Course Name:	Competitive Programming Lab	Semester:	IV
Date of Performance:	02/ 02/ 2026	DIV/ Batch No:	B2
Student Name:	Ashwera Hasan	Roll No:	16010124017

Experiment No: 2

Title: Implement Greedy algorithm

Aim of the Experiment: To understand and implement greedy algorithm for a given problem

Objective of the Experiment:

After completing this experiment, the student will be able to:

- **Understand the concept of greedy algorithms** and the principle of making locally optimal choices to achieve a global optimal solution.
- **Apply greedy strategies** to solve algorithmic problems by selecting the best immediate option at each step.
- **Design and implement C++ programs** using common greedy techniques such as sorting, priority queues, and interval scheduling.
- **Analyze the time and space complexity** of greedy algorithms and compare them with brute force and complete search approaches.
- **Identify problems suitable for greedy solutions** and justify the correctness of the greedy choice property.
- **Enhance logical reasoning and coding proficiency** by solving real-world and competitive programming problems using greedy methods.

COs to be achieved:

CO 1 Applying various problem-solving paradigms, enabling them to create and implement efficient algorithms for real-world challenges.

Theory:

Greedy algorithms are a class of algorithmic techniques that build a solution step by step by making the **locally optimal choice** at each stage, with the aim of achieving a **globally optimal solution**. Unlike brute force or complete search methods that explore all possible solutions, greedy algorithms focus on selecting the best immediate option without reconsidering previous choices.

Principle of Greedy Algorithms

The core idea behind greedy algorithms is the **greedy choice property**, which states that a global optimum can be reached by making locally optimal decisions. Once a choice is made, it is not changed later. This property makes greedy algorithms efficient and easy to implement, but it also means they are not suitable for all problems.

Another important concept is **optimal substructure**, where an optimal solution to a problem contains optimal solutions to its subproblems. Problems that satisfy both the greedy choice property and optimal substructure can be solved effectively using greedy algorithms.

Working Mechanism

A greedy algorithm typically follows these steps:

1. Define a criterion to make a local optimal choice.
2. Select the best option based on this criterion.
3. Reduce the problem size by fixing the chosen option.
4. Repeat the process until a complete solution is obtained.

Common techniques used in greedy algorithms include sorting, priority queues, and iterative selection based on a specific condition.

Time and Space Complexity

Greedy algorithms are generally efficient, often having time complexity of O(nlogn)O(n \log n)O(nlogn) due to sorting or priority queue operations. Space complexity is usually low, as greedy approaches often work iteratively and do not require extensive recursion or memory storage.

Advantages

- Simple and intuitive to understand
- Faster than brute force and complete search methods
- Easy to implement and debug
- Efficient for large input sizes

Limitations

- Not all problems can be solved using greedy algorithms
- Greedy solutions may fail if the problem does not satisfy the greedy choice property
- Correctness proof is required to ensure optimality

Problem Statements:

Difficulty (as per lab)	Problem	LeetCode mapping
Easy	Fractional knapsack	[LC 1710 – Maximum Units on a Truck]()
Medium	Activity selection	[LC 435 – Non-overlapping Intervals]()
Difficult	Candy	[Candy - LeetCode]()

Code :

Hard:

C++ Auto

```cpp
class Solution {
public:
    int candy(vector<int>& v) {
        int n=v.size();
        vector<int>l(n,1);
        vector<int>r(n,1);
        for(int i=1;i<n;i++){
            if(v[i]>v[i-1]){
                l[i] = l[i-1]+1;
            }
        }

        for(int i=n-2;i>=0;i--){
            if(v[i]>v[i+1]){
                r[i] = r[i+1]+1;
            }
        }

        int counter=0;
        for(int i=0;i<n;i++){
            counter+=max(l[i],r[i]);
        }
        return counter;
    }
};
```

Saved Ln 17, Col 10

Testcase | Test Result

Medium:

C++ Auto

```cpp
class Solution {
public:
    int eraseOverlapIntervals(vector<vector<int>>& intervals) {
        sort(intervals.begin(),intervals.end(), [](auto &a, auto &b){
                return a[1] < b[1];
            });
        int ans=0;
        int n = intervals.size();
        int end = intervals[0][1];
        for(int i=1;i<n;i++){
            if(intervals[i][0]<end) ans++;
            else{
                end = intervals[i][1];
            }
        }
        return ans;
    }
};
```

Easy:

