2021-09-17 - Pearls of Computer Science Core - Algorithmics

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

Duration: 1 hour

Generated on: Sep 17, 2021

Contents: Pages:

2021-09-17 - Pearls of Computer Science Core - Algorithmics

Course: B-CS Pearls of Computer Science Module - 202001021/202001022

Welcome to the digital exam for Pearl 001 Algorithmics.

- 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)**
- Please use the **BBB/Canvas chat for any question** during the exam.

Total number of points: 100

Number of questions: 10

1
5 pt.

Question 1a

Suppose you execute the following assignments in Python

```
clubs = ["Dance", "Theatre", "Sports"]
members = [68, 162, 92]
```

Here *clubs* is a list of club names at the University of Twente, and *members* is a list of the (fictional) number of active members in each club.

Write a Python condition (*not* an **if** statement) that tests whether the "Dance" club is the smallest of the three clubs.

2
5 pt.

Question 1b

Suppose you execute the following assignments in Python

```
clubs = ["Dance", "Theatre", "Sports"]
members = [68, 162, 92]
```

Here *clubs* is a list of club names at the University of Twente, and *members* is a list of the corresponding (fictional) number of active members in each club.

Assign to a new list '*smallest*' both the name and the number of active members of the club with the fewest active members.

3
5 pt.

Question 1c

Suppose you execute the following assignments in Python

```
clubs = ["Dance", "Theatre", "Sports"]
members = [68, 162, 92]
```

Here *clubs* is a list of club names at the University of Twente, and *members* is a list of the corresponding (fictional) number of active members in each club.

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 lowest to highest.

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

4

Question 2

Consider the following Python function:

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

5 pt. **a.** What is the **return value** upon calling compute([99, -8, 8, 20, 0])?

5 pt. **b.** What is the **return value** upon calling compute(["magic", "tanja", "hamster", "companion"])?

5 pt. **c.** 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.

5
10 pt.

Question 3

Consider again the Python function given in Question 2:

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

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

1. Approximately n

2. Approximately n^2

3. Approximately $n \cdot log_2 n$

4. Approximately $\sqrt{n}$

Motivate your answer.

6

Question 4

Consider again the Python function given in Question 2:

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

5 pt. **a.** How does the **return value** of the algorithm change if we replace **each** occurrence of "<" with a ">" instead, ie. if we replace **each** occurrence of "smaller than", with a "larger than"?

5 pt. **b.** How does the **return value** of the algorithm change if we delete lines 09 to 12?

5 pt. **c.** How does the **complexity** of the algorithm change if we delete lines 09 to 12? Motivate your answer.

7

Question 5a

10 pt.

Consider the following list

```
[8, 20, -4, 4, -18, 6]
```

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

8
10 pt.

Question 5b

Consider the following list

```
[8, 20, -4, 4, -18, 6]
```

Show how merge sort sorts this list by presenting how the list is split and zipped back together.

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

9
10 pt.

Question 6

After having studied all week for the Algorithmics test you lie awake one night and wonder if one could get an even faster search algorithm than binary search.

Then it dawns on you: *ternary search!* Rather than dividing a list into two equal parts, *ternary search* divides the list into *three* equal parts (left, middle, right) and then determines in which of these three parts the searched-for value lies.

Can this idea be realized, ie. could such an algorithm work in practice?

If not, why not?

If so, what would be the complexity of *ternary* search -- In particular, would it be fundamentally faster than binary search? Explain your answer.

10

Question 7

After an exciting first 10 weeks of studying at the University of Twente you accumulated a pile of books. You stacked them such that the largest book is at the very bottom, and the smallest at the very top -- A pyramid of books!

One fine day you decide to shift the entire book-pyramid to another location in your room -- But alas! The pile is too heavy.

There is no other solution: You must re-locate the books one-by-one. To make for a stable pile of books after the re-location, the biggest book needs to be at the bottom once again.

10 pt. **a.** (a) Write an algorithm in natural language with **unambiguous** and **numbered** instructions that re-locates all the books to another place one-by-one, such that the final stack has the same shape as before.

You may assume that there is enough space in your room to spread out books and sort them all you like.

*Do **not** give an answer in Python!*

5 pt. **b.** (b) What is the complexity of your algorithm in (a) in terms of the number of books n .

Thank you, your answers were saved. We'll inform you about your result once every test was corrected.