

National University of Singapore

School of Computing

IT5003 - Data Structures and Algorithms

Midterm Test

(S1 AY2026/27; Wed, 30 September 2026)

Time Allowed: 90 minutes

INSTRUCTIONS TO CANDIDATES:

1. Do **NOT** open this assessment paper until you are told to do so.
2. This assessment paper comprises TEN (10) printed pages, including this page.
3. This is an **Open Book Assessment**.
You cannot use any electronic device except one calculator.
4. You can use either pen or pencil. Just make sure that you write **legibly**!
5. Important tips: Pace yourself! Do **not** spend too much time on one (hard) question.
Read all the questions first! Some (sub-)questions might be easier than they appear.
The first two questions in Section B have Generative AI answers (not necessarily generated with the correct prompt) that you can use/fix/improve (or rewrite).
6. If you choose to write your own answer and for the last question in Section B,
you can use **pseudo-code** in your answer but beware of penalty marks for **ambiguous answer**.
You can use **standard, non-modified** classic algorithm in your answer by just mentioning its name, e.g. run MergeSort on list A, run QuickSort on list B, etc.
7. The total marks is 50. All the best :)

A Midterm Feedback (5 marks)

Write two paragraphs about your experience with AI in the first half of IT5003 of S1 AY2026/27.

The first paragraph is to elaborate how do you use AI specifically for this course so far.

The second paragraph is about your subjective judgment: Do you think using AI (as you described in the first paragraph) makes you more confident, neutral, or less confident in facing this pen-and-paper proctored paper without AI assistance?

You will be given 1 mark per each logical sentence that you write, for up to 5 marks for this section. You can distribute 1 + 4, 2 + 3, 3 + 2, or 4 + 1 sentences across the two questions.

B Application Questions ($3 \times 15 = 45$ marks)

B.1 Visibility Count (15 marks)

There are n ($n \geq 1$) students standing in a line from index 0 (leftmost) to index $n - 1$ (rightmost). You are given an integer list H of size n that represents the heights of the n students (in micrometers, i.e., $1 \leq H[i] \leq 2 \cdot 10^6$).

You are sitting on the rightmost of the last ($n - 1$ -th) student. A student can see you if all students to his/her right have a smaller height.

You need to output the number of students who can see you.

Here are several examples:

- Input: $H = [1540000, 1520000, 1530000, 1510000]$
 Output: 3
 Explanation: 3 students except student 1 (with height $H[1] = 1520000$) can see you.
- Input: $H = [1540000, 1530000, 1520000, 1510000]$
 Output: 4
 Explanation: All students can see you.
- Input: $H = [1510000, 1520000, 1530000, 1540000]$
 Output: 1
 Explanation: Only the rightmost student 3 (with height $H[3] = 1540000$) can see you.

Solve this problem in Python. To break ties among multiple correct solutions, prioritize in order:

1. Tightest worst-case time complexity.
2. Smallest auxiliary space complexity.

B.2 Find k^{th} Smallest (15 marks)

Given an $n \times n$ ($n > 1$) square matrix M containing n^2 distinct positive integers (each not more than 10^9) where each of the rows is sorted in ascending order, return the k^{th} (1-based) smallest element in M . **You are told that $k \in [1..2 \cdot n]$.**

Here are several examples:

- Input: $M = [[1, 7, 9], [4, 11, 13], [2, 14, 15]]$, $k = 4$
Output: 7
Explanation: The sorted order of all contents of M is $[1, 2, 4, \underline{7}, 9, 11, 13, 14, 15]$.
- Input: $M = [[100, 200], [50, 250]]$, $k = 2$
Output: 100
Explanation: The sorted order of all contents of M is $[50, \underline{100}, 200, 250]$.

Solve this problem in Python. To break ties among multiple correct solutions, prioritize in order:

1. Tightest worst-case time complexity.
2. Smallest auxiliary space complexity.

B.3 Multi-Agent AI Workflows (15 marks)

You are designing an enterprise multi-agent AI pipeline consisting of a linear sequence of n specialized model nodes, represented by a 0-indexed string `pipeline` of length n characters. Each node in the pipeline is currently in one of two execution modes:

- ‘A’ represents an autonomous Agent capable of passing context fluidly to the next step, or
- ‘C’ represents a static Chatbot node that requires manual prompt intervention, breaking autonomous data flow.

To handle a complex, long-horizon reasoning task without failing, the pipeline requires a continuous, uninterrupted sequence of at least k autonomous Agent nodes (‘A’). In a single optimization step, you can upgrade a static Chatbot node (‘C’) into an autonomous Agent node (‘A’) by wrapping it in an agentic framework. Return the minimum number of upgrades needed to ensure the pipeline contains a continuous sequence of at least k autonomous Agent nodes. **You are told that $k \in [1..n]$.**

Examples:

- Input: `pipeline = "CAACCAACAC"`, `k = 7`
 Output: 3
 Explanation: One optimal way to achieve 7 consecutive online servers is to **upgrade** the servers at index 0, 3, and 4, changing the status string to "AAAAAAACAC". It can be shown that it is impossible to achieve a continuous block of 7 Agents with fewer than 3 upgrades.
- Input: `pipeline = "CACAAAC"`, `k = 2`
 Output: 0
 Explanation: No upgrades are necessary because a continuous block of 3 autonomous Agents ("CACAAAC") already exists, satisfying the requirement of at least 2.

Solve this problem in Python. To break ties among multiple correct solutions, prioritize in order:

1. Tightest worst-case time complexity.
2. Smallest auxiliary space complexity.

The Answer Sheet for S1 AY2026/27

Write your Student Number in the box below using **(2B) pencil**. Do **NOT** write your name.

STUDENT NUMBER											
A											
U	☐	0	0	0	0	0	0	0	0	A	N
A	☒	1	1	1	1	1	1	1	1	B	R
HT	☐	2	2	2	2	2	2	2	2	E	U
NT	☐	3	3	3	3	3	3	3	3	H	W
		4	4	4	4	4	4	4	4	J	X
		5	5	5	5	5	5	5	5	L	Y
		6	6	6	6	6	6	6	6	M	
		7	7	7	7	7	7	7	7		
		8	8	8	8	8	8	8	8		
		9	9	9	9	9	9	9	9		

Section	Maximum Marks	Your Marks	Grading Remarks
A	5		
B	45		
Total	50		

Box A.1. Feedback about usage of AI in IT5003 S1 AY2026/27:

How do you use AI in this course so far?

Subjectively, are you now more confident, neutral, or less confident to do this paper?

Box B.1. Visibility Count

```
def count_students_who_can_see(H):  
    stack = []  
  
    for height in H:  
        while stack and stack[-1] <= height:  
            stack.pop()  
        stack.append(height)  
  
    return len(stack)  
  
H = [154, 152, 153, 151] # example 1 of B1, all values divided by 10000  
print(count_students_who_can_see(H)) # should be 3
```

My analysis for **the given code in page 6** is ($5 \times 1 = 5$ marks):

Ignoring time complexity, it gives correct output for all valid inputs: ☐ YES | ☐ NO, I will elaborate

Ignoring correctness, it runs in: ☐ $O(n^2)$ | ☐ $O(n \log n)$ | ☐ $O(n)$ | ☐ Other, I will elaborate

The **auxiliary** space that it uses is: ☐ $O(n \log n)$ | ☐ $O(n)$ | ☐ $O(\log n)$ | ☐ Other, I will elaborate

It can crash on at least one valid corner case input: ☐ YES, I will elaborate | ☐ NO

I can improve upon the Generated AI code: ☐ YES, I will elaborate | ☐ NO, it is the best possible

Elaboration of my analysis above (**10** marks):

Only if you need to improve upon the Generated AI code.

My faster+correct code runs in: ☐ $O(n \log n)$ | ☐ $O(n)$ | ☐ $O(\log n)$ | ☐ Other, as written above,
the **auxiliary** space that it uses is: ☐ $O(n)$ | ☐ $O(\log n)$ | ☐ $O(1)$ | ☐ Other, I will elaborate.

Box B.2. Find k^{th} Smallest

```
def kth_smallest(M, k):  
    flat = [val for row in M for val in row]  
    flat.sort()  
    return flat[k - 1]  
  
if __name__ == "__main__":  
    M1 = [[1, 7, 9], [4, 11, 13], [2, 14, 15]]  
    print(kth_smallest(M1, 4))  # 7  
  
    M2 = [[100, 200], [50, 250]]  
    print(kth_smallest(M2, 2))  # 100
```

My analysis for **the given code in page 8** is ($5 \times 1 = 5$ marks):

Ignoring time complexity, it gives correct output for all valid inputs: ☐ YES | ☐ NO, I will elaborate

Ignoring correctness, it runs in: ☐ $O(n^2)$ | ☐ $O(n \log n)$ | ☐ $O(n)$ | ☐ Other, I will elaborate

The **auxiliary** space that it uses is: ☐ $O(n^2)$ | ☐ $O(n \log n)$ | ☐ $O(n)$ | ☐ Other, I will elaborate

It can crash on at least one valid corner case input: ☐ YES, I will elaborate | ☐ NO

I can improve upon the Generated AI code: ☐ YES, I will elaborate | ☐ NO, it is the best possible

Elaboration of my analysis above (**10** marks):

Only if you need to improve upon the Generated AI code.

My faster+correct code runs in: ☐ $O(k \log n)$ | ☐ $O(n)$ | ☐ $O(\log n)$ | ☐ Other, as written above,
the **auxiliary** space that it uses is: ☐ $O(n^2)$ | ☐ $O(n \log n)$ | ☐ $O(n)$ | ☐ Other, as written above.

Box B.3. Multi-Agent AI Workflows (This is a STARRED task without Generative AI assistance)

- ☐ I give up, just give me the free 2 marks (you must leave the box below totally empty or cross it)
- ☐ I know how to solve this, I will elaborate below (pseudocode is allowed and encouraged):

My solution runs in $O(\text{-----})$ and uses $O(\text{-----})$ **auxiliary** space.

– END OF PAPER; All the Best –