Vikramaditya Reddy
2197367g.

ASSIGNMENT-6

Q1. 0-1 Knapsack Problem, Total weight limit = 8

Item no	1	2	3	4
Value	48	40	30	54
Weight	1	2	3	6

Sum[K,S] ← max[sum[K-1, S], sum[K-1, S-w[K] + v[K]]]

	S=0	1	2	3	4	5	6	7	8
K=0	0	0	0	0	0	0	0	0	0
1	0	48	48	48	48	48	48	48	48
2	0	48	40	48	48	48	48	48	48
3	0	48	40	48	48	48	78	78	78
4	0	48	40	48	48	48	78	78	(94)

Hence maximum profit is [94] with the items 2 and 4 with respetive weights 2+6 = 8.

Q2. Trace back algorithm.

Step 1: First find max value

Step 2: Start at bottom right cell (max_value)

Step 3: Check current value in cell and compare with value of the cell above. If its equal current corresponding to current value is not selected.

Step 4: If its not equal, current row is selected
Add its index to selected items
Subtract its W and value.
Move cell diagonally above and to the left

Step 5: Repeat Step 3&4 until you reach top of the table.

Step 6: Final selected items are the required items.

Q2. Traceback algorithm to determine the exact subset of items

```
def knap(weights,values,W):
    n=len(weights)

# 2D array
    table=[]
    for i in range(n+1):
        t=[0]*(W+1)
        table.append(t)

#sub-problems
    for i in range(1,n+1):
        for w in range(1,W+1):
            if weights[i-1] > w:
                table[i][w] = table[i -1][w]
            else:
# Choose Max value
                table[i][w]=max(table[i-1][w],values[i-1]+table[i-1][w-weights[i-1]])

#last cell
    max_value=table[-1][-1]
    return table,max_value


def backtrace(table,weights,values,W):
    n=len(weights)

#empty list
    items=[]
    i, w = n, W
    while i > 0 and w > 0:
        if table[i][w]!=table[i - 1][w]:
            items.append(i-1)
            w =w-weights[i-1]
        i=i-1
    items.reverse()

    return [(weights[i],values[i]) for i in items]


#Inputs from previous Question
weights=[1,2,3,6]
values=[8,40,30,54]
W=8
table, max_value=knap(weights,values,W)
items = backtrace(table,weights,values,W)

print("Max Value:",max_value)
print("Items:",items)
```

```
Max Value: 94
Items: [(2, 40), (6, 54)]
```

here the max value that does exceed weight limit of 8 is 94 and the items are 2 and 4 with weights 2 and 6 with corresponding values of 40 and 54

Q3. Dynamic programming algorithm to find longest monotonically increasing the subsequence of a sequence of n integers

(i/p)

Step1: Create an array X of size 'n' and initialize its value to 1.

Step2: Iterate through the input sequence.

Step3: For each element, Find all previous elements that are smaller than. current value and update it.

Step4: Find max value in array X and return it

(c) Time Complexity:

Time complexity is $O(n^2)$

Since we are using loops from

Q3.Longest monotonically increasing subsequence of a sequence of n integers.

```
def subseq(a):
    n = len(a)
    x = [1] * n
    for i in range(1, n):
        for j in range(i):
            if a[j]<a[i]:
                x[i]=max(x[i],x[j]+1)
    return max(x)
```

```
# Sequence
a =[53,91,50,1,70,88,12,19]
print("Length of subsequence is ", subseq(a))
```

```
Length of subsequence is  3
```

(a) Recurrence Relation:

$$X[i] = \max\{X[j]+1\} \quad \begin{cases} j < i \\ a[j] < a[i] \end{cases}$$

Base case is $X[i] = 1$, as subsequence ending at index i is element $a[i]$.

Q4) Dynamic Programming algorithm. to find change of n cents using Fewest coins for K denominations of coins.

(a) D[1.... K] are Denominations

c[n] be coins required

$$c[n] = \min\{c[n-D[i]]+1\} \quad \left.\begin{array}{l} \text{for } i=1 \text{ to } K \\ D[i] <= n. \end{array}\right\}$$

Base cases

$C[0] = 0$

$C[n] = \infty$ for $n<0$.

(b) Outer loop iterates n times, for K different denominations.

∴ Time complexity $\boxed{O(nk)}$.

(b) Step 1 : Initialize array C with length amt+1 (from 0 to amt).

Step 2 : C[0] = 0 for the base case with 0 coins

Step 3 : Compute minimum no. of coins C[i] = min(C[i], C[i-D.]+1)

Step 4 : Return C[amt].

Q4. Algorithm to find the change of n cents using the fewest number of coins given any set of k different coin denominations

```
def Coin(D,n):
    # Initializing an array to store upto n+1
    C=[float('inf')]*(n+1)
    C[0] = 0

# Computing minimum number of coins
    for i in range(1,n+1):
        for j in range(len(D)):
            if D[j]<=i:
                C[i]=min(C[i],C[i-D[j]]+1)
    return C[n]
```

```
coins=[1, 5, 10, 25]
amt=17
mincoins=Coin(coins,amt)
print(f"minimum coins required for {amt} cents are {mincoins}")
```

```
minimum coins required for 17 cents are 4
```

```
coins=[1, 5, 10, 25]
amt=75
mincoins=Coin(coins,amt)
print(f"Minimum coins required for {amt} cents are {mincoins}")
```

```
Minimum coins required for 75 cents are 3
```

```
coins=[1, 5, 10, 25]
amt=81
mincoins=Coin(coins,amt)
print(f"Minimum coins required for {amt} cents are {mincoins}")
```

```
Minimum coins required for 81 cents are 5
```