Experiment-8

Student Name: Gaurav Saini **UID:** 22BCS11085
Branch: BE-CSE **Section/Group:**NTPP_IOT_603_B
Semester:06 **Date of Performance:**28-03-2025

Subject Name: AP LAB-II **Subject Code:** 22CSP-351

1. **Aim:**
   a. **Max Units on a Truck**
   b. **Min Operations to Make Array Increasing**
   c. **Remove Stones to Maximize Total**

2. **Introduction to `Greedy Algorithm`:**
   Greedy algorithms are algorithms that take the best, immediate, or local, solution while looking for an answer. They identify the globally (overall) optimal solution for certain optimization problems but might identify less-than-optimal solutions for certain instances of other problems. It follows the problem-solving heuristic of making the locally optimal choice at every stage. They are known as greedy algorithms because even though finding an optimal solution for every smaller instance will give you an instant output, the algorithm does not take the larger problem into account. It never reconsiders any decisions after they are made.

3. Implementation/Code:

A. Max Units on a Truck

```
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 maxUnits = 0;
        for (const auto& box : boxTypes) {
            int boxCount = min(truckSize, box[0]);
            maxUnits += boxCount * box[1];
            truckSize -= boxCount;

            if (truckSize == 0) break;
        }

        return maxUnits;
    }
};    }
};
```

B. Min Operations to Make Array Increasing

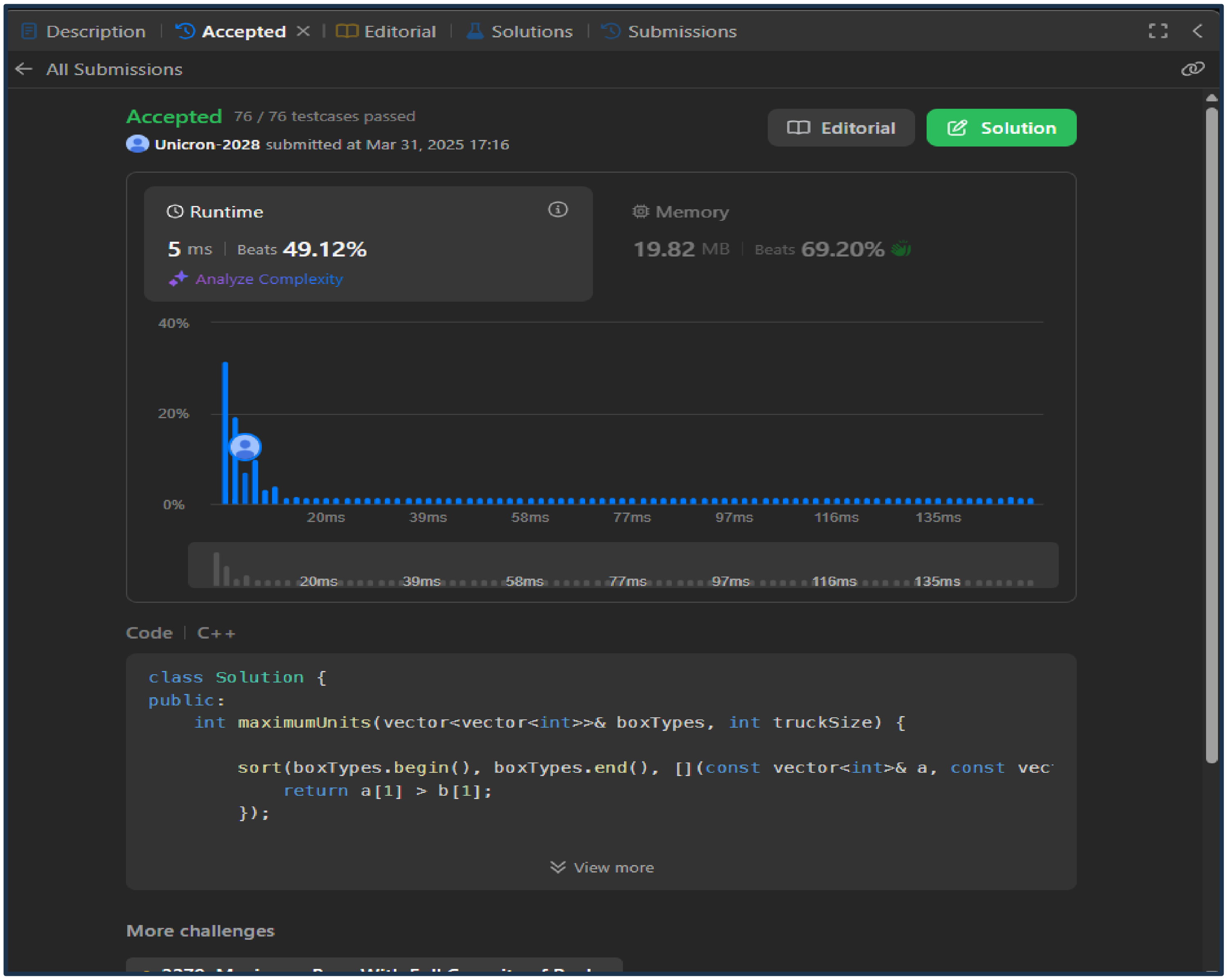
```
class Solution {
public:
    int minOperations(vector<int>& nums) {
        int operations = 0;
        for (int i = 1; i < nums.size(); i++) {
            if (nums[i] <= nums[i - 1]) {
                operations += (nums[i - 1] - nums[i] + 1);
                nums[i] = nums[i - 1] + 1;
            }
        }
        return operations;
    }
};
```

