Computational Thinking and Programming – A.Y. 2018/2019

Written examination – 04/02/2019

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 2 points.
- Section 2: understanding [max. score 4]. It contains an algorithm in Python, and you have explain what it does and 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 – Select the entities in the following list that can be used in a flowchart:

- flowline widget
- decision widget
- iterative widget
- back widget
- process widget
- quick widget

2 – Consider the following implementation of the *merge sort*:

```
def merge_sort(input_list):
    input_list_len = len(input_list)
    if input_list_len <= 1:
        return input_list
    else:
        mid = partition(input_list, 0, input_list_len, 0)
        left = merge_sort(input_list[0:mid])
        right = merge_sort(input_list[mid:input_list_len])
        return merge(left, right)
```

Identify the mistake in the aforementioned code and correct it.

3 – Write down a small function in Python that checks if two integers `i1` and `i2` are both even, and in that case returns *True* – otherwise it returns *False*.

4 – Describe the structure that a recursive function must have.

Section 2: understanding

Consider the following functions written in Python:

```
def r(gn, fn):
    g = ""
    f = None
    fl = list()
    for c in fn:
        fl.append(c)

    if len(fn) <= 1:
        return fn + gn
    else:
        for c in reversed(gn):
            if c >= fn[0]:
                g += c
            else:
                f = fn[1]
                g += f

        if f:
            fl.remove(f)
        else:
            fl.remove(fl[0])

        return r(g, "".join(fl))
```

Consider the variable `my_gn` containing the **string** of your given name as written in the first page but in **lowercase**, and the variable `my_fn` containing the **string** of your family name as written in the first page but in **lowercase**. What is the value returned by calling the function `r` as shown as follows:

```
r(my_gn, my_fn)
```

Section 3: development

The **BiGramsJaccardDistance** is a technique that calculates the difference between two strings by checking how many bigrams the two strings have in common overall. A bigram is defined as a sequence of two consecutive characters – e.g. the string `"john"` can be split in three bigrams: `"jo"`, `"oh"` and `"hn"`. Once created two collections of bigrams (one for each string), each of which cannot contain the same bigram twice or more times, the Jaccard distance is calculated by dividing the size of the intersection of these collections with the size of the union of the same collections.

Write an algorithm in Python – `def bigrams_jaccard(string_1, string_2)` – which implements the *BiGramsJaccardDistance* technique and returns the number defining the distance between the strings.