Experiment – 1.3

Student Name: Subhadip Patra **UID: 20BCS2543**

Branch: CSE **Section/Group: 707/B**

Subject Name: Competitive Coding II **Subject Code: 20CSP-351**

1. Aim: Last Stone Weight

2. Objective:

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.

Example 1:

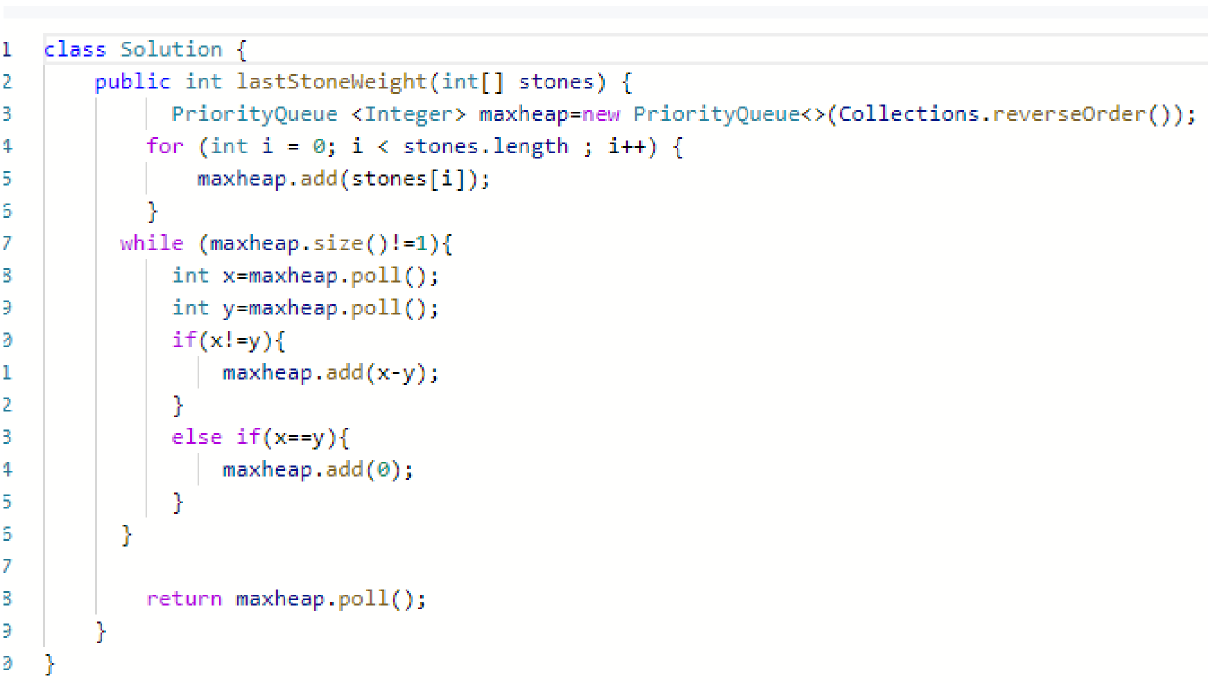
Input: stones = [2,7,4,1,8,1]
Output: 1

3. Code:

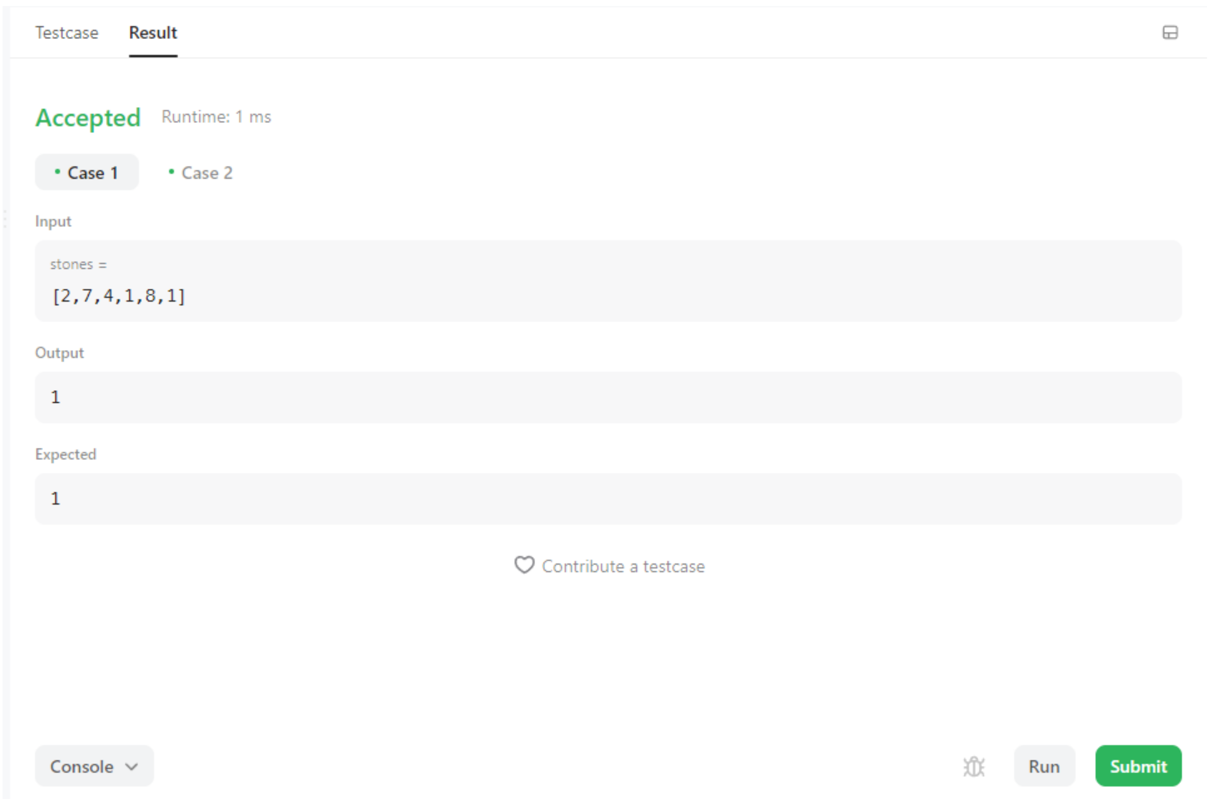
```
class Solution {
    public int lastStoneWeight(int[] stones) {
          PriorityQueue <Integer> maxheap=new
PriorityQueue<>(Collections.reverseOrder());
        for (int i = 0; i < stones.length ; i++) {
```

```
maxheap.add(stones[i]);
        }
      while (maxheap.size()!=1){
          int x=maxheap.poll();
          int y=maxheap.poll();
          if(x!=y){
              maxheap.add(x-y);
          }
          else if(x==y){
              maxheap.add(0);
          }
      }

        return maxheap.poll();
    }
}
```

4. Output:



```
class Solution {
    public int lastStoneWeight(int[] stones) {
        PriorityQueue <Integer> maxheap=new PriorityQueue<>(Collections.reverseOrder());
      for (int i = 0; i < stones.length ; i++) {
          maxheap.add(stones[i]);
      }
     while (maxheap.size()!=1){
        int x=maxheap.poll();
        int y=maxheap.poll();
        if(x!=y){
            maxheap.add(x-y);
        }
        else if(x==y){
            maxheap.add(0);
        }
     }

      return maxheap.poll();
    }
}
```



Accepted Runtime: 1 ms

Input

```
stones =
[2,7,4,1,8,1]
```

Output

```
1
```

Expected

```
1
```

5. Aim: Cheapest Flights Within K Stops

6. Objective:

There are n cities connected by some number of flights. You are given an array flights where flights[i] = [fromi, toi, 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.

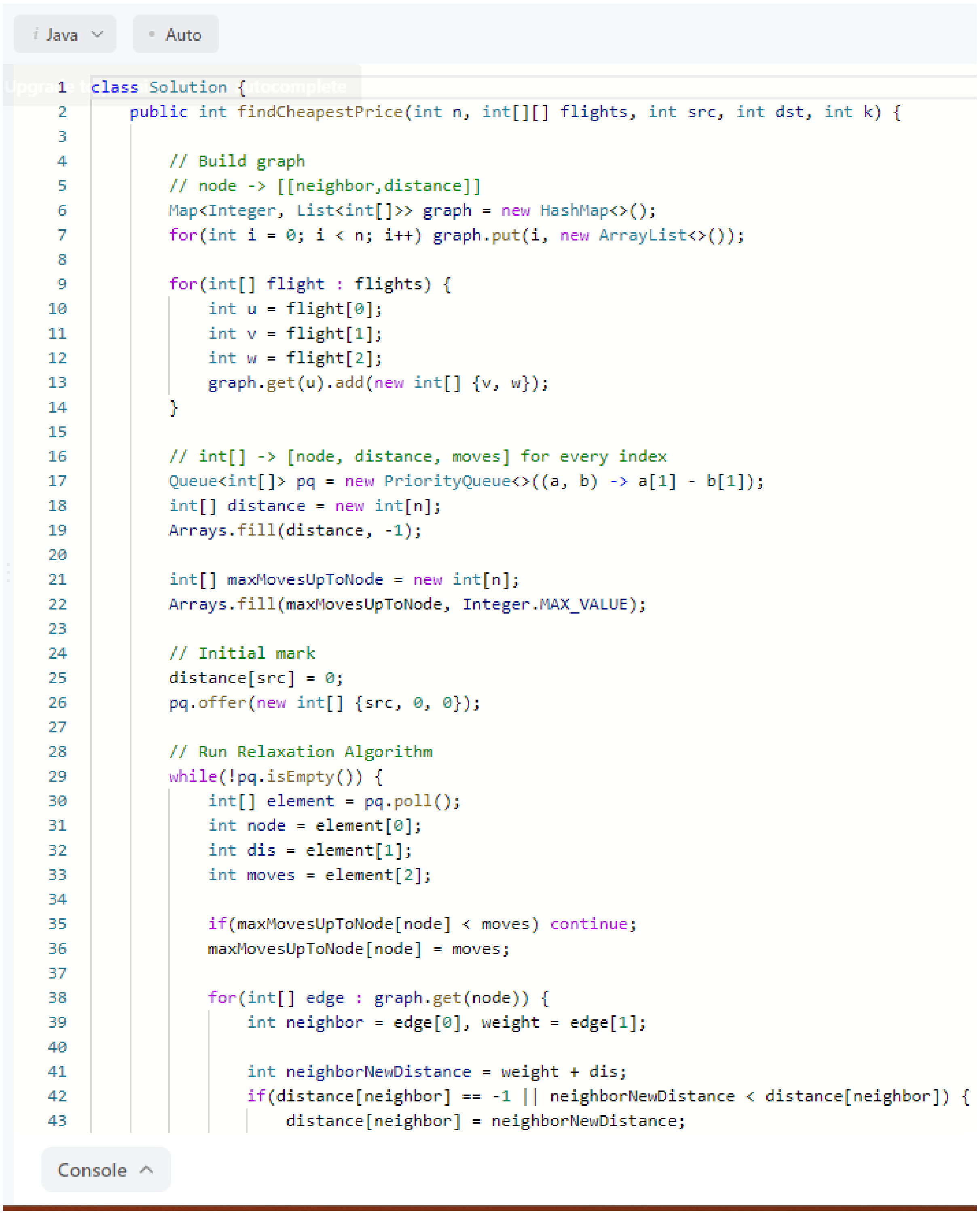
Input: n = 4, flights = [[0,1,100],[1,2,100],[2,0,100],[1,3,600],[2,3,200]], src = 0, dst = 3, k = 1
Output: 700

7. Code:

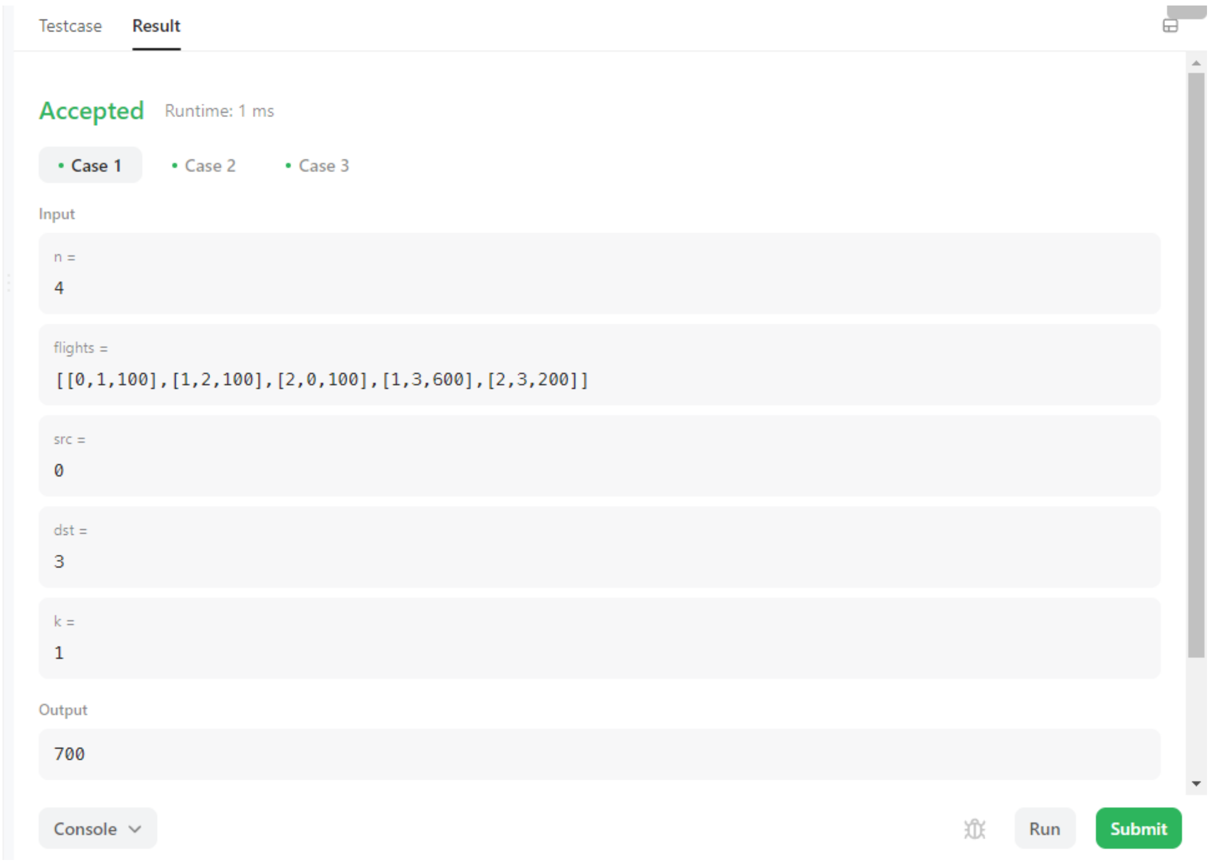
```
class Solution {
  public int findCheapestPrice(int n, int[][] flights, int src, int dst, int k) {

    // Build graph
    // node -> [[neighbor,distance]]
    Map<Integer, List<int[]>> graph = new HashMap<>();
    for(int i = 0; i < n; i++) graph.put(i, new ArrayList<>());

    for(int[] flight : flights) {
      int u = flight[0];
      int v = flight[1];
      int w = flight[2];
      graph.get(u).add(new int[] {v, w});
    }

    // int[] -> [node, distance, moves] for every index
    Queue<int[]> pq = new PriorityQueue<>((a, b) -> a[1] - b[1]);
    int[] distance = new int[n];
    Arrays.fill(distance, -1);

    int[] maxMovesUpToNode = new int[n];
    Arrays.fill(maxMovesUpToNode, Integer.MAX_VALUE);
```

```
        distance[src] = 0;
        pq.offer(new int[] {src, 0, 0});

        while(!pq.isEmpty()) {
            int[] element = pq.poll();
            int node = element[0];
            int dis = element[1];
            int moves = element[2];

            if(maxMovesUpToNode[node] < moves) continue;
            maxMovesUpToNode[node] = moves;

            for(int[] edge : graph.get(node)) {
                int neighbor = edge[0], weight = edge[1];

                int neighborNewDistance = weight + dis;
                if(distance[neighbor] == -1 || neighborNewDistance <
distance[neighbor]) {
                    distance[neighbor] = neighborNewDistance;
                }

                if(k != moves) {
                    pq.offer(new int[] {neighbor, neighborNewDistance, moves
+ 1});
                }
            }
        }


        return distance[dst];
    }
}
```

8. Output:



```java
class Solution {
    public int findCheapestPrice(int n, int[][] flights, int src, int dst, int k) {

        // Build graph
        // node -> [[neighbor,distance]]
        Map<Integer, List<int[]>> graph = new HashMap<>();
        for(int i = 0; i < n; i++) graph.put(i, new ArrayList<>());

        for(int[] flight : flights) {
            int u = flight[0];
            int v = flight[1];
            int w = flight[2];
            graph.get(u).add(new int[] {v, w});
        }

        // int[] -> [node, distance, moves] for every index
        Queue<int[]> pq = new PriorityQueue<>((a, b) -> a[1] - b[1]);
        int[] distance = new int[n];
        Arrays.fill(distance, -1);

        int[] maxMovesUpToNode = new int[n];
        Arrays.fill(maxMovesUpToNode, Integer.MAX_VALUE);

        // Initial mark
        distance[src] = 0;
        pq.offer(new int[] {src, 0, 0});

        // Run Relaxation Algorithm
        while(!pq.isEmpty()) {
            int[] element = pq.poll();
            int node = element[0];
            int dis = element[1];
            int moves = element[2];

            if(maxMovesUpToNode[node] < moves) continue;
            maxMovesUpToNode[node] = moves;

            for(int[] edge : graph.get(node)) {
                int neighbor = edge[0], weight = edge[1];

                int neighborNewDistance = weight + dis;
                if(distance[neighbor] == -1 || neighborNewDistance < distance[neighbor]) {
                    distance[neighbor] = neighborNewDistance;
```



Accepted Runtime: 1 ms

Input

```
n =
4
flights =
[[0,1,100],[1,2,100],[2,0,100],[1,3,600],[2,3,200]]
src =
0
dst =
3
k =
1
```

Output

```
700
```