In this lab you will experiment with three algorithm design techniques applied to the solution of three versions of the knapsack problem.

- Implement a backtracking algorithm that solves the optimization 0-1 knapsack problem. Instead of deciding whether we can take items worth a predefined amount of money, as described in class, in this version of the problem you need to find the highest-value load that can fit in the knapsack.

- Implement a greedy algorithm that solves the optimization continuous knapsack problem. This problem is identical the previous one, except that in this case we can take fractions of items. For example, if we take $\frac{3}{2}$ of an item that has value 2 and weight 3, the value of the fraction would be $\frac{3}{2}$ and its weight would be $\frac{9}{4}$. Hint: This can be solved in $O(n \log n)$ time, where $n$ is the number of items present.

- Implement a randomized algorithm that solves the optimization 0-1 knapsack problem. You should generate many random permutations (you can use the numpy np.random.permutation(n) function) of the items and, for every permutation, add the items to the knapsack one by one until the capacity is reached. At the end, return the permutation that resulted in the largest value.

- Implement a dynamic programming algorithm that solves the optimization integer knapsack problem. In this case, the thief can take multiple instances of an item. As before, you need to find the highest-value load that can fit in the knapsack. Hint: This problem is similar to the minimum coin problem described in class.

Compare the running times of the algorithms for various parameter values and, 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.