

WORKSHEET-8

Student Name: Abhishek Roushan

UID: 22BCS10187

Branch: CSE

Section/Group: NTPP-603-B

Semester: 6th

Date of Performance: 20/3/25

Subject Name: AP-2

Subject Code: 22CSP-351

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

- *`numberOfBoxesi` is the number of boxes of type `i`.*
- *`numberOfUnitsPerBoxi` 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`.*

Source Code:

```
class Solution {
public:
    int maximumUnits(vector<vector<int>>& boxTypes, int truckSize) {
        vector<int> buckets(1001, -1);
        int space_remaining_boxes = truckSize;
        int units_loaded = 0;
        for (int i = 0; i < boxTypes.size(); ++i) {
            if (buckets[ boxTypes[i][1] ] == -1) {
                buckets[ boxTypes[i][1] ] = boxTypes[i][0];
            } else { // already has a value
                buckets[ boxTypes[i][1] ] += boxTypes[i][0];
            }
        }
    }
};
```

```

    }

    for (int i = 1000; i >= 0; --i) {
        if (buckets[i] == -1) continue;

        if (buckets[i] > space_remaining_boxes) {
            units_loaded += space_remaining_boxes*i;
            return units_loaded;
        } else {
            units_loaded += buckets[i]*i
            space_remaining_boxes -= buckets[i];
        }
    }

    return units_loaded;
}
};

```

OUTPUT:

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

```
boxTypes =
[[1,3],[2,2],[3,1]]
```

```
truckSize =
4
```

Output

```
8
```

Expected

```
8
```

Question - 2

Aim(ii): 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 $\text{floor}(\text{piles}[i] / 2)$ 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.
- $\text{floor}(x)$ is the greatest integer that is smaller than or equal to *x* (i.e., rounds *x* down).

Source Code:

```
class Solution {
public:
    bool static help(int x,int y)
    {
        return x>y;
    }
    int minStoneSum(vector<int>& piles, int k) {

        int n=piles.size();
        priority_queue<int,vector<int>>pq(piles.begin(),piles.end());
        int ans=accumulate(piles.begin(),piles.end(),0);
        int i=0;
        while(k>0 && !pq.empty())
        {
            int temp=pq.top();
            pq.pop();
            ans-=(temp/2);
            pq.push(temp-temp/2);
            k--;
        }

        return ans;
    }
}
```

OUTPUT:

Accepted

Runtime: 0 ms

• Case 1

• Case 2

Input

piles =
[5,4,9]

k =
2

Output

12

Expected

12

🕒 Runtime



199 ms | Beats 77.39% 🏆

🔮 Analyze Complexity

💻 Memory

102.86 MB | B

15%

10%

5%

0%



Question - 3

Aim(iii):

You are given a string s and two integers x and y . You can perform two types of operations any number of times.

- *Remove substring "ab" and gain x points.*
- *For example, when removing "ab" from "cabxbae" it becomes "cxbae".*
- *Remove substring "ba" and gain y points.*
- *For example, when removing "ba" from "cabxbae" it becomes "cabxe".*
- *Return the maximum points you can gain after applying the above operations on s .*

Source Code:

```
class Solution {
    void getCount(string str, string sub, int& cnt1, int& cnt2) {

        char first = sub[0], second = sub[1];
        int i = 1;
        while(i < str.length()) {
            if(i > 0 && str[i-1] == first && str[i] == second) {
                cnt1++;
                str.erase(i-1, 2);
                i--;
                continue;
            }
            i++;
        }

        i = 1;
        while(i < str.length()) {
            if(i > 0 && str[i-1] == second && str[i] == first) {
                cnt2++;
                str.erase(i-1, 2);
                i--;
                continue;
            }
            i++;
        }
    }
};
```

```

        i++;
    }
    return;
}

public:
    int maximumGain(string s, int x, int y) {

        int mxABcnt = 0;
        int mxBAcnt = 0;
        int minBAcnt = 0;
        int minABcnt = 0;

        getCount(s, "ab", mxABcnt, minBAcnt);
        getCount(s, "ba", mxBAcnt, minABcnt);

        int operation1 = mxABcnt * x + minBAcnt * y;
        int operation2 = mxBAcnt * y + minABcnt * x;
        return max(operation1, operation2);
    }
};

```

OUTPUT :-

Accepted Runtime: 0 ms

• Case 1

• Case 2

Input

s =
"cdbcbbaaabab"

x =
4

y =
5

Output

19

Expected

19

Learning Outcome

1. We learnt about Greedy Programming.
2. We learnt about Priority Queue.
3. We learnt about Manipulating Strings.