2020-09-11 - Pearls of Computer Science Core - Pearl 001

Course: B-CS-MOD01-1A-202001022 B-CS Pearls of Computer Science Core 202001022

Duration: 1 hour

Generated on: Sep 17, 2021

Contents: Pages:

2020-09-11 - Pearls of Computer Science Core - Pearl 001

Course: B-CS-MOD01-1A-202001022 B-CS Pearls of Computer Science Core 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)**
- For technical questions (concerning the chromebooks, Remindo etc.): raise your mouse
- For pearl content question: raise your hand
- Total number of points: 100

1 Suppose you execute the following assignments in Python

5 pt.

```
room = ["Hopper", "Turing", "Lovelace"]
capacity = [129, 112, 236]
```

Here *room* is a list of room names, and *capacity* is a list of integers.

Write a Python condition (*not* an **if** statement) that tests whether room *i* has the largest capacity of the three rooms (note that you don't need to know *i*).

2 Suppose you execute the following assignments in Python

5 pt.

```
room = ["Hopper", "Turing", "Lovelace"]
capacity = [129, 112, 236]
```

Here *room* is a list of room names, and *capacity* is a list of integers.

Assign to a new list *largest* the name and the capacity of the room with the highest capacity.

3 Suppose you execute the following assignments in Python

5 pt.

```
room = ["Hopper", "Turing", "Lovelace"]
capacity = [129, 112, 236]
```

Here *room* is a list of room names, and *capacity* is a list of integers.

Write a sequence of assignments that is as short as possible, resulting in a change to *capacity* after which the integers are ordered from lowest to highest. (It is *not* correct to assign an entirely new value to *capacity*; You must modify the list by swapping elements.)

4

Question 2

Consider the following Python function:

```
01.def compute(data):
02.  even = []
03.  odd = []
04.  i = 0
05.  while i < len(data):
06.    if data[i] % 2 == 0:
07.      even.append(data[i])
08.    else:
09.      odd.append(data[i])
10.    i = i + 1
11.  return [odd, even]
```

5 pt. **a.** What is the return value upon calling compute([8, 21, 9, 19, 13, 1, 14])?

5 pt. **b.** Using your own words, what does the algorithm do?

5

Question 3

10 pt.

Consider again the Python function given in Question 2:

```
01.def compute(data):
02.  even = []
03.  odd = []
04.  i = 0
05.  while i < len(data):
06.    if data[i] % 2 == 0:
07.      even.append(data[i])
08.    else:
09.      odd.append(data[i])
10.    i = i + 1
11.  return [odd, even]
```

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.  even = []
03.  odd = []
04.  i = 0
05.  while i < len(data):
06.    if data[i] % 2 == 0:
07.      even.append(data[i])
08.    else:
09.      odd.append(data[i])
10.    i = i + 1
11.  return [odd, even]
```

Provide short explanations to the questions below and motivate your answers.

5 pt. **a.** What happens if line 10 is deleted?

5 pt. **b.** What happens if the condition *i* < *len(data)* in line 5 is replaced with *i* <= *len(data)*?

5 pt. **c.** What happens if *i = 0* in line 4 is replaced with *i = 1*?

7

10 pt.

Question 5a

Consider the following list

```
[13, 1, 18, 3, 21, 19]
```

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

8

10 pt.

Consider the following list

