Part A

Aim:
1. Dynamic programming
2. 0/1 Knapsack problem

Prerequisite: Any programming language

Outcome: Algorithms and their implementation

Theory:

Given weights and profits of n items, put these items in a knapsack of capacity W to get the maximum total profit in the knapsack.
In other words, given two integer arrays p[0..n-1] and wt[0..n-1] which represent profits and weights associated with n items respectively. Also given an integer W which represents knapsack capacity, find out the maximum value subset of p[] such that sum of the weights of this subset is smaller than or equal to W. You cannot break an item, either pick the complete item or don't pick it (0-1 property).

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 the 0/1 Knapsack using Dynamic Programming.
2	Find the run time complexity of the above algorithm

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

Part B

Algorithm:

Input: Capacity and Items with their respective profits(values) and weights
Output: Maximum profit that can be obtained using 0-1 Knapsack and the items that have contributed for maximum profit.

Algorithm for finding maximum profit:

```
def knapSack(capacity,weight,profit,num,k):
   for i in range(num + 1):
      for j in range(capacity + 1):
```

```
        if i == 0 or j == 0:
          K[i][j] = 0
        elif weight[i-1] <= j:
          K[i][j] = max(profit[i-1] + K[i-1][j-weight[i-1]],  K[i-1][j])

        else:
          K[i][j] = K[i-1][j]


  return K[num][capacity]
```

This will return the maximum profit that can be obtained.

To find the items that have contributed for maximum profit:

```
def contributors_maxprofit(K):
   j=len(K)
   contributors=[0]*num
   while(j>=0):

      if max_profit in K[j-2]:
         j-=1
     else:
         contributors[j-2]=1
         print("~ item ",j-1,end=" ")
         max_profit-=profit[j-2]
         j-=1
  return contributors
```

We can get contributors of maximum profit.

Code:

```
def knapSack(capacity,weight,profit,num):
  global K
  for i in range(num + 1):
    for j in range(capacity + 1):
      if i == 0 or j == 0:
        K[i][j] = 0
      elif weight[i-1] <= j:
        K[i][j] = max(profit[i-1] + K[i-1][j-weight[i-1]],  K[i-1][j])
```

```
            else:
                K[i][j] = K[i-1][j]


    return K[num][capacity]

profit =list(map(int,input('profit (value) : ').split(',')))
weight = list(map(int,input('respective weights : ').split(',')))
capacity = int(input('capacity : '))
num = len(profit)
K = [[0 for x in range(capacity + 1)] for x in range(num + 1)]

max_profit=knapSack(capacity,weight,profit,num)
print('_______________________________________________\n\nMaximum profit that can be obtained is',max_profit)
print('Items contributed for maximum profit are:',end=" ")
j=len(K)
contributors=[0]*num
while(j>=0):

    if max_profit in K[j-2]:
        j-=1
    else:
        contributors[j-2]=1
        print("~ item ",j-1,end=" ")
        max_profit-=profit[j-2]
        j-=1

print("\nTherefore,Contributors list is ",contributors)
```

Input and Output:

```
PS E:\books and pdfs\sem4 pdfs\DAA lab\week9> python .\knapsack01.py
profit (value) :  10,5,15,7,6,18,3
respective weights :  2,3,5,7,1,4,1
capacity :  15
__________________________________________

Maximum profit that can be obtained is 54
Items contributed for maximum profit are: ~ item  6 ~ item  5 ~ item  3 ~ item  2 ~ item  1
Therefore,Contributors list is  [1, 1, 1, 0, 1, 1, 0]
```

Run time complexity of 0/1 Knapsack: It takes O(nw) time to fill (n+1)(w+1) table entries.It takes O(n) time for tracing the solution since the tracing process traces the n rows. Thus, overall O(nw) time is taken to solve the 0/1 knapsack problem using dynamic programming. Therefore, The time complexity of the 0/1 Knapsack problem is O(nw) where n is the number of items and w is the capacity of the knapsack. Space complexity: O(n*w) As we are using a 2-D list instead of a 1-D list.
Observation & Learning: I have observed and learned that i) greedy approach doesn’t guarantee an optimal solution for the 0/1 knapsack problem and would guarantee an optimal solution only to the fractional knapsack problem. ii) 0–1 Knapsack problem, we are not allowed to break items. We either take the whole item or don’t take it. iii) Knapsack problem has a pseudo-polynomial solution and is thus called weakly NP-Complete
Conclusion: I have successfully implemented 0/1 knapsack problem in python programming language.