1. In the Knapsack problem, you are given a list of n items' weights $(w_1, w_2, ...w_n)$ and values $(v_1, v_2, ...v_n)$. The goal is to maximize the total value of the items in your knapsack, without exceeding the weight limit W. For each item in the list, you must choose whether to take it or to leave it.

   Design an $\mathcal{O}(nW)$ dynamic programming algorithm to solve the Knapsack problem.

2. (a) You are given an array of n positive integers $(a_1, a_2, ..., a_n)$. How can you use dynamic programming to determine whether the array can be partitioned into 2 subsets of equal sum? (*Hint: You can use an algorithm you already know....*)

   (b) What about partitioning into 3 subsets of equal sum? Design a dynamic programming algorithm.

3. In the Knapsack with Duplicates problem, you are given a list of n items' weights $(w_1, w_2, ...w_n)$ and values $(v_1, v_2, ...v_n)$. The goal is to maximize the total value of the items in your knapsack, without exceeding the weight limit W. You can choose to take zero, one, or more than one of each item.

   Consider this dynamic programming approach:

   Definition:

   $Value[w]$ is the maximum value achievable without exceeding weight w.

   Recursive Formula:

$$Value[w] = \max_{i:\, w_i \leq w} v_i + Value[w - w_i]$$

   (a) Provide the needed base case(s).

   (b) In what order should our algorithm fill the array $Value[w]$?

   (c) What is the runtime complexity of this algorithm? Is it polynomial?

   (d) After filling the array, $Value[W]$ will be the maximum total value. How could we then reconstruct the specific choice of items which achieves that optimal value?