Programming exercises - lectures 4-5

Exercise 1
Write a function `binary_search`, that takes as an input two arguments: a list `arr` of elements of type `int`, which can be assumed to be in the increasing order, and `target` of type int.

The function should return the position of `target` in the list `arr` (for simplicity you can assume that `target` is indeed present in the list `arr`), in time $O(\log n)$ using the binary search algorithm. You can find short visualization how the algorithm works in programming, lecture 4 slides.

Hint: Create two variables, `left` and `right` denoting positions of the left, and the right endpoint of the current range where the algorithm knows the element `target` needs to be. Update those positions accordingly in the `while` loop.

Bonus points: if `target` is not element of the list `arr`, your function should return `None`.

Example:

```
def binary_search(arr, target):
    ...

binary_search([1,5,7,12, 33], 5)
Out: 1
binary_search([1,5,7,12,33], 33)
Out: 4
binary_search([1,5,7,12,33], 4)
Out: None
```

Exercise 2
Write a function `all_subsets` that takes a number n, and returns a list of length 2^n - elements of this list should be themselves lists corresponding to all subsets of the set $\{0, 1, 2, \ldots n-1\}$.

Example:

```
def all_subsets(n):
    ...

all_subsets(2)
Out: [[], [1], [0], [0, 1]]
```

```
all_subsets(3)
Out: [[], [2], [1], [1, 2], [0], [0, 2], [0, 1], [0, 1, 2]]
```

The subsets can appear in any order in the output list. The elements in each subset should be given in the increasing order.

Hint:
I recommend using a recursion to solve this problem. You will also need to use the copy() method of the list, which has been mentioned during the first lecture. As a reminder, given a list

```
a = [1,2,3,4]
```

we can call `a.copy()` to create a new list with identical content as the list `a`. Now those two lists can be modified independently.

```
b = a.copy()
a.append(4)
print("a = ", a)
print("b = ", b)
```

In contrast, if instead we wrote

```
b = a
a.append(4)
print("a = ", a)
print("b = ", b)
```

Both variables `a` and `b` are referencing the same list in the memory. Modifying it, will affect both variables.