Test Pearl 001 — Algorithmics

Pearls of Computer Science (201300070) / Introduction to BIT (201300073)
14 september 2018, 13:45–14:45
Module coordinator: Doina Bucur
Instructor: Arend Rensink

- You may use 1 A4 sheet with your own notes for this test, as well as a *simple* calculator
- Scientific or graphical calculators, laptops, mobile phones, books etc. are not allowed. ***Put those in your bag now (with the sound switched off)!***
- Total number of points: 100

15 points **Question 1** Suppose that you execute the following assignments in Python:

```
progs = ["AM", "BIT", "TCS"]
studs = [40, 85, 210]
```

`progs` lists some study programme abbreviations (Applied Mathematics, Business and IT, Technical Computer Science) and `studs` lists the number of enrolled first-year students in those programmes in 2018.

(a) Write a Python condition (*not* an entire `if`-statement) that tests whether the second and third elements of `studs` are both more than twice as large as their predecessors.
(b) Write a Python assignment that assigns to a variable `bit` the abbreviation and number of first-year students in the programme Business and IT, taken from the lists above.
(c) Write a sequence of Python assignments that is as short as possible, resulting in a change to `progs` after which the study programmes are ordered from highest to lowest number of first-year students. (It is *not* correct to assign an entirely new value to `progs`; you must *modify* the list.)

15 points **Question 2** Analyse the following Python-function.

```
1  def compute(data):
2      result = True
3      i = 0
4      while i < len(data):
5          print("i=", i, ", result=", result)
6          if data[i] != data[len(data)-i-1]:
7              result = False
8          i = i+1
9      return result
```

(a) Show what happens in a call of `compute([63, 5, -1, 4, 63])`, by writing down
  (i) the output of the `print`-statement in line 5 every time it is executed;
  (ii) the return value of the call.
(b) What does the function `compute` actually do? *(Do not give a step-by-step explanation of the execution, but describe the purpose of the function.)*

10 points **Question 3** How many steps does `compute` (in Question 2) need in terms of the length (say n) of the input list `data`: approximately ("in the order of") $\log_2 n$, approximately $\sqrt{n}$, approximately n or approximately n^2? Explain your answer.

10 points **Question 4**

(a) What would change if the condition in line 4 of `compute` (Question 2) were replaced by the more complicated-looking "`i < len(data)/2 and result`"? Explain your answer.
(b) With the same replacement, does your answer to Question 3 change? If so, how; and if so, why not?

20 points **Question 5** Consider the list `[16, 7, -30, 5, 21, 0]`.

(a) Show how bubble sort sorts this list, by writing down the list after every single modification.
(b) Show how merge sort sorts this list, by schematically showing how the list is split and zipped back together.

15 points **Question 6** Suppose that you have stored the n words of a dictionary into a list of lists. The outer list has exactly 26 elements, one for each letter of the alphabet. Each of the 26 elements is an unordered list containing all the words starting with the corresponding letter. For the sake of this exercise, assume that the words are distributed evenly over the starting letters (which is not the case in reality, obviously).

(a) If you look up a word in this structure (i.e., you try to find out whether it exists), how many steps will that take on average, taking into account that there are n words in total and assuming that the word is eventually found? Explain your answer.
(b) If you start with a single unordered list of n words, how many steps will it take to create a list of lists with to the structure described above, containing exactly those words? Explain your answer.
(c) Suppose that you want to look up a number of words (say m) in an unordered list that contains n words. Under what circumstances (i.e., what values of m and n) does it pay off to convert the list into the structure described above and then look up the words, rather than to do a linear search in the original list? Again, you may assume that all words are eventually found. Explain your answer.

15 points **Question 7** The cards of the game *Magic: The Gathering* can be subdivided into the *commons*, *uncommons* and *rares*. You have a huge pile of *Magic: The Gathering* cards and you decide you want to give two rares to your twin nephews. However, to avoid starting a fight you want them to be identical cards. Unfortunately, you have not ordered your cards well (in fact your pile is more like a heap; something your parents tend to call a mess).

(a) Write an algorithm in natural language, with unambiguous, numbered instructions, that finds two identical rares or concludes that you don't have two identical rares, in as few steps as possible. Your instructions may only involve one or two cards at a time, never an arbitrary set of them.

You may assume that you have ample space to arrange the cards in any way you like. The cards do not have to end up where they started.

Do not try to give an answer in Python!

(b) How many steps does your algorithm need in the worst case, as a function of the number of cards, and assuming you in fact have a pair of identical rares? How many does it need in the best case? Explain your answers.