Code

C++ Auto

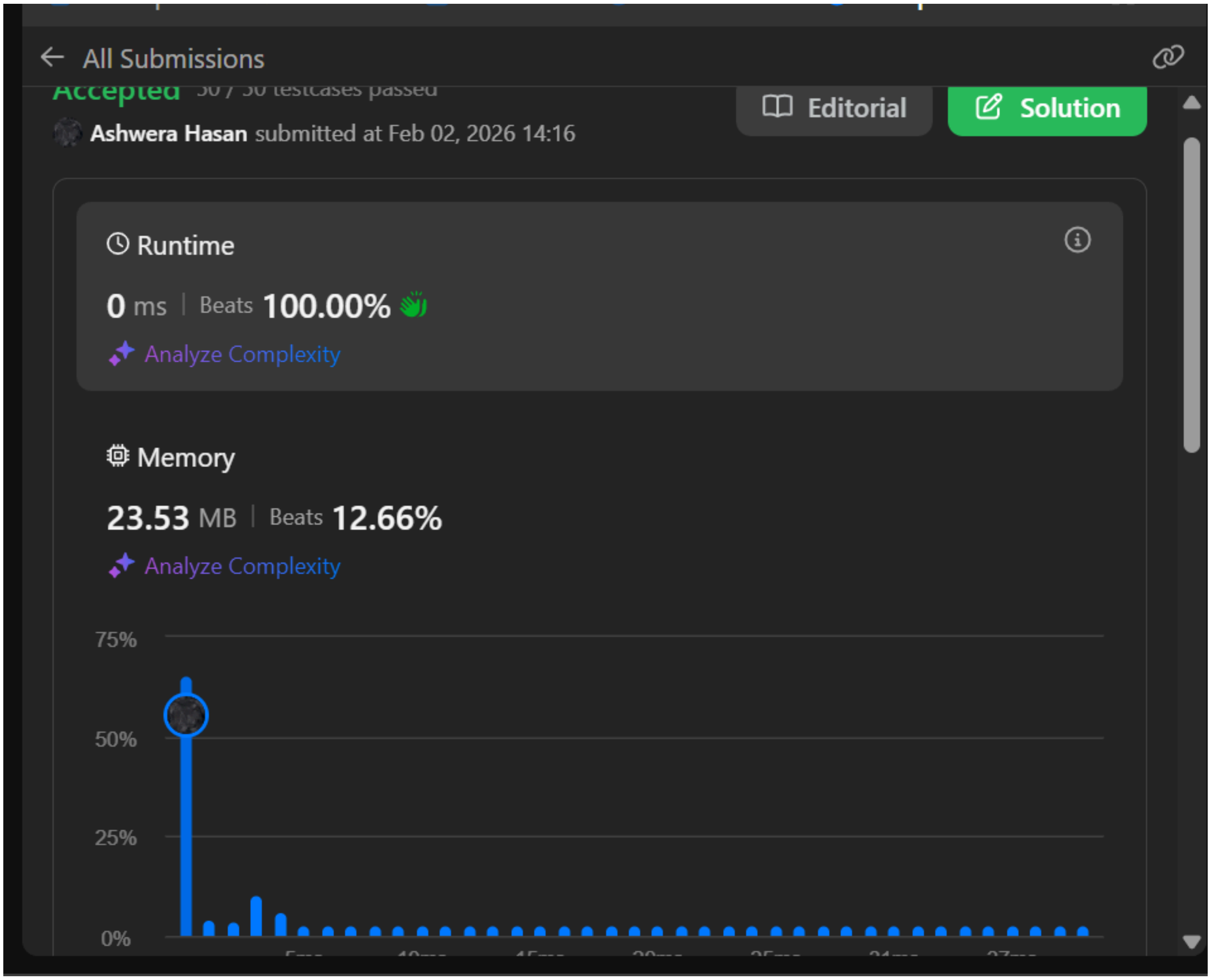
```cpp
class Solution {
public:
    int maximumUnits(vector<vector<int>>& boxTypes, int truckSize) {
        sort(boxTypes.begin(), boxTypes.end(),
            [](const vector<int>& a, const vector<int>& b) {
                return a[1] > b[1];
            });
        int n = boxTypes.size();
        int ans=0;
        for(int i=0;i<n;i++){
            ans += boxTypes[i][0] * boxTypes[i][1];
            truckSize-=boxTypes[i][0];
            while(truckSize<0){
                ans -= boxTypes[i][1];
                truckSize++;
                if(truckSize==0) return ans;
            }
        }
        return ans;
    }
```

