

Experiment 8

Student Name: Saurav Ashiwal

Branch: CSE

Semester: 6

Subject Name: AP lab-2

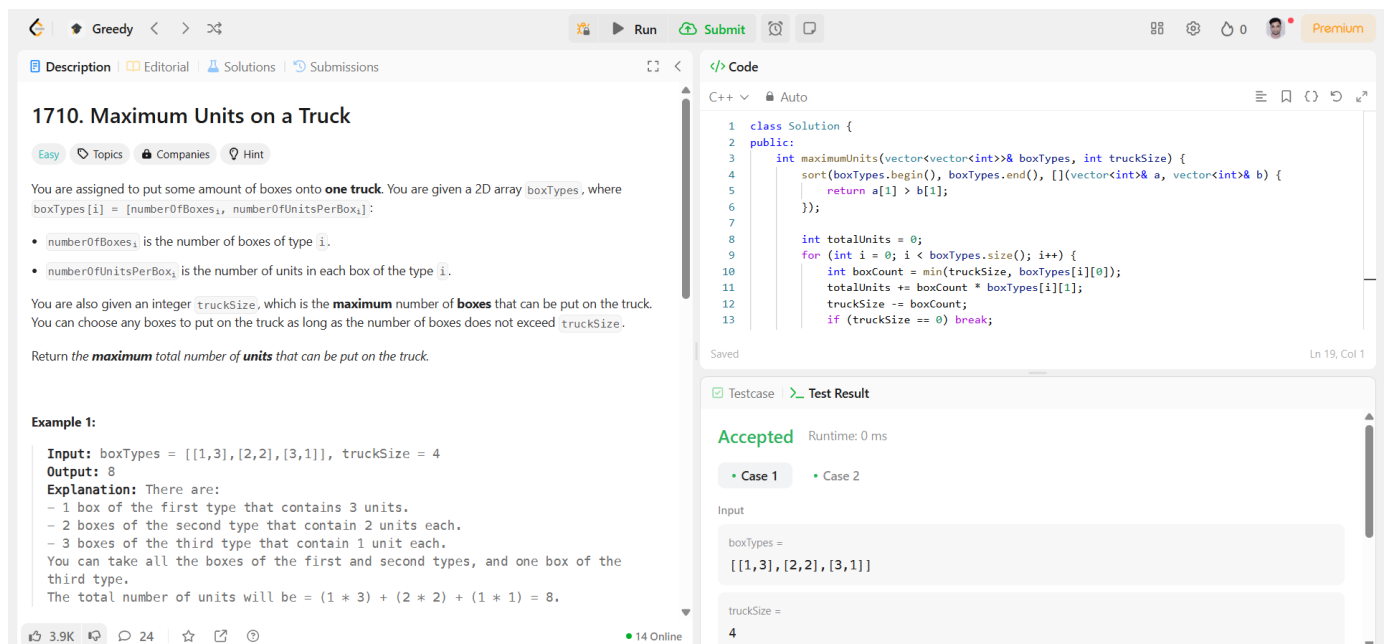
UID:22BCS13250

Section/Group: 614/B

Date of Performance:03/04/2025

Subject Code: 22CSP-351

Q 1. Max Units on a Truck



The screenshot displays a coding interface for the problem "1710. Maximum Units on a Truck". The problem description states: "You are assigned to put some amount of boxes onto **one truck**. You are given a 2D array `boxTypes`, where `boxTypes[i] = [numberOfBoxesi, numberOfUnitsPerBoxi]`:"

- `numberOfBoxesi` is the number of boxes of type `i`.
- `numberOfUnitsPerBoxi` 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:

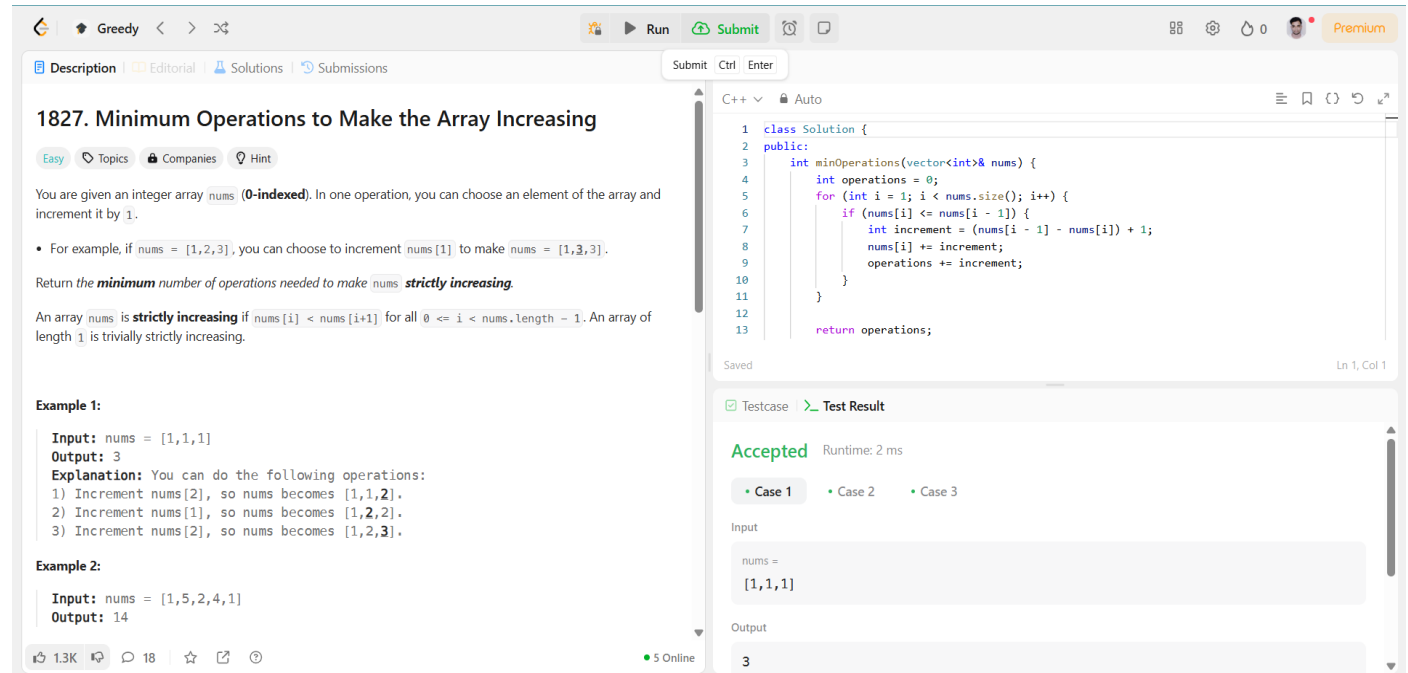
Input: `boxTypes = [[1,3],[2,2],[3,1]]`, `truckSize = 4`
Output: 8
Explanation: There are:
- 1 box of the first type that contains 3 units.
- 2 boxes of the second type that contain 2 units each.
- 3 boxes of the third type that contain 1 unit each.
You can take all the boxes of the first and second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

The code editor shows the following C++ solution:

```
1 class Solution {
2 public:
3     int maximumUnits(vector<vector<int>>& boxTypes, int truckSize) {
4         sort(boxTypes.begin(), boxTypes.end(), [](vector<int>& a, vector<int>& b) {
5             return a[1] > b[1];
6         });
7
8         int totalUnits = 0;
9         for (int i = 0; i < boxTypes.size(); i++) {
10             int boxCount = min(truckSize, boxTypes[i][0]);
11             totalUnits += boxCount * boxTypes[i][1];
12             truckSize -= boxCount;
13             if (truckSize == 0) break;
14         }
15         return totalUnits;
16     }
17 }
```

The test result shows "Accepted" with a runtime of 0 ms. The input is `boxTypes = [[1,3],[2,2],[3,1]]` and `truckSize = 4`.

