CS 330 homework

1)

Items	1	2	3	4
weight	22	8	6	2
value	6	5	3	5
weighted value v_i	0.2727	0.625	0.5	2.5

Knapsack = 100
Here it would be optimal for profit 250, choosing 50 item 4's, with value 5.

Example where it doesn't work

Items	1	2	3	4
Weight	99	100	98	97
value	50	51	50	50
weighted val	.50	.51	.5102	.5157

Knapsack = 100
Here, the algorithm would fit item 4 once with profit 50, but the real max profit is item 2:

	optimal sol'n	greedy rule's
Item	2	4
Profit	51	50
weight	100	97

2. Base Case:

$M(0) = 0$, $0 \le w \le W$ (max Knapsack capacity)

// $M(w)$ = Max obtainable value

Recurrence Relation:

$\max(\text{for } (i \ldots n)) \{ M(w - w_i) + v_i, \text{ if } w_i \le w \} = M(W)$

This equation finds all possibilities of items to add, and add it all to an array, then we use the max function to find the highest value. It produces an array with all possible values given that W = capacity, so it already forgets about greater weights that do not fit into knapsack.

3. You can use this relation with dynamic programming by defining another for loop for $0 \le w \le W$ and merging the base case into our code. Here's an example

```
def dynamic_prog(A[], N, W):
    // A[] is the given items, N is number of items, W is Knapsack capacity
    M[] = array of length W+1
    M[0] = 0
    // ^ base case
    for w to range(1, W+1): //end of Knapsack capacity
        current_max = 0
        for i to range(end of items N):
            if A[i](current itemweight) <= w:
                subprob = M[w - wi] + vi
                if subprob > current_max:
                    current_max = subprob
        M[w] = current_max
        // Keep adding the max to array comparing sub prob.
    print(end of M array)
    // this is the solution
    return(end of M array)
```

You find all possibilities and keep comparing to the sub problem to get an array with max value at the end.

This would be $O(nW)$ because there's a for loop with range(W) and one for range item length n. to get all possible values