Assignment 3

420-ENM-MT Algorithm, Pseudo-code and Design

Due: January 31st 2019 before class

Instructions

- This assignment is worth **5% of your final grade**;
- You must submit three (3) files on **Omnivox** (four if you answer the bonus question). Submit theory questions as a single **pdf** and coding questions as `.py` files;
- **Read every question carefully**;
- Some problems require you to write code. Make sure to read the instructions and submit the corresponding `.py` files with your assignment;
- Your code will be tested with **Python 3.7**;
- Using online resources is allowed, but you should cite your sources for any algorithm or code you did not write yourself. You must prove you understood the solution by properly commenting it, otherwise you will be given a grade of zero for **plagiarism**.

Problem 1 (15 points)

Consider the following recursive functions.

```
function fibonacci(n):
    if n <= 1:
        return n
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
```

```
function factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

a) (5 points) Draw the recursion tree generated by calling **`fibonacci(4)`**.

b) (5 points) Draw the recursion tree generated by calling **`factorial(4)`**.

c) (5 points) By referring to the recursion trees drawn above, explain why recursion (the recursion fairy) is more efficient at solving `factorial` than `fibonacci`.

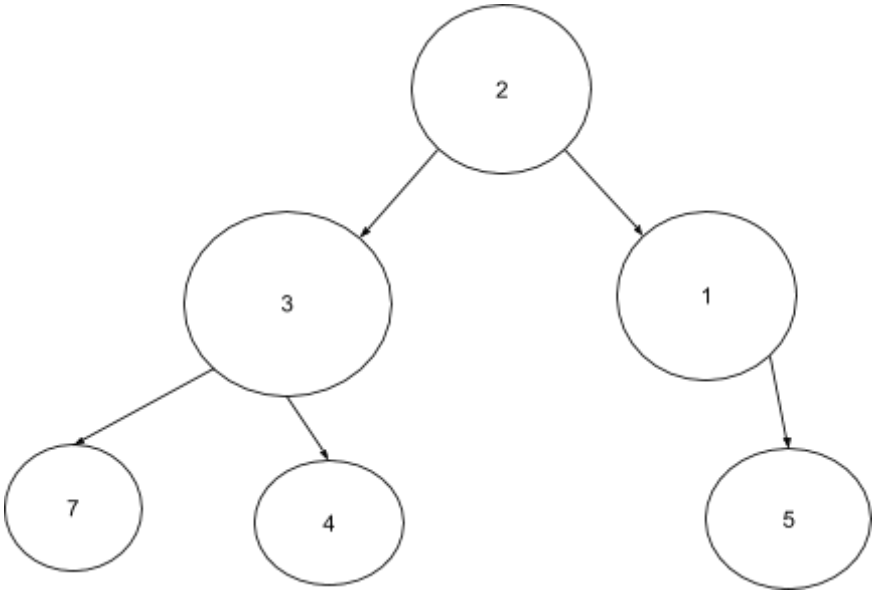
Problem 2 (20 points)

For each of the following question, write down an algorithm using **pseudo-code**.

a) (10 points) Write down an algorithm which takes the root of a tree as argument and returns the size of the tree.

b) (10 points) Suppose you are given a tree in which **values are numbers**.

Example:



Write down an algorithm which takes the root of such a tree as argument and returns the sum of all numbers in the tree.

By example, the algorithm should return **22** for the above tree.

Problem 3 (20 points)

This question has a theoretic and a Python part. Make sure to submit both.

We previously had to find an algorithm to decide whether a list of numbers is already sorted in increasing order or not. In this question, we will write a **Python recursive function** `is_increasing` to solve this problem.

A list is considered to be increasing if each of its element is greater than or equal to its predecessor. Remember that a constant or empty list is considered to be increasing.

Examples:

- `is_increasing([ ])` → `True`
- `is_increasing([1, 1, 1, 1])` → `True`
- `is_increasing([0, 3, 5, 7, 13])` → `True`
- `is_increasing([1, 3, 4, 0]` → `False`

a) (5 points) What is the base case of the algorithm `is_increasing`?

b) (5 points) What is the recursion step of the algorithm `is_increasing`? In other words, how do you break down the problem to a smaller one to ask the recursion fairy?

c) (10 points) Write a **Python recursive function** `is_increasing` which takes a list of numbers as argument and returns `True` if it is increasing and `False` if it is not.

You are not allowed to use any Python library as well as built-in sorting methods or functions.

Answer this question by filling in the file `is_increasing.py`, then submit it with your assignment.

Problem 4 (20 points)

This question has a theoretic and a Python part. Make sure to submit both.

A key feature available across most text-based programs, whether they are text editor, web browser or pdf reader, is to be able to search a given word in the text. By example, it is likely that if you press CTRL-F right now, the program you are using will prompt you for a word to search and highlight in this document.

String-searching algorithms are an important category of algorithms which are meant to efficiently find one or more occurence of a word or sentence into text.

In this problem we will write a recursive function `word_in_string` which finds whether a word in contained in a text.

Examples:

- `word_in_string("algorithm", "algorithms are fun!")` → `True`
- `word_in_string("", "empty strings are everywhere")` → `True`
- `word_in_string("computer", "computer")` → `True`
- `word_in_string("samosas", "Menu: pizza, sandwich")` → `False`
- `word_in_string("pseudocode", "")` → `False`

a) (5 points) What are the base cases (there are two) of the algorithm `word_in_string`?

b) (5 points) What is the recursion step of the algorithm `word_in_string`? In other words, how do you break down the problem to a smaller one?

c) (10 points) Write a **Python recursive function** `word_in_string` which takes two strings as argument and returns `True` if the first string is contained into the second string and `False` if it is not.

You are not allowed to use any Python library, any built-in string methods or the *`in`* operator, but you are allowed to use slicing.

Answer this question by filling in the file `word_in_string.py`, then submit it with your assignment.

Problem 5 - Bonus Question (15 points)

The following problem is optional. Although, you will get bonus points for answering it correctly.

In class we wrote a recursive function to find the minimum element in a list.

You are now asked to write a **Python recursive function** to find the n^{th} smallest element of a list.

Examples:

- `nth_min([5, 6, 2, 0], 1)` → `0`
- `nth_min([1, 4, 3, 2], 3)` → `3`
- `nth_min([3, 5, 1, 8], 4)` → `8`

Using **Python**, write a recursive function `nth_min` which takes a list as first argument and a positive integer `n` as second argument and returns the n^{th} smallest element in the list.

You are not allowed to use any Python library, but you can use the built-in functions `min` and `max`.

Answer this question by filling in the file `nth_min.py`, then submit it with your assignment.