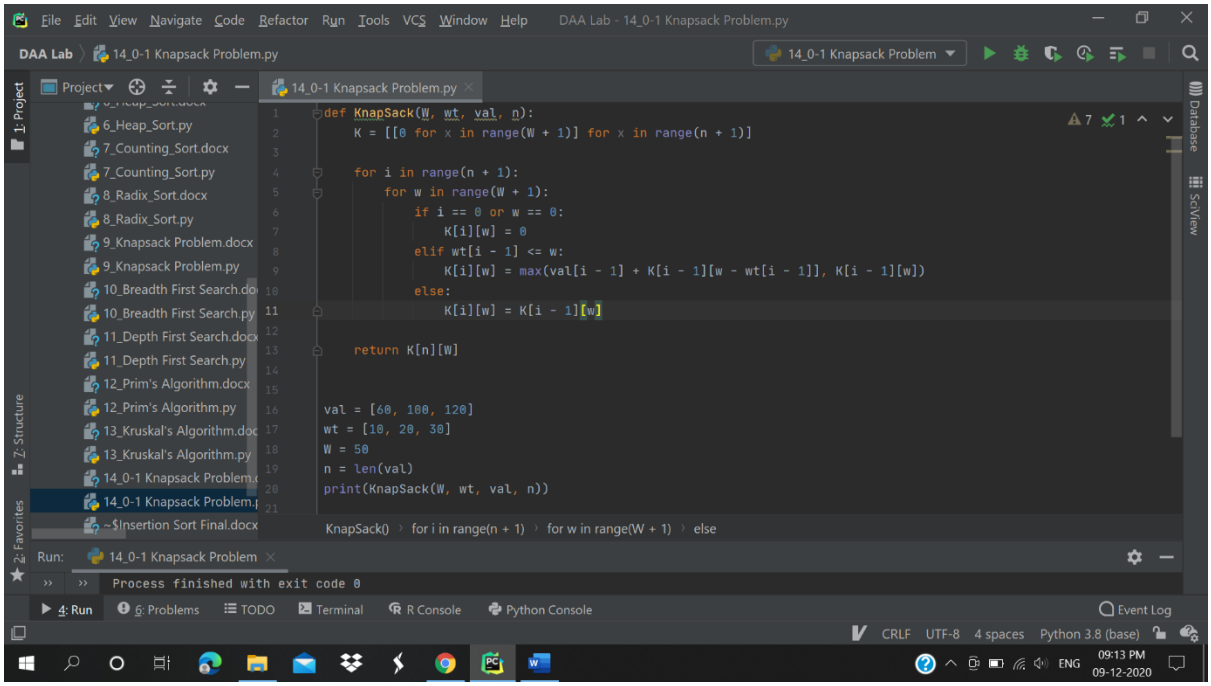
Objective: **Implementation and analysis of 0-1Knapsack**

0-1 Knapsack

Given weights and values of n items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack. In other words, given two integer arrays val[0..n-1] and wt[0..n-1] which represent values and weights associated with n items respectively. Also given an integer W which represents knapsack capacity, find out the maximum value subset of val[] 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).

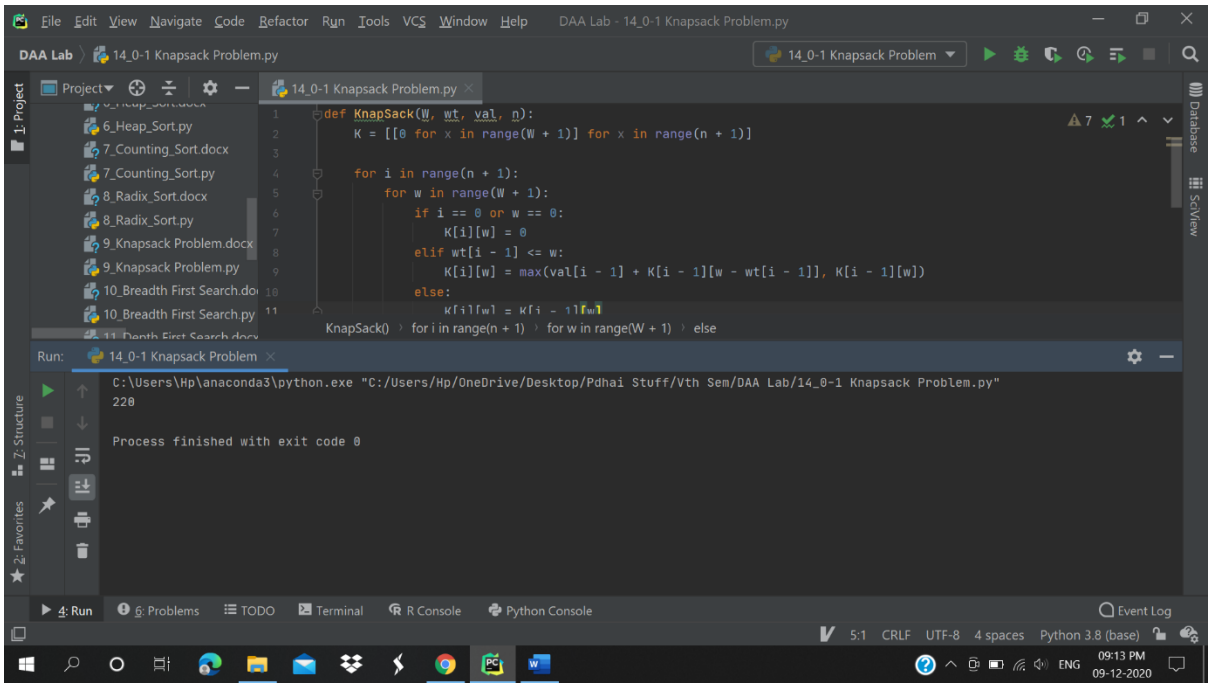
Code:

```
def KnapSack(W, wt, val, n):
    K = [[0 for x in range(W + 1)] for x in range(n + 1)]

    for i in range(n + 1):
        for w in range(W + 1):
            if i == 0 or w == 0:
                K[i][w] = 0
            elif wt[i - 1] <= w:
                K[i][w] = max(val[i - 1] + K[i - 1][w - wt[i - 1]], K[i - 1][w])
            else:
                K[i][w] = K[i - 1][w]

    return K[n][W]


val = [60, 100, 120]
wt = [10, 20, 30]
W = 50
n = len(val)
print(KnapSack(W, wt, val, n))
```


File Edit View Navigate Code Refactor Run Tools VCS Window Help
DAA Lab - 14_0-1 Knapsack Problem.py
DAA Lab
14_0-1 Knapsack Problem.py
14_0-1 Knapsack Problem
Project
14_0-1 Knapsack Problem.py
6_Heap_Sort.py
7_Counting_Sort.docx
7_Counting_Sort.py
8_Radix_Sort.docx
8_Radix_Sort.py
9_Knapsack Problem.docx
9_Knapsack Problem.py
10_Breadth First Search.doc
10_Breadth First Search.py
11_Depth First Search.docx
11_Depth First Search.py
12_Prim's Algorithm.docx
12_Prim's Algorithm.py
13_Kruskal's Algorithm.doc
13_Kruskal's Algorithm.py
14_0-1 Knapsack Problem.p
14_0-1 Knapsack Problem.p
~$Insertion Sort Final.docx
def KnapSack(W, wt, val, n):
K = [[0 for x in range(W + 1)] for x in range(n + 1)]
for i in range(n + 1):
for w in range(W + 1):
if i == 0 or w == 0:
K[i][w] = 0
elif wt[i - 1] <= w:
K[i][w] = max(val[i - 1] + K[i - 1][w - wt[i - 1]], K[i - 1][w])
else:
K[i][w] = K[i - 1][w]
return K[n][W]
val = [60, 100, 120]
wt = [10, 20, 30]
W = 50
n = len(val)
print(KnapSack(W, wt, val, n))
KnapSack() for i in range(n + 1) for w in range(W + 1) else
Run: 14_0-1 Knapsack Problem
Process finished with exit code 0
4: Run
6: Problems
TODO
Terminal
R Console
Python Console
Event Log
CRLF UTF-8 4 spaces Python 3.8 (base)
ENG
09:13 PM
09-12-2020

Output:

220


File Edit View Navigate Code Refactor Run Tools VCS Window Help
DAA Lab - 14_0-1 Knapsack Problem.py
DAA Lab
14_0-1 Knapsack Problem.py
14_0-1 Knapsack Problem
Project
14_0-1 Knapsack Problem.py
6_Heap_Sort.py
7_Counting_Sort.docx
7_Counting_Sort.py
8_Radix_Sort.docx
8_Radix_Sort.py
9_Knapsack Problem.docx
9_Knapsack Problem.py
10_Breadth First Search.doc
10_Breadth First Search.py
def KnapSack(W, wt, val, n):
K = [[0 for x in range(W + 1)] for x in range(n + 1)]
for i in range(n + 1):
for w in range(W + 1):
if i == 0 or w == 0:
K[i][w] = 0
elif wt[i - 1] <= w:
K[i][w] = max(val[i - 1] + K[i - 1][w - wt[i - 1]], K[i - 1][w])
else:
KnapSack() for i in range(n + 1) for w in range(W + 1) else
Run: 14_0-1 Knapsack Problem
C:\Users\Hp\anaconda3\python.exe "C:/Users/Hp/OneDrive/Desktop/Pdhai Stuff/Vth Sem/DAA Lab/14_0-1 Knapsack Problem.py"
220
Process finished with exit code 0
4: Run
6: Problems
TODO
Terminal
R Console
Python Console
Event Log
5:1 CRLF UTF-8 4 spaces Python 3.8 (base)
ENG
09:13 PM
09-12-2020

Time Complexity: O(N*W).
where ‘N’ is the number of weight element and ‘W’ is capacity. As for every weight element we traverse through all weight capacities 1<=w<=W.

Auxiliary Space: O(N*W).
The use of 2-D array of size ‘N*W’.