**A1-B3-42**

**Amber Shukla**

**DAA LAB**

**PRACTICAL NO. 3**

**Aim:** Perform Fractional Knapsack for the given scenario

**A.** Load the truck using different methods: Minimum weight, Maximum profit,

Profit/weight ratio. Compute the total profit using each method and infer the best

performing method.

B. Compute the time required in each method.

Given Data:

 Capacity of truck 850 Kgs

 Weight in kg for each box:

[7, 0, 30, 22, 80, 94, 11, 81, 70, 64, 59, 18, 0, 36, 3, 8, 15, 42, 9, 0, 42, 47, 52, 32, 26, 48, 55,6, 29, 84, 2, 4, 18, 56, 7, 29, 93, 44, 71, 3, 86, 66, 31, 65, 0, 79, 20, 65, 52, 13]

 Profit in Rs for each box:

[ 360, 83, 59, 130, 431, 67, 230, 52, 93, 125, 670, 892, 600, 38, 48, 147, 78, 256, 63, 17, 120,164, 432, 35, 92, 110, 22, 42, 50, 323, 514, 28, 87, 73, 78, 15, 26, 78, 210, 36, 85, 189, 274,43, 33, 10, 19, 389, 276, 312 ]

**Code:**

#include <bits/stdc++.h>

#include <chrono>

using namespace std;

using namespace std::chrono;

#define pb push\_back

#define pop pop\_back

double trucksize = 850, n = 50;

// In Vector Box Dimention 1 is Weight || Dimention 2,0 is Profit || Dimention 2,1 is Profit/Weight

double max\_profit(vector<vector<double>> box)

{

    double profit = 0, current\_weight = 0;

    sort(box.begin(), box.end(), [](const vector<double> &a, const vector<double> &b)

         { return a[1] > b[1]; });

    int i = 0;

    while (current\_weight < trucksize && i < n)

    {

        if (current\_weight + box[i][0] >= trucksize)

        {

            double remaining\_space = trucksize - current\_weight;

            if (box[i][0] > 0)

            { // Avoid division by zero

                profit += (box[i][1] \* (remaining\_space / box[i][0]));

            }

            current\_weight = trucksize;

            break;

        }

        current\_weight += box[i][0];

        profit += box[i][1];

        i++;

    }

    return profit;

}

double min\_weight(vector<vector<double>> box)

{

    double profit = 0, current\_weight = 0;

    sort(box.begin(), box.end(), [](const vector<double> &a, const vector<double> &b)

         { return a[0] < b[0]; });

    int i = 0;

    while (current\_weight < trucksize && i < n)

    {

        if (current\_weight + box[i][0] >= trucksize)

        {

            double remaining\_space = trucksize - current\_weight;

            if (box[i][0] > 0)

            { // Avoid division by zero

                profit += (box[i][1] \* (remaining\_space / box[i][0]));

            }

            current\_weight = trucksize;

            break;

        }

        current\_weight += box[i][0];

        profit += box[i][1];

        i++;

    }

    return profit;

}

double pw\_ratio(vector<vector<double>> box)

{

    double profit = 0, current\_weight = 0;

    sort(box.begin(), box.end(), [](const vector<double> &a, const vector<double> &b)

         { return a[2] > b[2]; });

    int i = 0;

    while (current\_weight < trucksize && i < n)

    {

        if (current\_weight + box[i][0] >= trucksize)

        {

            double remaining\_space = trucksize - current\_weight;

            if (box[i][0] > 0)

            { // Avoid division by zero

                profit += (box[i][1] \* (remaining\_space / box[i][0]));

            }

            current\_weight = trucksize;

            break;

        }

        current\_weight += box[i][0];

        profit += box[i][1];

        i++;

    }

    return profit;

}

int main()

{

    vector<vector<double>> box(n, vector<double>(3, 0));

    cout << "Enter Weight of all Boxes" << endl;

    for (int i = 0; i < n; i++)

        cin >> box[i][0];

    cout << "Enter Profit Values of all Boxes" << endl;

    for (int i = 0; i < n; i++)

    {

        cin >> box[i][1];

        (box[i][0] != 0) ? (box[i][2] = (box[i][1] / box[i][0])) : (box[i][2] = 9999); // Profit/Weight Calculation

    }

    auto start\_time = high\_resolution\_clock::now();

    cout << "\n\nProfit By Maximum Profit Method: " << max\_profit(box) << endl;

    auto end\_time = high\_resolution\_clock::now();

    auto time\_taken = duration\_cast<microseconds>(end\_time - start\_time);

    cout << "Time taken: " << time\_taken.count() << " microseconds" << endl;

    start\_time = high\_resolution\_clock::now();

    cout << "\nProfit By Minimum Weight Method: " << min\_weight(box) << endl;

    end\_time = high\_resolution\_clock::now();

    time\_taken = duration\_cast<microseconds>(end\_time - start\_time);

    cout << "Time taken: " << time\_taken.count() << " microseconds" << endl;

    start\_time = high\_resolution\_clock::now();

    cout << "\nProfit By Profit/Weight Ratio Method: " << pw\_ratio(box) << endl;

    end\_time = high\_resolution\_clock::now();

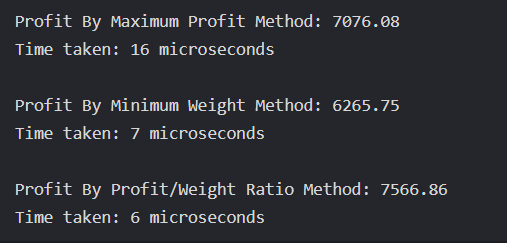
    time\_taken = duration\_cast<microseconds>(end\_time - start\_time);

    cout << "Time taken: " << time\_taken.count() << " microseconds" << endl;

    return 0;

}

**Output:**



**LeetCode:**

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 totalUnits = 0;

int Total\_Boxes = 0;

for (auto& box : boxTypes) {

int Box\_Count = box[0];

int Box\_Units = box[1];

int Box\_Min = min(Box\_Count, truckSize - Total\_Boxes);

totalUnits += Box\_Min \* Box\_Units;

Total\_Boxes += Box\_Min;

if (Total\_Boxes == truckSize) {

break;

}

}

return totalUnits;

}

};

****

**PART (B)**

[SUBMISSION ON HackerRank]

**Aim:** Huffman coding and decoding

<https://www.hackerrank.com/challenges/tree-huffman-decoding/problem>

**Code:**

void decode\_huff(node \*root, string s)

{

    node \*temp = root;

    char digit;

    for (int i = 0; i <= s.size(); i++)

    {

        if (i == s.size())

            digit = s[i - 1];

        else

            digit = s[i];

        if (temp->left == NULL && temp->right == NULL)

        {

            cout << temp->data;

            temp = root;

            i--;

            continue;

        }

        if (digit == '0')

            temp = temp->left;

        if (digit == '1')

            temp = temp->right;

    }

}

**Outputs:**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.