Green University of Bangladesh
Department of Computer Science and Engineering(CSE)

Faculty of Sciences and Engineering

Semester: (Spring, Year:2021), B.Sc. in CSE (Day)

LAB REPORT NO 6

Course Title: Algorithm

Course Code:206 Section:DB

Lab Experiment Name: Using Greedy algorithm to find minimum number of coins and fractional knapsack

Student Details

	Name	ID
1.	Shamim Ahmed	201902067

Course Teacher's Name : Monoshi Kumar Roy

[For Teachers use only: Don't Write Anything inside this box]

L ab Report Status Marks: …… **Signature:……** .

Objective(s)

- Using Greedy algorithm to find minimum number of coins
- To find the maximum prince in a limited weight

Problem analysis

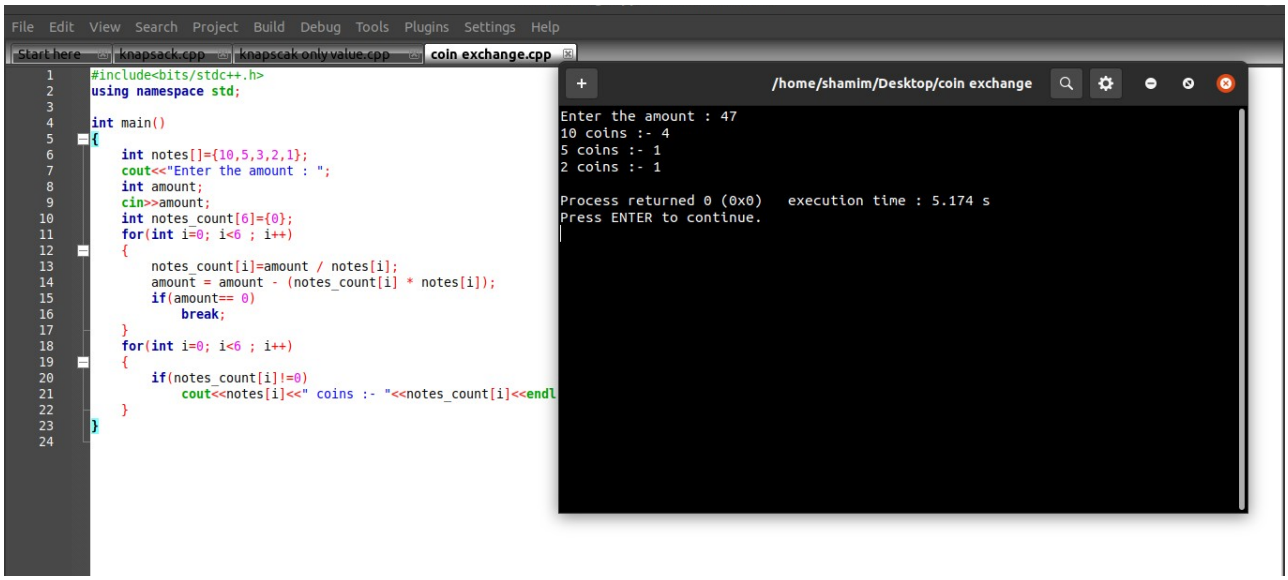
Problem Statement 1: I have to give n taka to my friend. I have {1,2,3,5,10} coins. I can use one coin many times. Have to write a code to take n as input, and my program should show which coin I would exchange so that I have to give minimum number of coins to my friend.

Implementation in C++ Problem Statement 1:

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

int main()
{
   int notes[]={10,5,3,2,1};
   cout<<"Enter the amount : ";
   int amount;
   cin>>amount;
   int notes_count[6]={0};
   for(int i=0; i<6 ; i++)
   {
      notes_count[i]=amount / notes[i];
      amount = amount - (notes_count[i] * notes[i]);
      if(amount== 0)
         break;
   }
   for(int i=0; i<6 ; i++)
   {
      if(notes_count[i]!=0)
         cout<<notes[i]<<" coins :- "<<notes_count[i]<<endl;
   }
}
```

Sample Input/Output for minimum coin:



Time Complexity for Problem Statement 1:

Time complexity of the greedy coin change algorithm will be o(n). If I was sort my array than the complexity would be o(nlogn). But here I use {10,5,3,2,1} which already sorted. So dosent require to sort my coins array so the complexity would be 0(n).

Problem Statement 2: A list of items, their corresponding weight and values are given. We know using per weight value sorting technique will help us to maximize our profit. However, in my code I have to apply both per weight value vs only item value sorting (sort the items on the basis of their value) and show a comparison that per weight value is giving us maximum profit. (knapsack Weight=60kg).
We see this is a problem of finding max. So,

- **Step 1**: For each item, compute its value / weight ratio.
- **Step 2**: Arrange all the items in decreasing order of their value / weight ratio.
- **Step 3**: Start putting the items into the knapsack beginning from the item with the highest ratio. Putting as many items as we can into the knapsack.

Algorithm for problem statement 2

```
float fractionalKnapsack(int K, item A[], int N)
{
        //Sort the items on the basis of increasing ratio of V/W
        sort(A, A+N)
        double totalValue = 0.0
        int currUsedWeight = 0
         for(int i = 0 to N-1)
        {
                if(A[i].weight < K)
         {
                totalValue = totalValue + A[i].value
                currUsedWeight = currUsedWeight + A[i].weight
```

