**Task 1: Code and analyze solutions to following problem with given strategies:**

1. **Knap Sack using greedy approach**
2. **Knap Sack using dynamic approach**

**i. Knapsack Using Greedy Approach**

The greedy approach for the Knapsack problem involves making the most optimal choice at each step with the hope of finding the global optimum. However, this approach does not always yield the best solution for the Knapsack problem, but it's useful in some variations like the Fractional Knapsack problem.

**ii. Knapsack Using Dynamic Programming**

The dynamic programming (DP) approach solves the Knapsack problem by building up a solution using previously calculated results. This approach ensures that you find the optimal solution by considering all possible combinations of items.

**CODE:**

#include <iostream>

#include <algorithm>

#include <vector>

using namespace std;

struct Item {

    int profit;

    int weight;

    float pwRatio;

    Item(int profit, int weight) : profit(profit), weight(weight) {

        this->pwRatio = (float)profit / weight;

    }

};

bool cmpByRatio(Item a, Item b) {

    return a.pwRatio > b.pwRatio;

}

bool cmpByWeight(Item a, Item b) {

    return a.weight < b.weight;

}

bool cmpByProfit(Item a, Item b) {

    return a.profit > b.profit;

}

float knapSackGreedyByRatio(vector<Item> obj, int capacity) {

    sort(obj.begin(), obj.end(), cmpByRatio);

    float totalProfit = 0.0;

    int currentWeight = 0;

    for (auto item : obj) {

        if (currentWeight + item.weight <= capacity) {

            currentWeight += item.weight;

            totalProfit += item.profit;

        } else {

            int remainingWeight = capacity - currentWeight;

            totalProfit += item.profit \* ((double)remainingWeight / item.weight);

            break;

        }

    }

    return totalProfit;

}

float knapSackGreedyByWeight(vector<Item> obj, int capacity) {

    sort(obj.begin(), obj.end(), cmpByWeight);

    float totalProfit = 0.0;

    int currentWeight = 0;

    for (auto item : obj) {

        if (currentWeight + item.weight <= capacity) {

            currentWeight += item.weight;

            totalProfit += item.profit;

        } else {

            int remainingWeight = capacity - currentWeight;

            totalProfit += item.profit \* ((double)remainingWeight / item.weight);

            break;

        }

    }

    return totalProfit;

}

float knapSackGreedyByProfit(vector<Item> obj, int capacity) {

    sort(obj.begin(), obj.end(), cmpByProfit);

    float totalProfit = 0.0;

    int currentWeight = 0;

    for (auto item : obj) {

        if (currentWeight + item.weight <= capacity) {

            currentWeight += item.weight;

            totalProfit += item.profit;

        } else {

            int remainingWeight = capacity - currentWeight;

            totalProfit += item.profit \* ((double)remainingWeight / item.weight);

            break;

        }

    }

    return totalProfit;

}

int knapSackDynamic(vector<Item> &obj, int capacity) {

    int n = obj.size();

    vector<vector<int>> dp(n + 1, vector<int>(capacity + 1, 0));

    for (int i = 1; i <= n; ++i) {

        for (int w = 1; w <= capacity; ++w) {

            if (obj[i - 1].weight <= w) {

                dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - obj[i - 1].weight] + obj[i - 1].profit);

            } else {

                dp[i][w] = dp[i - 1][w];

            }

        }

    }

    return dp[n][capacity];

}

void printObj(vector<Item> obj){

    cout<<"Item  Profit  Weight\n";

    int i = 1;

    for(auto item : obj){

        cout<<i<<"     "<<item.profit<<"      "<<item.weight<<endl;

        i++;

    }

    cout<<endl;

}

int main() {

    int capacity = 20;

    vector<Item> obj = {

        {25, 18},

        {24, 15},

        {15, 10}

    };

    printObj(obj);

    float maxProfitByProfit = knapSackGreedyByProfit(obj, capacity);

    cout << "Max profit in knap sack (by profit) is : " << maxProfitByProfit << endl;

    float maxProfitByWeight = knapSackGreedyByWeight(obj, capacity);

    cout << "Max profit in knap sack (by weight) is : " << maxProfitByWeight << endl;

    float maxProfitByRatio = knapSackGreedyByRatio(obj, capacity);

    cout << "Max profit in knap sack (by ratio) is : " << maxProfitByRatio << endl;

    int maxProfitDynamic = knapSackDynamic(obj, capacity);

    cout << "Max profit in knap sack (dynamic approach) is : " << maxProfitDynamic <<endl;

    return 0;

}

**OUTPUT:**

