# 2023-09-15 - Pearls of Computer Science Core - Algorithmics

## Course: B-CS-MOD01-1A-202001021/202001022 B-CS Pearls of Computer Science Module - 202001021/202001022 – TEST 1

---

**Duration:** 1 hour

**Generated on:** Sep 13, 2023

**Contents:** Pages:

# 2023-09-15 - Pearls of Computer Science Core - Algorithmics

## Course: B-CS Pearls of Computer Science Module - 202001021/202001022 – TEST 1

---

Welcome to the *graded* digital test of the Algorithmics Pearl.

- 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)
- A Python interpreter is available on the Chromebook. **Feel free to use it**.
- For *technical* questions (concerning the Chromebooks, Remindo etc.): **raise your mouse**.
- For *pearl content* question: **ask the Pearl teacher in the BBB chat available in one of your tabs**.

Total number of points: 100

**Number of questions:** 8

**1** Suppose you execute the following assignments in Python:

```
associations = ["Interactief", "Proto", "Scintilla"]
members = [482, 267, 368]
```

Here *associations* is a list of student associations at the University of Twente, and *members* is a list of their corresponding (fictional) numbers of active members.

5 pt. **a.** Write a Python condition (**not** an *if-statement*) that tests whether the "Interactief" association is the **largest** of the three associations.

5 pt. **b.** Assign to a new list '*smallest*' both the name and the number of active members of the association with the **fewest** active members.

5 pt. **c.** Write a sequence of assignments that is **as short as possible**, resulting in a change to 'members' after which the number of active members are ordered from **highest to lowest**.

(**NB:** It is not correct to assign an entirely new value to members. You must modify the list by swapping elements.)

**2** Consider the following Python function:

```
01. def compute(arr):
02.     n = len(arr)
03.     i = 0
04.
05.     while i < n:
06.         k = i
07.         j = i + 1
08.
09.         while j < n:
10.             if arr[j] < arr[k]:
11.                 k = j
12.             j += 1
13.
14.         arr[i], arr[k] = arr[k], arr[i]
15.         i += 1
16.
17.     return arr
```

2 pt. **a.** What is the **return value** upon calling *compute(5198)?*

2 pt. **b.** What is the **return value** upon calling *compute([5,1,9,8])*

2 pt. **c.** What is the **return value** upon calling *compute("kayak")?*

4 pt. **d.** Using your own words, what does the algorithm do?

**Hint:** Do *not* list line-by-line what the algorithm does, but state what the function *compute* does as a whole.

**3** Consider again the following Python function:

10 pt.

```
01. def compute(arr):
02.     n = len(arr)
03.     i = 0
04.
05.     while i < n:
06.         k = i
07.         j = i + 1
08.
09.         while j < n:
10.             if arr[j] < arr[k]:
11.                 k = j
12.             j += 1
13.
14.         arr[i], arr[k] = arr[k], arr[i]
15.         i += 1
16.
17.     return arr
```

Assume that we input a list *arr* of length *n*. How many steps does *compute* need to finish?

1. Approximately $n$
2. Approximately $n^2$
3. Approximately $log_2(n)$
4. Approximately $n \cdot log_2(n)$

Motivate your answer as precisely as you can.

**4** Consider again the following Python function:

```
01. def compute(arr):
02.     n = len(arr)
03.     i = 0
04.
05.     while i < n:
06.         k = i
07.         j = i + 1
08.
09.         while j < n:
10.             if arr[j] < arr[k]:
11.                 k = j
12.             j += 1
13.
14.         arr[i], arr[k] = arr[k], arr[i]
15.         i += 1
16.
17.     return arr
```

5 pt. **a.** What happens to **the algorithmic complexity** of the algorithm if we change line 07 from "*j = i + 1"* to "*j = 0"*?

Briefly motivate your answer.

5 pt. **b.** What happens to **the algorithmic complexity** if we indent line 12 to the same level as line 11, i.e., if *j += 1* became part of the body of the if statement in line 10.

Briefly motivate your answer.

**5** Consider the following list:

10 pt.

[-3, -9, 23, -15, 5, 16]

Show how bubble sort sorts this list by writing down the list after every single modification.

**6** Consider the following list:

10 pt.

[-3, -9, 23, -15, 5, 16]

Show how merge sort sorts this list by writing down the list after every single modification.

Write down every change the algorithm makes to the list(s) in a new line.

**7** Suppose that you are a computer science teacher at the University of Twente. As part of your course you hand out 100 uniquely enumerated Arduinos labeled with numbers 1 to 100 to your students.

A week later each student is asked to return their Arduino. You want to ensure that you received each Arduino back, and if not, you want to know which numbers are missing.

Here are two ways to go about this:

1. Put all Arduinos you have in a pile and linearly go through numbers 1 to 100 to see which ones are missing, or
2. First sort all Arduinos you have according to their number label and then apply binary search to go through numbers 1 to 100 to see which ones are missing.

Naturally, as a computer science teacher you want to do this as efficiently as possible.

(**Hint**: Simply counting them is not enough, since we would not know which Arduinos are missing. You must choose to argue for either option 1 or 2)

10 pt. **a.** Which method do you choose and why? Briefly explain your reasoning and support your answer with the algorithmic complexity of both methods.

5 pt. **b.** Can you think of an even more efficient algorithm to find the missing Arduinos? Briefly sketch out your algorithm in words.

**8** After three years of studying at the University of Twente it is finally time for the presentation of your BSc thesis. For the occasion you would like to wear a pair of matching socks; you do not care which ones.

But alas! After three years of studying you gathered a lot of single socks without a matching second one. You are not even sure that you have a single pair of matching socks left at all.

How do you go about finding a pair of matching socks, or alternatively conclude that you do, in fact, not have one pair of matching socks?

15 pt. **a.** Write an algorithm in natural human language, with **unambiguous and numbered** instructions, that allows you to reach your goal in as few steps as possible. Your instructions may only involve **one or two socks** at a time, never an arbitrary number of them.

**Hint:** Do *not* write Python code.

5 pt. **b.** How many steps does *your* algorithm need in the worst case, as a function of the number of socks *n* in your box.

Thank you, your answers were saved. We will inform you about your result once every test was *manually* corrected.

An answer key will be posted on Canvas soon.