Roll No: 33343

Code:

```
#include <bits/stdc++.h>
using namespace std;


struct item{
    float weight;
    float profit;
    float pbyw; //profit by weight ratio
};


string prd(const float x, const int decDigits, const int width) {
    stringstream ss;
    ss << fixed << right;
    ss.fill(' ');        // fill space around displayed #
    ss.width(width);     // set  width around displayed #
    ss.precision(decDigits); // set # places after decimal
    ss << x;
    return ss.str();
}


// merge function
// type parameter is used for sorting based on profit by weight ratio(1), by profit(2), by weight(3)
void merge(item items[], int start, int mid, int end, int type){
    int lSize = mid-start+1;
    int rSize = end-mid;


    item leftArray[lSize];
    item rightArray[rSize];
    for(int i=0; i<lSize; i++) leftArray[i] = items[i+start];
    for(int i=0; i<rSize; i++) rightArray[i] = items[i+mid+1];


    int i=0, j=0, k=start;
    while(i<lSize && j<rSize){
        if(type==1){
            if(leftArray[i].pbyw > rightArray[j].pbyw){
                items[k++] = leftArray[i++];
            }else{
                items[k++] = rightArray[j++];
            }
        }
        if(type==2){
            if(leftArray[i].profit > rightArray[j].profit){
                items[k++] = leftArray[i++];
            }else{
                items[k++] = rightArray[j++];
            }
        }
        if(type==3) {
            if(leftArray[i].weight < rightArray[j].weight){
                items[k++] = leftArray[i++];
            }else{
                items[k++] = rightArray[j++];
            }
        }
    }

    while(i<lSize) items[k++] = leftArray[i++];
    while(j<rSize) items[k++] = rightArray[j++];
}

// merge sort function
// type parameter is used for sorting based on profit by weight ratio(1), by profit(2), by weight(3)
void mergeSort(item items[], int start, int end, int type){
    if(start>=end) return;

    int mid = (end+start)/2;

    mergeSort(items, start, mid, type);
    mergeSort(items, mid+1, end, type);
    merge(items, start, mid, end, type);
}
// type parameter for fractional knapsack or 1/0 based
void calc_profit(int capacity, item items[], int n, int type){
    cout << "item picked" << endl;
    cout << "Item weight\t item profit \t total profit"<<endl;
    int total_profit= 0;
    for(int i=0; i<n; i++){
        if(capacity - items[i].weight >= 0){
            capacity -= items[i].weight;
            total_profit += items[i].profit;
            cout << prd(items[i].weight, 0, 8) << " | " << prd(items[i].profit, 0, 15) << " | " <<prd(total_profit, 2, 10) << "\n";

        }else{
            if(type == 1){
                total_profit += (capacity/items[i].weight) * items[i].profit;
                string str =  (capacity>0) ? "yes - original weight= "+to_string(items[i].weight): "no";
                cout << prd(capacity, 0, 8) << " | " << prd(items[i].profit, 0, 15) << " | " <<prd(total_profit, 2, 10) << " | Picked ?" << str << "\n";
                capacity = 0;
            }
            if(capacity == 0) break;
        }
    }
    cout << "\nTotal profit is: " << total_profit << endl;
    cout << "Is bag empty: " << (capacity<=0 ?
"no" : "yes") << endl;


}


int main(){

    int n, capacity;

    cout << "Enter the count of items: ";

    cin >> n;

    cout << "Enter capacity of bag: ";

    cin >> capacity;


    item items[n];


    cout << "Enter the items weight: ";

    int w;

    for(int i=0; i<n; i++){

        cin >> w;

        items[i].weight = w;

    }

    cout << "Enter the items profit: ";

    int p;

    for(int i=0; i<n; i++){

        cin >> p;

        items[i].profit = p;

        items[i].pbyw =
items[i].profit/items[i].weight;

    }

    cout << "\n\nAvailable information\n";

    cout << "Items: " << n << endl;

    cout << "Capacity: " << capacity << endl
<< endl;


    int type=0;

    cout << "\n\nBased on profit by weight
ratio\n";

    cout << "1.Fractional knapsack \n2.1/0
knapsack: \nEnter your choice : ";

    cin >> type;

    mergeSort(items, 0, n-1, 1);

    calc_profit(capacity, items, n, type);


    cout << "\n\nBased on profit\n";

    cout << "1.Fractional knapsack \n2.0/1
knapsack: ";

    cin >> type;

    mergeSort(items, 0, n-1, 2);

    calc_profit(capacity, items, n, type);


    cout << "\n\nBased on weight\n";

    cout << "1.Fractional knapsack \n2.0/1
knapsack: ";

    cin >> type;

    mergeSort(items, 0, n-1, 3);

    calc_profit(capacity, items, n, type);


    return 0;

}
```

Output:

```
Enter the count of items: 6

Enter capacity of bag: 15

Enter the items weight: 2 4 2 6 4 3

Enter the items profit: 6 8 9 10 5 6



Available information

Items: 6

Capacity: 15


Based on profit by weight ratio

1.Fractional knapsack

2.1/0 knapsack:

Enter your choice : 1

item picked

Item weight     item profit     total profit

       2 |              9 |       9.00

       2 |              6 |      15.00

       3 |              6 |      21.00

       4 |              8 |      29.00

       4 |             10 |      35.00 | Picked ?yes -
original weight= 6.000000


Total profit is: 35

Is bag empty: no
```

```
Based on profit

1.Fractional knapsack

2.0/1 knapsack: 2

item picked

Item weight     item profit     total profit

       6 |             10 |      10.00

       2 |              9 |      19.00

       4 |              8 |      27.00

       3 |              6 |      33.00


Total profit is: 33

Is bag empty: no



Based on weight

1.Fractional knapsack

2.0/1 knapsack: 1

item picked

Item weight     item profit     total profit

       2 |              6 |       6.00

       2 |              9 |      15.00

       3 |              6 |      21.00

       4 |              5 |      26.00

       4 |              8 |      34.00

       0 |             10 |      34.00 | Picked ?no


Total profit is: 34

Is bag empty: no
```