Discover. Learn. Empower. ENGINEERING

WORKSHEET-8

Student Name: Arpit Srivastava **UID:** 22BCS10378

Branch: CSE **Section/Group:** NTPP-603-B

Semester: 6th **Date of Performance:** 20/3/25

Subject Name: AP-2 **Subject Code:** 22CSP-351

: You are assigned to put some amount of boxes onto one truck. You are

1710. Maximum Units on a Truck

Easy | Topics | Companies | Hint

You are assigned to put some amount of boxes onto **one truck**. You are given a 2D array `boxTypes`, where `boxTypes[i] = [numberOfBoxes_i, numberOfUnitsPerBox_i]`:

- `numberOfBoxes_i` is the number of boxes of type `i`.
- `numberOfUnitsPerBox_i` is the number of units in each box of the type `i`.

You are also given an integer `truckSize`, which is the **maximum** number of **boxes** that can be put on the truck. You can choose any boxes to put on the truck as long as the number of boxes does not exceed `truckSize`.

Return *the **maximum** total number of **units** that can be put on the truck.*

Example 1:

```
Input: boxTypes = [[1,3],[2,2],[3,1]], truckSize = 4
Output: 8
Explanation: There are:
- 1 box of the first type that contains 3 units.
- 2 boxes of the second type that contain 2 units each.
- 3 boxes of the third type that contain 1 unit each.
You can take all the boxes of the first and second types, and one box of the third type.
The total number of units will be = (1 * 3) + (2 * 2) + (1 * 1) = 8.
```

Code

C++ Auto

```cpp
        int ans=0;
        for(auto box: boxTypes){
            int x=min(box[0],truckSize);  //choose minimum boxes from available boxes and capacity left
            ans+=(x*box[1]);  //adding units in ans
            truckSize-=x;  //reduce the capacity
            if(!truckSize) break;  //capacity full
        }
        return ans;
    }
};
```

space_remaining_boxes -= buckets[i];

Saved Ln 20, Col 1

Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

boxTypes =

[[1,3],[2,2],[3,1]]

truckSize =

4

Output

8

Expected

• Case 1 • Case 2

Input

boxTypes =

[[5,10],[2,5],[4,7],[3,9]]

truckSize =

10

Output

91

1962. Remove Stones to Minimize the Total

Medium | Topics | Companies | Hint

You are given a **0-indexed** integer array `piles`, where `piles[i]` represents the number of stones in the i^{th} pile, and an integer `k`. You should apply the following operation **exactly** `k` times:

- Choose any `piles[i]` and **remove** `floor(piles[i] / 2)` stones from it.

Notice that you can apply the operation on the **same** pile more than once.

Return *the **minimum** possible total number of stones remaining after applying the* `k` *operations*.

`floor(x)` is the **greatest** integer that is **smaller** than or **equal** to `x` (i.e., rounds `x` down).

Example 1:

```
Input: piles = [5,4,9], k = 2
Output: 12
Explanation: Steps of a possible scenario are:
- Apply the operation on pile 2. The resulting piles are [5,4,5].
- Apply the operation on pile 0. The resulting piles are [3,4,5].
The total number of stones in [3,4,5] is 12.
```

Example 2:

```
Input: piles = [4,3,6,7], k = 3
Output: 12
Explanation: Steps of a possible scenario are:
- Apply the operation on pile 2. The resulting piles are [4,3,3,7].
- Apply the operation on pile 3. The resulting piles are [4,3,3,4].
- Apply the operation on pile 0. The resulting piles are [2,3,3,4].
The total number of stones in [2,3,3,4] is 12.
```

Constraints:

Code

C++ Auto

```cpp
class Solution {
public:
    int minStoneSum(vector<int>& piles, int k) {
        priority_queue<int>maxHeap;
        //sare piles ki value ko heap mein daalo
        for(int i = 0;i<piles.size();i++){
            maxHeap.push(piles[i]);
        }
        while(k--){
            int maxElement = maxHeap.top();
            maxHeap.pop();
            maxElement = maxElement - floor(maxElement/2);//operation to be performed
            maxHeap.push(maxElement);
        }
        int sum = 0;
        while(!maxHeap.empty()){
            sum += maxHeap.top();
            maxHeap.pop();
        }
        return sum;
    }
};
```

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

piles =
[5,4,9]

k =
2

Output

12

Expected

12

• Case 1 • Case 2

Input

piles =
[4,3,6,7]

k =
3

Output

12

Expected

12

2071. Maximum Number of Tasks You Can Assign

Hard | Topics | Companies | Hint

You have `n` tasks and `m` workers. Each task has a strength requirement stored in a **0-indexed** integer array `tasks`, with the `i^th` task requiring `tasks[i]` strength to complete. The strength of each worker is stored in a **0-indexed** integer array `workers`, with the `j^th` worker having `workers[j]` strength. Each worker can only be assigned to a **single** task and must have a strength **greater than or equal** to the task's strength requirement (i.e., `workers[j] >= tasks[i]`).

Additionally, you have `pills` magical pills that will **increase a worker's strength** by `strength`. You can decide which workers receive the magical pills, however, you may only give each worker **at most one** magical pill.

Given the **0-indexed** integer arrays `tasks` and `workers` and the integers `pills` and `strength`, return *the* ***maximum*** *number of tasks that can be completed.*

Example 1:

```
Input: tasks = [3,2,1], workers = [0,3,3], pills = 1, strength = 1
Output: 3
Explanation:
We can assign the magical pill and tasks as follows:
- Give the magical pill to worker 0.
- Assign worker 0 to task 2 (0 + 1 >= 1)
- Assign worker 1 to task 1 (3 >= 2)
- Assign worker 2 to task 0 (3 >= 3)
```

Code

C++ | Auto

```cpp
class Solution {
public:
    bool check(vector<int>& tasks, vector<int>& workers, int pills, int strength,int index)
    {
        multiset<int> st;
        for(auto it:workers)
        {
            st.insert(it);
        }
        for(int i=index-1;i>=0;i--)
        {
            auto it=st.lower_bound(tasks[i]);
            if(it!=st.end())
            {
                st.erase(it);
            }
            else
            {
                if(pills<=0)
                {
```

Code

C++ Auto

```
                return false;
            }
            else
            {
                it=st.lower_bound(tasks[i]-strength);
                if(it!=st.end())
                {
                    st.erase(it);
                    pills--;
                }
                else
                {
                    return false;
                }
            }
        }
    }
    return true;
}
int maxTaskAssign(vector<int>& tasks, vector<int>& workers, int pills, int strength) {
    sort(tasks.begin(),tasks.end());
```

Code

C++ Auto

```
    sort(workers.begin(),workers.end());
    int low=0;
    int high=min(workers.size(),tasks.size());
    while(low<high)
    {
        int mid=(low+high+1)/2;
        if(check(tasks,workers,pills,strength,mid)==true)
        {
            low=mid;
        }
        else
        {
            high=mid-1;
        }
    }
    return high;
}
};
```

Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

tasks =
[3,2,1]

workers =
[0,3,3]

pills =
1

strength =
1

Output

3

Expected

3

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

tasks =
[5,4]

workers =
[0,0,0]

pills =
1

strength =
5

Output

1

Expected

1

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

tasks =

[10,15,30]

workers =

[0,10,10,10,10]

pills =

3

strength =

10

Output

2

Expected

2