Experiment-8

Student Name: Abhinav Paswan **UID:** 22BET10332
Branch: BE-IT **Section/Group:** 22BET-IOT-701(A)
Semester: 6th **Date of Performance:**28-03-2025
Subject Name: AP LAB-II **Subject Code:** 22ITP-351

Problem 1:-

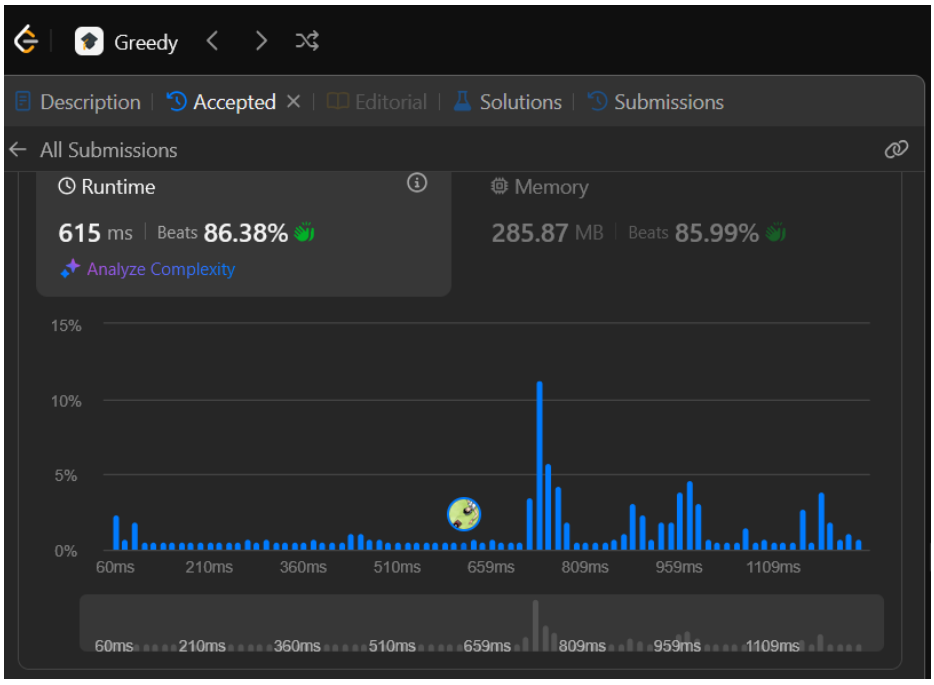
You have n tasks and m workers. Each task has a strength requirement stored in a **0-indexed** integer array tasks, with the i^{th} task requiring tasks[i] strength to complete. The strength of each worker is stored in a **0-indexed** integer array workers, with the j^{th} worker having workers[j] strength. 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.*

Code:

```
class Solution {
public:
   bool canAssign(int k, vector<int>& tasks, vector<int>& workers, int pills, int strength) {
      multiset<int> available_workers(workers.end() - k, workers.end());
      int pill_count = pills;

      for (int i = k - 1; i >= 0; i--) {
         int task = tasks[i];

         auto it = available_workers.rbegin();
         if (*it >= task) {
            available_workers.erase(prev(available_workers.end()));
         }
         else if (pill_count > 0) {
            auto weak_it = available_workers.lower_bound(task - strength);
            if (weak_it != available_workers.end()) {
               available_workers.erase(weak_it);
               pill_count--;
            }
            else {
               return false;
            }
         }
```

```
            else {
                return false;
            }
        }
        return true;
    }

    int maxTaskAssign(vector<int>& tasks, vector<int>& workers, int pills, int strength) {
        sort(tasks.begin(), tasks.end());
        sort(workers.begin(), workers.end());

        int left = 0, right = min((int)tasks.size(), (int)workers.size());
        while (left < right) {
            int mid = (left + right + 1) / 2;
            if (canAssign(mid, tasks, workers, pills, strength))
                left = mid;
            else
                right = mid - 1;
        }
        return left;
    }
};
```

OUTPUT:



Problem 2 :-

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.

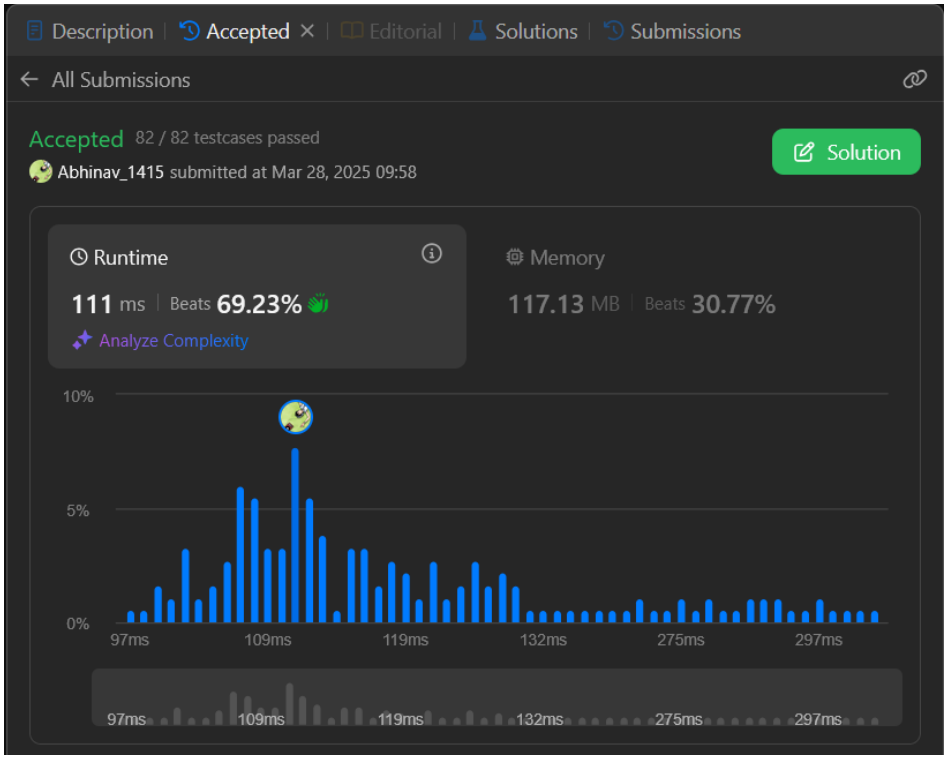
Return the minimum number of operations needed to make target a subsequence of arr.

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.

Code:

```
class Solution {
public:
   int minOperations(vector<int>& target, vector<int>& arr) {
     unordered_map<int, int> index_map;
      int n = target.size();

      for (int i = 0; i < n; i++) {
         index_map[target[i]] = i;
      }

      vector<int> transformed;
      for (int num : arr) {
         if (index_map.find(num) != index_map.end()) {
            transformed.push_back(index_map[num]);
         }
      }

      vector<int> lis;
      for (int idx : transformed) {
         auto it = lower_bound(lis.begin(), lis.end(), idx);
         if (it == lis.end()) {
            lis.push_back(idx);
         } else {
            *it = idx;
         }
      }

      return n - lis.size();
   }
};
```

OUTPUT:


Description
Accepted
Editorial
Solutions
Submissions
All Submissions
Accepted 82 / 82 testcases passed
Abhinav_1415 submitted at Mar 28, 2025 09:58
Solution
Runtime
111 ms | Beats 69.23%
Analyze Complexity
Memory
117.13 MB | Beats 30.77%
10%
5%
0%
97ms
109ms
119ms
132ms
275ms
297ms

Problem 3:-

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".

