AD - assignment 2

Task 1

Let $N(C, i)$ denote the number the number of ways to spend exactly C DKK with prices $p_1, \dots, p_i$. This formula can be written recursively as

$$N(C, i) = \begin{cases} 1 & , C = 0 \\ 0 & , C < 0 \text{ or } c \neq 0, i = 0 \\ N(C, i-1) + N(C - p_i, i-1) & , \text{else} \end{cases}$$

Task 2

The intuition about the the recursion, is that if an extra $i + 1$ item is added, then all of the solution that existed when there were only i items, must still be valid. As we already know how to spend exactly C with i items, then when one new item is added, we only have to figure out how to include this in the budget as all other possibilities where this item is not included is already accounted for. Hence the last part of the recursion $N(C - p_i, i - 1)$.

Proof by induction:
Base case: $i = 1$

$$N(C, 1) = N(C, 0) + N(\overbrace{C - p_i}^{C'}, 0) = \begin{cases} 1 & , C' = 0 \\ 0 & , \text{else} \end{cases}$$

So with one product the only way to spend exactly C is if the price of the product equals C, which is the same as the first parameter in $N(C, i)$ equal 0. Hence the base case holds.
Assume it hold for i then we want to show that it also holds for $i + 1$.

$$N(C, i + 1) = N(C, i) + N(C - p_{i+1}, i)$$

which is exactly what we want to show.

Per induction the formula is correct. It is not difficult to see that the problem exhibits optimal substructure and overlapping problems, as the $i - 1, i - 2, \dots, 0$ problems are all part of problem i and the solution to those problem does not depend on i there is optimal substructure. As problem $i - 1$ contains the problems for $i - 2, \dots, 0$ it share all but one problem of the same problems as the original problem i hence there are overlapping problems.

Task 3

Below is an dynamic programming memorization algorithm with an $O(nC)$ runtime. There are made som assumtions that C is an integer, p is a list of positive integers prices. Furthermore; we also assume the list p have some properties p.last which takes the last element in the list in constant time, and p.notLast which return a sublist with all but the last element, this also can be done in constant time. m is a two dimenaional (zero indexed) array, with $C + 1$ rows and p.length $+ 1$ columns and it is initialized with the values -1 at all index.

```
N(C, p, m)
    if C < 0
       return 0
    i = p.length
    if m[C, i] != -1
       return m[C, n]
    if C == 0
      m[C, i] = 1
       return 1
    if i == 0
```

```
10      m[C, i] = 1
11      return 1
12  m[C, i] = N(C, p.notLast, m) + N(C − p.last, p.notLast, m)
13  return m[C, i]
```

Task 4

Show correctness: Show that for every input it will halt with the correct output.

The running time if the algorithm is $O(iC)$ as lines 1-11, 13 all take constant time. The recursion in line 12 all terms are at most calculated once and there are at most ic subproblems to solve, hence the algorithm takes $O(iC)$ time.

The memory usage of the algorithm is also $O(iC)$ as this is the size of the m array which is the larges data we need to save.