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

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

    while(i<lSize) items[k++] = lArr[i++];
    while(j<rSize) items[k++] = rArr[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, 15) << " | " << 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, 15) << " | " << 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 data\n";
    cout << "Items: " << n << endl;
    cout << "Capacity: " << capacity << endl << endl;

    int type=0;
    cout << "\n\nBased on profit by weight ration\n";
    cout << "1.Fractional knapsack 2.1/0 knapsack: ";
    cin >> type;
    mergesort(items, 0, n-1, 1);
    calc_profit(capacity, items, n, type);



    cout << "\n\nBased on profit\n";
    cout << "1.Fractional knapsack 2.1/0 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 2.1/0 knapsack: ";
    cin >> type;
    mergesort(items, 0, n-1, 3);
    calc_profit(capacity, items, n, type);

    return 0;
}
```

Output:

c**ase 1: (based on weight, profit, and the ratio)**

```
@somesh4545 ➜ /workspaces/TE-Labs/DAA (main) $ g++ fractional_knapsack.cpp && ./a.out
Enter the count of items: 5
Enter capacity of bag: 25
Enter the items weight: 5 10 15 8 1
Enter the items profit: 15 20 30 40 10

Available data
Items: 5
Capacity: 25

Based on profit by weight ration
1.Fractional knapsack 2.1/0 knapsack: 1
item picked
Item weight     item profit     total profit
          1 |          10 |     10.00
          8 |          40 |     50.00
          5 |          15 |     65.00
         11 |          30 |     87.00 | Picked ?yes - original weight= 15.000000

Total profit is: 87
Is bag empty: no

Based on profit
1.Fractional knapsack 2.1/0 knapsack: 1
item picked
Item weight     item profit     total profit
          8 |          40 |     40.00
         15 |          30 |     70.00
          2 |          20 |     74.00 | Picked ?yes - original weight= 10.000000

Total profit is: 74
Is bag empty: no

Based on weight
1.Fractional knapsack 2.1/0 knapsack: 2
item picked
Item weight     item profit     total profit
          1 |          10 |     10.00
          5 |          15 |     25.00
          8 |          40 |     65.00
         10 |          20 |     85.00

Total profit is: 85
Is bag empty: yes
```

case 2: (when capacity of bag is more than total weight)

```
@somesh4545 ➜ /workspaces/TE-Labs/DAA (main) $ g++ fractional_knapsack.cpp && ./a.out
Enter the count of items: 5
Enter capacity of bag: 40
Enter the items weight: 5 10 15 8 1
Enter the items profit: 15 20 30 40 10

Available data
Items: 5
Capacity: 40

Based on profit by weight ration
1.Fractional knapsack 2.1/0 knapsack: 1
item picked
Item weight     item profit     total profit
          1 |            10 |     10.00
          8 |            40 |     50.00
          5 |            15 |     65.00
         15 |            30 |     95.00
         10 |            20 |    115.00

Total profit is: 115
Is bag empty: yes

Based on profit
1.Fractional knapsack 2.1/0 knapsack: 2
item picked
Item weight     item profit     total profit
          8 |            40 |     40.00
         15 |            30 |     70.00
         10 |            20 |     90.00
          5 |            15 |    105.00
          1 |            10 |    115.00

Total profit is: 115
Is bag empty: yes

Based on weight
1.Fractional knapsack 2.1/0 knapsack: 1
item picked
Item weight     item profit     total profit
          1 |            10 |     10.00
          5 |            15 |     25.00
          8 |            40 |     65.00
         10 |            20 |     85.00
         15 |            30 |    115.00

Total profit is: 115
Is bag empty: yes
```