Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

Algorithms and Complexity (COMP 314)

Lab 4

Submitted To:

Dr. Rajani Chulyadyo

Department of Computer Science and Engineering

Submitted By:

Mission Shrestha (54)

Year/Sem: III/II

Submission Date: 9th May, 2023

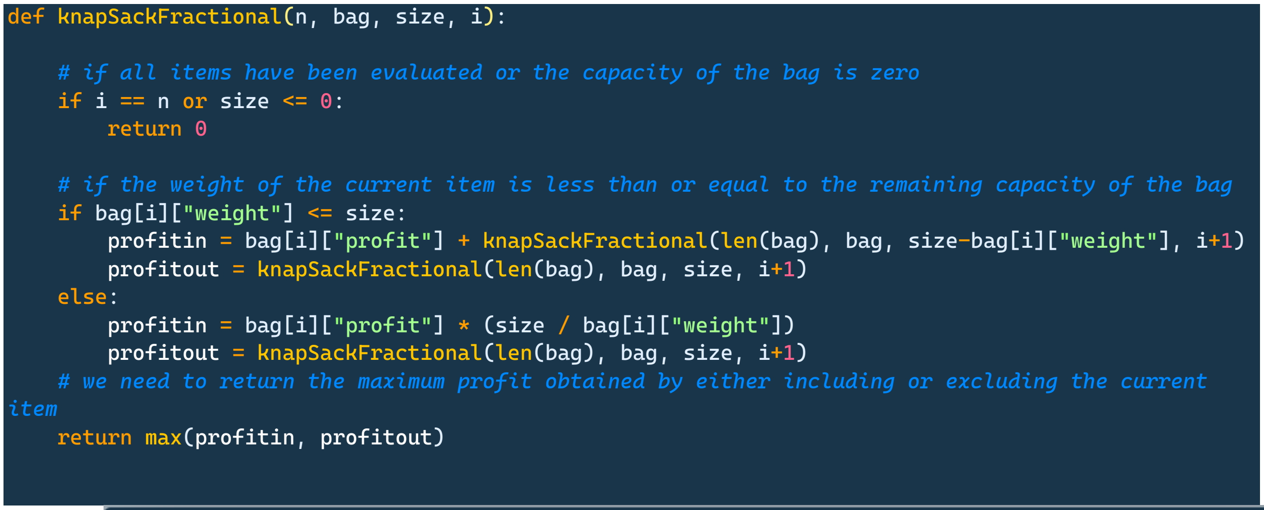
Solving Knapsack problem using different algorithm design strategies.

1. Brute-force method (Fractional Knapsack)

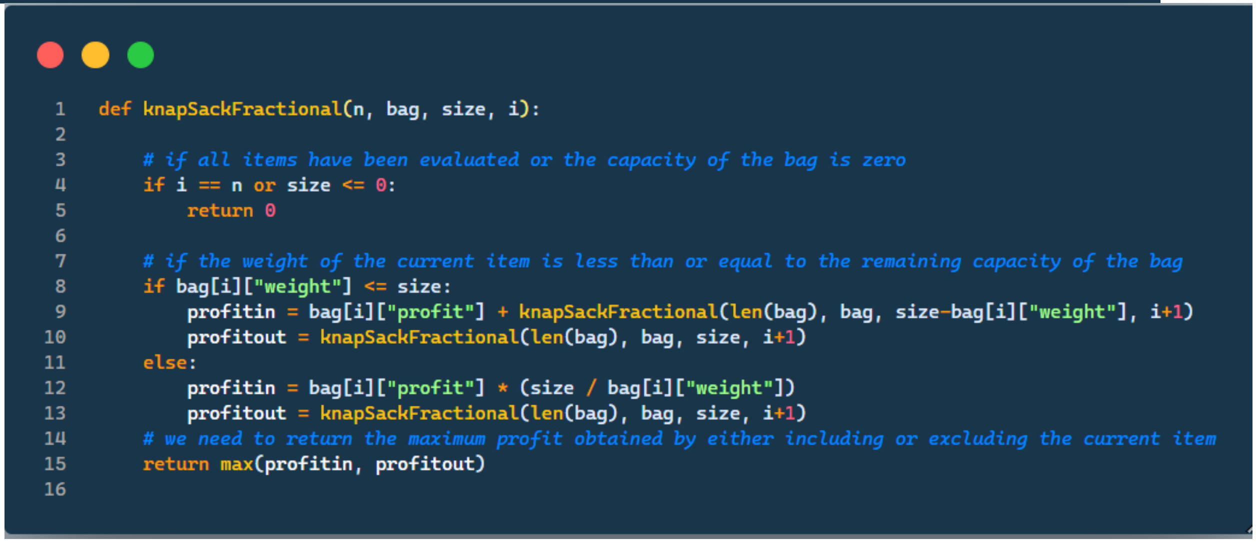
a. Pseudocode:

```
1. function knapSackFractional(n, bag, size, i):
    a. if i & n are equal OR size is less than or equal to 0:
        i. return 0

    b. if bag[i]["weight"] is less than or equal to size:
        i. profitin = bag[i]["profit"] + knapSackFractional (n, bag, size -
           bag[i]["weight"], i + 1)

        ii. profitout= knapSackFractional(n, bag, size, i + 1)

    c. else:
        i. profitin = bag[i]["profit"] * (size / bag[i]["weight"])
        ii. profitout = knapSackFractional(len(bag), bag, size, i+1)

    d. Return the maximum profit obtained by either including or excluding
       the current item
```