C. Remove Stones to Maximize Total

```
#include <vector>
#include <queue>
using namespace std;

class Solution {
public:
    int minStoneSum(vector<int>& piles, int k) {
        priority_queue<int> maxHeap(piles.begin(), piles.end()); // Max heap to get
the largest pile

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

        int total = 0;
        while (!maxHeap.empty()) {
            total += maxHeap.top();
            maxHeap.pop();
        }

        return total;
    }
};
```

4. Output

A. Max Units on a Truck



Accepted 76 / 76 testcases passed

Unicron-2028 submitted at Mar 31, 2025 17:16

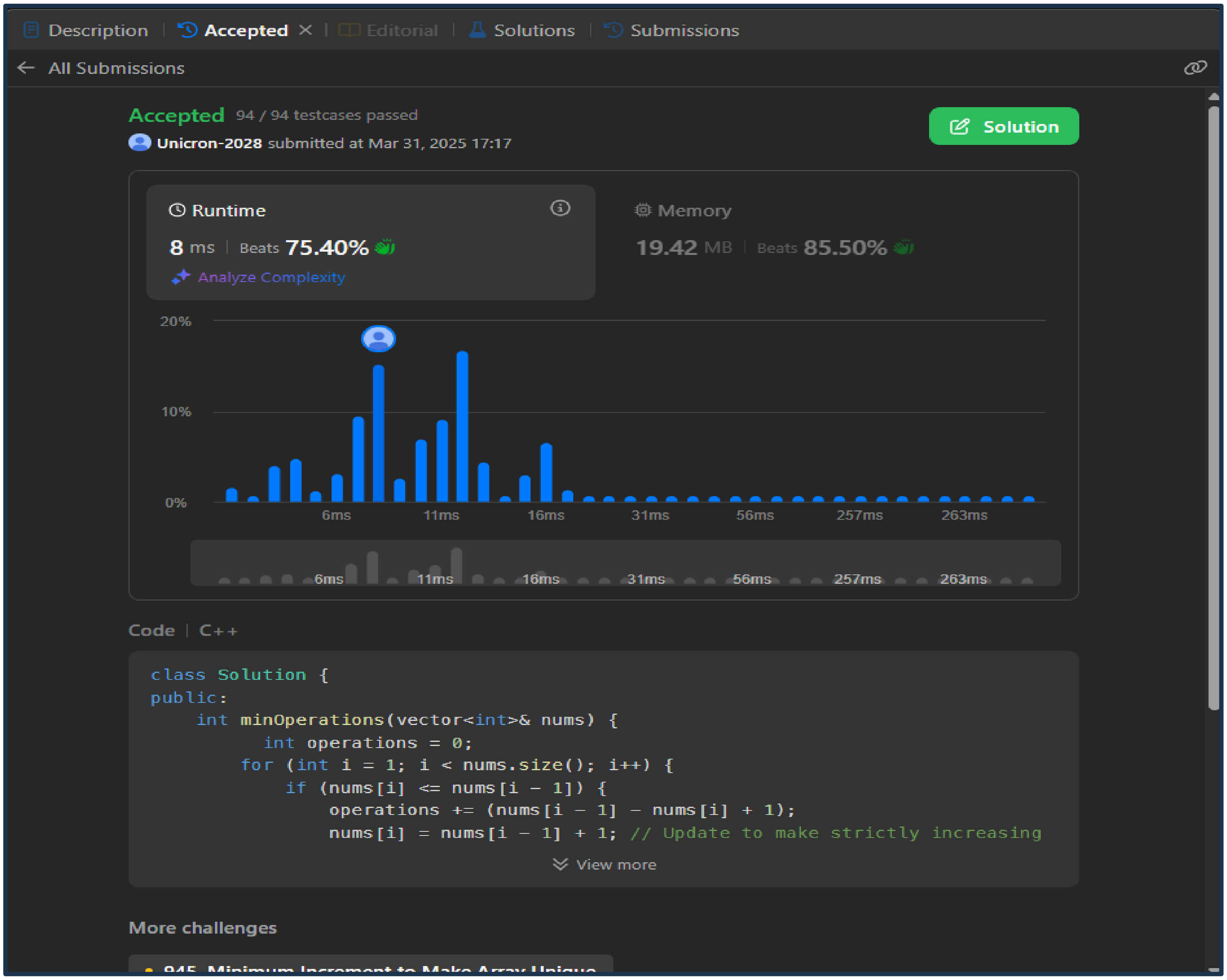
Runtime: 5 ms | Beats 49.12%

Memory: 19.82 MB | Beats 69.20%

Code | C++

```
class Solution {
public:
    int maximumUnits(vector<vector<int>>& boxTypes, int truckSize) {

        sort(boxTypes.begin(), boxTypes.end(), [](const vector<int>& a, const vec
            return a[1] > b[1];
        });
```

