Advanced Programming LAB II

EXPERIMENT - 8

Submitted by,

Jiya | 22BCS14856

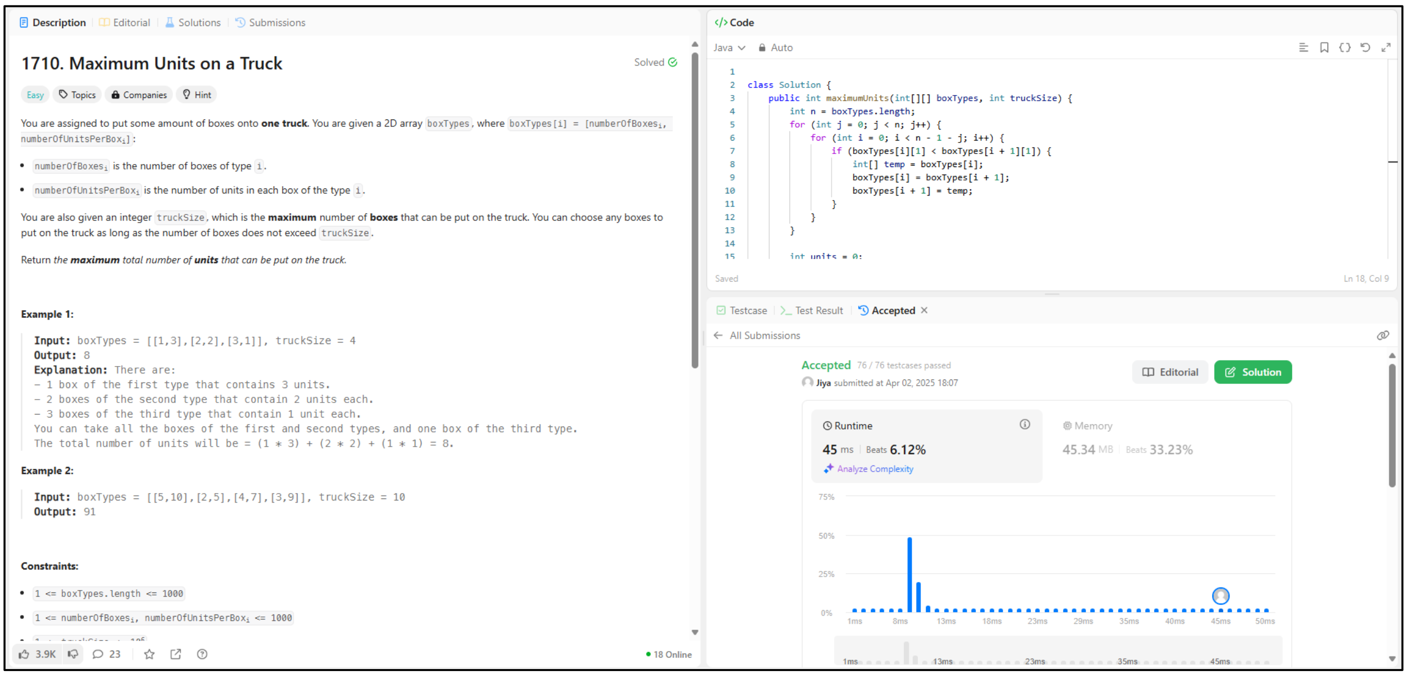
22BCS_FL_IOT-601 (A)

1710. Maximum Units on a Truck

https://leetcode.com/problems/maximum-units-on-a-truck/description/

```java
class Solution {
    public int maximumUnits(int[][] boxTypes, int truckSize) {
        int n = boxTypes.length;
        for (int j = 0; j < n; j++) {
            for (int i = 0; i < n - 1 - j; i++) {
                if (boxTypes[i][1] < boxTypes[i + 1][1]) {
                    int[] temp = boxTypes[i];
                    boxTypes[i] = boxTypes[i + 1];
                    boxTypes[i + 1] = temp;
                }
            }
        }

        int units = 0;
        int boxs = 0;
        for (int i = 0; i < n; i++) {
            if (boxs + boxTypes[i][0] <= truckSize) {
                units += boxTypes[i][0] * boxTypes[i][1];
                boxs += boxTypes[i][0];
            } else if (boxs < truckSize) {
                int remaining = truckSize - boxs;
                units += remaining * boxTypes[i][1];
                break;
            }
        }
        return units;
    }
}
```



1710. Maximum Units on a Truck

You are assigned to put some amount of boxes onto **one truck**. You are given a 2D array `boxTypes`, where `boxTypes[i] = [numberOfBoxes_i, numberOfUnitsPerBox_i]`:

- `numberOfBoxes_i` is the number of boxes of type `i`.
- `numberOfUnitsPerBox_i` is the number of units in each box of the type `i`.

