The knapsack problem is defined as follows: given a set of \( i_0, ..., i_{n-1} \) items, each with a weight \( w_i \) and a value \( v_i \), determine which items you should pick to maximize their total value while keeping the overall weight to at most \( W \), the limit of your knapsack (i.e., a backpack). For example, if the items weights are \( w = \{1, 2, 3, 4\} \) and their values are \( v = \{2, 7, 9, 8\} \), and \( W = 5 \), we should take items \( i_1 \) and \( i_2 \) with weights 2 and 3, and a total value of 16.

- Using the subset sum algorithm given in class as model, implement a solution to the problem described above.

- In a variant of the knapsack problem, that are an unlimited number of items of various types. Each type of item has a weight \( w_i \) and a value \( v_i \) and the task is to determine how many items of each type to take in order to maximize total value without exceeding the knapsack’s capacity \( W \). This problem cannot be easily solved with a backtracking algorithm, but, fortunately, it can be solved using dynamic programming. Using as model the algorithm to determine the minimum number of coins necessary to give change described in class, implement a solution to this problem using dynamic programming.

- Consider another variation of the knapsack problem where you can take fractions of items, form 0 up to the complete item. For example, if you take \( \frac{3}{4} \) of item \( i_k \), its weight would be \( \frac{3}{4} w_k \) and its value would be \( \frac{3}{4} v_k \) (you can imagine that the items are metal bars that can be cut precisely). This problem is easier than the other two variants and can be solved in \( O(n \log n) \) using a greedy approach. Design and implement a greedy algorithm to solve this problem, called the fractional knapsack problem.

As usual, write a report describing your results. Given the little time available, a demo will not be required, thus it is very important that your report accurately reflects your work.