b. Source Code:



```python
def knapSackFractional(n, bag, size, i):

    # if all items have been evaluated or the capacity of the bag is zero
    if i == n or size <= 0:
        return 0

    # if the weight of the current item is less than or equal to the remaining capacity of the bag
    if bag[i]["weight"] <= size:
        profitin = bag[i]["profit"] + knapSackFractional(len(bag), bag, size-bag[i]["weight"], i+1)
        profitout = knapSackFractional(len(bag), bag, size, i+1)
    else:
        profitin = bag[i]["profit"] * (size / bag[i]["weight"])
        profitout = knapSackFractional(len(bag), bag, size, i+1)
    # we need to return the maximum profit obtained by either including or excluding the current
item
    return max(profitin, profitout)
```



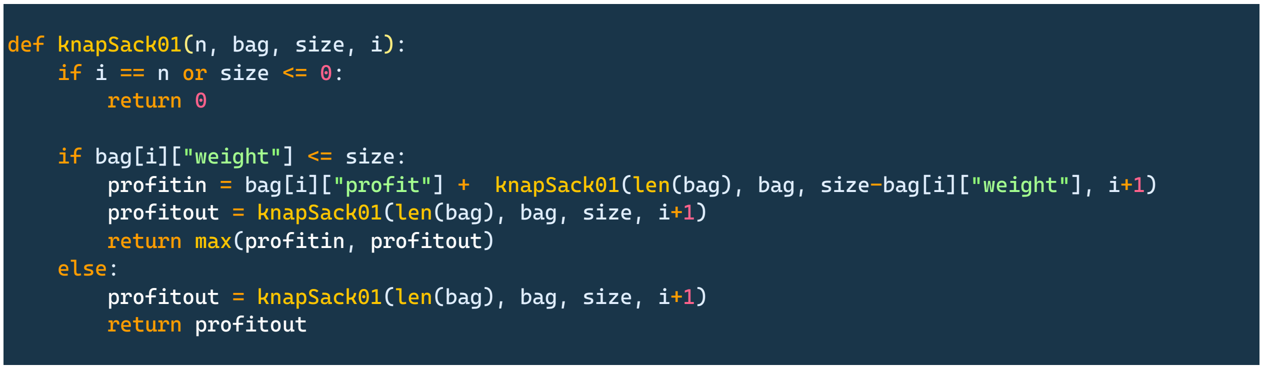
```python
def knapSackFractional(n, bag, size, i):

    # if all items have been evaluated or the capacity of the bag is zero
    if i == n or size <= 0:
        return 0

    # if the weight of the current item is less than or equal to the remaining capacity of the bag
    if bag[i]["weight"] <= size:
        profitin = bag[i]["profit"] + knapSackFractional(len(bag), bag, size-bag[i]["weight"], i+1)
        profitout = knapSackFractional(len(bag), bag, size, i+1)
    else:
        profitin = bag[i]["profit"] * (size / bag[i]["weight"])
        profitout = knapSackFractional(len(bag), bag, size, i+1)
    # we need to return the maximum profit obtained by either including or excluding the current item
    return max(profitin, profitout)