You are also given an integer `truckSize`, which is the **maximum** number of **boxes** that can be put on the truck. You can choose any boxes to put on the truck as long as the number of boxes does not exceed `truckSize`.

Return *the **maximum** total number of **units** that can be put on the truck.*

Example 1:

```
Input: boxTypes = [[1,3],[2,2],[3,1]], truckSize = 4
Output: 8
Explanation: There are:
- 1 box of the first type that contains 3 units.
- 2 boxes of the second type that contain 2 units each.
- 3 boxes of the third type that contain 1 unit each.
You can take all the boxes of the first and second types, and one box of the third type.
The total number of units will be = (1 * 3) + (2 * 2) + (1 * 1) = 8.
```

Example 2:

```
Input: boxTypes = [[5,10],[2,5],[4,7],[3,9]], truckSize = 10
Output: 91
```

Constraints:

- `1 <= boxTypes.length <= 1000`
- `1 <= numberOfBoxes_i, numberOfUnitsPerBox_i <= 1000`

```java
class Solution {
    public int maximumUnits(int[][] boxTypes, int truckSize) {
        int n = boxTypes.length;
        for (int j = 0; j < n; j++) {
            for (int i = 0; i < n - 1 - j; i++) {
                if (boxTypes[i][1] < boxTypes[i + 1][1]) {
                    int[] temp = boxTypes[i];
                    boxTypes[i] = boxTypes[i + 1];
                    boxTypes[i + 1] = temp;
                }
            }
        }

        int units = 0;
```

Accepted 76 / 76 testcases passed
Jiya submitted at Apr 02, 2025 18:07

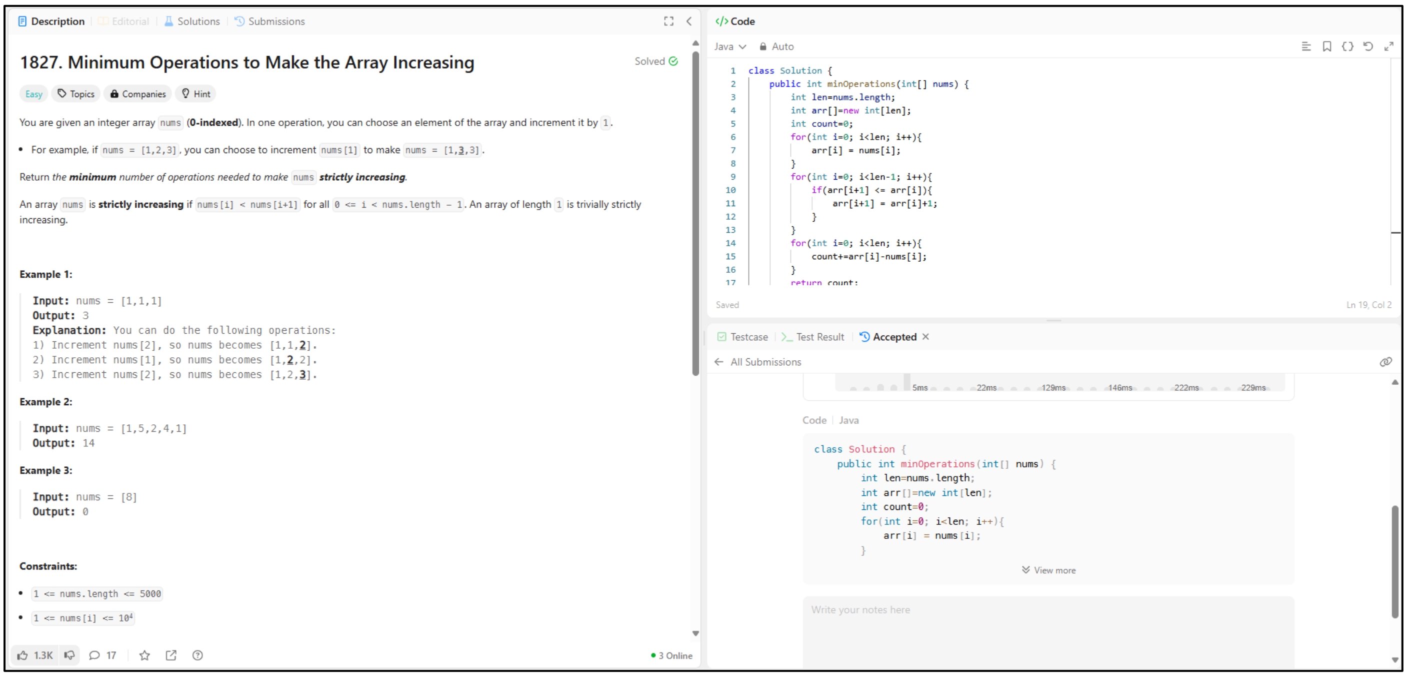
Runtime
45 ms | Beats 6.12%

Memory
45.34 MB | Beats 33.23%

1827. Minimum Operations to Make the Array Increasing

https://leetcode.com/problems/minimum-operations-to-make-the-array-increasing/description/

```java
class Solution {
    public int minOperations(int[] nums) {
        int len=nums.length;
        int arr[]=new int[len];
        int count=0;
        for(int i=0; i<len; i++){
            arr[i] = nums[i];
        }
        for(int i=0; i<len-1; i++){
            if(arr[i+1] <= arr[i]){
                arr[i+1] = arr[i]+1;
            }
        }
        for(int i=0; i<len; i++){
            count+=arr[i]-nums[i];
        }
        return count;
    }
}
```



1827. Minimum Operations to Make the Array Increasing

You are given an integer array `nums` (**0-indexed**). In one operation, you can choose an element of the array and increment it by `1`.

- For example, if `nums = [1,2,3]`, you can choose to increment `nums[1]` to make `nums = [1,3,3]`.

Return *the* ***minimum*** *number of operations needed to make* `nums` ***strictly increasing***.

An array `nums` is **strictly increasing** if `nums[i] < nums[i+1]` for all `0 <= i < nums.length - 1`. An array of length `1` is trivially strictly increasing.

Example 1:

```
Input: nums = [1,1,1]
Output: 3
Explanation: You can do the following operations:
1) Increment nums[2], so nums becomes [1,1,2].
2) Increment nums[1], so nums becomes [1,2,2].
3) Increment nums[2], so nums becomes [1,2,3].
```

Example 2:

```
Input: nums = [1,5,2,4,1]
Output: 14
```

Example 3:

```
Input: nums = [8]
Output: 0
```

Constraints:

- `1 <= nums.length <= 5000`
- `1 <= nums[i] <= 10^4`

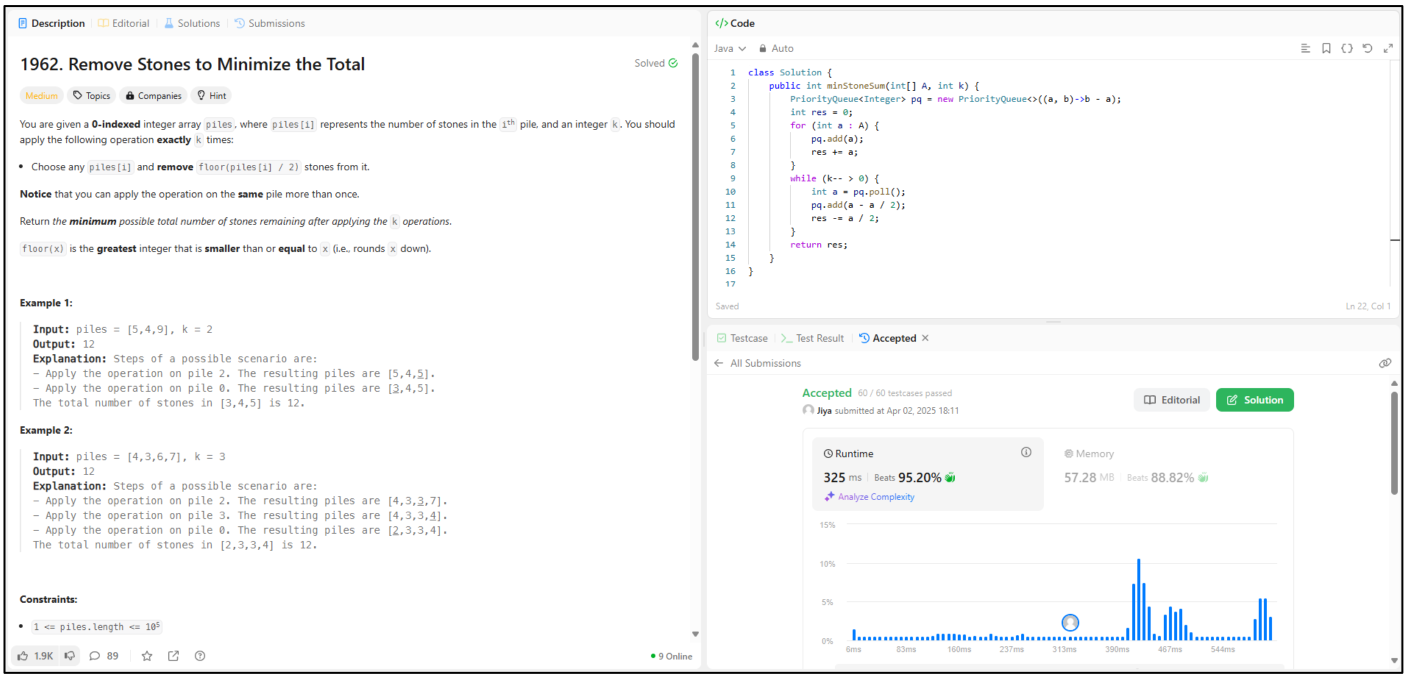
```java
class Solution {
    public int minOperations(int[] nums) {
        int len=nums.length;
        int arr[]=new int[len];
        int count=0;
        for(int i=0; i<len; i++){
            arr[i] = nums[i];
        }
        for(int i=0; i<len-1; i++){
            if(arr[i+1] <= arr[i]){
                arr[i+1] = arr[i]+1;
            }
        }
        for(int i=0; i<len; i++){
            count+=arr[i]-nums[i];
        }
        return count;
```

```java
class Solution {
    public int minOperations(int[] nums) {
        int len=nums.length;
        int arr[]=new int[len];
        int count=0;
        for(int i=0; i<len; i++){
            arr[i] = nums[i];
        }
```

