“**Experiment1.3”**

**Aim:**

To demonstrate the concept of [Heap model](https://lms.cuchd.in/mod/url/view.php?id=880092).

**Objective:**

• The objective is to build problem solving capability and to learn the basic concepts of data structures.

• The implementation of Last stone weight problem brushes up the concept of array and heap.

• The implementation of Repeated String Match problem brushes up the concept of strings and string matching.

**Problem 1: “ Last stone weight”**

<https://leetcode.com/problems/last-stone-weight/>

You are given an array of integers stones where stones[i] is the weight of the ith stone.

We are playing a game with the stones. On each turn, we choose the heaviest two stones and smash them together. Suppose the heaviest two stones have weights x and y with x <= y. The result of this smash is:

If x == y, both stones are destroyed, and

If x != y, the stone of weight x is destroyed, and the stone of weight y has new weight y - x.

At the end of the game, there is at most one stone left.

Return the weight of the last remaining stone. If there are no stones left, return 0.

**Code:**

class Solution {

public:

    int lastStoneWeight(vector<int>& stones) {

        priority\_queue<int> pq;            // SC: O(n)

        for(int i=0;i<stones.size();i++) pq.push(stones[i]);

        while(pq.size()>1){                 // TC: O(n)

            int x= pq.top(); pq.pop();

            int y= pq.top(); pq.pop();

            if(x>y) pq.push(x-y);

        }

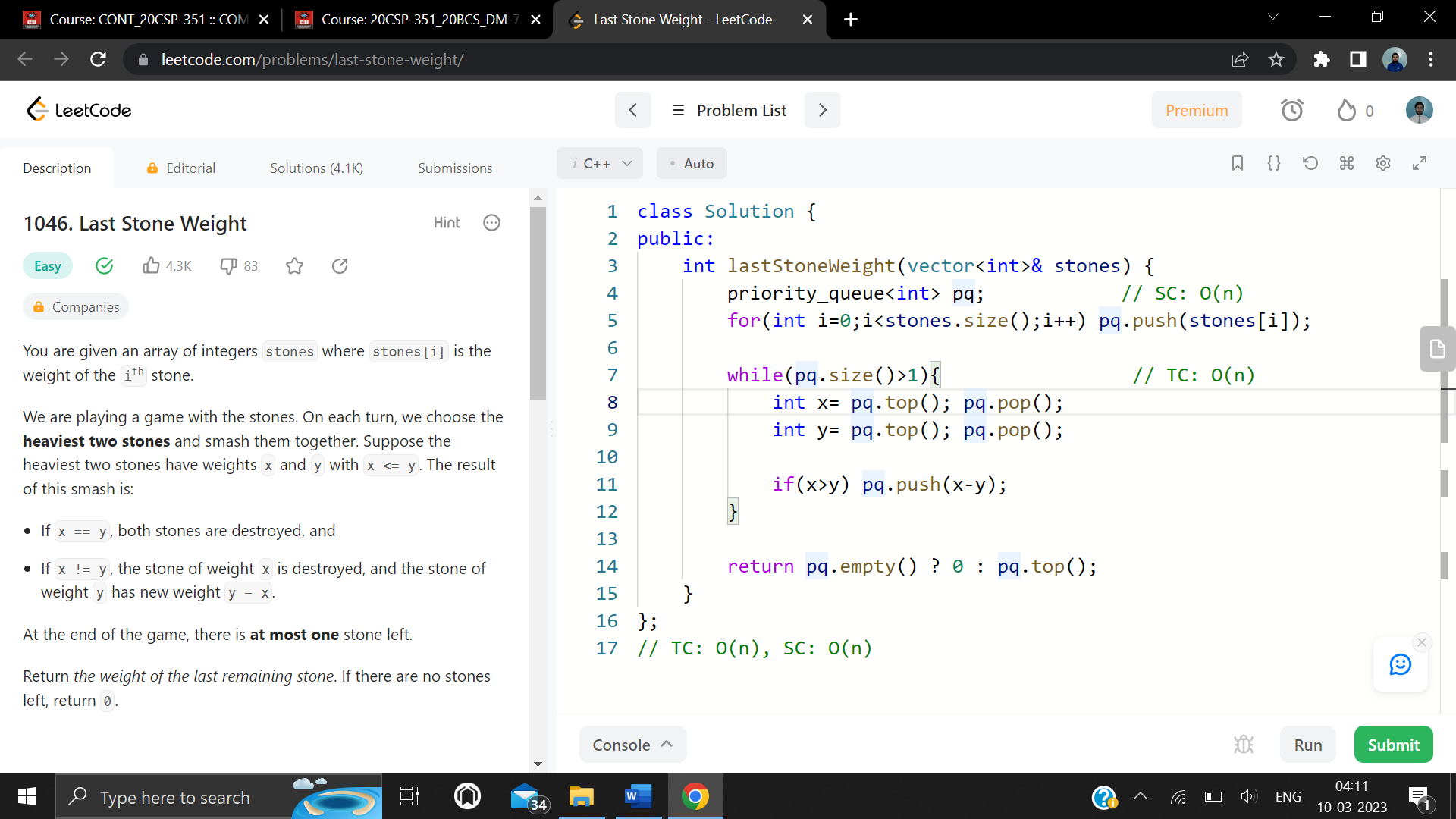
        return pq.empty() ? 0 : pq.top();

    }

};

// TC: O(n), SC: O(n)

**Output:**



**Problem 2: “Cheapest flights with k shops”**

<https://leetcode.com/problems/cheapest-flights-within-k-stops/>

There are n cities connected by some number of flights. You are given an array flights where flights[i] = [fromtoi, pricei] indicates that there is a flight from city fromi to city toi with cost pricei..

You are also given three integers src, dst, and k, return the cheapest price from src to dst with at most k stops. If there is no such route, return -1.

**Code:**

class Solution {

public:

    int findCheapestPrice(int n, vector<vector<int>>& flights, int src, int dst, int k) {

        vector<vector<pair<int, int>>> adj(n);

        for (auto e : flights) {

            adj[e[0]].push\_back({e[1], e[2]});

        }

        vector<int> stops(n, numeric\_limits<int>::max());

        priority\_queue<vector<int>, vector<vector<int>>, greater<vector<int>>> pq;

        // {dist\_from\_src\_node, node, number\_of\_stops\_from\_src\_node}

        pq.push({0, src, 0});

        while (!pq.empty()) {

            auto temp = pq.top();

            pq.pop();

            int dist = temp[0];

            int node = temp[1];

            int steps = temp[2];

            // We have already encountered a path with a lower cost and fewer stops,

            // or the number of stops exceeds the limit.

            if (steps > stops[node] || steps > k + 1) continue;

            stops[node] = steps;

            if (node == dst) return dist;

            for (auto& [neighbor, price] : adj[node]) {

                pq.push({dist + price, neighbor, steps + 1});

            }

        }

        return -1;

    }

};

**Output:**

