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 20, 2023

Contents: Pages:

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

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** (click on the Windows button in the bottom left to find either Notepad++ or Python3.10 -> IDLE).
- 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** (minimize the window in the top right corner by clicking on the two arrows if you cannot find it).

Total number of points: 100

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, or explain why no better algorithm can exist.

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.

Correction model

1.
15 pt.

a.

Correction criterion	Points
* -2 if an "if" is included * -2 for missing 'and' * -2 if both < and <= (> >= resp.) are used * -3 for the wrong list (association vs members) * -3 if the smallest association was used instead of the largest (< vs >) * -1 if the same element was used for comparison and everything else is correct, eg. members[0] < members[1] and members[0] < members[1] * -2 for syntax mistakes, eg. => instead of >= * -5pts for members[0] > members[1] > members[2] While this is allowed in Python, it's evaluated pairwise and will give the wrong answer Answer: We will accept both > and >= to determine the 'largest' association (< and <= respectively): 1. members[0] > members[1] and members[0] > members[2] 2. members[0] >= members[1] and members[0] >= members[2]	5 points
Total points:	*5 points*

b.

Correction criterion	Points
This often yields 0 points if it is not quite correct: * -4pt if 'append' or 'extend' are used * -4pt if "smallest=" is missing and only brackets are present * -3pt for wrong or missing brackets * -2pt for extra step * -1pt for one missing bracket * -3pt for absence of any index Answer: smallest = [associations[1], members[1]] NB: Forming an entirely new list is only partially correct. The intent is to access the existing lists, e.g., smallest = ["Proto", 267] is only partially correct (2pt).	5 points
Total points:	*5 points*

c.

Correction criterion	Points
* 0 if 'members' is assigned a new list * -1 for additional swap * -2 for every additional assignment beyond the first * -2 for wrong order * -2 for syntax mistake * -3 for wrong list 'associations' * -2 missing comma(s) Answer: In Python we can do a variable swap like: members[1], members[2] = members[2], members[1]	5 points
Total points:	*5 points*

2.
10 pt.

a.

Correction criterion	Points
Answer: The function will throw a TypeError, because line 02 will then ask for the length of an integer. This operation is not allowed in Python., as one asks for the length of a list. Note that this is not an issue for the list with one element [5198]	2 points
Total points:	*2 points*

b.

Correction criterion	Points
Answer: The return value is the sorted list [1,5,8,9]	2 points
Total points:	*2 points*

c.

Correction criterion	Points
The Python function will throw a TypeError, since the function cannot handle input of type string, and "kayak" is a string	2 points
Total points:	*2 points*

d.

Correction criterion	Points
* -2 If ordering/sorting is not mentioned at all * -2 only mentioning 'sorting' without explanation, e.g., ascending order, increasing Answer: This is "Selection Sort", a sorting algorithm that sorts numbers in ascending order. NB: A correct answer does not have to name the algorithm;	4 points
Total points:	*4 points*

3.
10 pt.

Correction criterion	Points
Points are given for (i) the correct answer (3pt) and (ii) for the correct motivation (7pt). Thus, only stating the correct complexity will typically get less than half of the available points. In particular: * "It behaves like bubble sort" alongside an explanation gives 10pt, but missing the explanation can get a maximum of 7 pt because N^2 was correctly identified but not necessarily explained * Only mentioning "because of the nested loop" and no additional explanation is insufficient 7pt * -2pt per vague/ambiguous statement Answer: The complexity of the algorithm is n^2. Note that the initial ordering of the input list does not matter, since the best case complexity (i.e. list is already ordered) AND the worst case complexity (list is ordered in reverse) of selection sort are both n^2. This is due to the nested while loop that has n as an upper bound and whose parameters i and j increment linearly by 1 with each step. The inner loop is reset (line 07) after the outer loop's first iteration.	10 points
Total points:	*10 points*

4.
10 pt.

a.

Correction criterion	Points
Answer: While this does make the algorithm more inefficient by checking values that are already sorted, the algorithmic complexity does *not* change and remains n^2. This is due to the fact that the algorithmic complexity only depends on the asymptotic behaviour of the input list's length and *not* any constant associated with it.	5 points
Total points:	*5 points*

b.

Correction criterion	Points
Answer: In the worst case the algorithm will not terminate at all. This is the case if ever there is an iteration in which arr[k] < arr[j], as variable j will then never be increased, and the inner while loop turns into an infinite loop.	5 points
Total points:	*5 points*

5.
10 pt.

Correction criterion	Points
This question is strictly assessed. * -3 for every mistake, e.g. skipping a step * up to -10 for a systematic error Answer: The consecutive steps are --- Start --- [-3, -9, 23, -15, 5, 16] --- Round 1 --- [-9, -3, 23, -15, 5, 16] [-9, -3, -15, 23, 5, 16] [-9, -3, -15, 5, 23, 16] [-9, -3, -15, 5, 16, 23] --- Round 2 --- [-9, -15, -3, 5, 16, 23] --- Round 3 --- [-15, -9, -3, 5, 16, 23] --- Final --- [-15, -9, -3, 5, 16, 23]	10 points
Total points:	*10 points*

6.
10 pt.

Correction criterion	Points
This question is strictly assessed. -3 if pairs are not split -5 if zipping is not included -5 if lists are mixed up * up to -10 for systematic/algorithmic mistakes NB: The version of merge sort that was treated in the lecture and tutorials splits lists of odd lengths such that the even number is on the right hand side; However, the answer will naturally be graded correctly if the presented splitting method consistently puts it on the left hand side. Answer: [-3, -9, 23, -15, 5, 16] ---------- [-3, -9, 23] [-15, 5, 16] ---------- * [-3] [-9, 23] ** [-9] [23] * [-9,-3, 23] ---------- > [-15] [5, 16] >> [5, 16] > [-15, 5, 16] ---------- [-15, -9, -3, 5, 16, 23]	10 points
Total points:	*10 points*

7.
15 pt.

a.

Correction criterion	Points
There are three components to this question: 1) Laying out the complexity of option 1 (3 pts) 2) Laying out the complexity of option 2 (4 pts) 3) Comparing both and drawing a conclusion (3 pts) Points are deducted for unclear or ambiguous answers. Answer: Option 1: This is a clear application of linear search. Each of the numbers 1 to 100 needs to be found in the pile. In the worst case, the number will be the last one to find. With n = 100, this leads to a worst case complexity of n*n, or 100*100 = 10.000. Option 2: This is a clear application of a sorting algorithm (e.g., merge sort) followed by binary search. Applying merge sort has a one-time complexity of n*log(n). One application of binary search has the complexity of log(n). Since we need to find which ones are missing (conversely, which ones we have), we need to do a total of n searches. Thus, the complexity of searching is n*log(n), leading to an overall complexity of n*log(n) + n*log(n) = 2*n*log(n). With n = 100 this yields 2*100*log(100) which is approximately 460. With n = 100 it follows that 2*n*log(n) < n^2 we should use the second option, i.e. 460 < 10.000	10 points
Total points:	*10 points*

b.

Correction criterion	Points
For example, an even faster algorithm could be: 1. Sorting the arduinos from lowest to highest with merge sort for a total of n*log(n) steps 2. Going through the sortest list of Arduinos *once* linearly and writing down all the missing ones for a total n n steps. Note that the last step is possible only because the Arduinos are numbered consecutively from 1 to 100. This algorithm has a total complecity of n*log(n) + n = log(n) * (n+1) and with n = 100 this yields 202.	5 points
Total points:	*5 points*

8.
20 pt.

a.