1962. Remove Stones to Minimize the Total

https://leetcode.com/problems/remove-stones-to-minimize-the-total/description/

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



1962. Remove Stones to Minimize the Total

Solved

Medium · Topics · Companies · Hint

You are given a **0-indexed** integer array `piles`, where `piles[i]` represents the number of stones in the `ith` pile, and an integer `k`. You should apply the following operation **exactly** `k` times:

- Choose any `piles[i]` and **remove** `floor(piles[i] / 2)` stones from it.

Notice that you can apply the operation on the **same** pile more than once.

Return *the **minimum** possible total number of stones remaining after applying the* `k` *operations.*

`floor(x)` is the **greatest** integer that is **smaller** than or **equal** to `x` (i.e., rounds `x` down).

Example 1:

```
Input: piles = [5,4,9], k = 2
Output: 12
Explanation: Steps of a possible scenario are:
- Apply the operation on pile 2. The resulting piles are [5,4,5].
- Apply the operation on pile 0. The resulting piles are [3,4,5].
The total number of stones in [3,4,5] is 12.
```

Example 2:

```
Input: piles = [4,3,6,7], k = 3
Output: 12
Explanation: Steps of a possible scenario are:
- Apply the operation on pile 2. The resulting piles are [4,3,3,7].
- Apply the operation on pile 3. The resulting piles are [4,3,3,4].
- Apply the operation on pile 0. The resulting piles are [2,3,3,4].
The total number of stones in [2,3,3,4] is 12.
```

Constraints:

- `1 <= piles.length <= 10^5`

1.9K · 89 · 9 Online

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

Saved · Ln 22, Col 1

Testcase · Test Result · Accepted

All Submissions

Accepted 60 / 60 testcases passed

Jiya submitted at Apr 02, 2025 18:11

Editorial · Solution

Runtime 325 ms | Beats 95.20%

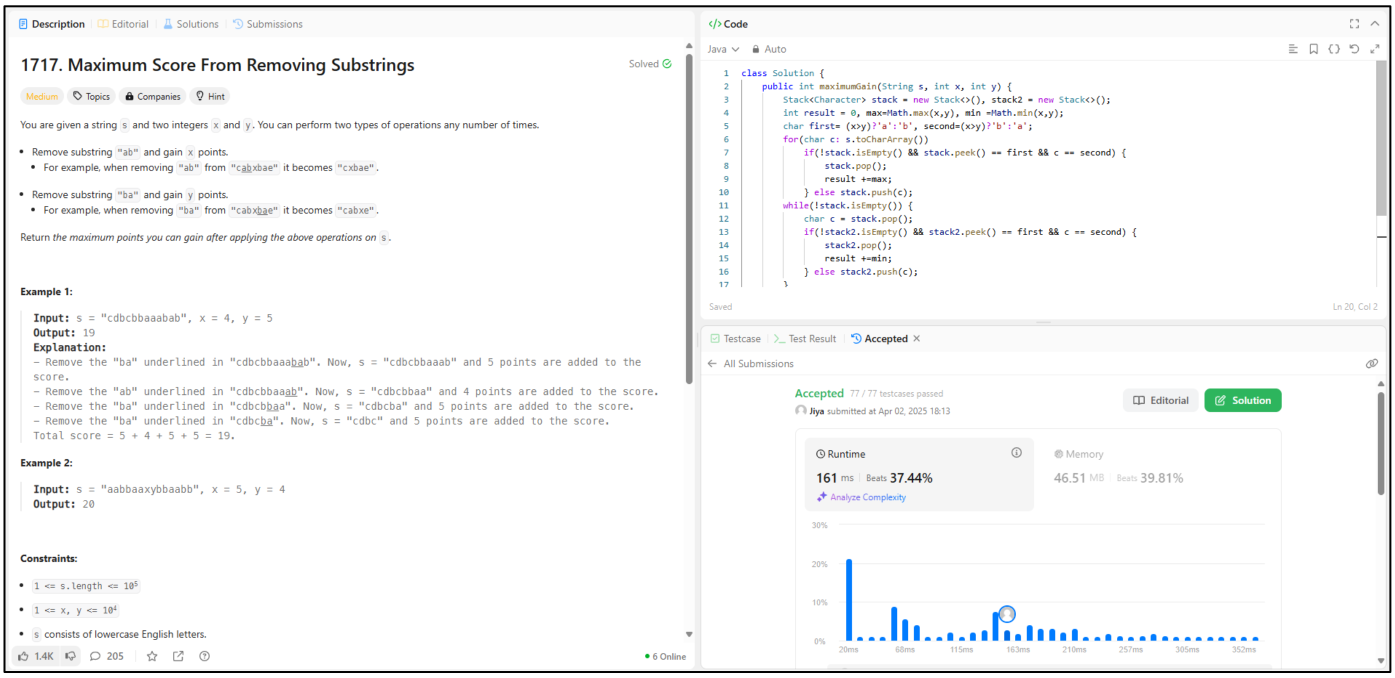
Analyze Complexity

Memory 57.28 MB | Beats 88.82%

1717. Maximum Score From Removing Substrings

https://leetcode.com/problems/maximum-score-from-removing-substrings/description/

```java
class Solution {
    public int maximumGain(String s, int x, int y) {
        Stack<Character> stack = new Stack<>(), stack2 = new Stack<>();
        int result = 0, max=Math.max(x,y), min =Math.min(x,y);
        char first= (x>y)?'a':'b', second=(x>y)?'b':'a';
        for(char c: s.toCharArray())
            if(!stack.isEmpty() && stack.peek() == first && c == second) {
                stack.pop();
                result +=max;
            } else stack.push(c);
        while(!stack.isEmpty()) {
            char c = stack.pop();
            if(!stack2.isEmpty() && stack2.peek() == first && c == second) {
                stack2.pop();
                result +=min;
            } else stack2.push(c);
        }
        return result;
    }
}
```



1717. Maximum Score From Removing Substrings

Solved

You are given a string s and two integers x and y. You can perform two types of operations any number of times.

- Remove substring "ab" and gain x points.
  - For example, when removing "ab" from "cabxbae" it becomes "cxbae".
- Remove substring "ba" and gain y points.
  - For example, when removing "ba" from "cabxbae" it becomes "cabxe".

Return *the maximum points you can gain after applying the above operations on* s.

Example 1:

```
Input: s = "cdbcbbaaabab", x = 4, y = 5
Output: 19
Explanation:
- Remove the "ba" underlined in "cdbcbbaaabab". Now, s = "cdbcbbaaab" and 5 points are added to the
score.
- Remove the "ab" underlined in "cdbcbbaaab". Now, s = "cdbcbbaa" and 4 points are added to the score.
- Remove the "ba" underlined in "cdbcbbaa". Now, s = "cdbcba" and 5 points are added to the score.
- Remove the "ba" underlined in "cdbcba". Now, s = "cdbc" and 5 points are added to the score.
Total score = 5 + 4 + 5 + 5 = 19.
```

Example 2:

```
Input: s = "aabbaaxybbaabb", x = 5, y = 4
Output: 20
```

Constraints:

- 1 <= s.length <= 10^5
- 1 <= x, y <= 10^4
- s consists of lowercase English letters.

```java
class Solution {
    public int maximumGain(String s, int x, int y) {
        Stack<Character> stack = new Stack<>(), stack2 = new Stack<>();
        int result = 0, max=Math.max(x,y), min =Math.min(x,y);
        char first= (x>y)?'a':'b', second=(x>y)?'b':'a';
        for(char c: s.toCharArray())
            if(!stack.isEmpty() && stack.peek() == first && c == second) {
                stack.pop();
                result +=max;
            } else stack.push(c);
        while(!stack.isEmpty()) {
            char c = stack.pop();
            if(!stack2.isEmpty() && stack2.peek() == first && c == second) {
                stack2.pop();
                result +=min;
            } else stack2.push(c);
        }
```

Saved   Ln 20, Col 2

Accepted 77 / 77 testcases passed
Jiya submitted at Apr 02, 2025 18:13

Runtime
161 ms | Beats 37.44%

Memory
46.51 MB | Beats 39.81%

1713. Minimum Operations to Make a Subsequence

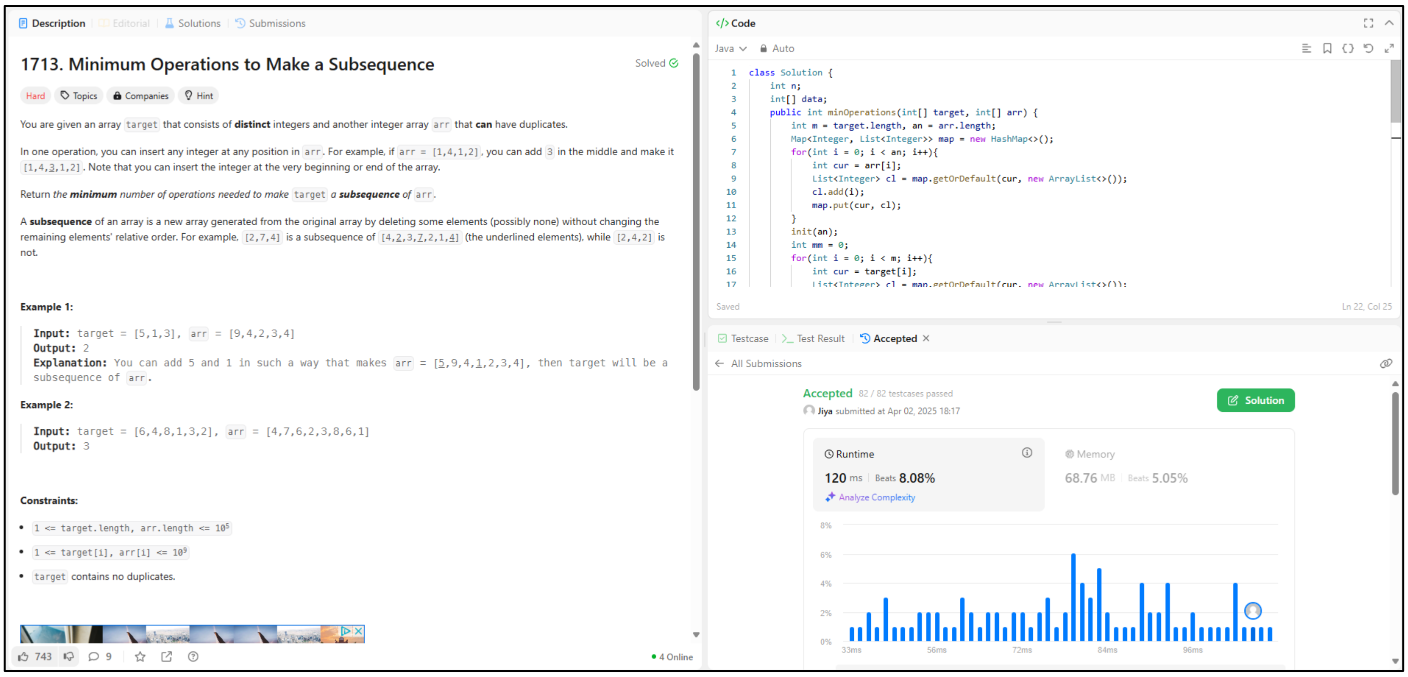
https://leetcode.com/problems/minimum-operations-to-make-a-subsequence/description/

```
class Solution {
    int n;
    int[] data;
    public int minOperations(int[] target, int[] arr) {
        int m = target.length, an = arr.length;
        Map<Integer, List<Integer>> map = new HashMap<>();
        for(int i = 0; i < an; i++){
            int cur = arr[i];
            List<Integer> cl = map.getOrDefault(cur, new ArrayList<>());
            cl.add(i);
            map.put(cur, cl);
        }
        init(an);
        int mm = 0;
        for(int i = 0; i < m; i++){
            int cur = target[i];
            List<Integer> cl = map.getOrDefault(cur, new ArrayList<>());
            int clen = cl.size();
            for(int j = clen - 1; j >=0; j--){
                int pos = cl.get(j);
                int prev = query(0, pos, 0,0,n);
                int curv = prev + 1;
                update(pos, curv);
                //System.out.println(pos +" "+ curv);
                if(mm < curv) mm = curv;
            }
        }
        return m - mm;
    }
    void init(int n_){
        n = 1;
        while(n < n_) n *= 2;
        data = new int[2 * n - 1];
        for(int i = 0; i < 2 * n - 1; i++)data[i] =0;
    }

    void update(int k, int a){
        k += n - 1;
        data[k] = a;
        while(k > 0){
            k = (k - 1)/2;
            data[k] = Math.max(data[k * 2 + 1], data[k * 2 + 2]);
        }
    }

    int query(int a, int b, int k, int l, int r){
        if(r <= a || b <= l)return 0;
        if(a <= l && r <= b){
            return data[k];
        } else {
            int vl = query(a,b,k * 2 + 1, l, (l + r)/2);
            int vr = query(a,b,k * 2 + 2, (l + r)/2, r);
            return Math.max(vl, vr);
        }

    }
}
```



