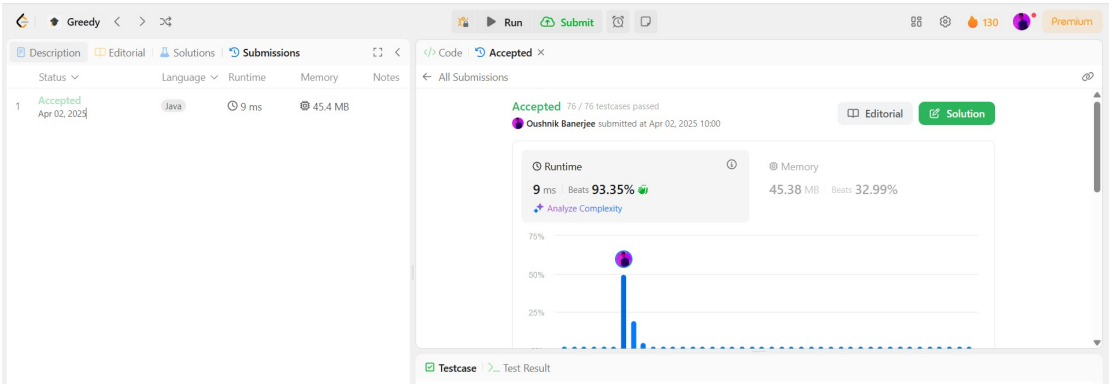
AP Experiment 8

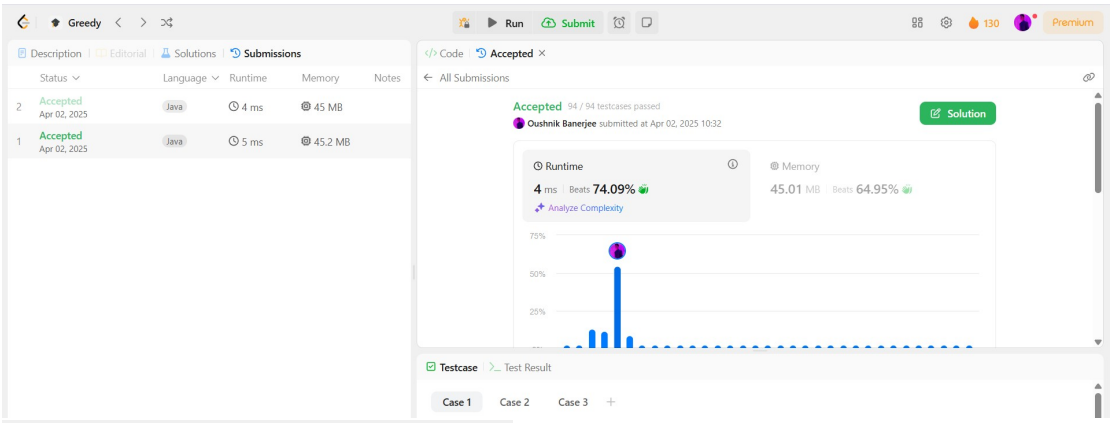
Maximum Units on a Truck

```
class Solution {
    public int maximumUnits(int[][] boxTypes, int truckSize) {
        Arrays.sort(boxTypes, (a,b)-> b[1]-a[1]);
        int res=0;
        for(int a[]: boxTypes){
            int min=Math.min(truckSize, a[0]);
            res+=(a[1]*min);
            truckSize-=a[0];
            if(truckSize<=0) break;
        }
        return res;
    }
}
```