Correction criterion	Points
* -5 If loops are not correctly specified * -5 If multiple steps are combined into one; Applies per combined steps * -5 if sorting is assumed to be possible without commenting on it, e.g., "apply merge sort" * -5 for the absence of natural language in lieu of actual Python code * -3 Steps are not enumerated * -2 for unclear descriptions of steps * 0 to 5 points if the wrong algorithm was implemented (depending on discrepancy) The answer below is only an example, other working algorithms are possible and equally correct. Answer: 1 Pick the first sock from the box and use it to start a row of socks 2 If the box is empty go to step 5 3 Pick a sock from the box and put it at the end of the row 4 Go back to step 3 5 Put a marker (A) on the first sock 6 If (A) is at the end of the row, there is no matching pair; Exit the algorithm 7 Put a marker (B) next to (A) 8 If the socks at (A) and (B) are identical, exit the algorithm with those two socks 9 If (B) is at the end of the row, shift (A) one to the right and go back to step 7 10 Otherwise, shift (B) to the right and go back to step 8	15 points
Total points:	*15 points*

b.

Correction criterion	Points
Of course the answer should correspond to part (a): * -3 answer of part (b) is not matching to part (a) Note that points for the correct complexity of the algorithm in (a) can be collected, even if the algorithm in (a) is not correct. Answer: In the worst case, steps 6 to 10 are executed as many times as there are pairs of socks, which is in the order of n^2 if there are n socks total. The other instructions are executed less often, so they do not affect the complexity in a fundamental way. The solution by sorting is in the order of n*log(n), but requires a sorting criterion first.	5 points
Total points:	*5 points*

Caesura

Points scored	Grade
100	10
99	9.9
98	9.8
97	9.7
96	9.6
95	9.5
94	9.4
93	9.3
92	9.2
91	9.1
90	9.0
89	8.9
88	8.8
87	8.7
86	8.6
85	8.5
84	8.4
83	8.3
82	8.2
81	8.1
80	8.0
79	7.9
78	7.8
77	7.7
76	7.6
75	7.5
74	7.4
73	7.3
72	7.2
71	7.1
70	7.0
69	6.9
68	6.8
67	6.7
66	6.6
65	6.5
64	6.4
63	6.3
62	6.2
61	6.1
60	6.0
59	5.9
58	5.8
57	5.7
56	5.6
55	5.5
54	5.4
53	5.3
52	5.3
51	5.2
50	5.1
49	5.0
48	4.9
47	4.8
46	4.8
45	4.7
44	4.6
43	4.5
42	4.4
41	4.4
40	4.3

39	4.2
38	4.1
37	4.0
36	3.9
35	3.9
34	3.8
33	3.7
32	3.6
31	3.5
30	3.5
29	3.4
28	3.3
27	3.2
26	3.1
25	3.0
24	3.0
23	2.9
22	2.8
21	2.7
20	2.6
19	2.6
18	2.5
17	2.4
16	2.3
15	2.2
14	2.1
13	2.1
12	2.0
11	1.9
10	1.8
9	1.7
8	1.7
7	1.6
6	1.5
5	1.4
4	1.3
3	1.2
2	1.2
1	1.1
0	1.0

Question identifiers

These identifiers can be used to track the exact origin of the question. Use these identifiers together with the identifier of this document when sending in comments about the questions, so that your comment can be connected precisely with the question you are referring to.

Document identifier: 13401-17779

Question number	Question identifier	Version identifier
1	129928	adda410b-001c-4ca8-874e-a83df48f83ec
2	129929	38bea22e-0a7b-4dbd-96f4-4cb01216bb2a
3	129930	0b476989-a980-411a-8e3f-287f35d9ceba
4	129931	32f08785-d6a9-458b-a297-f7f6e95ada26
5	129932	0d6ca91b-ce77-47e6-bd6c-f8f613611fec
6	129986	36a455cf-ea8b-4486-afc5-89c77881564a
7	129933	9dcbd1de-2f4a-4c6c-a697-415063ee6e34
8	129934	0f8a30e3-76ea-4984-8c43-c39aa698d779