National University of Singapore

School of Computing

IT5003 - Data Structures and Algorithms

Midterm Test

(S2 AY2025/26; Wed, 04 March 2026)

Time Allowed: 80 minutes

INSTRUCTIONS TO CANDIDATES:

1. Do **NOT** open this assessment paper until you are told to do so.
2. This assessment paper contains TWO (2) sections.
   It comprises EIGHT (8) 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.
6. Section B of this paper has special format.
   Read the instructions carefully.
7. The total marks is 50. All the best :)

A Midterm Feedback ($2 \times 1 = 2$ marks)

Prof Halim had run IT5003 for 11 iterations. For this S2 AY2025/26, he is implementing these changes:

- What is your opinion about having the optional LeetCode exercises in this course, especially knowing that the course lecturer had solved 1 out of *any random* 2 LeetCode tasks by the start of this semester and that the tasks came from *real* technical interviews? More importantly, do you attempt these optional exercises? None/a few/some/all/much more than the ones listed.
- What is your opinion about the second IT5003 midterm test (this paper) and further increasing its importance from 10% to 16%? And about the new 'interesting' format?

B Application Questions ($3 \times 16 = 48$ marks)

Suppose for all three application questions below, you copy-pasted the entire problem statement and asked your Generative AI friend 'talkTwinSeek' to generate Python solutions for you. This GenAI generates three Python code that are shown in the answer sheet (assume that all necessary imports and the main function are added automatically).

For all three application problems of this special midterm paper, your task is to check 'talkTwinSeek' outputs. There are a few possibilities and there is a custom marking scheme depending on how accurate is your analysis and/or (updated) solutions:

AC). If you think the generated Python code is 100% correct (AC), just declare that it is correct.

WA). If you think the generated Python code is wrong (WA), identify the bug(s), explain why it is/they are buggy, and suggest as minimal fix as possible to the buggy part(s).

TLE). If you think the generated Python code is correct but is (too) slow (TLE), identify which part(s) is/are slow - analyze this slow time complexity, rewrite those part(s) with the faster solution(s), and re-analyze the faster time complexity. Note that in class/programming assignments, we generally say $\Omega(n^2)$ — i.e., n^2 or worse – is too slow and $O(n \log n)$ — i.e., $n \log n$ or faster – is needed, but if the generated code is $O(\log n)$ and you know the $O(1)$ solution, you should still suggest this improvement.

RTE). If you think the generated Python code is generally fast enough and is the correct algorithm, but it has not handled specific corner case(s) that can cause the code to crash (RTE), identify such corner case(s), and suggest as minimal fix as possible to the buggy part(s).

MLE). If you think the generated Python code uses too much memory. Analyze this big space complexity, rewrite the required part(s), then re-analyze the smaller space complexity.

Note that the generated code can be WA, TLE, RTE, and/or MLE at the same time. In this case, you may actually want to rewrite the whole thing. You are allowed to use **pseudo-code** instead of Python. But beware of penalty marks for **ambiguous answer**. As usual, you can use **standard, non-modified** classic algorithm in your answer by just mentioning its name, e.g. run Merge Sort on

list L, use Stack S, use Binary Min Heap $MinPQ$, etc. However, if your minimal fix is minor, you are allowed to just annotate directly in the generated Python code.

Remember to (re)-analyze the tightest worst-case time complexity of the fixed solution, if any.

B.1 Sort Bit String (16 marks)

You are given a bit string s of n ($n \geq 2$) characters that contains just '0' (zero) and '1' (one). In one move, you can choose any two adjacent characters in s and swap them. Compute the minimum number of moves to sort the bit string so that all the zeroes are on the left of all the ones?

Example 1: `s = "10"`
Output 1: 1
Explanation: One move is enough, e.g., `s = "10"` → `"01"`

Example 2: `s = "1010"`
Output 2: 3
Explanation: We need at least 3 moves, e.g., `s = "1010"` → `"1001"` → `"0101"` → `"0011"`

Example 3: `s = "00000111101000001101011010 10"`
Output 2: 77
Explanation: For you to self-check your solution on slightly larger test case

Solve this problem in Python and analyze its time complexity.

B.2 Simple Linked List Task (16 marks)

You are given the `head` of a Singly Linked List (SLL) with n vertices ($n \geq 1$). Delete all vertices whose values (values are positive integers not more than 99) occur *more than once* time in the SLL. Keep the ordering of the remaining vertices as in the input order. See Figure 1 and 2 for an example.

Figure 1: An example SLL; Value 2, 3, 6 appears 3x/2x/2x, respectively

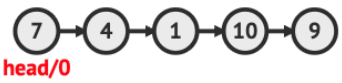


Figure 2: After vertices that occur more than once are deleted

Solve this problem in Python and analyze its time complexity.

B.3 Minimum Number of Moves (16 marks)

You are given an integer list `nums` with n integers ($n \geq 2$) and an integer t. All integers in this task are positive integer not more than 10^9. In one move, you can do the following: select the smallest integer x and the second smallest integer y from `nums`, remove both of them from `nums`, and reinsert $4 \cdot x + 2 \cdot y$ to the front (specifically at index 0) of list `nums`. Your task is to compute the minimum number of moves needed so that all elements of `nums` are $\geq t$.

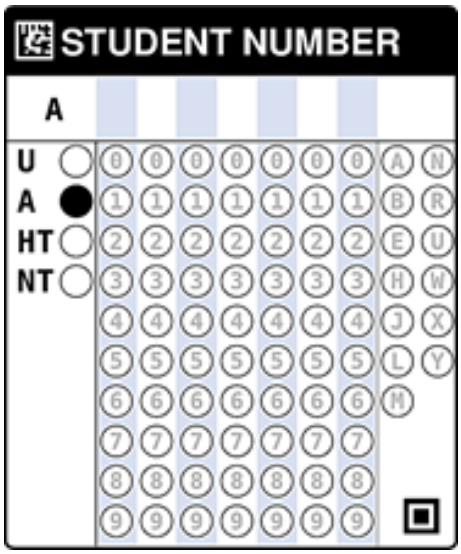
For this task, it is guaranteed that after some number of operations, all elements of `nums` are $\geq t$.

Example: `nums = [7, 1, 2, 3]` and $t = 8$
Output: 2
Explanation: After the first move, we have `nums = [8, 7, 3]`.
After the second move, we have `nums = [26, 8]`, all elements are ≥ 8.

Solve this problem in Python and analyze its time complexity.

The Answer Sheet for Semester 2 AY2025/26

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



Section	Maximum Marks	Your Marks	Grading Remarks
Total	50		

Box A.1. Feedback about IT5003 S2 AY2025/26:

What is your opinion about the optional LeetCode exercises? Do you attempt them?

What is your opinion about this new 'interesting' 2nd IT5003 midterm format?

Box B.1. Sort Bit String

```
class Solution:
    def sortBinary(self, s: str) -> int:
        s = list(s) # convert to list of chars
        ans, n = 0, len(s)
        for i in range(n):
            for j in range(n-1-i):
                if s[j] > s[j+1]:
                    s[j], s[j+1] = s[j+1], s[j]
                    ans += 1
        return ans
```

My analysis of the generated code above is:

That solution runs in $O($_____$)$.

Box B.2. Simple Linked List Task

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

class Solution:
    def deleteDuplicates(self, head: ListNode) -> ListNode:
        prev = None
        curr = head
        while curr:
            count = 0
            checker = head
            while checker:
                if checker.val == curr.val:
                    count += 1
                checker = checker.next
            if count > 1:
                prev.next = curr.next
            else:
                prev = curr
            curr = curr.next
        return head
```

My analysis of the generated code above is:

That solution runs in $O($_____$)$.

Box B.3. Minimum Number of Moves

```
class Solution:
    def minMoves(self, nums, t):
        moves = 0
        while True:
            all_good = True
            for value in nums:
                if value < t:
                    all_good = False
                    break
            if all_good:
                return moves
            nums.sort(reverse = True)
            x, y = nums[0], nums[1]
            nums.pop(0)
            nums.pop(0)
            new_value = 4 * x + 2 * y
            nums.insert(0, new_value)
            moves += 1
        return moves
```

My analysis of the generated code above is:

That solution runs in $O($_____$)$.

– END OF PAPER; All the Best –