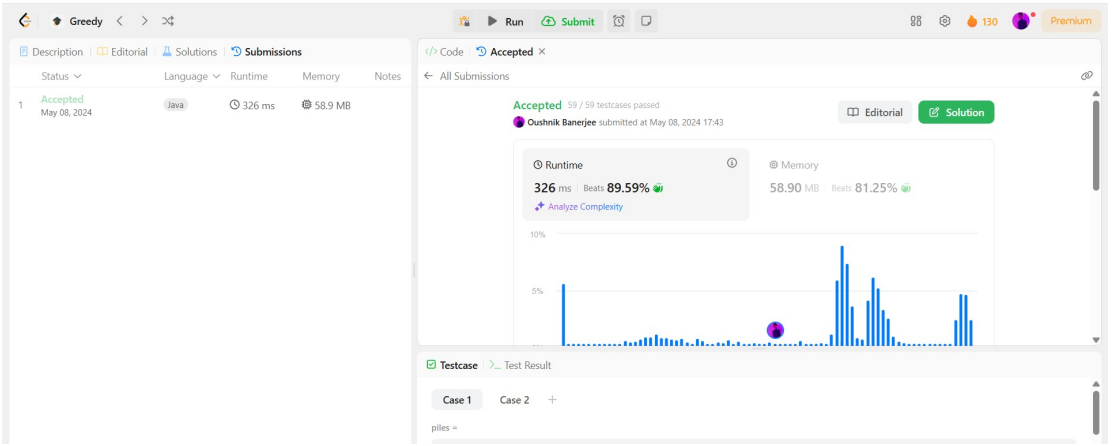
Minimum Operations to Make the Array Increasing

```
class Solution {
    public int minOperations(int[] nums) {
        int res = 0;
        for (int i = 1; i < nums.length; i++) {
            if (nums[i] <= nums[i - 1]) {
                int diff = nums[i - 1] - nums[i] + 1;
                res += diff;
                nums[i] += diff;
            }
        }
        return res;
    }
}
```



Remove Stones to Minimize the Total

```
class Solution {
    public int minStoneSum(int[] piles, int k) {
        int res=0;
        PriorityQueue<Integer> pq= new PriorityQueue<>((a,b) -> b-a);
        for(int num: piles){
            pq.add(num);
            res+= num;
        }
        for(int i=0; i<k; i++){
            int num= pq.poll();
            pq.add(num-num/2);
            res-=num/2;
        }
        return res;
    }
}
```



Maximum Score From Removing Substrings

```
class Solution {
    public int maximumGain(String s, int x, int y) {
        if (x > y) return process(s, 'a', 'b', x, y);
```

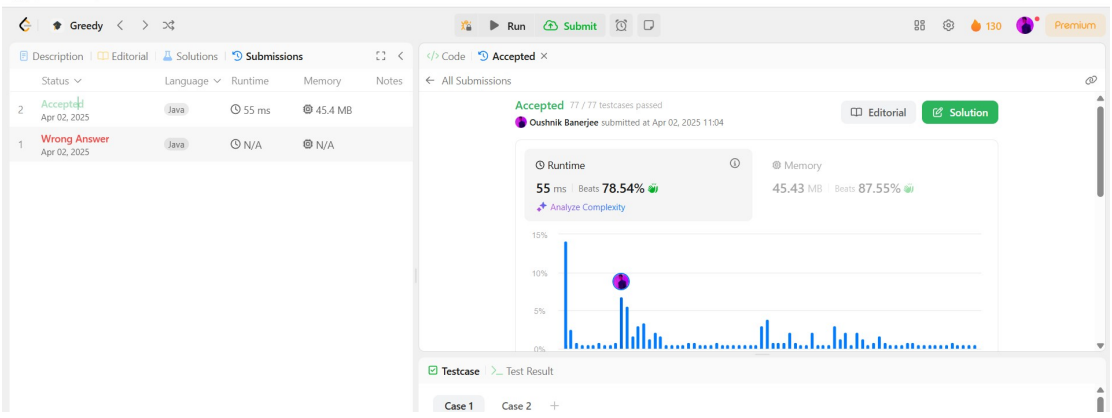
```
        return process(s, 'b', 'a', y, x);
    }

    private int process(String s, char first, char second, int high, int low) {
        int score = 0;
        StringBuilder stack1 = new StringBuilder();

        for (char c : s.toCharArray()) {
            if (stack1.length() > 0 && stack1.charAt(stack1.length() - 1) == first && c ==
second) {
                stack1.setLength(stack1.length() - 1);
                score += high;
            } else {
                stack1.append(c);
            }
        }

        StringBuilder stack2 = new StringBuilder();
        for (char c : stack1.toString().toCharArray()) {
            if (stack2.length() > 0 && stack2.charAt(stack2.length() - 1) == second && c
== first) {
                stack2.setLength(stack2.length() - 1);
                score += low;
            } else {
                stack2.append(c);
            }
        }

        return score;
    }
}
```

