

A greedy algorithm is an algorithm that follows the problem solving heuristic of making the locally optimal choice each stage with the hope of finding the global optimum.

A simple solution is to consider all subsets of items and calculate the total weight and value of all subsets. Consider the only subsets whose total weight is smaller than W . From all such subsets, pick the maximum value subset. To consider all subsets of items, there can be two cases for every item.

1. **Case 1:** The item is included in the optimal subset.
2. **Case 2:** The item is not included in the optimal set.

Therefore, the maximum value that can be obtained from 'n' items is the max of the following two values.

Maximum value obtained by n-1 items and W weight (excluding nth item)

Value of nth item plus maximum value obtained by n-1 items and W minus the weight of the nth item.

Below is the implementation of the above approach:

```
#include <iostream>

#include <bits/stdc++.h>

using namespace std;

typedef struct {
    int value;
    int weight;
    float density;
```

```
}Item;
```

```
void input(Item items[],int sizeOfItems){
```

```
    cout << "Enter total " << sizeOfItems << " item's values and weight" << endl;
```

```
    for(int i=0; i<sizeOfItems; i++){
```

```
        cout << "Enter " << i+1 << " Value ";
```

```
        cin >> items[i].value;
```

```
        cout << "Enter " << i+1 << " Weight ";
```

```
        cin >> items[i].weight;
```

```
    }
```

```
}
```

```
void display(Item items[],int sizeOfItems){
```

```
    cout << "values:  ";
```

```
    for(int i=0; i<sizeOfItems; i++){
```

```
        cout << items[i].value << "\t";
```

```
    }
```

```
    cout << endl << "weight:  ";
```

```
    for(int i=0; i<sizeOfItems; i++){
```

```
        cout << items[i].weight << "\t";
```

```
}  
cout << endl;  
}
```

```
bool compare(Item i1, Item i2){  
    return (i1.density > i2.density);  
}
```

```
float knapsack(Item items[],int sizeOfItems, int W){  
    float totalValue=0, totalWeight=0;  
  
    //calculating density of each item  
    for(int i=0; i<sizeOfItems; i++){  
        items[i].density = items[i].value/items[i].weight;  
    }
```

```
    //sorting w.r.t to density using compare function  
    sort(items, items+sizeOfItems,compare);
```

```
    for(int i=0; i<sizeOfItems; i++){  
        if(totalWeight + items[i].weight<= W){  
            totalValue += items[i].value ;
```

```

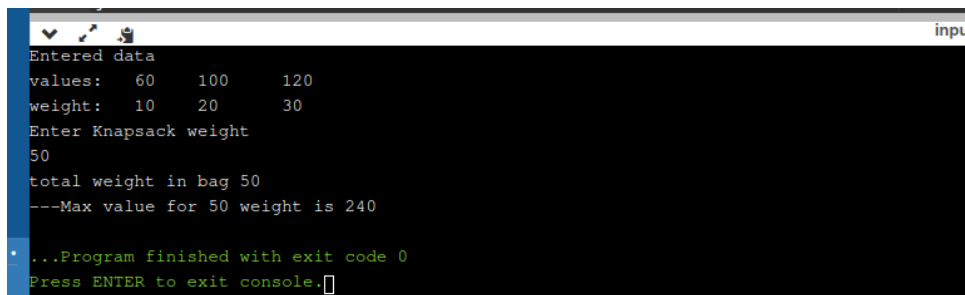
        totalWeight += items[i].weight;
    } else {
        int wt = W-totalWeight;
        totalValue += (wt * items[i].density);
        totalWeight += wt;
        break;
    }
}

cout << "total weight in bag " << totalWeight<<endl;
return totalValue;
}

int main()
{
    int W,n;
    cout<<"Enter number of items: ";
    cin>>n;
    Item items[n];
    input(items,n);
    cout << "Entered data \n";
    display(items,n);
    cout<< "Enter Knapsack weight \n";
    cin >> W;

```

```
float mxVal = knapsack(items,n,W);  
cout << "---Max value for "<< W <<" weight is "<< mxVal;  
  
return 0;  
}
```



```
input  
Entered data  
values: 60 100 120  
weight: 10 20 30  
Enter Knapsack weight  
50  
total weight in bag 50  
---Max value for 50 weight is 240  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Complexity Analysis:

- **Time Complexity:** $O(2^n)$.
- As there are redundant subproblems.
- **Auxiliary Space :** $O(1)$.
- As no extra data structure has been used for storing values.

