Objective: Implementation and analysis of Knapsack problem

Knapsack problem

Given a set of items, each with a weight and a value, determine a subset of items to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

The knapsack problem is in combinatorial optimization problem. It appears as a subproblem in many, more complex mathematical models of real-world problems. One general approach to difficult problems is to identify the most restrictive constraint, ignore the others, solve a knapsack problem, and somehow adjust the solution to satisfy the ignored constraints.

Knapsack Algorithm:

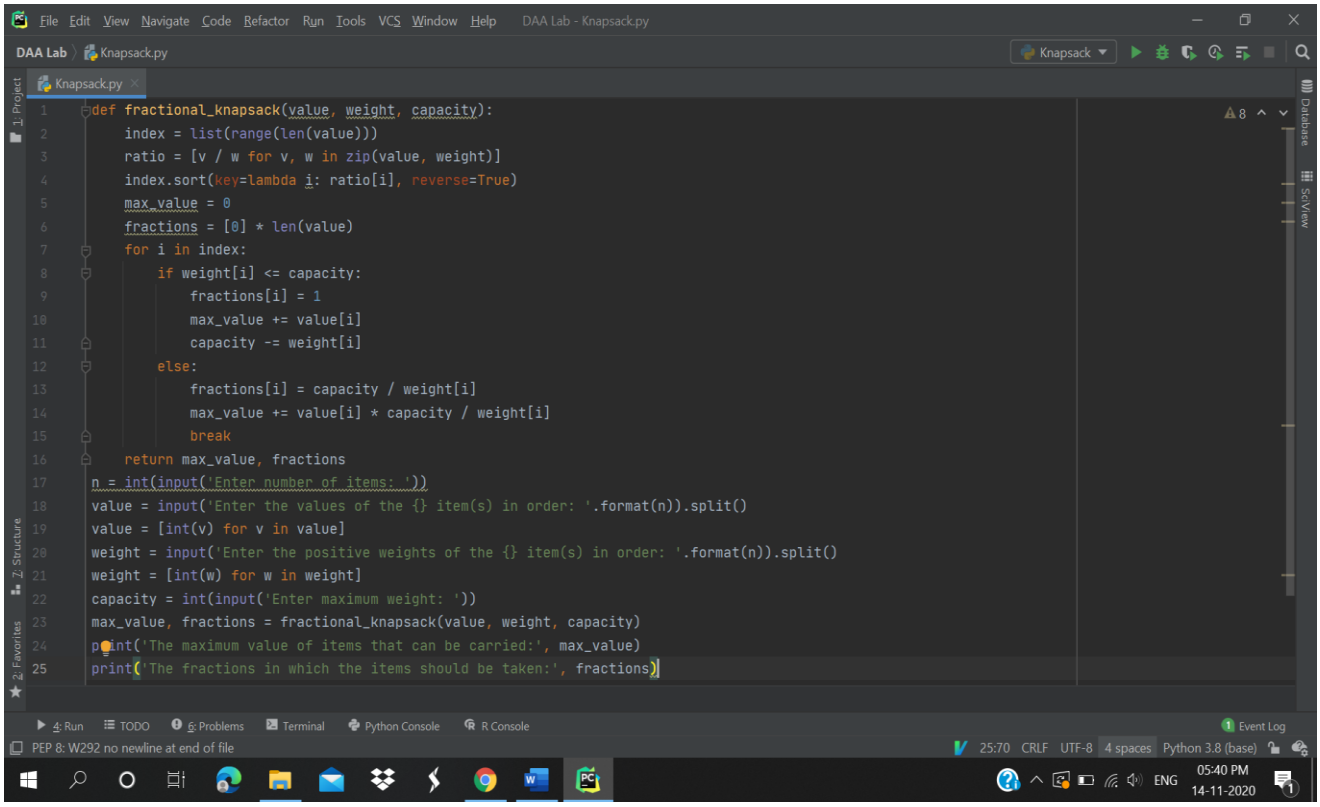
```
Greedy-Fractional-Knapsack (w[1..n], p[1..n], W)
for i = 1 to n
  do x[i] = 0
weight = 0
for i = 1 to n
  if weight + w[i] ≤ W then
    x[i] = 1
    weight = weight + w[i]
  else
    x[i] = (W - weight) / w[i]
    weight = W
    break
return x
```

