Cost analysis of the algorithm (c_2 → problem_1.py)

```
def knapsackProblem(itemValues,itemWeights,maxCapacity):
    n = len(itemValues)                                          #c1=1
    dpTable = [[0 for _ in range(maxCapacity+1)] for _ in range(n+1)]
#c2=n.m

    for i in range(1, n+1):                                      #c3=n
        for capacity in range(1, maxCapacity+1):                 #c4=n.m
            dpTable[i][capacity] = dpTable[i-1][capacity]        #c5=n.m

            if itemWeights[i-1] <= capacity:                     #c6=n.m
                items = itemValues[i-1] + dpTable[i-1][capacity -
itemWeights[i-1]]                                                #c7=n.m
                if items > dpTable[i][capacity]:                 #c8=n.m
                    dpTable[i][capacity] = items                 #c9=n.m

    maxValue = dpTable[n][maxCapacity]                           #c10=1
    selectedItems = []                                           #c11=1
    capacity = maxCapacity                                       #c12=1

    for i in range(n, 0, -1):                                    #c13=n
        if dpTable[i][capacity] != dpTable[i-1][capacity]:       #c14=n
            selectedItems.append(i-1)                            #c15=n
            capacity = capacity - itemWeights[i-1]               #c16=n

    selectedItems.reverse()                                      #c17=n
    return maxValue, selectedItems                               #c18=1
```

- Basic operation:
  $c_4, c_5, c_6, c_7, c_8, c_9 = n.m$

- Time complexity calculation:
  $T(n) = (c_4 + c_5 + c_6 + c_7 + c_8 + c_9).(n.m)$
  $T(n) = 6(n.m)$
  $T(n) = n.m$
  $T(n) \in O(n.m)$

- Solving the recurrence:
  $T(n) = T(n-1) + 6m,\ n >= 1, T(0) = 0$
  $T(1) = T(1-1) + 6m = T(0) + 6m = 0 + 6m = 6m$
  $T(2) = T(2-1) + 6m = T(1) + 6m = 6m + 6m = 12m$

$$T(3) = T(3-1) + 6m = T(2) + 6m = 12m + 6m = 18m$$
$$T(4) = T(4-1) + 6m = T(3) + 6m = 18m + 6m = 24m$$
$$T(n) = \sum_{i=1}^{n} 6.m$$