Computational Thinking and Programming – A.Y. 2019/2020

Written examination – 09/07/2020

Given name:	
Family name:	
Matriculation number:	
University e-mail:	
Group name:	

Is it your first try? Yes | No

The examination is organised in three different sections:

- Section 1: basic questions [max. score: 8]. It contains four simple questions about the topics of the whole course. Each question requires a short answer. Each question answered correctly will give you either 2 points (full answer) or 1 point (partial answer).
- Section 2: understanding [max. score 4]. It contains an algorithm in Python, and you have to report the particular results of some of its executions according to specific input values.
- Section 3: development [max. score 4] It describes a particular computational problem to solve, and you are asked to write an algorithm in Python for addressing it.

You have 1 hour and 30 minutes for completing the examination. By the final deadline, you should deliver only the original text (i.e. this document) with the definitive answers to the various exercises that must to be written with a pen – pencils are not permitted. You can keep all the draft papers that you may use during the examination for your convenience – blank sheets will be provided to you on request.

Section 1: basic questions

1 – Which of the following algorithmic techniques *are not* natively based on recursion:

- backtracking
- brute-force
- divide and conquer
- dynamic programming
- greedy

2 – Identify the mistake(s) and propose appropriate modifications in the linear search algorithm implemented in the following Python code:

```
def linear_search(input_list, value_to_search):
    for position, item in enumerate(input_list):
        if item != value_to_search:
            return item
```

3 – Write down a small function in Python that takes in input two numbers and returns `True` if their subtraction is an even number, otherwise it returns `False`.

4 – What are the ordered structures and what are their main characteristics in terms of adding and removing elements?

Section 2: understanding

Consider the following functions written in Python:

```
def fun(given_name, family_name, mat_number):
    n = len(given_name) - len(family_name)
    if n < 0:
        n = n * -1

    num_l = list()
    for idx, item in enumerate(mat_number):
        num_l.append(int(item) + idx + n)

    name_l = list()
    for c in given_name + family_name:
        if c != " ":
            name_l.append(c)

    result = list()
    name_len = len(name_l)
    for idx in num_l:
        c = name_l[idx % name_len]
        result.append(c)

    return result
```

Consider the variables `my_given_name` and `my_family_name` containing the strings of your given name and family name respectively, both in **lower case**, and the variable `my_mat_number` containing the **string** of your matriculation number (ten digits). What is the value returned by calling the function `fun` as shown as follows:

```
fun(my_given_name, my_family_name, my_mat_number)
```

Section 3: development

Co-citation Proximity Analysis (**CPA**) is a similarity measure that uses citations to assess semantic similarity between documents using the part (section, paragraph, sentence) where the citations happen in the document. This similarity measure assumes that the documents cited in close proximity to each other tend to be more strongly related than those documents cited farther apart. For instance, suppose that a document is composed by three sections (1, 2, and 3) and that in section 1 it cites documents A and B, in section 2 it cites document C, and in section 3 it cites document A, C and D. Then the **Citation Proximity Index** (**CPI**) for each set of documents cited by an examined document is calculated by the following formula:

$$CPI(doc1_{cited}, doc2_{cited}) = \frac{1}{2^n}$$

where n stands for the minimal distance in levels between citations to *doc1* and *doc2*. In the example above, where we use the section number to define the level, CPI(A,B) = 1 (since n = 0), CPI(A,C) = 1 (since n = 0), CPI(A,D) = 1 (since n = 0), CPI(B,C) = 0.5 (since n = 1), CPI(B,D) = 0.25 (since n = 2), and CPI(C,D) = 1 (since n = 0).

Write an algorithm in Python – `def cpa(doc_sections)` – which returns the CPI of all the documents cited in the various sections of the document provided in the input list. Each item in the input list represents a section of the document in consideration, and it is defined as the set of the documents (identified by strings) that are cited in that section. For instance, the input of the example presented above should be `[{"A", "B"}, {"C"}, {"A", "C", "D"}]`. The function returns a dictionary of pairs where, in each pair, the key is a tuple representing two of the cited documents in the input document and the value is the related CPI – e.g. `{("A", "B"): 1, ("A", "C"): 1, …}`. The dictionary contains all the possible pairs between the cited documents – please remember that, for instance, `("A", "C")` is the same as `("C", "A")`, thus it should be represented only once in the dictionary.