Code:

```
def fractional_knapsack(value, weight, capacity):
    index = list(range(len(value)))
    ratio = [v / w for v, w in zip(value, weight)]
    index.sort(key=lambda i: ratio[i], reverse=True)
    max_value = 0
    fractions = [0] * len(value)
    for i in index:
        if weight[i] <= capacity:
            fractions[i] = 1
```

```
            max_value += value[i]
            capacity -= weight[i]
        else:
            fractions[i] = capacity / weight[i]
            max_value += value[i] * capacity / weight[i]
            break
    return max_value, fractions


n = int(input('Enter number of items: '))
value = input('Enter the values of the {} item(s) in order:
'.format(n)).split()
value = [int(v) for v in value]
weight = input('Enter the positive weights of the {} item(s) in order:
'.format(n)).split()
weight = [int(w) for w in weight]
capacity = int(input('Enter maximum weight: '))
max_value, fractions = fractional_knapsack(value, weight, capacity)
print('The maximum value of items that can be carried:', max_value)
print('The fractions in which the items should be taken:', fractions)
```



Output:

Enter number of items: 3

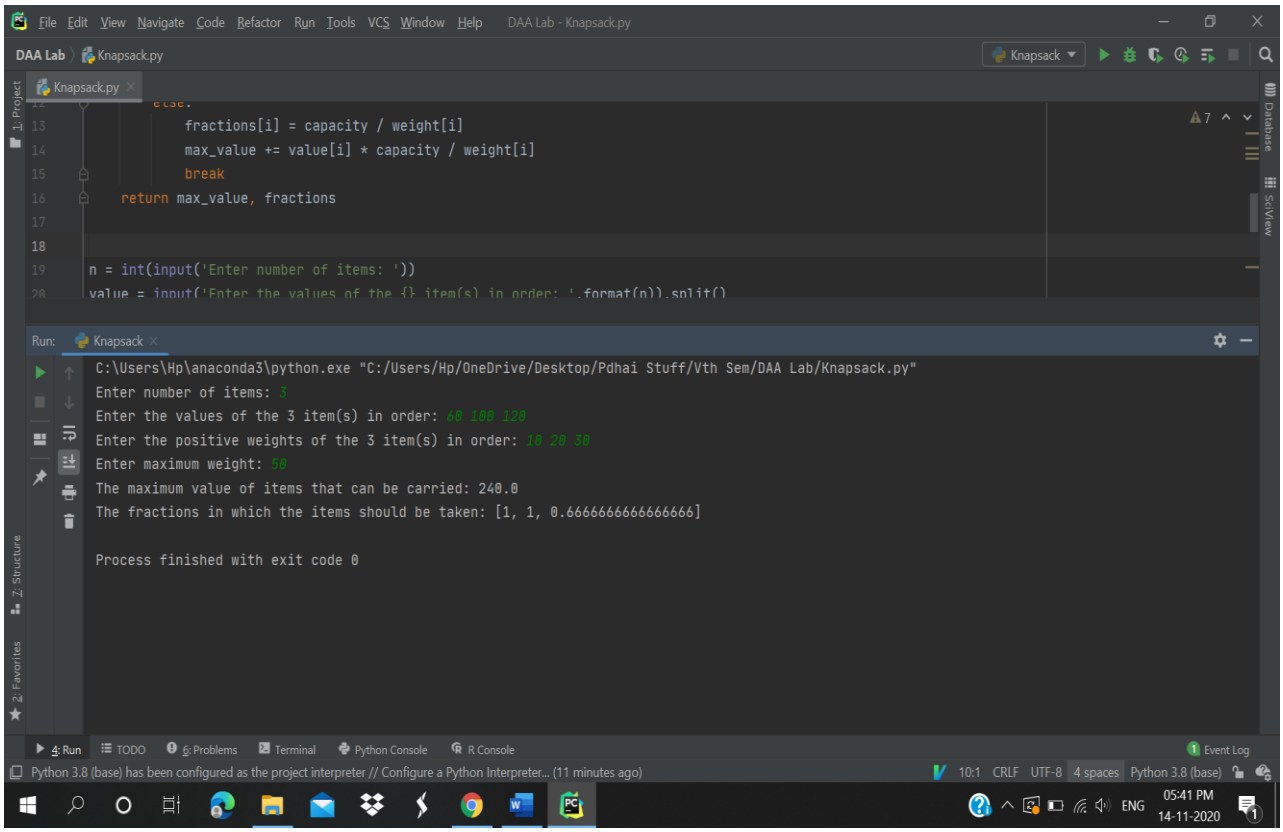
Enter the values of the 3 item(s) in order: 60 100 120

Enter the positive weights of the 3 item(s) in order: 10 20 30

Enter maximum weight: 50

The maximum value of items that can be carried: 240.0

The fractions in which the items should be taken: [1, 1, 0.6666666666666666]



Time Complexities:

If the provided items are already sorted into a decreasing order of pi/wi , then the while loop takes a time in *O(n)*; Therefore, the total time including the sort is in *O (n log n)*.

Knapsack Sort Applications:

- Finding the least wasteful way to cut raw materials
- portfolio optimization
- Cutting stock problems