```
[13, 1, 18, 3, 21, 19]
```

Show how merge sort sorts this list, by presenting how the list is split and zipped back together, i.e. write down every change the algorithm makes to the list in a new line.

9
15 pt.

Question 6

You are given two (unordered) lists with student names belonging to two student houses. You want to find out whether the lists are the same. Consider two ways of doing this:

1. Sort each list by name and compare both lists element by element, and
2. For each name on list 1 try to find the matching name on list 2.

Assuming the lists are very long, which of the two methods is faster? Explain your answer as accurately as possible.

10 Assume there is a global pandemic, and to protect yourself and others from an infection, you have gathered a large collection of face masks in your wardrobe. The masks are ordered from small to big. One day you're feeling brave, and want to order them from big to small instead!

10 pt. **a.** Provide an algorithm in *human language* with **unambiguous** and **numbered** instructions that yields the desired outcome as fast as possible. Your instructions may refer to at most 2 masks at the same time -- Never an arbitrary number of them.

You may assume that you have ample space outside your wardrobe to re-arrange the masks in any way you like, but the algorithm needs to stop with every mask being at the correct place inside the wardrobe.

Do not give an answer in Python!

5 pt. **b.** How many steps does your algorithm take in terms of number of masks n .

Thank you, your exam has been saved. You will be notified about your grade after your answers were checked by the correction team. Stay safe!

Correction model

1.
5 pt.

Correction criterion	Points
-2 if an "if" is included -1 for missing 'and' -3 for the wrong list -3 for using > rather than >= -2 for other syntax mistakes Answer: capacity[i] >= capacity[0] and capacity[i] >= capacity[1] and capacity[i] >= capacity[2] In particular, avoid solving this for general lists/list sizes. The question very specifically asks to provide a condition that tests whether "room i" hast the largest capacity of the *three* rooms. No need to think of a solution for list size >3.	5 points
Total points:	*5 points*

2.
5 pt.

Correction criterion	Points
This often yields 0 points, if it is not quite correct. -4 if 'append' is used -2 for wrong brackets -3 for missing brackets -1 for one missing bracket -2 for a 2-dimensional list -3 for absence of any index Answer: largest = [room[2], capacity[2]]	5 points
Total points:	*5 points*

3.
5 pt.

Correction criterion	Points
-2 for every additional assignment beyond the first 0pts if capacity is assigned a new list -2 for wrong order -3 for wrong list (room names) Answer: capacity[0], capacity[1] = capacity[1], capacity[0]	5 points
Total points:	*5 points*

4.
10 pt.

a.

Correction criterion	Points
-2 if odd (even resp) numbers are in wrong order, but adhere to [odd, even] Beware the additional inner square brackets. Answer: [[21, 9, 19, 13, 1], [8, 14]]	5 points
Total points:	*5 points*

b.

Correction criterion	Points
Full points are given to colloquial answers, as long as they are correct on some level. Answer: It splits a list of integers into two lists comprised of a list of its odd numbers, and its even numbers respectively.	5 points
Total points:	*5 points*

5.
10 pt.

Correction criterion	Points
+5 for the correct answer +5 for the correct motivation, e.g. the answer contains something similar to 'every element' and 'exactly once' Answer: The list is traversed exactly once, hence every element is checked exactly once. The algorithm takes approximately n steps.	10 points
Total points:	*10 points*

6.
15 pt.

a.

Correction criterion	Points
Correct answer as long as similar to below. Answer: The function does not stop/terminate/finish, given that the loop condition is i < len(data)	5 points
Total points:	*5 points*

b.

Correction criterion	Points
Answers are correct as long as they represent the one below on some level Answer: This gives an "index out of range" error	5 points
Total points:	*5 points*

c.

Correction criterion	Points
Answer: The first element of data is skipped	5 points
Total points:	*5 points*

7.
10 pt.

Correction criterion	Points
This question is strictly assessed. -3 for every mistake, e.g. skipping a step up to -8 for a systematic error Answer: The consecutive steps are [13, 1, 18, 3, 21, 19] [1, 13, 18, 3, 21, 19] [1, 13, 3, 18, 21, 19] [1, 13, 3, 18, 19, 21] [1, 3, 13, 18, 19, 21]	10 points
Total points:	*10 points*

8.
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 -8 for systematic mistakes Answer: [13, 1, 18, 3, 21, 19] is split into [13, 1, 18] and [3, 21, 19] - [13, 1, 18] is split into [13] and [1, 18] -- [1, 18] is split into [1] and [18] -- [1] and [18] are merged into [1, 18] - [13] and [1, 18] are merged into [1, 13, 18] - [3, 21, 19] are split into [3] and [21, 19] -- [21, 19] are split into [21] and [19] -- [21] and [19] are merged into [19, 21] - [3] and [19, 21] are merged into [3, 19, 21] [1, 13, 18] and [3, 19, 21] are merged into [1, 3, 13, 18, 19, 21]	10 points
Total points:	*10 points*

9.
15 pt.

Correction criterion	Points
+5 For the correct method +5 for giving correct complexity of method 1 +5 for giving correct complexity of method 2 Alternatively +5 if the explanation is correct, but does not go into detail Answer: Sorting the names and then comparing is faster. Looking up each name takes n steps, and has to be done n times, thus complexity n^2. Sorting a list takes n*log n, and has to be done twice, so 2n*log n. Additionally we have to compare n elements, so n+2n log n. Example: n = 8, then n^2 = 64, and n+2n log n = 56	15 points
Total points:	*15 points*

10.
15 pt.

a.

Correction criterion	Points
-2 to -6 for ambiguities -5 if an answer is given in Python -8 if more than two masks are used in an instruction, e.g. 'Take all masks and put them in reversed order' Answer: An example algorithm could look like the one below, but every answer that adheres to the restrictions and fulfills the desired outcome gets awarded full points: We assume there is a table to temporarily store the masks outside: 1. Take the right-most mask out of the wardrobe and put it right of the most recently placed mask on the table, or to the very left if the table is still empty 2. If there are no more masks in the wardrobe, go to step 3; else go to step 1 3. Take the left-most mask from the table and place it to the right of the most recently placed mask in the wardrobe, or to the left-most position of the wardrobe is empty 4. If there are no more masks on the table we are done, else we go to step 3	10 points
Total points:	*10 points*

b.

Correction criterion	Points
This gives 0 or 5 points, hardly anything in between. Points are given if (b) reflects the steps of the algorithm in (a) correctly, even if (a) is wrong. Answer: For the example algorithm above, it takes 2n steps, given that each of the n masks is picked up twice.	5 points
Total points:	*5 points*

Caesura

Applied guessing score: 0 pt

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: 3072-8119

Question number	Question identifier	Version identifier
1	34017	c4ca5577-3409-b736-5630-212e9faa8bad
2	34022	60c5ecdf-d307-09dc-b157-3f5c73ce75a0
3	34027	e026f723-96e2-8e2a-465f-6ffe9a249a94
4	34052	815fa95e-37c8-d55d-c3c4-3a191df0a2b6
5	34057	029de39e-a09c-dabd-7313-920913a519d1
6	34062	e6e2d4b2-1c2a-2727-b104-4733fdc9bfe0
7	34042	cc67f439-5164-f2c2-97bf-e0d80380b009
8	34047	20d774bf-fb63-2961-f58a-1adef78d0987
9	34037	0972b10c-a455-c17b-4400-0c6377126efd
10	34032	20548e39-cca3-2ace-acd3-061f15d1ebc4