Experiment 2.1

Student Name: Gaurav Kumar **UID: 22MCC20177**

Branch: Computer **Section/Group:- 1/B**

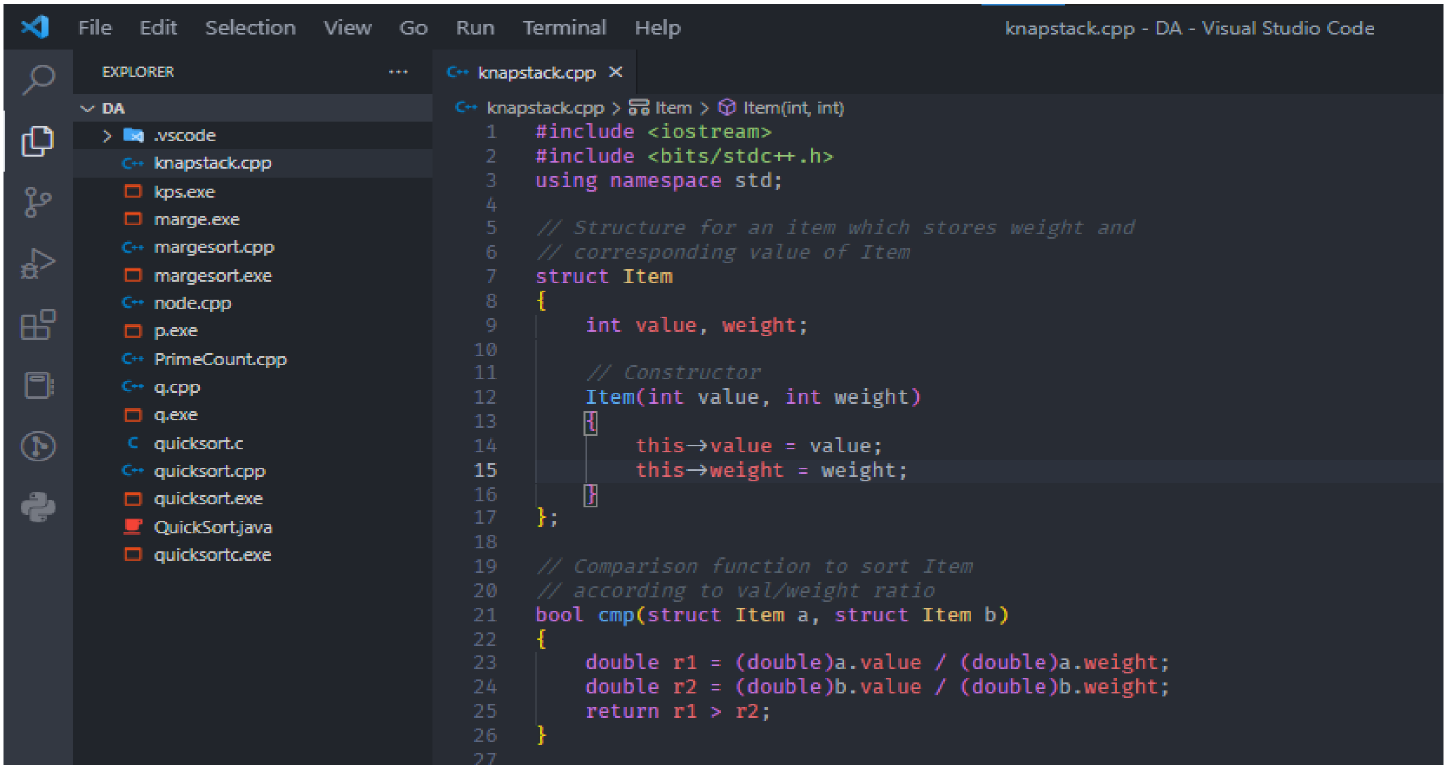
Semester: One **Date of Performance: 27/11/2022**

Subject Name:- Design & Analysis of Algorithms Lab **Subject Code: 22CAP-646**

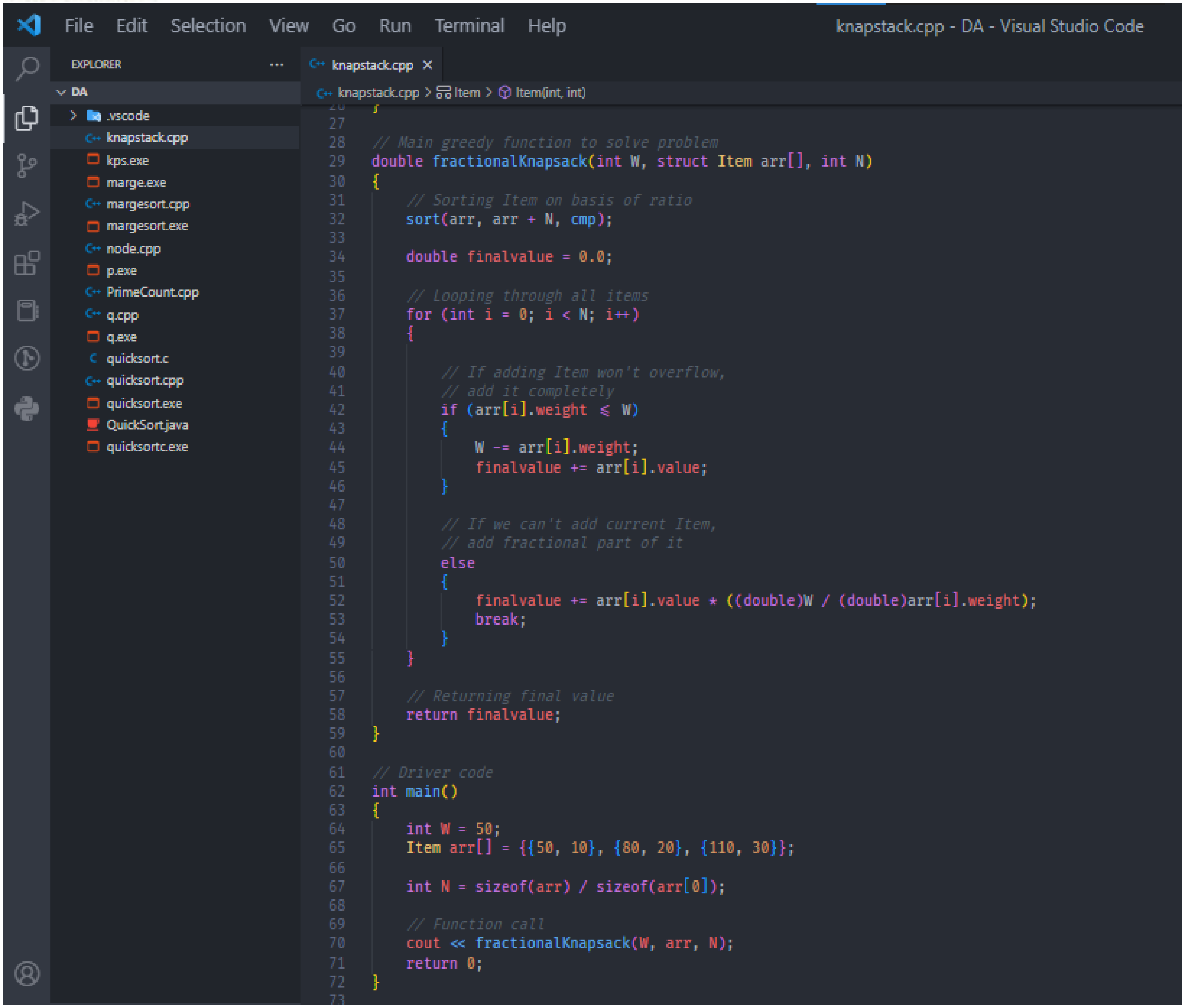
Task to be done:

(a) Implement Fractional Knapsack problem using Greedy algorithm.
Take Input:
Items as (value, weight) pairs
arr[] = {{50, 10}, {80, 20}, {110, 30}}
Knapsack Capacity, W = 50;

