Answer:

a) The obvious combinational approach takes theta(2^n), because for each of the n items, we either select it or not select it (2 possibilities), so there are 2^n combinations, and we search all of them, excluding invalid combinations and comparing all valid ones to obtain the one with maximum value. This approach is extremely slow – the museum guards would find you before the algorithm was done.

b)

Define $m[i, w]$ to be the maximum value that can be attained with weight less than or equal to w using items up to i (first i items).

We can define $m[i, w]$ recursively as follows:

- $m[0, w] = 0$ -> 0 items have 0 value
- $m[i, w] = m[i-1, w]$ if $w_i > w$ (the new item is more than the current weight limit)

  ->If the item exceeds the weight limit, then look at the next item

- $m[i, w] = \max(m[i-1, w], m[i-1, w-w_i] + v_i)$ if $w_i \leqslant w$.

  ->If it doesn't exceed the weight limit, choose whether or not to take this item based on whether or not the recursion on the rest of the items will yield a larger value

The solution can then be found by calculating $m[n, W]$. To do this efficiently we can use a table to store previous computations.

The following is pseudo code for the dynamic program:

```
// Input:
// Values (stored in array v)
// Weights (stored in array w)
// Number of distinct items (n)
// Knapsack capacity (W)

for j from 0 to W do:
    m[0, j] := 0

for i from 1 to n do:
    for j from 0 to W do:
        if w[i-1] > w[j] then:
```

```
13              m[i, j] := m[i-1, j]
14          else:
15              m[i, j] := max(m[i-1, j], m[i-1, j-(i-1)] + v[i-1])
```

This runs in O(nW), where n is the number of items and W is the number of pounds the backpack can carry.