Part A

Aim:

1. Greedy Method
2. Design and analysis algorithms for Fractional Knapsack

Prerequisite: Any programming language

Outcome: Algorithms and their implementation

Theory:

Given weights and profits of n items, we need to put these items in a knapsack of capacity W to get the maximum total value in the knapsack.

In **Fractional Knapsack**, we can break items for maximizing the total value of knapsack. This problem in which we can break an item is also called the fractional knapsack problem.

Procedure:

1. Design algorithm and find best, average and worst-case complexity
2. Implement algorithm in any programming language.
3. Paste output

Practice Exercise:

S.no	Statement
1	Implement Fractional Knapsack using greedy method
3	Find the best, worst and average case complexity for 1.

Instructions:

1. Design, analysis and implement the algorithms.
2. Paste the snapshot of the output in input & output section.

Part B

Algorithm :

finding maximum profit that can be obtained using greedy approach.(fractional knapsack)

Input: List of items along with their values(profits) and weights

Output: maximum profit that can be obtained with the given items along with the items to be selected to get maximum profit

```
def knapsack(obj,value,weight,capacity):
    pw=[0.0]*len(obj)
    for i in range(len(obj)):
        pw[i]=value[i]/weight[i]
    max_value=0
    c1=capacity
    while(capacity>0):
        m=pw.index(max(pw))
        if capacity>weight[m]:
```

max_value+=value[m]
capacity-=weight[m]
items_taken.append(obj[m])
weights_taken.append(weight[m])
pw[m]=0.0
else:
max_value+=(capacity*value[m])/weight[m]
weights_taken.append(capacity)
capacity-=capacity
items_taken.append(obj[m])
pw[m]=0.0
printing maximum profit obtained,items taken, and their respective weights

Code:

Fractional Knapsack:

```
def knapsack(obj,value,weight,capacity):
    pw=[0.0]*len(obj)
    for i in range(len(obj)):
        pw[i]=value[i]/weight[i]
        # if weight[i]>capacity:
        #     pw[i]=0.0
    max_value=0
    c1=capacity
    while(capacity>0):
        m=pw.index(max(pw))
        if capacity>weight[m]:
            max_value+=value[m]
            capacity-=weight[m]
            items_taken.append(obj[m])
            weights_taken.append(weight[m])
            pw[m]=0.0
        else:
            max_value+=(capacity*value[m])/weight[m]
            weights_taken.append(capacity)
            capacity-=capacity
            items_taken.append(obj[m])
            pw[m]=0.0
    return max_value

items_taken=[]
weights_taken=[]
obj=list(map(str,input('list of items : ').split()))
value=list(map(int,input('Corresponding values : ').split()))
weight=list(map(int,input('Corresponding weights : ').split()))
capacity=int(input('capacity : '))
```

```
max_value=knapsack(obj,value,weight,capacity)
print("maximum profit obtained with given capacity from given list of items
with corresponding profits and weights is",max_value)
print("items taken :",items_taken)
print("weights are :",weights_taken)
```

Input & Output:

```
PS E:\books and pdfs\sem4 pdfs\DAA lab\week6> python .\fractional_knapsack.py
list of items : A B C D E F G H
Corresponding values : 10 5 15 7 6 18 3 45
Corresponding weights : 2 3 5 7 1 4 1 16
capacity : 15
maximum profit obtained with given capacity from given list of items with corresponding profits and weights is 57.625
items taken : ['E', 'A', 'F', 'C', 'G', 'H']
weights are : [1, 2, 4, 5, 1, 2]
PS E:\books and pdfs\sem4 pdfs\DAA lab\week6>
```

Best, worst and average case complexity for Fractional Knapsack problem:

Time complexity

If we use quick sort for sorting, time complexity for sorting will be will be O(n log n) and O(n) to run over the loop .

Time complexity of the sorting + Time complexity of the loop to maximize profit = O(nlog n) + O(n) = O(nlog n)

Best case: O(n log n)
Average case: O(n log n)
Worst case: O(n^2) (as quick sort worst case time complexity is O(n^2) total time complexity of fractional knapsack problem will be equal to O(n^2)+O(n)=O(n^2))

But instead of quick sort , If we consider bubble sort or selection sort algorithms then complexity of dorting would be O(n^2) so total total complexity would be O(n^2)+ O(n)=O(n^2)

Space complexity:

space complexity for Fractional Knapsack problem O(n)

Observation & Learning:

I have observed and learned that

i) greedy approach would guarantee optimal solution only to fractional knapsack problem and doesnt guarantee optimal solution for 0-1 knapsack problem.

ii) Sorting of items will be done in decreasing order of value(i.e. profit)/weight ratio.

iii) Brute-force solution would be used to try all possible subset with all different fraction but that will be too much time taking but efficient solution is to use Greedy approach

Conclusion:

I have implemented fractional knapsack problem using greedy approach in python language.

Questions:

1. Is it possible to solve 0/1 Knapsack using Greedy Method? Justify your answer.

Answers:

1) 0-1 Knapsack cannot be solved by Greedy approach. Greedy approach does not ensure an optimal solution. In many instances, Greedy approach may give an optimal solution.

As objects / items are indivisible greedy approach doesnt gaurenty optimal solution.

eg: If capacity is 6 for below example

Item	Value	Size	Value/Size
A	5.5	4	1.38
B	4	3	1.33
C	4	3	1.33

If we use greedy approach for above problem for 0-1 knapsack,
As we cannot divide objects we can select only one object i.e. ITEM A with weight 4 and value **5.5** (maximum value/size among those)

But if we use fractional knapsack for same problem optimal solution would be, 5.5+ 4*(2/3)
where we can select ITEM A as well as ⅔ th portion of B
so optimal value using fractional knapsack would be **8.16666**

Therefore it is clearly proved that greedy approach for 0-1 knapsack problem doesnot guarantee optimal solution