2. Min Operations to Make Array Increasing



127. Minimum Operations to Make the Array Increasing

Easy Topics Companies Hint

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 $0 \leq i < \text{nums.length} - 1$. 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

```
1 class Solution {
2 public:
3     int minOperations(vector<int>& nums) {
4         int operations = 0;
5         for (int i = 1; i < nums.size(); i++) {
6             if (nums[i] <= nums[i - 1]) {
7                 int increment = (nums[i - 1] - nums[i]) + 1;
8                 nums[i] += increment;
9                 operations += increment;
10            }
11        }
12        return operations;
13    }
```

Accepted Runtime: 2 ms

Case 1 Case 2 Case 3

Input

`nums =`
`[1,1,1]`

Output

3

1.3K 18 5 Online

3. Remove Stones to Maximize Total

Greedy < > ↺

Run Submit ⌂

8 0 Premium

Description Editorial Solutions Submissions

Submit Ctrl Enter

1962. Remove Stones to Minimize the Total

Medium Topics Companies Hint

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

- Choose any `piles[i]` and **remove** $\text{floor}(\text{piles}[i] / 2)$ 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.

$\text{floor}(x)$ 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

1.9K 89 ☆ ↗ ⌂

8 Online

C++ Auto

```
1 class Solution {
2 public:
3     int minStoneSum(vector<int>& piles, int k) {
4         priority_queue<int> pq;
5         for (int pile : piles) pq.push(pile);
6
7         while (k--) {
8             int top = pq.top();
9             pq.pop();
10            top -= top / 2;
11            pq.push(top);
12        }
13    }
14 }
```

Saved Ln 23, Col 1

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

piles =
[5,4,9]

k =
2

4.. Max Score from Removing Substrings

Greedy < > ↺

Run Submit

0 Premium

Description Editorial Solutions Submissions

1717. Maximum Score From Removing Substrings Solved ✓

Medium Topics Companies Hint

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 = "cdbcb"` and 5 points are added to the score.
- Remove the `"ba"` underlined in `"cdbcba"`. Now, `s = "cd"` and 5 points are added to the score.

</> Code

C++ Auto

```
1 class Solution {
2 public:
3     int maximumGain(string s, int x, int y) {
4         stack<char> st;
5         int total = 0;
6
7         string first, second;
8         int firstScore, secondScore;
9
10        if (x > y) {
11            first = "ab";
12            firstScore = x;
13            second = "ba";
```

Saved Ln 1, Col 1

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

s =

"cdbcbbaaabab"

x =

4

1.4K 205 22 Online

5. Min Operations to Make a Subsequence

Greedy < > ↺

Run Submit ⌛

0 0 Premium

Description Editorial Solutions Submissions

1713. Minimum Operations to Make a Subsequence

Hard Topics Companies Hint

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

743 9 7 Online

Code

C++ Auto

```
1 class Solution {
2 public:
3     int minOperations(vector<int>& target, vector<int>& arr) {
4         unordered_map<int, int> pos;
5         for (int i = 0; i < target.size(); ++i) {
6             pos[target[i]] = i;
7         }
8
9         vector<int> seq;
10        for (int num : arr) {
11            if (pos.count(num)) {
12                int index = pos[num];
13                auto it = lower_bound(seq.begin(), seq.end(), index);
```

Saved Ln 25, Col 1

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

target =

[5,1,3]

arr =

[9,4,2,3,4]

6. Max Number of Tasks You Can Assign

Greedy < > ↺

Run Submit ⌵

0 Premium

Description Editorial Solutions Submissions

2071. Maximum Number of Tasks You Can Assign

Hard Topics Companies Hint

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 ($3 >= 3$)

566 6 ☆ ↗ ⌵

7 Online

Code

C++ Auto

```
1 class Solution {
2 public:
3     bool canAssign(int k, vector<int>& tasks, vector<int>& workers, int pills, int strength) {
4         multiset<int> ws(workers.begin() - k, workers.end());
5         int p = pills;
6
7         for (int i = k - 1; i >= 0; --i) {
8             auto it = ws.lower_bound(tasks[i]);
9             if (it != ws.end()) {
10                 ws.erase(it);
11             } else {
12                 if (p == 0) return false;
13                 it = ws.lower_bound(tasks[i] - strength);
```

Saved Ln 40, Col 1

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

tasks =

[3,2,1]

workers =

[0,3,3]