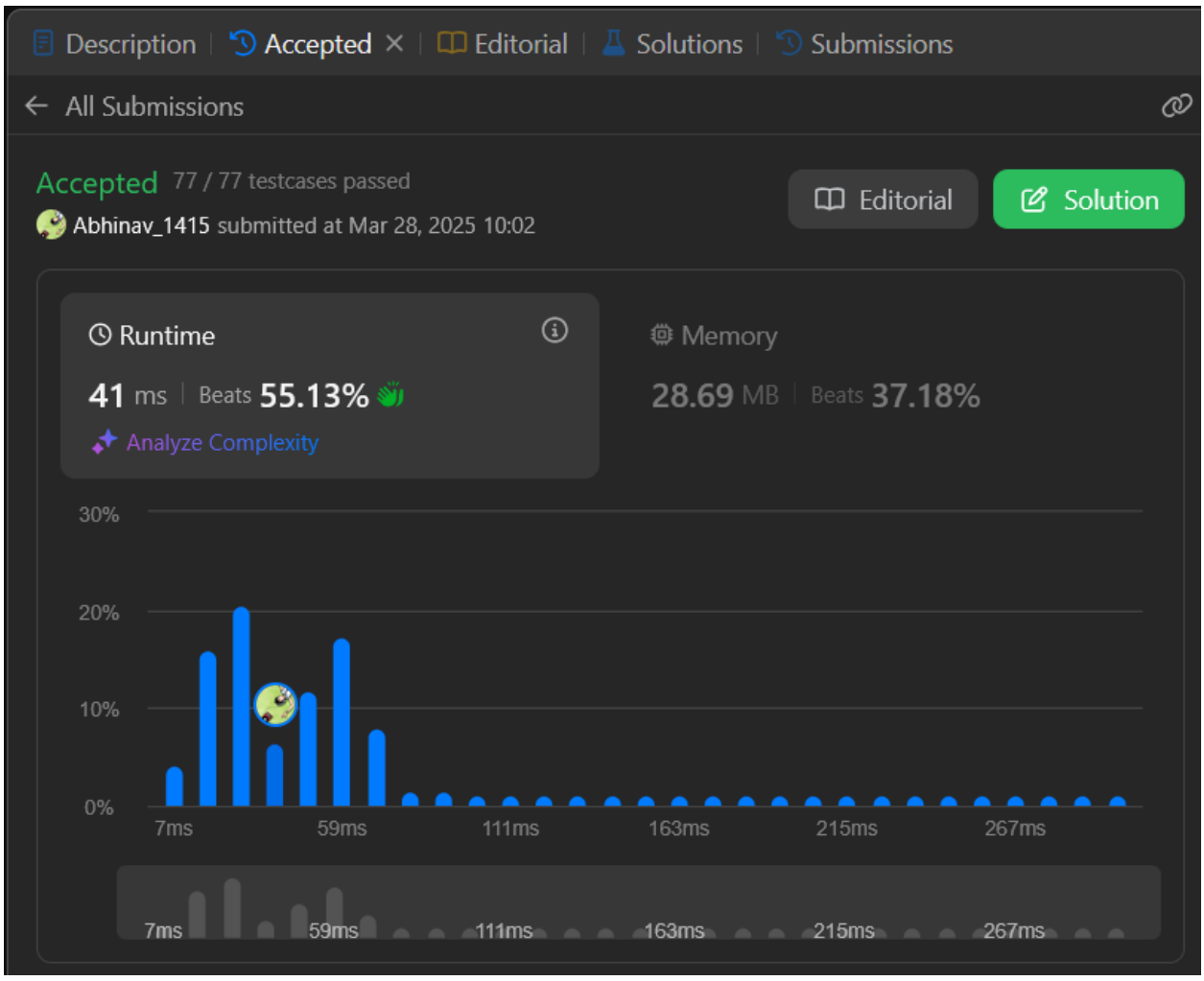
Return the maximum points you can gain after applying the above operations on s.

Code:

```
class Solution {
public:
    int maximumGain(string s, int x, int y) {
        char firstA = 'a', firstB = 'b', secondA = 'b', secondB = 'a';
        if (y > x) {
            swap(x, y);
            swap(firstA, secondA);
            swap(firstB, secondB);
        }

        stack<char> st;
        int score = 0;
        string reduced = "";

        for (char c : s) {
            if (!st.empty() && st.top() == firstA && c == firstB) {
                st.pop();
                score += x;
            } else {
                st.push(c);
            }
        }

        while (!st.empty()) {
            reduced += st.top();
            st.pop();
        }
        reverse(reduced.begin(), reduced.end());

        for (char c : reduced) {
            if (!st.empty() && st.top() == secondA && c == secondB) {
```

```
            st.pop();
            score += y;
        } else {
            st.push(c);
        }
    }

    return score;
  }
};
```

OUTPUT:



Problem 4:-

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.