1713. Minimum Operations to Make a Subsequence

Solved

You are given an array `target` that consists of **distinct** integers and another integer array `arr` that **can** have duplicates.

In one operation, you can insert any integer at any position in `arr`. For example, if `arr = [1,4,1,2]`, you can add `3` in the middle and make it `[1,4,3,1,2]`. Note that you can insert the integer at the very beginning or end of the array.

Return *the **minimum** number of operations needed to make* `target` *a **subsequence** of* `arr`.

A **subsequence** of an array is a new array generated from the original array by deleting some elements (possibly none) without changing the remaining elements' relative order. For example, `[2,7,4]` is a subsequence of `[4,2,3,7,2,1,4]` (the underlined elements), while `[2,4,2]` is not.

Example 1:

```
Input: target = [5,1,3], arr = [9,4,2,3,4]
Output: 2
Explanation: You can add 5 and 1 in such a way that makes arr = [5,9,4,1,2,3,4], then target will be a
subsequence of arr.
```

Example 2:

```
Input: target = [6,4,8,1,3,2], arr = [4,7,6,2,3,8,6,1]
Output: 3
```

Constraints:

- `1 <= target.length, arr.length <= `10^5
- `1 <= target[i], arr[i] <= `10^9
- `target` contains no duplicates.

```java
class Solution {
    int n;
    int[] data;
    public int minOperations(int[] target, int[] arr) {
        int m = target.length, an = arr.length;
        Map<Integer, List<Integer>> map = new HashMap<>();
        for(int i = 0; i < an; i++){
            int cur = arr[i];
            List<Integer> cl = map.getOrDefault(cur, new ArrayList<>());
            cl.add(i);
            map.put(cur, cl);
        }
        init(an);
        int mm = 0;
        for(int i = 0; i < m; i++){
            int cur = target[i];
            List<Integer> cl = map.getOrDefault(cur, new ArrayList<>());
```