```

2. Brute-force method (0/1 Knapsack)

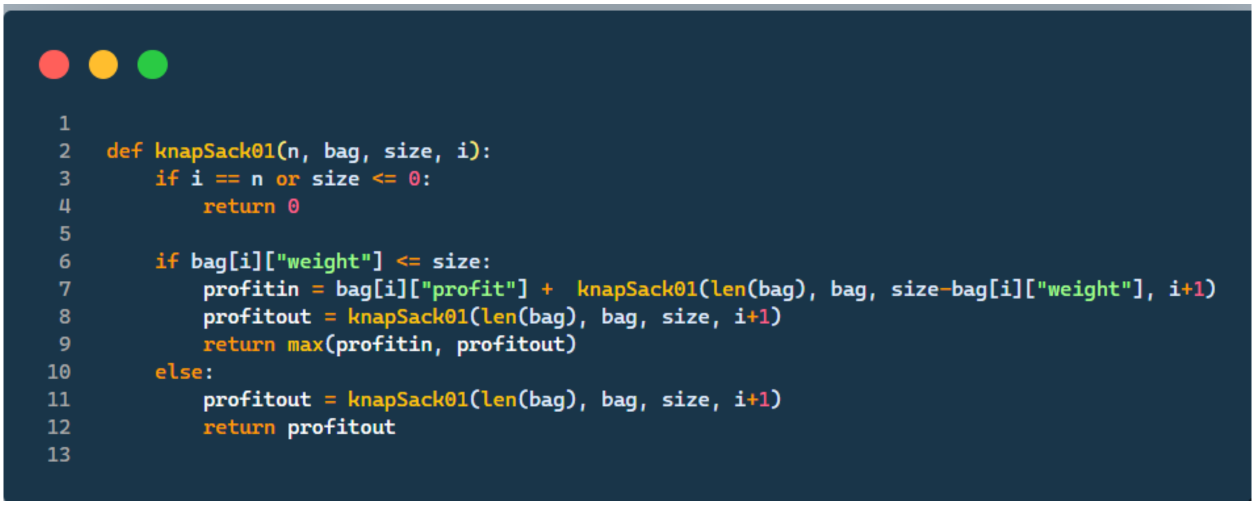
a. Pseudocode:

```
1. function knapSack01(n, bag, size, i):
    a. if i & n are equal OR size is less than or equal to 0:
        i. return 0

    b. if bag[i]["weight"] is less than or equal to size:
        i. profitin = bag[i]["profit"] + knapSack01(len(bag), bag, size-bag[i]["weight"], i+1)
        ii.  profitout = knapSack01(len(bag), bag, size, i+1)
        iii. Return the maximum profit obtained by either including or
             excluding the current item
    c. else:
        i.   profitout = knapSack01(len(bag), bag, size, i+1)
        ii.  return profitout
```

b. Source Code



```python
def knapSack01(n, bag, size, i):
    if i == n or size <= 0:
        return 0

    if bag[i]["weight"] <= size:
        profitin = bag[i]["profit"] +  knapSack01(len(bag), bag, size-bag[i]["weight"], i+1)
        profitout = knapSack01(len(bag), bag, size, i+1)
        return max(profitin, profitout)
    else:
        profitout = knapSack01(len(bag), bag, size, i+1)
        return profitout
```