Saved Ln 19, Col 20

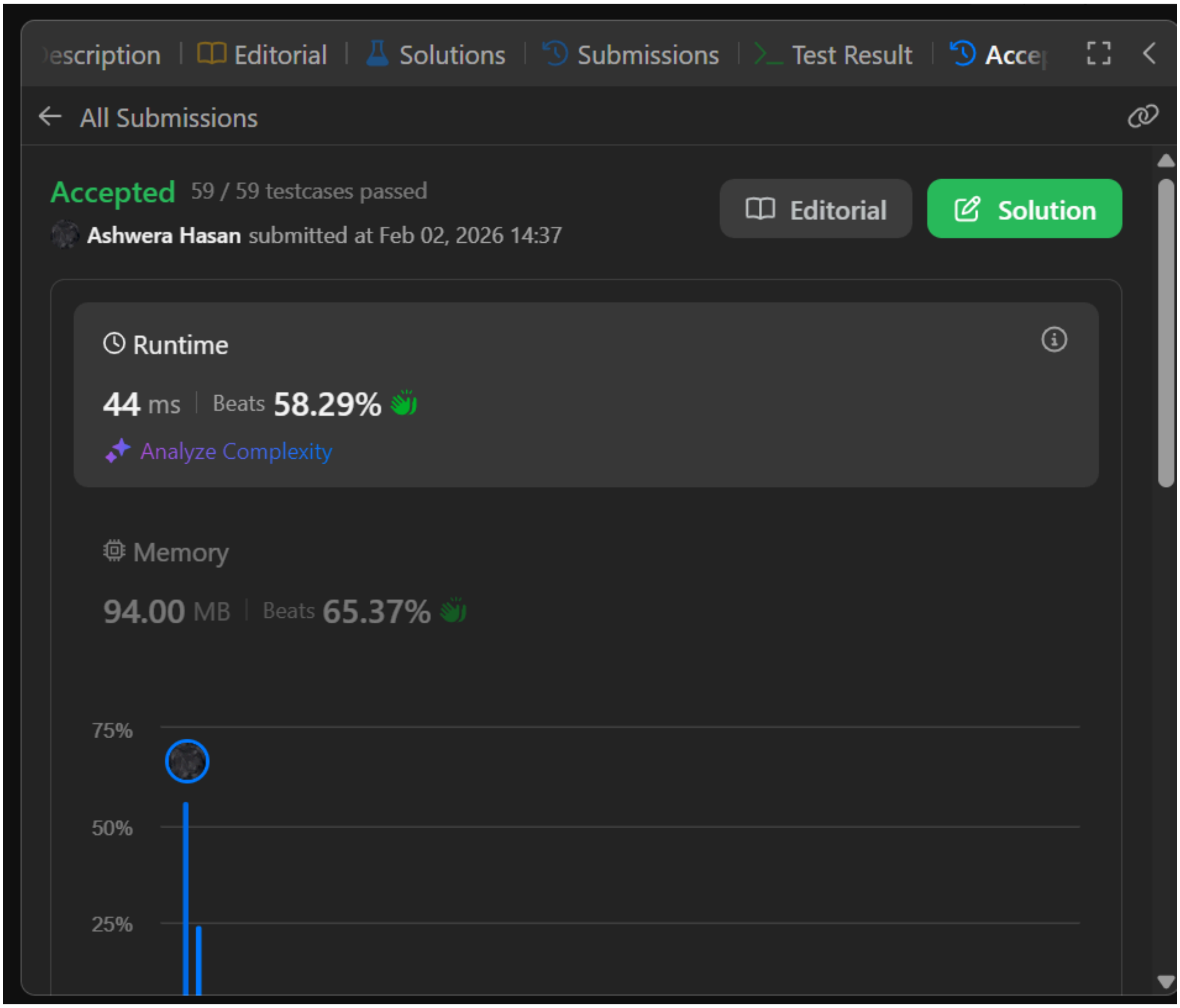
Testcase | Test Result

Output:

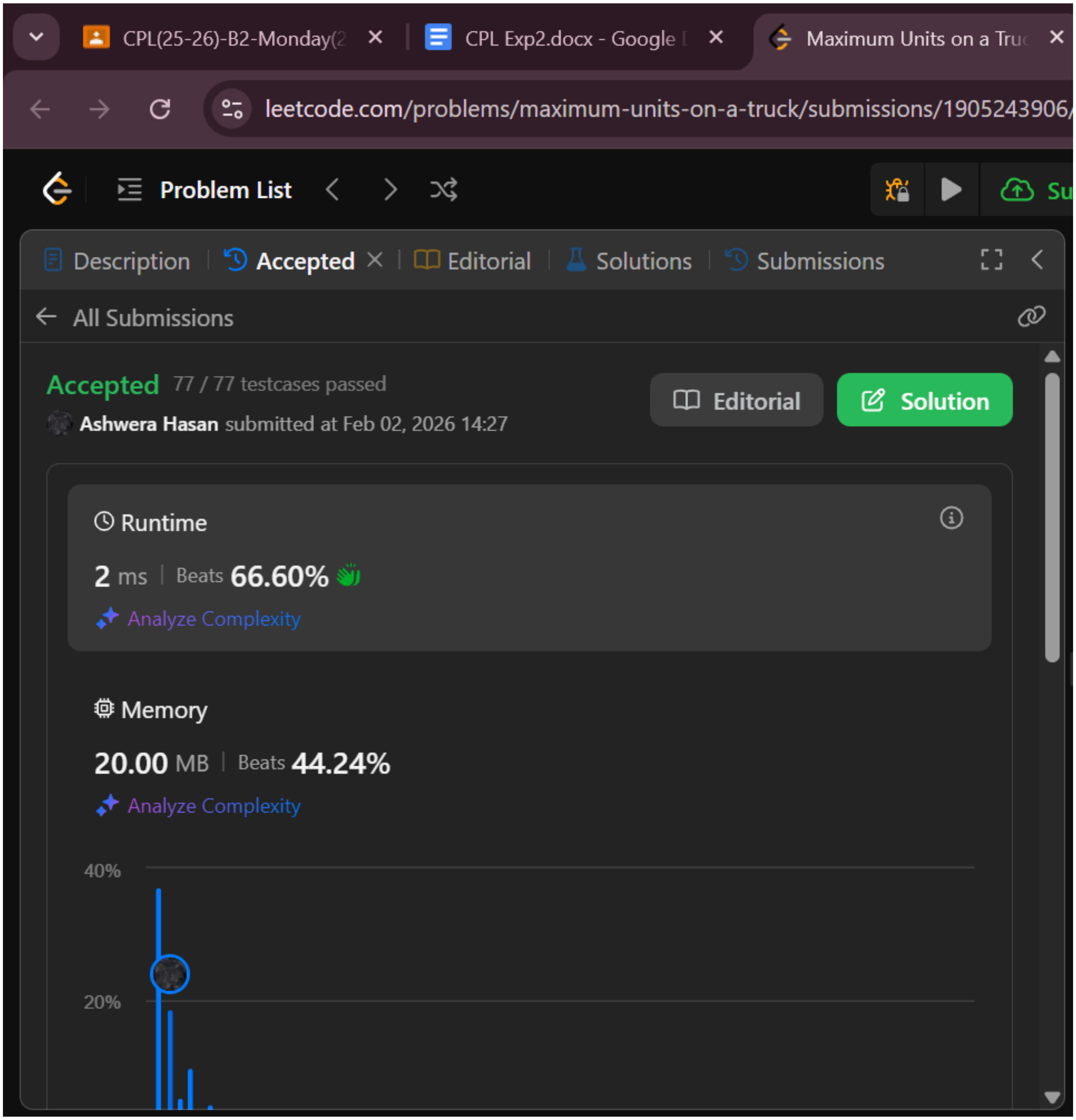
Hard:


All Submissions
Accepted
Editorial
Solution
Ashwera Hasan submitted at Feb 02, 2026 14:16
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
23.53 MB | Beats 12.66%
Analyze Complexity
75%
50%
25%
0%

Medium:


Description
Editorial
Solutions
Submissions
Test Result
All Submissions
Accepted 59 / 59 testcases passed
Ashwera Hasan submitted at Feb 02, 2026 14:37
Editorial
Solution
Runtime
44 ms | Beats 58.29%
Analyze Complexity
Memory
94.00 MB | Beats 65.37%
75%
50%
25%

Easy:



Post Lab Subjective/Objective type Questions:
1. What is the greedy choice property and optimal substructure? Explain why both are important for solving problems using greedy algorithms. The greedy choice property means that a globally optimal solution can be reached by making a locally optimal (greedy) choice at each stage. This property is crucial because it allows the algorithm to focus only on the current situation, leading to a much simpler and

often more efficient design compared to algorithms like dynamic programming, which might explore many possibilities.

2. State one advantage and one limitation of greedy algorithms.
   Advantages: The greedy solution is faster than bruteforce and not very difficult to understand. It helps build intuition.
   Disadvantage: The greedy solution does not necessarily work for all problems.

Conclusion:

The experiment was successful in practically implementing the greedy approaches to problems. How a left and right subset vectorization of problems can help coming to solutions quickly and how sorting using custom comparator can help sort by some specific column/row in 2-d vectors

Signature of faculty in-charge with Date: