

Grade: 9.1 Course: Diamonds of Computer Science 20250033 Test name: 2025-09-30 Software Diamond Algorithmics - Exam Status: Closed Type: Exam Time left: 298 min. and 45 sec.

No filters selected; all answers are displayed

Question 1

Answered on: September 30, 2025 - 3:51 PM Duration: 7 min. and 35 sec. Score: 15 of 15 pts.

15 pts.

Response model changed by M. Gerhold on Oct 20, 2025, 4:13 PM for reason "Post exam correction fix"

Given are the following lists in Python

```
Python:
movies = ["Inception", "Titanic", "Avatar"]
rating = [8.8, 7.9, 7.8]
```

The list 'movies' contains titles of popular movies and 'ratings' contains their respective (fictional) ratings from 1 (bad) to 10 (good).

Write a Python condition (not an if-statement) that tests whether 'Inception' has the highest rating of these three movies.

5 pts.

```
rating[0] > rating[1] and rating[0] > rating [2]
```

Status: Finished with checking

- * -2 if an "if" is included
- * -2 for missing 'and'
- * -2 if both < and <= (> >= resp.) are used
- * -3 for the wrong list (movies vs rating)
- * -3 if the lowest rated language was used instead of the most popular (< vs >)
- * -1 if the same element was used for comparison and everything else is correct, eg.
rating[0] < rating[1] and rating[0] < rating[1]
- * -1 for rating[0], rating[1], rating[2] = rating[2], rating[1], rating[0]
- * -2 for syntax mistakes, eg. => instead of >=

Answer: We will accept both > and >= to determine the 'highest rated' language (< and <= respectively):

- rating[0] > rating[1] and rating[0] > rating[2]
- rating[0] >= rating[1] and rating[0] >= rating[2]

5 of 5 pts.

Assign to a new list 'lowest_rated' both the movie and its rating that has the lowest rating. You must access the elements directly from the existing lists 'movies' and 'ratings' using indexing.

5 pts.

```
lowest_rated = [movies[2], rating[2]]
```

Status: Finished with checking

- This often yields 0 points if it is not quite correct:
 - * -4pt if 'append' or 'extend' are used
 - * -4pt if "lowest_rated=" 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: lowest_rated = [movies[2], rating[2]]

NB: Forming an entirely new list is only partially correct. The intent is to access existing lists, e.g., "lowest_rated = ["Avatar", 7.8] is only partially correct (2pt)

5 of 5 pts.

Write a sequence of assignments that is as short as possible, resulting in a change to the list 'ratings' after which the numbers are ordered from **lowest to highest**.

5 pts.

NB: It is not correct to assign an entirely new value to 'ratings'. You must modify the list by swapping elements.

```
rating[0], rating[2] = rating[2], rating[0]
```

Status: Finished with checking

- * 0 if 'rating' 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 'movies'
 - * -2 missing comma(s)
- points
- Answer: In Python we can do a variable swap like: rating[0], rating[2] = rating[2], rating[0]

5 of 5 pts.

Question 2

Answered on: September 30, 2025 - 3:57 PM Duration: 4 min. and 31 sec. Score: 6 of 10 pts.

6 pts.

Consider the following Python function:

```
Python:
01. def compute(arr):
02.     result = 0
03.     i = 0
04.     while i < len(arr):
05.         j = 0
06.         while j < len(arr):
```

```

07.         if arr[i] > arr[j] and arr[i] - arr[j] > result:
08.             result = arr[i] - arr[j]
09.             j = j + 1
10.             i = i + 1
11.         return result

```

What is the return value upon calling compute([5])?

0

Status: Finished with checking

- The result is 0; In particular, the algorithm never reaches result = arr[i] - arr[j].
- 2 of 2 pts.**

2 pts.

What is the return value upon calling compute([4,2,0])?

4

Status: Finished with checking

- The answer should be 4, as the maximum difference between 4 to 0 is 4.
- 2 of 2 pts.**

2 pts.

What is the result of calling compute(["kayak","canoe","speedboat"])?

TypeError

Status: Finished with checking

- The result should be a type error, failing in the line of the 'if statement'.
- 2 of 2 pts.**

2 pts.

Using your own words, what does the algorithm *actually* do?

NB: Do not list line-by-line what the algorithm does, but describe the overall purpose of the function.

The function will return the highest number in a given list.

Status: Finished with checking

- The algorithm finds the 'maximum difference' of elements in a list.
- 0 of 4 pts.**

0 pts.

Question 3

Answered on: September 30, 2025 - 3:58 PM Duration: 1 min. and 55 sec. Score: 10 of 10 pts.

10 pts.

Consider again the algorithm of Question 2:

```

Python:
01. def compute(arr):
02.     result = 0
03.     i = 0
04.     while i < len(arr):
05.         j = 0
06.         while j < len(arr):
07.             if arr[i] > arr[j] and arr[i] - arr[j] > result:
08.                 result = arr[i] - arr[j]
09.                 j = j + 1
10.             i = i + 1
11.     return result

```

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

- Approximately n
- Approximately n^2
- Approximately $\log_2(n)$
- Approximately $n \cdot \log_2(n)$

Clearly state your choice and motivate your answer as precisely as you can.

NB: Just stating the complexity will not give any points. Your answer needs to be motivated.

The answer is 2. Approximately n^2 . The reason for this is that it will iterate twice through the list. One time at line 4 and one time at line 6. Meaning that for every item in the list (n) you will go through every item in the list (n). Meaning that the complexity is $n \cdot n = n^2$.

Status: Finished with checking

- Points are given for (i) the correct answer (3pt) and (ii) for the correct motivation (7pt).

In particular:

- Saying "It behaves like a nested loop algorithm" with an explanation gives, but missing the explanation can get a maximum of 7pt because $O(n^2)$ was correctly identified but not necessarily explained.
- Only mentioning "because of the nested loops" without further clarification is insufficient, resulting in a maximum of 7pt.
- 3pt per vague/ambiguous statement.

Complexity: n^2 steps.

Reasoning: Two full nested loops each run n times, so $n \cdot n = n^2$ iterations of the if. Inside each iteration there's a constant-time check ($arr[i] > arr[j]$ and, if needed, $arr[i] - arr[j] > result$) and at most one constant-time assignment. So total work is $c_1 \cdot n^2 + c_2 \cdot n + c_3 = O(n^2)$. Best/average/worst are all quadratic because both loops always go through the full ranges regardless of data.

10 of 10 pts.

10 pts.

Question 4

Answered on: September 30, 2025 - 4:06 PM Duration: 7 min. and 26 sec. Score: 12.5 of 15 pts.

12.5 pts.

Response model changed by M. Gerhold on Sep 30, 2025, 6:34 PM for reason "Changed points for 4b from 1 to 5"

Consider, once again, the algorithm of Question 2:

```
Python:
01. def compute(arr):
02.     result = 0
03.     i = 0
04.     while i < len(arr):
05.         j = 0
06.         while j < len(arr):
07.             if arr[i] > arr[j] and arr[i] - arr[j] > result:
08.                 result = arr[i] - arr[j]
09.                 j = j + 1
10.             i = i + 1
11.     return result
```

What happens to the **algorithmic complexity** of 'compute' if we replace the line 'j = 0' with 'j = i' in line 5?

Briefly motivate your answer.

5 pts.

This means that instead of "for every element in the list -> it will go through every element in the list" it now is: for every element in the list -> it will go through every element in the list but a bit less, since you are looking at fewer elements in the second iteration. That said the complexity will still be $n * n$, since the complexity doesn't change, just the amount of elements it needs to look through in the second iteration which will result in the program being a bit faster.

Status: Finished with checking

1. Changing $j = 0$ to $j = i$ makes the inner loop run $n-i$ times for each i , so the total number of iterations is $n + (n-1) + \dots + 1 = n(n+1)/2$, which is still $\Theta(n^2)$. In other words, we roughly halve the comparisons (we no longer check both (i, j) and (j, i)), but only by a constant factor; the asymptotic complexity remains quadratic.

Only stating "still quadratic" will only give half of the points. An explanation as to why the complexity remains the same is needed to get the other half.

5 of 5 pts.

What happens to the **return value** of 'compute' if we delete the clause 'arr[i] > arr[j] and' from the condition in line 7?

Briefly motivate your answer.

2.5 pts.

It will not change the return value, the return value would be the same if you didn't delete that line.

Status: Finished with checking

1. Removing 'arr[i] > arr[j] and' does not change the result. Since result starts at 0 and only increases, the remaining test $arr[i] - arr[j] > result$ can only succeed when $arr[i] > arr[j]$ anyway (positive difference), and the $j == i$ case yields 0, which never beats 'result'. The function still returns the maximum difference between any two elements, i.e., $\max(arr) - \min(arr)$.

2.5 of 5 pts.

Comment:

“ why? ”

What happens to the **algorithmic complexity** of 'compute' if we delete the clause 'arr[i] > arr[j] and' from the condition in line 7?

Briefly motivate your answer.

5 pts.

The complexity will not change, since it is still a loop with a loop; so a second iteration without the first iteration = $n * n = n^2$. The only thing that will change is that the program will run slightly faster, since there are fewer instructions, but since the complexity doesn't change. This difference in speed is small.

Status: Finished with checking

1. The complexity remains $\Theta(n^2)$: both while loops still cover all n times n index pairs and the body does constant-time work per iteration. Dropping the first clause removes some negligible steps and thus slightly increases the constant factor (we always evaluate $arr[i] - arr[j] > result$), but it does not change the complexity.

5 of 5 pts.

Question 5

Answered on: September 30, 2025 - 4:11 PM Duration: 5 min. and 24 sec. Score: 10 of 10 pts.

10 pts.

Consider the following list:

```
Python:
[9, -2, 4, -6, 3, 1]
```

Show how bubble sort sorts this list by writing down the list after every single swap the algorithm performs (i.e., every modification). Continue until the list is in ascending order.

```
start
[9, -2, 4, -6, 3, 1]
round1
[-2, 9, 4, -6, 3, 1]
[-2, 4, 9, -6, 3, 1]
[-2, 4, -6, 9, 3, 1]
[-2, 4, -6, 3, 9, 1]
[-2, 4, -6, 3, 1, 9]
round2
[-2, 4, -6, 3, 1, 9] (is the same as end of round1)
[-2, -6, 4, 3, 1, 9]
[-2, -6, 3, 4, 1, 9]
```

10 pts.

Status: Finished with checking

1. This question is strictly assessed

* -3 for every mistake, e.g. skipping a step
* up to -10 for a systematic error that indicates that the sorting algorithm has not been understood sufficiently well.

Answer: The consecutive steps are

```
-- Start --  
[9, -2, 4, -6, 3, 1]  
-- Round 1 --  
[-2, 9, 4, -6, 3, 1]  
[-2, 4, 9, -6, 3, 1]  
[-2, 4, -6, 9, 3, 1]  
[-2, 4, -6, 3, 9, 1]  
[-2, 4, -6, 3, 1, 9]  
-- Round 2 --  
[-2, -6, 4, 3, 1, 9]  
[-2, -6, 3, 4, 1, 9]  
[-2, -6, 3, 1, 4, 9]  
-- Round 3 --  
[-6, -2, 3, 1, 4, 9]  
[-6, -2, 1, 3, 4, 9]  
-- Done --  
[-6, -2, 1, 3, 4, 9]
```

10 of 10 pts.

Question 6

Answered on: September 30, 2025 - 4:14 PM Duration: 2 min. and 39 sec. Score: 10 of 10 pts.

10 pts.

Response model changed by M. Gerhold on Sep 30, 2025, 6:33 PM for reason "Changing points from 1 to 10 for all"

Consider again the list:

```
Python:  
[9, -2, 4, -6, 3, 1]
```

Show how merge sort sorts this list by writing down the list(s) after every single modification (each split and each merge operation). Write each change on a new line.

```
[9, -2, 4, -6, 3, 1]  
-----  
[9, -2, 4] [-6, 3, 1]  
-----  
*[9] [-2, 4]  
**[-2] [4]  
**[-2, 4]  
*[-2, 4, 9]  
-----  
>[-6] [3, 1]  
>>[3] [1]
```

10 pts.

Status: Finished with checking

```
1. Whole List  
[9, -2, 4, -6, 3, 1]  
--Split 1--  
[9, -2, 4] [-6, 3, 1]  
-- Splitting first half further --  
[9] [-2, 4]  
-- Single element, done --  
[9]  
-- Splitting further --  
[-2] [4]  
-- Single element, done --  
[-2]  
-- Single element, done --  
[4]  
-- merging two single elements --  
[-2, 4]  
-- merging tuple with single element --  
-- First half done--  
[-2, 4, 9]  
-- Splitting second half further --  
[-6] [3, 1]  
-- Single element, done --  
[-6]  
-- Splitting further --  
[3] [1]  
-- Single element, done --  
[3]  
-- Single element, done --  
[1]  
-- merging two single elements --  
[1, 3]  
-- merging tuple with single element --  
-- second half done--  
[-6, 1, 3]  
=== Done ===  
[-6, -2, 1, 3, 4, 9]  
10 of 10 pts.
```

Question 7

Answered on: September 30, 2025 - 4:26 PM Duration: 12 min. and 5 sec. Score: 12 of 15 pts.

12 pts.

Response model changed by M. Gerhold on Sep 30, 2025, 6:35 PM for reason "Changed b and c from 1 to 5 points"

Assume you work in the course administration at the University and the new year has just begun. You have an **unsorted** list of students with student 6-digit IDs (e.g. 123456) for 1000 new students.

Throughout the day you will need to answer membership queries like: "Is this ID already in the list?"

Here are two strategies to do this:

- **Strategy A:** For each query, use linear search to scan the list from start to finish until found/not found.
- **Strategy B:** First sort the list once (use a sorting algorithm of your choice), then answer each query with a binary search.

Clearly, there are also other things to do during the day, so you would like to be very efficient with your work.

Which of the two strategies do you use if you expect to receive **3 queries during the day**? Briefly motivate your answer.

5 pts.

The first strategy (A) has a complexity of queries * 1000, which in this case = 3 * 1000 = 3000
The second strategy (B), I will use the fastest/most efficient sorting algorithm: merge sort; which has a complexity of $1000 * \log_2(1000) +$ the binary search algorithm, which is queries * $\log_2(1000)$, resulting in $1000 * \log_2(1000) +$ queries * $\log_2(1000)$, which is roughly 9996. Since $3000 < 9996$, I would choose strategy A (By the way, I use the math module for python to calculate these 'exact' values.)

Word count: 90, character count: 501

Status: Finished with checking

1. Assuming linear search takes n steps and merge sort + binary search takes $n \cdot \log_2 n + 100 \cdot \log_2 n$ steps, where n is 1000 (the number of students) the comparison is between

Strat A: $3 * 1000 = 3000$

Strat B: $1000 * \log_2 1000 + 3 * \log_2 1000 \sim 9995$

So for these few queries option A is better.

5 of 5 pts.

Which of the two strategies do you use if you expect to receive **30 queries during the day**? Briefly motivate your answer.

5 pts.

You can use the same formulas as in the first question, with a different number of queries.
Strategy A = $30 * 1000 = 30000$
Strategy B = $1000 * \log_2(1000) + 30 * \log_2(1000)$, which is roughly 10265
Since $30000 > 10265$, I would choose strategy B. (By the way, I use the math module for python to calculate these 'exact' values.)

Word count: 59, character count: 322

Status: Finished with checking

1. Assuming linear search takes n steps and merge sort + binary search takes $n \cdot \log_2 n + 100 \cdot \log_2 n$ steps, where n is 1000 (the number of students) the comparison is between

Strat A: $30 * 1000 = 30.000$

Strat B: $1000 * \log_2 1000 + 30 * \log_2 1000 \sim 10265$

So for these queries option B is better.

5 of 5 pts.

In general, for which number of queries is **Strategy A better than Strategy B**, and for which number of queries is **Strategy B better than Strategy A**?

2 pts.

Up until 10 queries, strategy A is better. From 11 queries and onwards, strategy B is better.

Word count: 18, character count: 96

Status: Finished with checking

1. Assuming linear search takes n steps and merge sort + binary search takes $n \cdot \log_2 n + 100 \cdot \log_2 n$ steps, where n is 1000 (the number of students) the comparison is between

Strat A: $n * 1000$

Strat B: $1000 * \log_2 1000 + n * \log_2 1000$

We have to solve the inequality $1000 * \log_2 1000 + n * \log_2 1000 < n * 1000$, i.e. when does Strat B become better than Strat A. Thus, B becomes the better option at around (equal to or more than) 11 queries.

2 of 5 pts.

Question 8

Answered on: September 30, 2025 - 4:44 PM Duration: 18 min. and 11 sec. Score: 15 of 15 pts.

15 pts.

Over time you've collected a pile of **USB sticks** on your desk, each labeled with its storage capacity in **GB** (an integer). The pile is **unsorted**. You want to determine **both** the **smallest** and the **largest** capacity present.

Write an algorithm in **natural/human language**, with **numbered** and **unambiguous** steps, that finds **both** the smallest and the largest capacity of USB sticks.

10 pts.

Your actions may only ever involve **one or two USB sticks at a time** (e.g., "I am sorting ALL sticks in size" is **not** a valid operation).

NB: Do not write Python code!

1. pick 1 usb stick out of the pile of usb stick to start a line of usb sticks
2. pick 1 usb stick out of the pile and put it next in line of the already existing line of usb sticks
3. if the pile of usb sticks is empty go to step 5
4. go to step 2
5. pick the first 2 usb stick in line to keep aside and remember which one has the smallest size and which one has the largest size
6. if there are no usb sticks in line you haven't taken/hold yet, go to step 11
7. take the first usb stick in line of which you haven't taken/hold yet, and compare it your current usb stick with the smallest size -> if the usb stick you are holding is smaller than the current usb stick that is the smallest. discard the previous smallest size usb stick and keep aside the usb you were holding as the new usb stick with the smallest size (so keep the new smallest size usb stick)
8. if you didn't discard a usb stick in step 7, compare the usb stick you are holding with the current usb stick with the largest size -> if the usb stick you are holding is larger than the current usb stick that is the largest size, discard the previous largest size usb stick and keep aside the usb stick you were holding as the new usb stick with the largest size (so you keep the new largest size usb stick)
9. if you didn't discard a usb stick in step 7 and 8, discard the usb stick
10. go to step 6
11. the current smallest usb stick you were keeping aside is the smallest usb stick of all the usb sticks
12. the current largest usb stick you were keeping aside is the largest usb stick of all the usb sticks

Word count: 323, character count: 1573

Status: Finished with checking

1. we give the solution for finding the maximum. minimum runs analogously.

- 1) Take the first stick, this is now the biggest stick
 - 2) Take the next stick and compare the one we're holding with this new one (A vs B 1 comparison).
 - 3) Keep the bigger stick of these two in your hand, put the other on a 'checked' pile.
 - 4) If more ('unchecked') sticks are available, go to step 2; otherwise you have looked at all USB sticks and are holding the biggest one
- 10 of 10 pts.**

Approximately how many **comparisons** does your algorithm need as a function of the number of USB sticks n ? Give an expression and a brief justification.

For both the largest usb stick and the smallest usb sticks we look trough the pile of usb sticks, meanign that there are $2*n$ comparions made. Furthermore i first layed out the sticks from the pile in line, b

Word count: 39, character count: 207

Status: Finished with checking

1. Finding the maximum (minimum resp.) in a list of has the complexity n . Hence, searching for both minimum AND maximum means we'd have to do it twice.

Given the solution in part (a) that means the complexity would be $2n$ (although the 2 is just a constant here, n would also work).

5 of 5 pts.

5 pts.

Show one question at a time