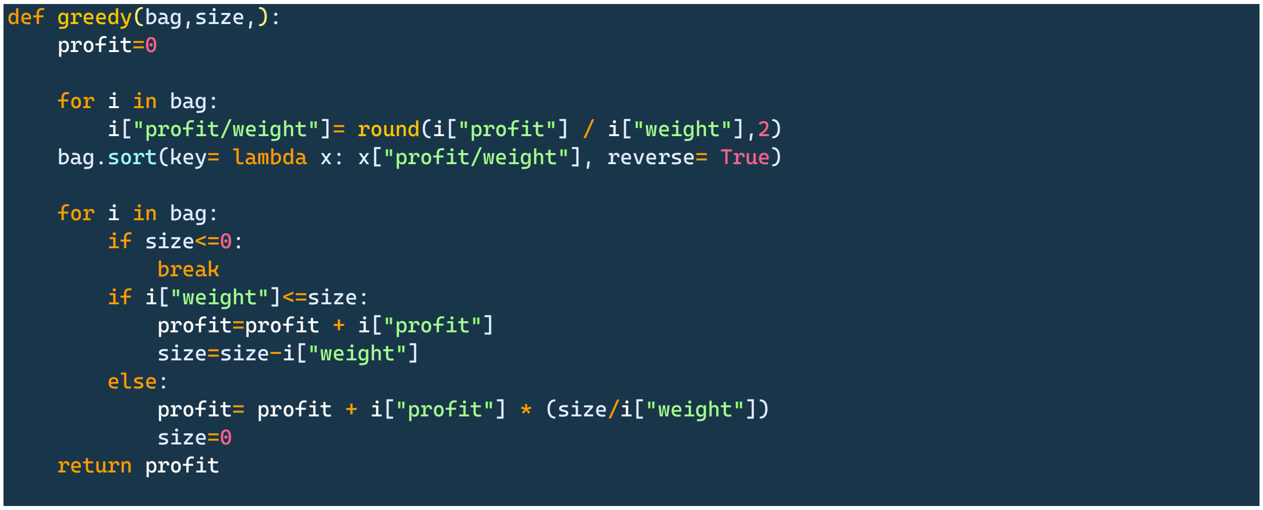
```python

def knapSack01(n, bag, size, i):
    if i == n or size <= 0:
        return 0

    if bag[i]["weight"] <= size:
        profitin = bag[i]["profit"] +  knapSack01(len(bag), bag, size-bag[i]["weight"], i+1)
        profitout = knapSack01(len(bag), bag, size, i+1)
        return max(profitin, profitout)
    else:
        profitout = knapSack01(len(bag), bag, size, i+1)
        return profitout

```

3. Greedy method (Fractional Knapsack)

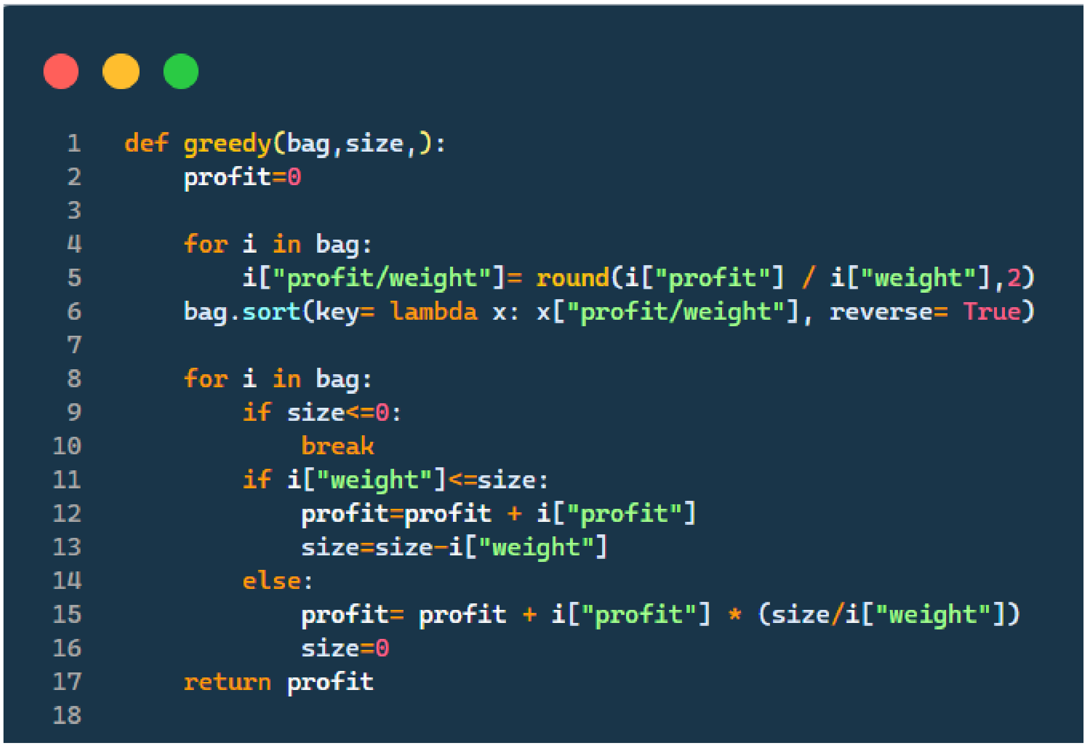
a. Pseudocode:

```
1. function greedy(bag, size):
    a. profit = 0
    b. for i in bag:
        i. i["profit/weight"] = round(i["profit"] / i["weight"], 2)
    c. sort(bag, by = "profit/weight", in descending order)
    d. for i in bag:
        i. if size is less than or equal to 0:
            1. break
        ii. if i["weight"] is less than or equal to size:
            1. profit = profit + i["profit"]
            2. size = size - i["weight"]
        iii. else:
            1. profit = profit + i["profit"] * (size/i["weight"])
            2. size = 0
    e. return profit
```