Accepted 82 / 82 testcases passed

Jiya submitted at Apr 02, 2025 18:17

Runtime 120 ms | Beats 8.08%

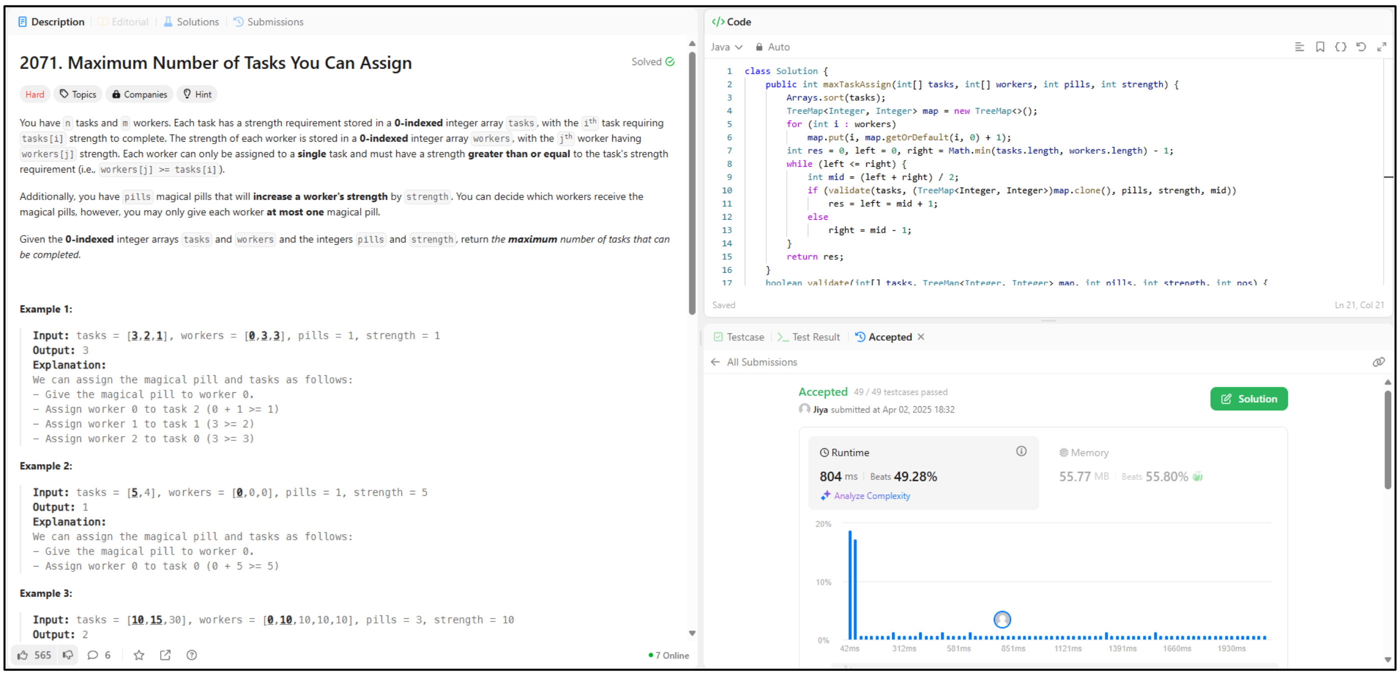
Analyze Complexity

Memory 68.76 MB | Beats 5.05%

2071. Maximum Number of Tasks You Can Assign

https://leetcode.com/problems/maximum-number-of-tasks-you-can-assign/description/

```
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;
    }
}
```



2071. Maximum Number of Tasks You Can Assign

You have `n` tasks and `m` workers. Each task has a strength requirement stored in a **0-indexed** integer array `tasks`, with the `i`th task requiring `tasks[i]` strength to complete. The strength of each worker is stored in a **0-indexed** integer array `workers`, with the `j`th worker having `workers[j]` strength. Each worker can only be assigned to a **single** task and must have a strength **greater than or equal** to the task's strength requirement (i.e., `workers[j] >= tasks[i]`).

Additionally, you have `pills` magical pills that will **increase a worker's strength** by `strength`. You can decide which workers receive the magical pills, however, you may only give each worker **at most one** magical pill.

Given the **0-indexed** integer arrays `tasks` and `workers` and the integers `pills` and `strength`, return *the **maximum** number of tasks that can be completed.*

Example 1:

```
Input: tasks = [3,2,1], workers = [0,3,3], pills = 1, strength = 1
Output: 3
Explanation:
We can assign the magical pill and tasks as follows:
- Give the magical pill to worker 0.
- Assign worker 0 to task 2 (0 + 1 >= 1)
- Assign worker 1 to task 1 (3 >= 2)
- Assign worker 2 to task 0 (3 >= 3)
```

Example 2:

```
Input: tasks = [5,4], workers = [0,0,0], pills = 1, strength = 5
Output: 1
Explanation:
We can assign the magical pill and tasks as follows:
- Give the magical pill to worker 0.
- Assign worker 0 to task 0 (0 + 5 >= 5)
```

Example 3:

```
Input: tasks = [10,15,30], workers = [0,10,10,10,10], pills = 3, strength = 10
Output: 2
```

```
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) {
```

Accepted 49 / 49 testcases passed
Jiya submitted at Apr 02, 2025 18:32

Runtime
804 ms | Beats 49.28%

Memory
55.77 MB | Beats 55.80%