```
        }
        else
        {
                int availableSpace = K - currUsedSpace
                //Taking the Fraction
                totalValue = totalValue + A[i].value*(availableSpace/A[i].weight) break
        }
        }
}
```

Implementation in C++ Problem Statement 2:

```
#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++)
   {
       cin >> items[i].weight;
       cin >> items[i].value;

   }
}

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;


   for(int i=0; i<sizeOfItems; i++)
   {
      items[i].density = (float)items[i].value/items[i].weight;
   }


   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 << "\nTotal weight in bag : " << totalWeight<<endl;
   return totalValue;
}
bool comp(Item i1, Item i2)
{
   return (i1.value> i2.value);
}

float knapsack_0nly_value(Item items[],int sizeOfItems, int W)
{
   float totalValue=0, totalWeight=0;


   sort(items, items+sizeOfItems,comp);

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

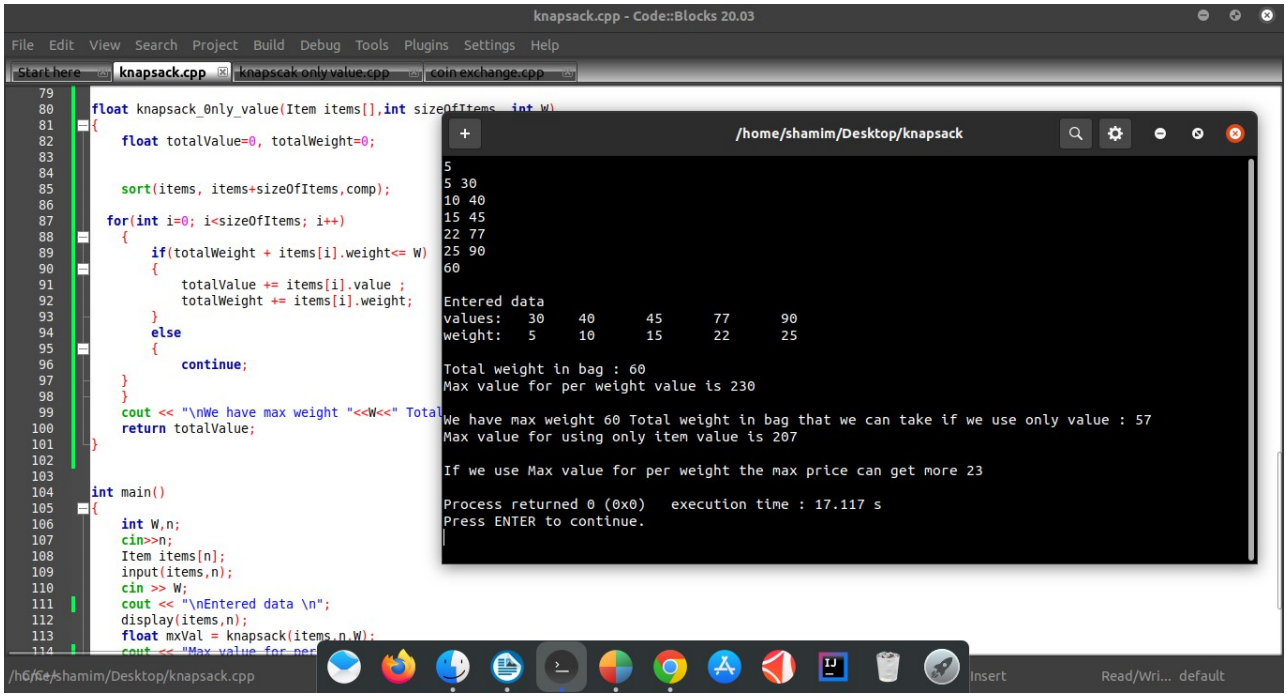
```
            totalValue += items[i].value ;
            totalWeight += items[i].weight;
        }
        else
        {
            continue;
    }
    }
    cout << "\nWe have max weight "<<W<<" Total weight in bag that we can take if we use only
value : " << totalWeight<<endl;
    return totalValue;
}


int main()
{
    int W,n;
    cin>>n;
    Item items[n];
    input(items,n);
    cin >> W;
    cout << "\nEntered data \n";
    display(items,n);
    float mxVal = knapsack(items,n,W);
    cout << "Max value for per weight value is "<< mxVal<<endl;
     float mxVal2 = knapsack_0nly_value(items,n,W);
    cout << "Max value for using only item value is "<< mxVal2<<endl;

    cout<<"\nIf we use Max value for per weight the max price can get more "<<mxVal-
mxVal2<<endl;


    return 0;
}
```

Sample Input/Output for fractional knapsack:



Time Complexity for Problem Statement 2:

The main time taking step is the sorting of all items in decreasing order of their value / weight ratio.

- If the items are already arranged in the required order, then while loop takes O(n) time.
- The average time complexity for sorting is O(nlogn).
- Therefore, total time taken including the sort is O(nlogn).

Discussion & Conclusion:

When we was using only only item value sorting we could take 57 kg where the limitation was 60. The maximum price was 207. But when we apply per weight value we was able to take all 60 kg the maximum price was 230 . So if we use per weight value we would be able to take maximum price in a limited wight than using only item value sorting