b. Code:



```python
def greedy(bag,size,):
    profit=0

    for i in bag:
        i["profit/weight"]= round(i["profit"] / i["weight"],2)
    bag.sort(key= lambda x: x["profit/weight"], reverse= True)

    for i in bag:
        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
```

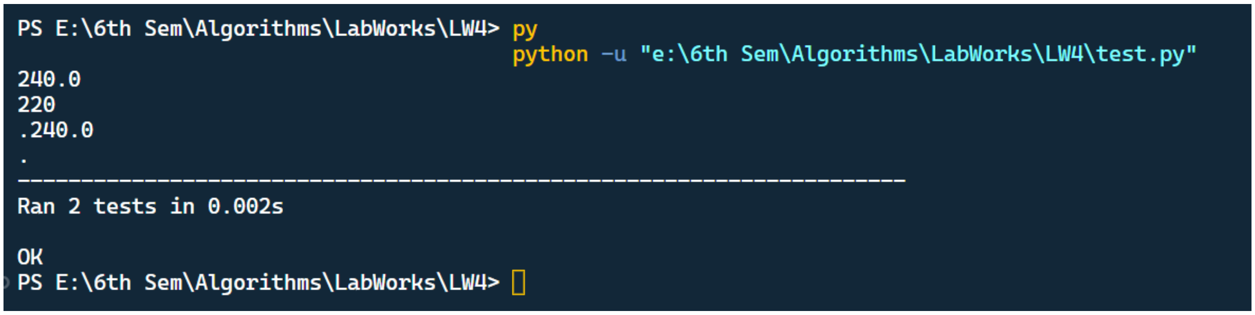


```python
def greedy(bag,size,):
    profit=0

    for i in bag:
        i["profit/weight"]= round(i["profit"] / i["weight"],2)
    bag.sort(key= lambda x: x["profit/weight"], reverse= True)

    for i in bag:
        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

```

4. Test cases

```python
import unittest
from greedy import greedy
from brute_force import knapSackFractional, knapSack01


class KnapSackTestCase(unittest.TestCase):
    def test_greedy(self):

        box = [
            {"profit": 60, "weight": 10},
            {"profit": 100, "weight": 20},
            {"profit": 120, "weight": 30},
        ]
        size = 50
        profit = greedy(box, size)
        print(profit)

        self.assertEqual(profit, 240)

    def test_brute(self):

        box = [
            {"profit": 60, "weight": 10},
            {"profit": 100, "weight": 20},
            {"profit": 120, "weight": 30},
        ]
        size = 50
        fractionalProfit = knapSackFractional(len(box), box, size, 0)
        print(fractionalProfit)
        self.assertEqual(fractionalProfit, 240)
        zeroneProfit = knapSack01(len(box), box, size, 0)
        print(zeroneProfit)
        self.assertEqual(zeroneProfit, 220)


if __name__ == "__main__":
    unittest.main()
```

When test cases were run:



```
PS E:\6th Sem\Algorithms\LabWorks\LW4> py
                                       python -u "e:\6th Sem\Algorithms\LabWorks\LW4\test.py"
240.0
220
.240.0
.
----------------------------------------------------------------------
Ran 2 tests in 0.002s

OK
PS E:\6th Sem\Algorithms\LabWorks\LW4> 
```

Conclusion:

The Knapsack problem were solved by using both Greedy method (for fractional knapsack) and Brute-force method (for both 0/1 knapsack and fractional knapsack).

The test cases were implemented and program was checked for the test cases. The program passed the test cases.