Steps for experiment/practical: copy and paste your code here/screenshots

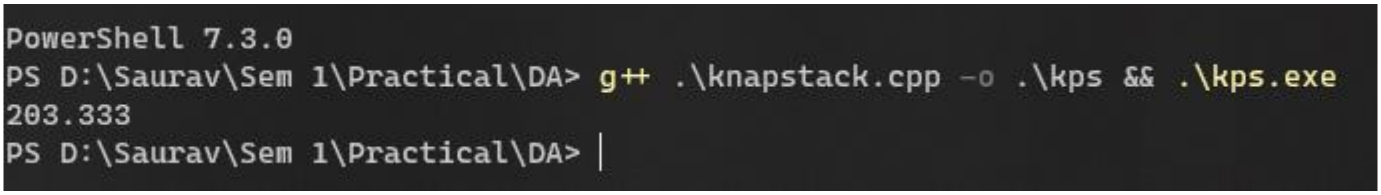


```cpp
#include <iostream>
#include <bits/stdc++.h>
using namespace std;

// Structure for an item which stores weight and
// corresponding value of Item
struct Item
{
    int value, weight;

    // Constructor
    Item(int value, int weight)
    {
        this->value = value;
        this->weight = weight;
    }
};

// Comparison function to sort Item
// according to val/weight ratio
bool cmp(struct Item a, struct Item b)
{
    double r1 = (double)a.value / (double)a.weight;
    double r2 = (double)b.value / (double)b.weight;
    return r1 > r2;
}
```



```cpp
}

// Main greedy function to solve problem
double fractionalKnapsack(int W, struct Item arr[], int N)
{
    // Sorting Item on basis of ratio
    sort(arr, arr + N, cmp);

    double finalvalue = 0.0;

    // Looping through all items
    for (int i = 0; i < N; i++)
    {

        // If adding Item won't overflow,
        // add it completely
        if (arr[i].weight <= W)
        {
            W -= arr[i].weight;
            finalvalue += arr[i].value;
        }

        // If we can't add current Item,
        // add fractional part of it
        else
        {
            finalvalue += arr[i].value * ((double)W / (double)arr[i].weight);
            break;
        }
    }

    // Returning final value
    return finalvalue;
}

// Driver code
int main()
{
    int W = 50;
    Item arr[] = {{50, 10}, {80, 20}, {110, 30}};

    int N = sizeof(arr) / sizeof(arr[0]);

    // Function call
    cout << fractionalKnapsack(W, arr, N);
    return 0;
}
```

Output (screenshots)