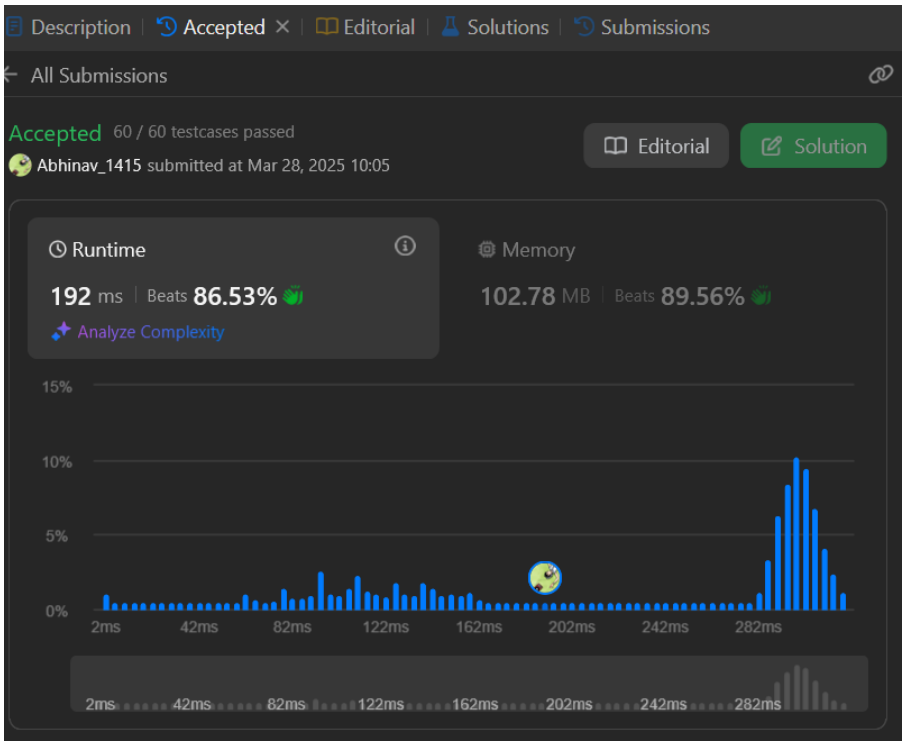
Return the minimum possible total number of stones remaining after applying the k operations.

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

Code:

```
class Solution {
public:
    int minStoneSum(vector<int>& piles, int k) {
    priority_queue<int> maxHeap(piles.begin(), piles.end());
    int totalSum = accumulate(piles.begin(), piles.end(), 0);

    while (k-- > 0) {
       int largest = maxHeap.top();
       maxHeap.pop();
       int removed = largest / 2;
       totalSum -= removed;
       maxHeap.push(largest - removed);
    }

    return totalSum;
  }
};
```

OUTPUT:


Description
Accepted
Editorial
Solutions
Submissions
All Submissions
Accepted 60 / 60 testcases passed
Abhinav_1415 submitted at Mar 28, 2025 10:05
Editorial
Solution
Runtime
192 ms | Beats 86.53%
Analyze Complexity
Memory
102.78 MB | Beats 89.56%
15%
10%
5%
0%
2ms
42ms
82ms
122ms
162ms
202ms
242ms
282ms

Problem 5:-

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].

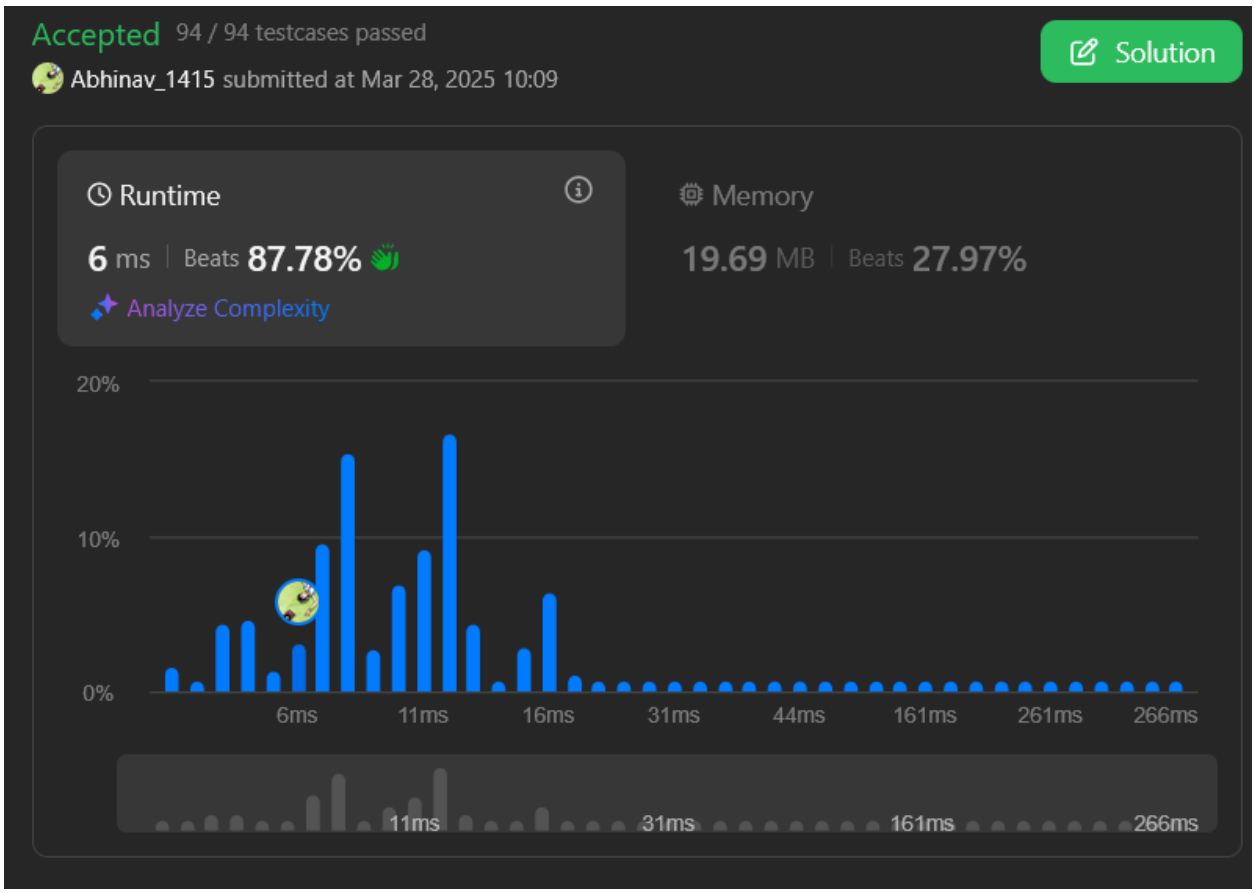
Return *the **minimum** number of operations needed to make* nums ***strictly increasing***.

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.

Code:-

```
class Solution {
public:
    int minOperations(vector<int>& nums) {

        int operations = 0;

        for (int i = 1; i < nums.size(); i++) {
            if (nums[i] <= nums[i - 1]) {
                int diff = (nums[i - 1] + 1 - nums[i]);
                operations += diff;
                nums[i] = nums[i - 1] + 1;
            }
        }

        return operations;
    }
};
```

OUTPUT:


Accepted 94 / 94 testcases passed
Abhinav_1415 submitted at Mar 28, 2025 10:09
Solution
Runtime
6 ms | Beats 87.78%
Analyze Complexity
Memory
19.69 MB | Beats 27.97%
20%
10%
0%
6ms
11ms
16ms
31ms
44ms
161ms
261ms
266ms
11ms
31ms
161ms
266ms

Problem 6:

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$_i$]:

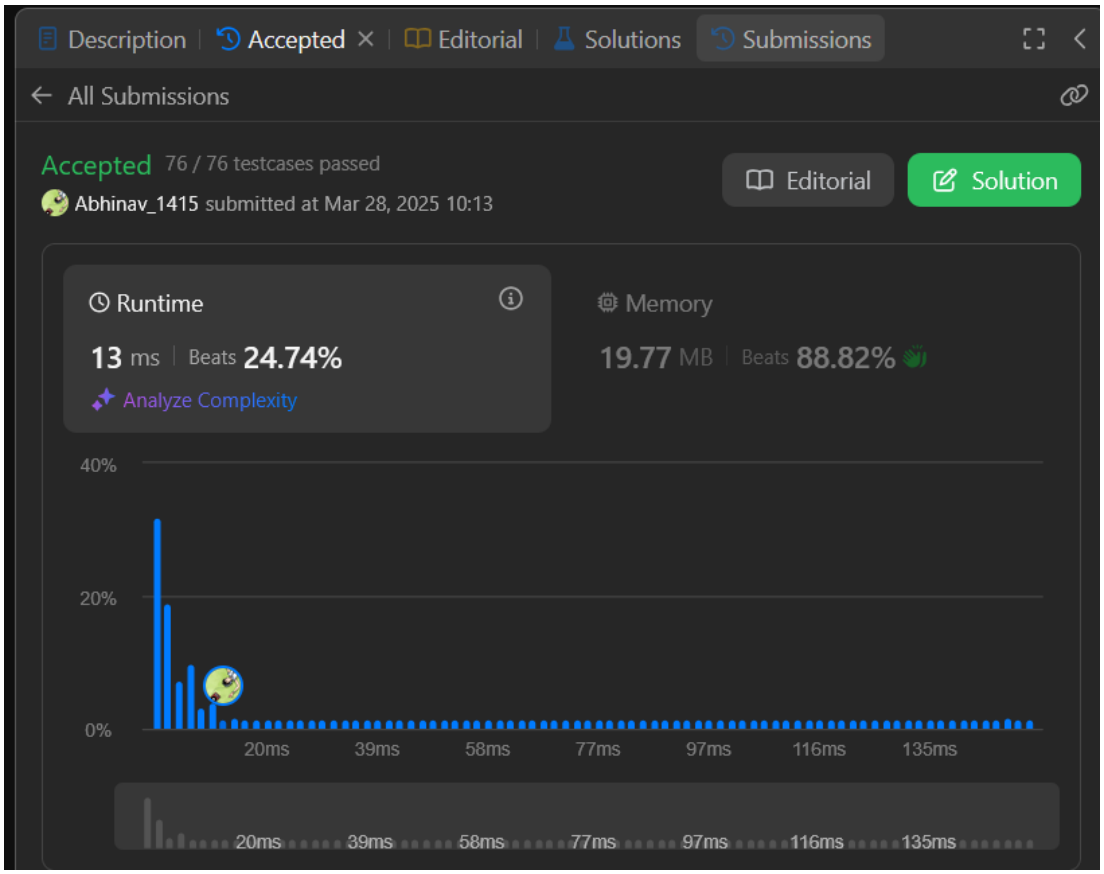
- numberOfBoxes$_i$ is the number of boxes of type i.
- numberOfUnitsPerBox$_i$ is the number of units in each box of the type i.

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.*

CODE:

```
class Solution {
public:
    int maximumUnits(vector<vector<int>>& boxTypes, int truckSize) {
        sort(boxTypes.begin(), boxTypes.end(), [](vector<int>& a, vector<int>& b) {
            return a[1] > b[1];
        });

        int totalUnits = 0;
        for (auto& box : boxTypes) {
            int boxesToTake = min(truckSize, box[0]);
            totalUnits += boxesToTake * box[1];
            truckSize -= boxesToTake;

            if (truckSize == 0) break;
        }

        return totalUnits;
    }
};
```

OUTPUT:


Description
Accepted
Editorial
Solutions
Submissions
All Submissions
Accepted 76 / 76 testcases passed
Abhinav_1415 submitted at Mar 28, 2025 10:13
Editorial
Solution
Runtime
13 ms | Beats 24.74%
Analyze Complexity
Memory
19.77 MB | Beats 88.82%
40%
20%
0%
20ms
39ms
58ms
77ms
97ms
116ms
135ms