Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

Lab 4

Algorithms and Complexity (COMP 314)

Submitted By:

Mani Dumaru

Roll no.: 15

CE 3rd Year/ 2nd Semester

Submitted to:

Dr. Rajani Chulyadyo

Department of Computer Science and Engineering

Submission Date: 3rd June 2023

Solving Knapsack problem using different algorithm design strategies.

1. Brute Force Method:
   a. *0/1 Knapsack using Brute Force strategy*

   Source Code: 01Brute.py

   Pseudocode:

```
01knapsackBrute (length, data, size, init):
        if size <= 0: return;
        if (weight of current data) <= size:
                profit_including _data = data.profit + 01knapsackBrute(length, data, size - data.weight, init+1)
                profit_excluding_data = 01knapsackBrute(length, data, size, init+1)
                return max of (profit_including_data, profit_excluding_data)
        else:
                profit_excluding_data = 01knapsackBrute(length, data, size, init+1)
                return profit_excluding_data
```

   b. *Fractional Knapsack using Brute Force*

   Source Code: fractionalKnapsack.py

   Pseudocode:

```
FractionalBrute (length, data, size, init):
        if size <= 0: return;
        if (weight of current data) <= size:
                profit_including _data = data.profit + FractinalBrute(length, data, size - data.weight, init+1)
                profit_excluding_data = FractionalBrute(length, data, size, init+1)
        else:
                profit_including = data.profit * (size/data.weight)
```

```
            excluding_profit = FractionalBrute(length, data, size, init+1)
        return max(profit_including, profit, excluding)
```

2. Greedy Approach:

Source Code: greedyKnapsack.py

Pseudocode:

```
greedyKnapsack (data, size):
        profit = 0
        for every i in data:
            i[“profit/weight”] = i.profit / i.weight

        sort data in descending order of profit/weight

        for every i in data:
            if size <= 0:
                break
            if i.weight <= size:
                profit = profit + i.profit
                size = size – i.weight
            else:
                profit = profit + i.profit / (size/i.weight)
                size = 0
        return profit
```

Test Cases:

Source Code: test.py

Output:

```
D:\CE-2019\Sem 6\lab works\Algorithm\lab4>python test.py
...
----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
```

Conclusion:

Hence, the knapsack problem was solved using two different approaches of algorithm design strategies. Brute Force and Greedy approach. The test cases were written for each of these approach to check if they give the correct result from the given manual data. The algorithm turned out to be true for given set of data for knapsack problem.