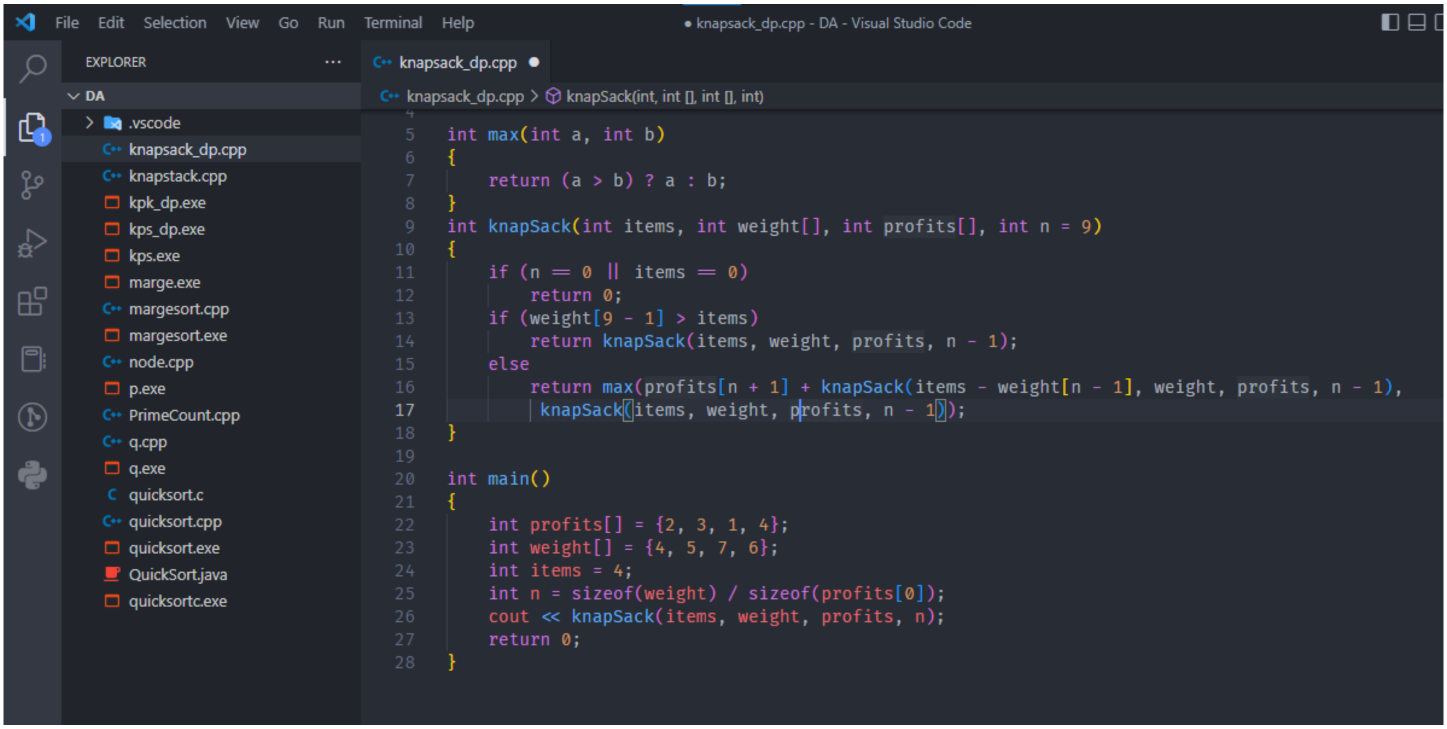
```
PowerShell 7.3.0
PS D:\Saurav\Sem 1\Practical\DA> g++ .\knapstack.cpp -o .\kps && .\kps.exe
203.333
PS D:\Saurav\Sem 1\Practical\DA> |
```

Task to be done:

(b) Implement 0/1 Knapsack problem using dynamic programming.
Take Input
Weights: {4, 5, 7, 6}
Profits: {2, 3, 1, 4}
The weight of the knapsack is 9 kg

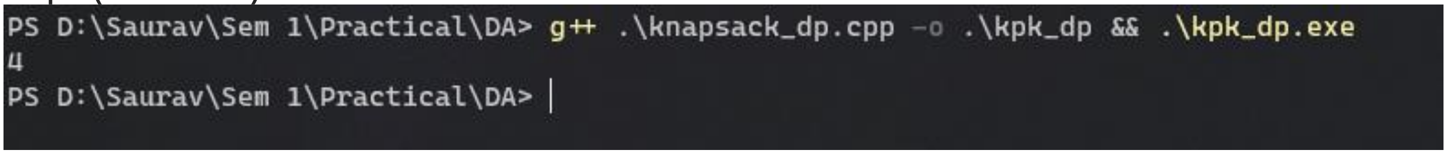
The number of items is 4

Steps for experiment/practical: copy and paste your code here/screenshots



```cpp
int max(int a, int b)
{
    return (a > b) ? a : b;
}
int knapSack(int items, int weight[], int profits[], int n = 9)
{
    if (n == 0 || items == 0)
        return 0;
    if (weight[9 - 1] > items)
        return knapSack(items, weight, profits, n - 1);
    else
        return max(profits[n + 1] + knapSack(items - weight[n - 1], weight, profits, n - 1),
         knapSack(items, weight, profits, n - 1));
}

int main()
{
    int profits[] = {2, 3, 1, 4};
    int weight[] = {4, 5, 7, 6};
    int items = 4;
    int n = sizeof(weight) / sizeof(profits[0]);
    cout << knapSack(items, weight, profits, n);
    return 0;
}
```

Output (screenshots)



```
PS D:\Saurav\Sem 1\Practical\DA> g++ .\knapsack_dp.cpp -o .\kpk_dp && .\kpk_dp.exe
4
PS D:\Saurav\Sem 1\Practical\DA> 
```

Evaluation Grid:

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.	Demonstration and Performance (Quiz)		22
2.	Worksheet		8