B. Min Operations to Make Array Increasing



Accepted 94 / 94 testcases passed

Unicron-2028 submitted at Mar 31, 2025 17:17

8 ms | Beats 75.40%

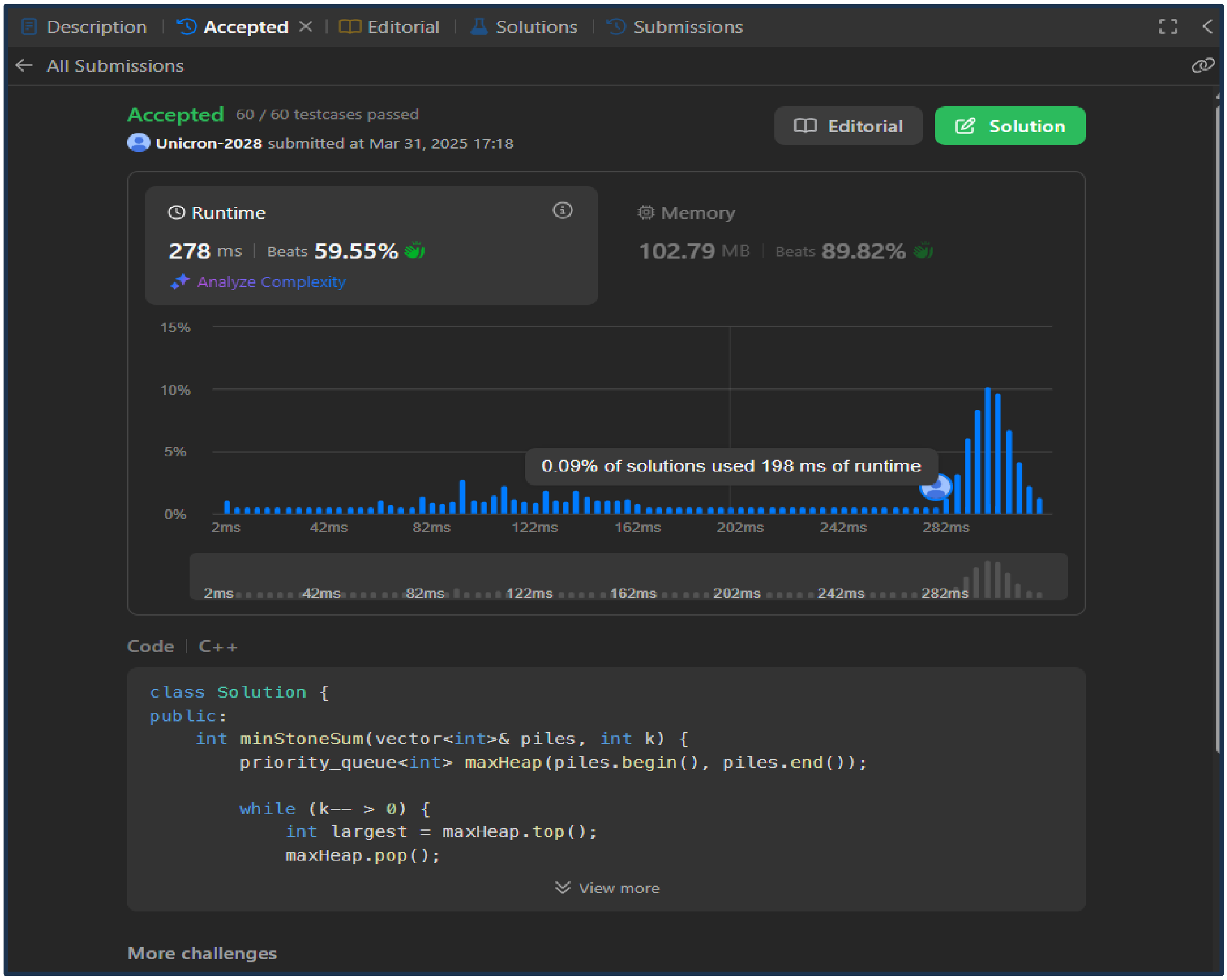
19.42 MB | Beats 85.50%

Code | C++

```
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]) {
                operations += (nums[i - 1] - nums[i] + 1);
                nums[i] = nums[i - 1] + 1; // Update to make strictly increasing
```

More challenges

C. Remove Stones to Maximize Total



Accepted 60 / 60 testcases passed

Unicron-2028 submitted at Mar 31, 2025 17:18

Runtime

278 ms | Beats 59.55%

Memory

102.79 MB | Beats 89.82%

0.09% of solutions used 198 ms of runtime

Code | C++

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

        while (k-- > 0) {
            int largest = maxHeap.top();
            maxHeap.pop();
```

5. Learning Outcomes:

 Understanding Heaps (Priority Queues)
- Learn how to use a max heap to efficiently extract and modify the largest element.
- Understand the use of priority_queue in C++ for optimization.

 Greedy Algorithm Approach
- Grasp the greedy strategy of always reducing the largest pile to minimize the total sum.

 Time Complexity Analysis
- Learn how heap operations like insertion and extraction work in O(log n)O(\log n)O(logn) time.
- Understand how the algorithm runs efficiently even for large constraints.

 Practical Application of Data Structures
- Experience how heaps optimize real-world problems involving repeated selection of the maximum element.