Experiment 8

Student Name: Arun **UID:** 22BET10320
Branch: Information Technology **Section/Group:** 22BET_IOT-701/A
Semester: 6th **Subject Code:** 22ITP-351

Problem: 1

Aim: Maximum Units on a Truck

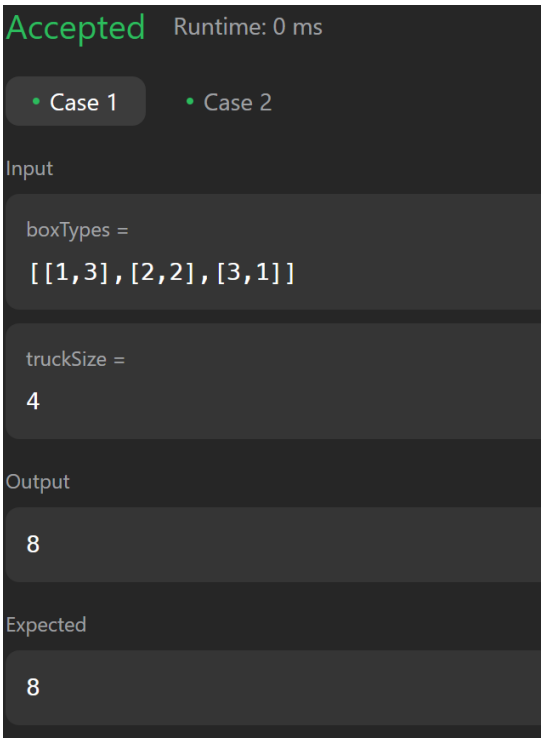
Code:

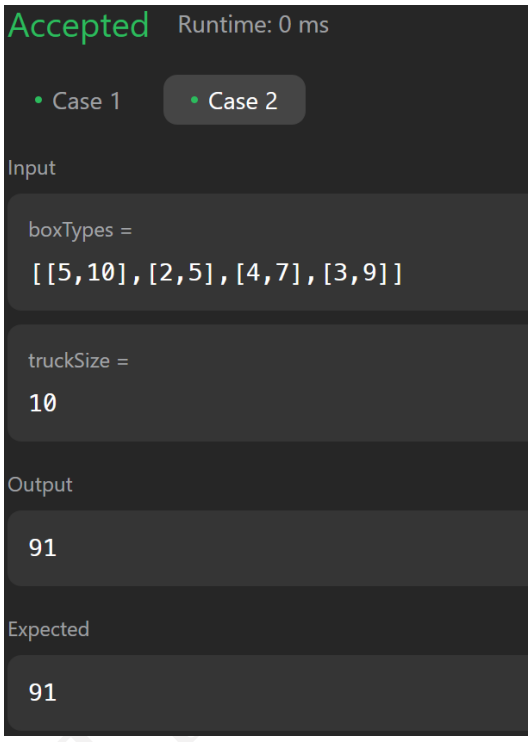
```
class Solution {
public:
   int maximumUnits(vector<vector<int>>& boxTypes, int truckSize) {
      sort(boxTypes.begin(),boxTypes.end(),[](const vector<int>& a,const vector<int>& b){
         return a[1] > b[1];
      });
      int mx = 0;
      for(auto i:boxTypes)
      {
         /*if(truckSize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

          if (truckSize == 0) break;


         int boxesToTake = min(i[0], truckSize);
         mx += boxesToTake * i[1];
         truckSize -= boxesToTake;
      }
      return mx;
   }
};
```

Output:





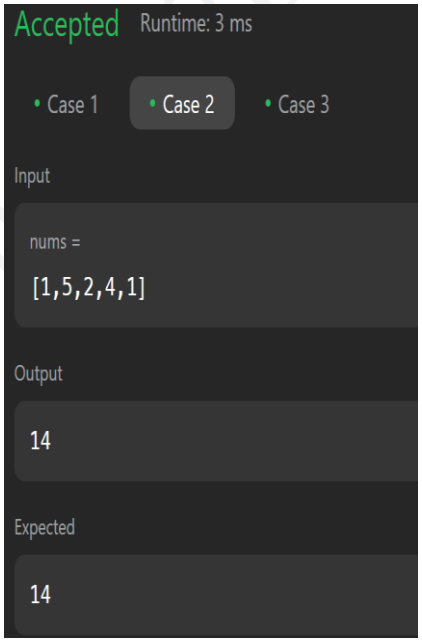
Problem: 2

Aim: Minimum Operations to Make the Array Increasing

Code:

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

Output:



