Experiment-1.3

Student Name: Aayush Gurung **UID:20BCS5323**
Branch: CSE **Section/Group:DM_607(A)**
Semester: 6 **Date of Performance:24-02-2023**
Subject Name: CC LAB **Subject Code: 20CSP-351**

Aim:

To demonstrate the concept of Heap model

Objective

Problem 1:
Last Stone Weight

Approach:
We'll create a priority queue and push all the elements in the priority queue and then run a loop and remove the largest two elements from the queue and if they are equal then we'll continue otherwise we'll push the absolute difference of both the elements in the queue and after the loop stops, if the queue is empty then we'll return 0 otherwise the last element in the queue.

Algorithm:

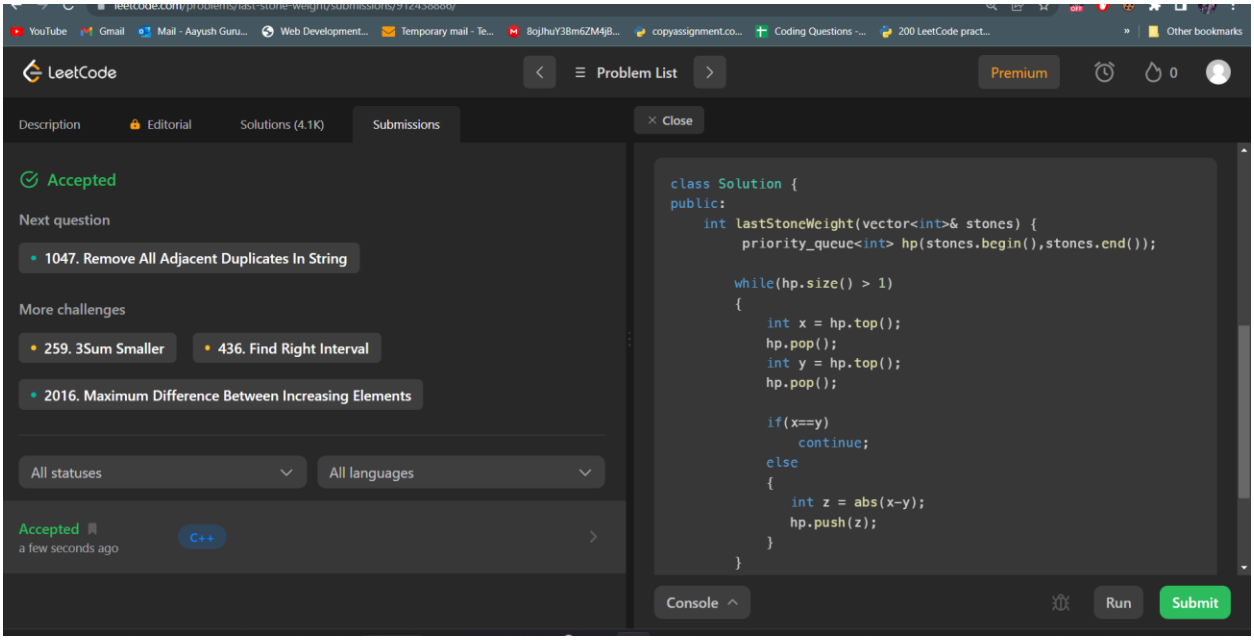
1. Declare a priority queue hp and initialize it with the given array.
2. Now start a loop with a condition that hp.size()>1.
3. Now declare a variable x and put top of hp in that and pop hp and do similar thing with another variable y.
4. If x == y then continue otherwise push abs(x-y) in the hp
5. After the loop ends return 0 if hp is empty else return the last element in the hp.

Code:

```
class Solution {
public:
    int lastStoneWeight(vector<int>& stones) {
         priority_queue<int> hp(stones.begin(),stones.end());

        while(hp.size() > 1)
        {
            int x = hp.top();
            hp.pop();
```

```
            int y = hp.top();
            hp.pop();

            if(x==y)
                continue;
            else
            {
               int z = abs(x-y);
               hp.push(z);
            }
        }
        if(hp.empty()) return 0;
        return hp.top();

    }
};
```

Output:



Problem 2:

Cheapest Flights within K stops

Approach:
Breadth first search

Algorithm:

- maintain a adj list with src -> {dst,cost}
- make a queue for pending node(pn)
- push stops,node,cost
- a distance vector & dist of src = 0
- perform normal BFS & check if adj[node] dist can be change within stops <=k
- if yes then update dist[currnode] & push to queue {stops+1,{currnode,updatedcost}}
- return dist[dst]

Code:

```
class Solution {
public:
    int findCheapestPrice(int n, vector<vector<int>>& flights, int src, int dst, int
k) {
        vector<pair<int,int>> adj[n];
        for(auto it : flights){
            adj[it[0]].push_back({it[1],it[2]});
        }
        queue<pair<int,pair<int,int>>> pn;
        pn.push({0,{src,0}});
        vector<int> dist(n,1e9);
        dist[src] = 0;
        while(!pn.empty()){
            auto front = pn.front();
            pn.pop();
            int stops = front.first;
            int node = front.second.first;
            int distance = front.second.second;
            if(stops>k)continue;
            for(auto it:adj[node]){
                int adjnode = it.first;
                int d = it.second;
                if(distance + d<dist[adjnode]&&stops<=k){
                    dist[adjnode] = distance + d;
                    pn.push({stops+1,{adjnode,distance+d}});
                }
```

```
            }
        }
        if(dist[dst]==1e9)return -1;
        return dist[dst];

    }
};
```

Output: