Recursion – Exercises
ENSEA/FAME Computer Science

Exercise 1 – The subset sum problem

Consider the following problem: given a set of integers t and an integer s, find a subset of t whose sum is equal to s. For example, if $t = \{1, 2, 3, 4\}$ and $s = 5$, then the subsets $\{1, 4\}$ and $\{2, 3\}$ are solutions to this problem. If $s = 0$, we consider that $\emptyset \subseteq t$ is a solution of the subset sum problem.

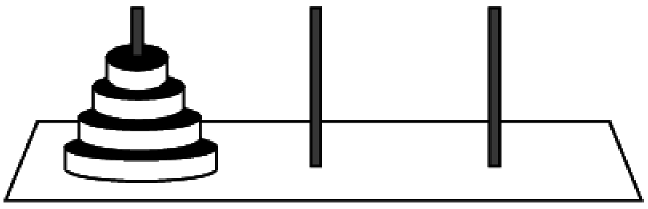
Question 1. Write a function `def subsetsum(s, t)` (whose arguments are an integer `s` and a list `t`) which returns:

- a sublist of `t` whose sum is `s`, if there is any,
- otherwise it returns `None`.

Question 2. What is the complexity of this algorithm?

Exercise 2 – The Hanoi towers

The *Hanoi towers* is a game, invented by the French mathematician Édouard Lucas. It consists of three rods, and a number of disks of different sizes which can slide on any rod. Let n be the number of disks. The game starts in the following configuration:



The scope of the game is to move all disks on the last rod, with the following rules:

1. you can move only one disk at a time,
2. each move consists of taking the upper disk from one of the stack and placing it on top of another stack,
3. no disk may be placed on top of a smaller disk.

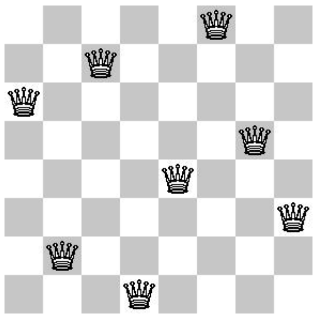
Question 1. Write a function `def hanoi(n)` which prints the list of moves to be made. For example,

```
>>> hanoi(2)
1 -> 2
1 -> 3
2 -> 1
```

Question 2. Find out the number of moves, depending on n.

Exercise 3 – The n queens problem

We want to place n queens on an $n \times n$ chessboard, such that no queen threatens another, *i.e* we require that no two queens share the same row, column or diagonal. Here is a possible solution for $n = 8$.



Question 1. Find a lower bound for the complexity of the brute force method.

We label the rows and the columns from 0 to $n - 1$. A configuration is represented by a list of couples indicating the queens' positions. For example, the configuration shown in the picture is represented by `[(0, 5), (1, 2), (2, 0), (3, 6), (4, 4), (5, 7), (6, 1), (7, 3)]`.

Question 2. Create a function `def check(queens)` which returns a boolean indicating whether the configuration `queens` is a valid solution to the queens problem.

```
>>> check([(0, 5), (1, 2), (2, 0), (3, 6), (4, 4), (5, 7), (6, 1), (7, 3)])
True
>>> # The function also returns True is the solution is partial
>>> # (there is less than n queens)
>>> check([(3, 6), (4, 4), (5, 7), (6, 1), (7, 3)])
True
>>> check([(0, 0), (1, 1)])
False
```

Question 3. Write a function `def findqueens(n)` which returns

- a solution to the n queens problem, if it exists,
- otherwise `None`.