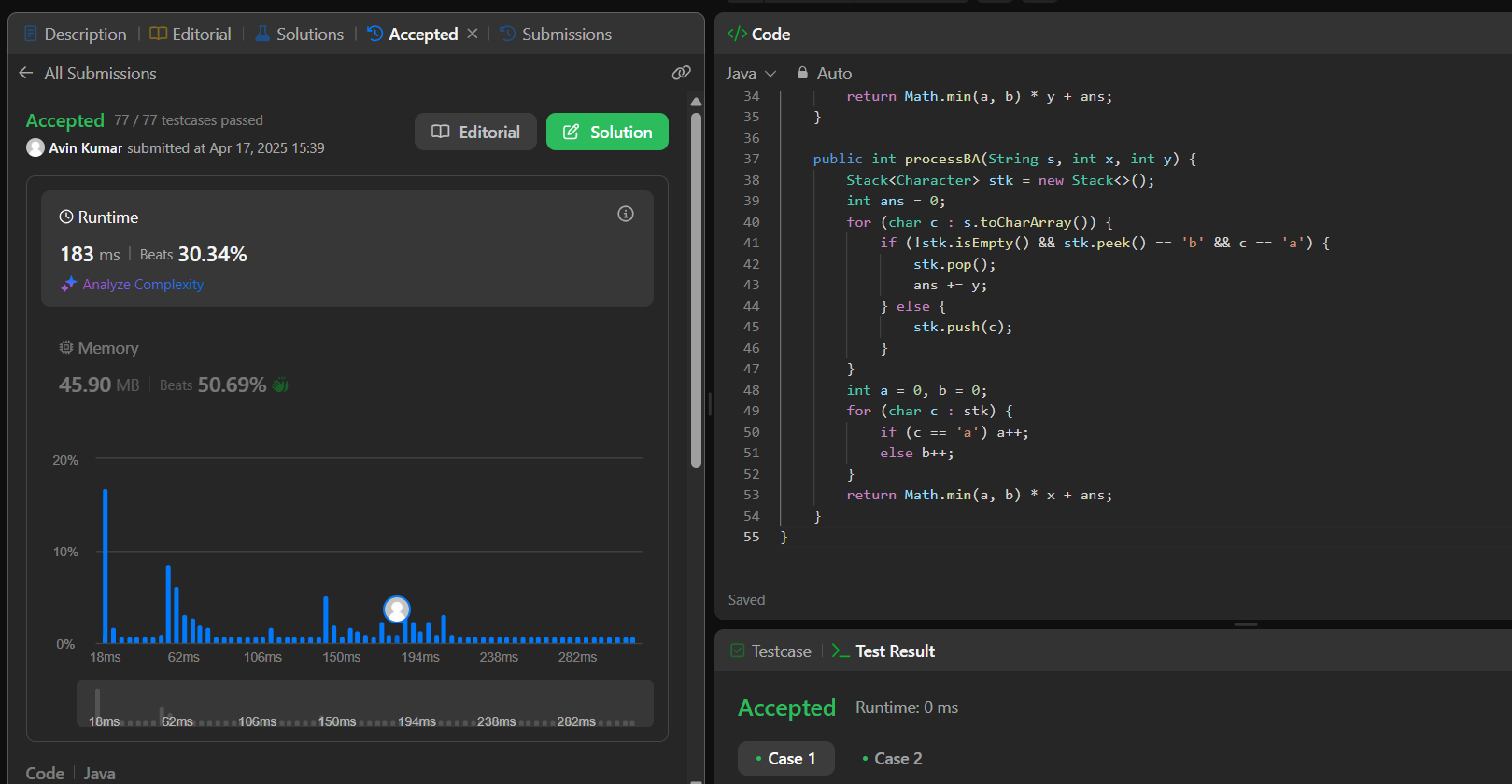
else b++;

}

return Math.min(a, b) \* x + ans;

}

}



[**1713. Minimum Operations to Make a Subsequence**](https://leetcode.com/problems/minimum-operations-to-make-a-subsequence/)**:**

**Code:**

class Solution {

    public int minOperations(int[] target, int[] arr) {

        int n = target.length;

        Map<Integer, Integer> map = new HashMap<>();

        for(int i = 0; i < n; i++) {

            map.put(target[i], i);

        }

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

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

            if(!map.containsKey(arr[i])) {

                continue;

            }

            array.add(map.get(arr[i]));

        }

        int maxLen = 0;

        int[] tails = new int[n + 1];

        for(int i = 0; i <= n; i++) {

            tails[i] = -1;

        }

        for(int num: array) {

            int index = findMinIndex(tails, maxLen, num);

            if(tails[index] == -1) {

                maxLen++;

            }

            tails[index] = num;

        }

        return n - maxLen;

    }

    public int findMinIndex(int[] tails, int n, int val) {

        int low = 0;

        int ans = n;

        int high = n - 1;

        while(low <= high) {

            int mid = (high + low) / 2;

            if(tails[mid] >= val) {

                ans = mid;

                high = mid - 1;

            }

            else {

                low = mid + 1;

