Imagine you are a thief who has broken inside a museum, but you only have a small backpack to carry your loot. You want to maximize the value of the things you steal, while minimizing their weight (so you can fit more inside the backpack). For example, you can have:

-A painting that weighs 10 pounds and is worth 1 million dollars

-A vase that weighs 2 pounds and is worth 200 thousand dollars

-A piece of jewelry that weights 1 pound and is worth 10 million dollars

Being a very smart thief, you decide to create an algorithm that will calculate the best combination for you, so that, when you enter the museum and avail the items at your disposal, you can just input them into the program. Since you are under risk of being discovered by the guards at any moment, make this algorithm run in optimum time.

a) The "dumb" approach would be to calculate every possible combination of items, then choose the one with the highest value that still fits in the backpack. Calculate the time complexity of this algorithm.
b) Since you really need this heist to go well, you decide to create a faster algorithm. You call your computer geek friends from Purdue, and they give you some hints:
   - If the backpack can fit W pounds, the best way to utilize that weight capacity is the combination of the best ways to utilize the individual subsets of w pounds each (where w and W are integers).
   - With this metric, the constraints you have to consider are whether or not w is larger than the weight of the partition you are analyzing now and if there are better ways of utilizing this weight. The only decision you have to make is whether or not to take an object with you.
   - Using this, you can think of a metric m[i,w] that is defined as the maximum value of items among the first i items that weigh at most w pounds. If you order the items by value, you can keep computing m[i,w] from m[i-1,w] until you computer m[n,W] where n is the number of items and W is the weight constraint.

     Note: Define v[i] as the value of item i

   Given all this, construct a recursive algorithm that works in better time than the one from part a and give its runtime complexity.