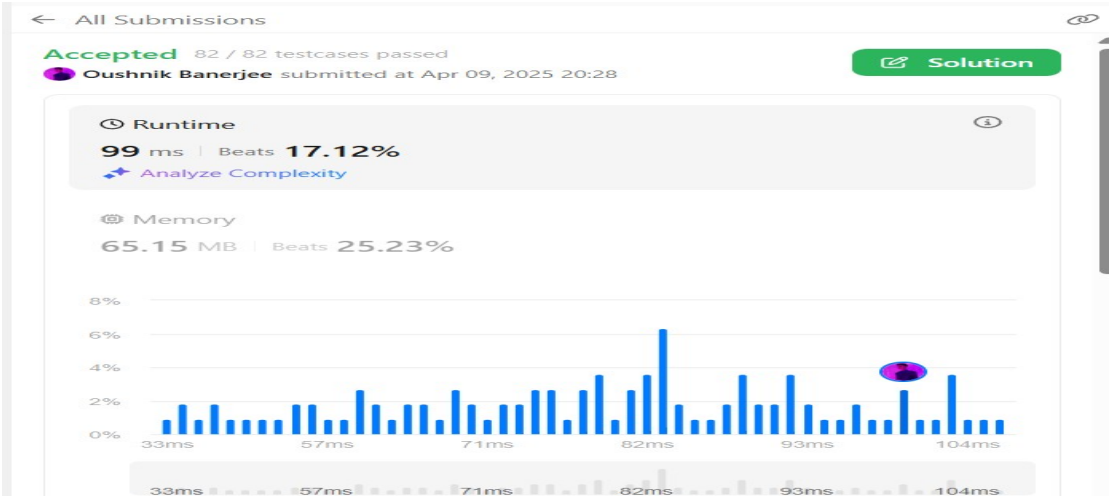


Minimum Operations to Make a Subsequence

```
class Solution {
    public int minOperations(int[] target, int[] A) {
    Map<Integer, Integer> h = new HashMap<>();
```

```
        for (int i = 0; i < target.length; ++i)
            h.put(target[i], i);

        ArrayList<Integer> stack = new ArrayList<>();
        for (int a : A) {
            if (!h.containsKey(a)) continue;
            if (stack.size() == 0 || h.get(a) > stack.get(stack.size() - 1)) {
                stack.add(h.get(a));
                continue;
            }
            int left = 0, right = stack.size() - 1, mid;
            while (left < right) {
                mid = (left + right) / 2;
                if (stack.get(mid) < h.get(a))
                    left = mid + 1;
                else
                    right = mid;
            }
            stack.set(left, h.get(a));
        }
        return target.length - stack.size();
    }
}
```



Maximum Number of Tasks You Can Assign

```
class Solution {
    public int maxTaskAssign(int[] tasks, int[] workers, int pills, int strength) {
        Arrays.sort(tasks);
        TreeMap<Integer, Integer> map = new TreeMap<>();
        for (int i : workers)
            map.put(i, map.getOrDefault(i, 0) + 1);
        int res = 0, left = 0, right = Math.min(tasks.length, workers.length) - 1;
        while (left <= right) {
```

```
            int mid = (left + right) / 2;
            if (validate(tasks, (TreeMap<Integer, Integer>)map.clone(), pills, strength,
mid))
                res = left = mid + 1;
            else
                right = mid - 1;
        }
        return res;
    }
    boolean validate(int[] tasks, TreeMap<Integer, Integer> map, int pills, int strength,
int pos) {
        for (; pos >= 0; pos--) {
            int maxStrength = map.lastKey(), t = tasks[pos];
            if (pills > 0 && strength + maxStrength < t || pills == 0 && maxStrength < t)
                return false;
            if (maxStrength < t) {
                t -= strength;
                pills--;
            }
            int matchStrength = map.ceilingKey(t);
            if (map.get(matchStrength) > 1)
                map.put(matchStrength, map.get(matchStrength) - 1);
            else
                map.remove(matchStrength);
        }
        return true;
    }
}
```