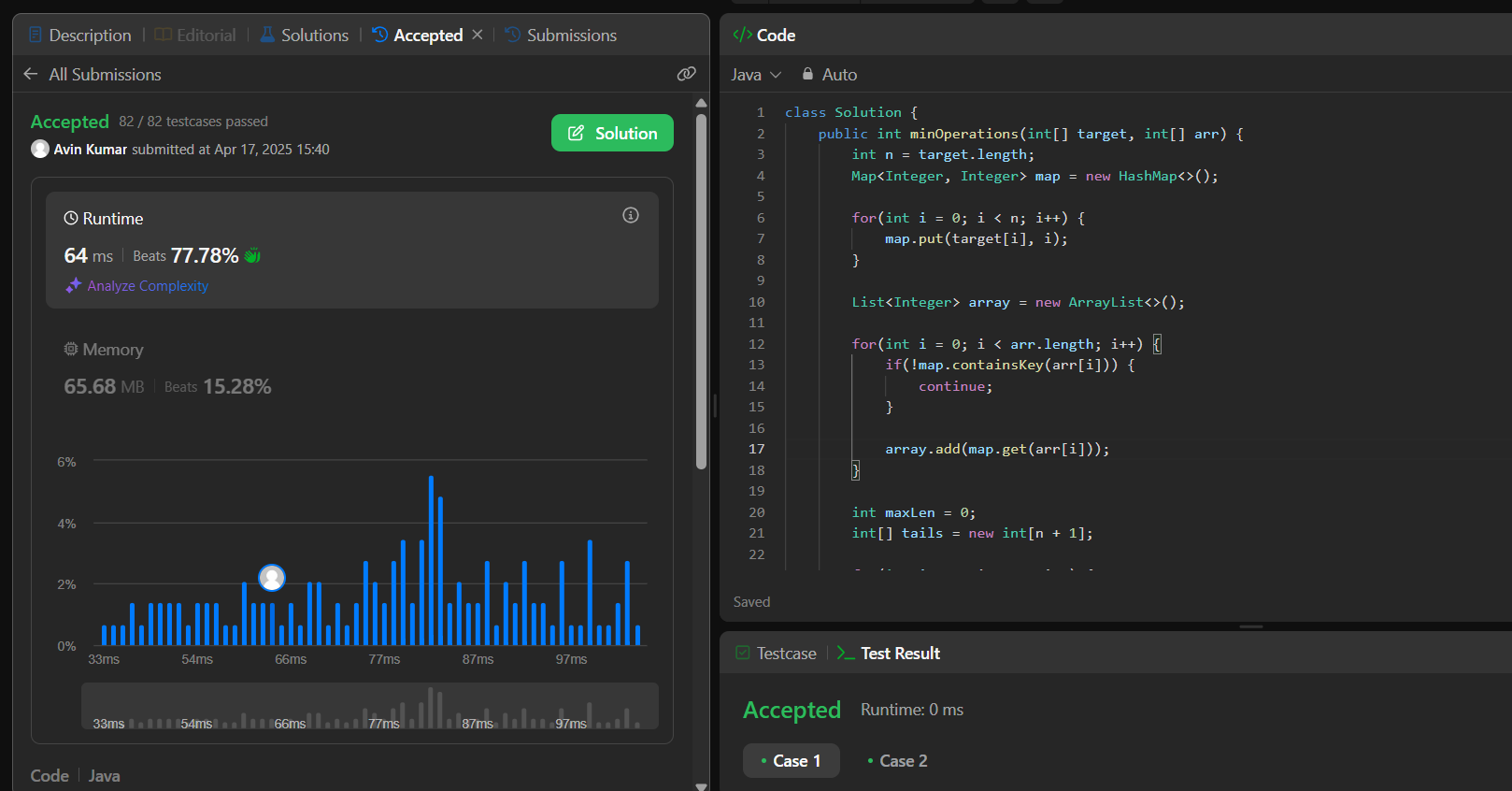
            }

        }

        return ans;

    }

}



[**2071. Maximum Number of Tasks You Can Assign**](https://leetcode.com/problems/maximum-number-of-tasks-you-can-assign/)**:**

**Code:**

class Solution {

    public int maxTaskAssign(int[] tasks, int[] workers, int pills, int strength) {

        Arrays.sort(tasks);

        Arrays.sort(workers);

        int low=0;

        int high=tasks.length;

        int ans=0;

        while(low<=high)

        {

            int mid=(low+high)/2;

            if(isPossible(tasks,workers,pills,strength,mid))

            {

                ans=mid;

                low=mid+1;

            }else{

                high=mid-1;

            }

        }

        return ans;

    }

    public boolean isPossible(int tasks[],int workers[],int pills,int strength,int n)

    {

        if(Math.min(workers.length,tasks.length)<n)

            return false;

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

        for(int i=0;i<n;i++)

        {

            tm.put(tasks[i],tm.getOrDefault(tasks[i],0)+1);

        }

        for(int j=workers.length-n;j<workers.length;j++)

        {

            int smallest=tm.firstKey();

            if(workers[j]<smallest)

            {

                if(pills==0)

                    return false;

                pills--;

                if(tm.floorKey(workers[j]+strength)!=null)

                    smallest=tm.floorKey(workers[j]+strength);

                else

                    return false;

            }

            if(tm.get(smallest)==1)

                tm.remove(smallest);

            else

                tm.put(smallest,tm.get(smallest)-1);

        }

        return true;

    }

}