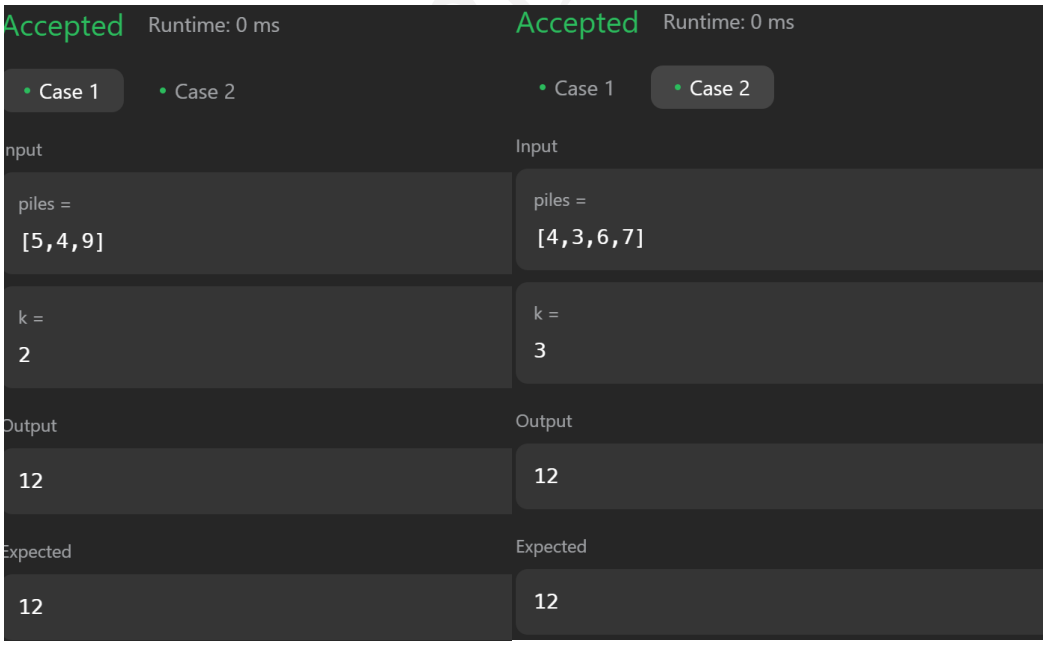
Problem: 3

Aim: Remove Stones to Minimize the Total

Code:

```
class Solution {
public:
   int minStoneSum(vector<int>& piles, int k) {
        priority_queue<int>pq(piles.begin(),piles.end()); //will copy the vector to the priority queue
       int ans=0;
       for(int i=0;i<k;i++){
          int tp=pq.top(); //top element will always be the largest element
          pq.pop();
          tp-=(tp/2);
          pq.push(tp);
       }
       while(!pq.empty()){
          ans+=pq.top();   //adding the left stones, after k operations
          pq.pop();
       }
       return ans;
   }
};
```

Output:



Problem: 4

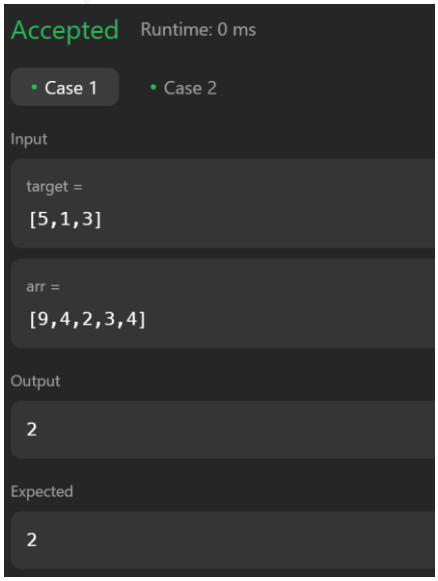
Aim: Minimum Operations to Make a Subsequence

Code:

```
class Solution {
public:
    int minOperations(vector<int>& target, vector<int>& A) {
    unordered_map<int, int> h;
    int n = A.size();
    for (int i = 0; i < target.size(); ++i)
      h[target[i]] = i;

    vector<int> stack;
    for (int a : A) {
      if (h.find(a) == h.end()) continue;
      if (stack.empty() || h[a] > stack.back()) {
        stack.push_back(h[a]);
        continue;
      }
      int left = 0, right = stack.size() - 1, mid;
      while (left < right) {
        mid = (left + right) / 2;
        if (stack[mid] < h[a])
          left = mid + 1;
        else
          right = mid;
      }
      stack[left] = h[a];
    }
    return target.size() - stack.size();
  }
};
```

Output:



Problem: 5

Aim: Maximum Number of Tasks You Can Assign

Code:

```
class Solution {
public:
    int maxTaskAssign(vector<int>& tasks, vector<int>& workers, int p, int strength) {
        int n = tasks.size(), m = workers.size();

        // Sorting the tasks and workers in increasing order
        sort(tasks.begin(), tasks.end());
        sort(workers.begin(), workers.end());
        int lo = 0, hi = min(m, n);
        int ans;

        while(lo <= hi) {
            int mid = lo + (hi - lo) / 2;
            int count = 0;
            bool flag = true;

            // Inserting all workers in a multiset
            multiset<int> st(workers.begin(), workers.end());

            // Checking if the mid smallest tasks can be assigned
            for(int i = mid - 1; i >= 0; i--) {

                // Case 1: Trying to assing to a worker without the pill
                auto it = prev(st.end());
                if(tasks[i] <= *it) {

                    // Case 1 satisfied!
                    st.erase(it);
                } else {

                    // Case 2: Trying to assign to a worker with the pill
                    auto it = st.lower_bound(tasks[i] - strength);
                    if(it != st.end()) {

                        // Case 2 satisfied!
                        count++;
                        st.erase(it);
                    } else {

                        // Case 3: Impossible to assign mid tasks
                        flag = false;
                        break;
                    }
                }
```

```
            // If at any moment, the number of pills require for mid tasks exceeds
            // the allotted number of pills, we stop the loop
            if(count > p) {
                flag = false;
                break;
            }
        }

        if(flag) {
            ans = mid;
            lo = mid + 1;
        } else {
            hi = mid - 1;
        }
    }
    return ans;
  }
};
```

Output: