National University of Singapore

School of Computing

IT5003 - Data Structures and Algorithms

Final Assessment

(Semester 1 AY2025/26)

Time Allowed: 2 hours

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 FOURTEEN (14) printed pages, including this page.
3. This is an **Open Book Assessment**.
   You cannot use any electronic device except one non-programmable 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. 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 Inorder Traversal on BST T, BFS on graph G, Dijkstra's on graph G', etc.
7. The total marks is 100. All the best :)

A MCQs ($10 \times 2.5 = 25$ marks)

Select the **best unique** answer for each question. Each correct answer is worth 2.5 marks.

1. Which of the following functions has the tightest worst-case time complexity bound of $O(n)$?

   a). $A(n) = n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots$

   b). $B(n) = n^2 + \sqrt{n} + 10^8$

   c). $C(n) = n \cdot \sum_{x=1}^{n} \frac{1}{x}$

   d). $D(n) = 1 + 2 + 3 + \cdots + n$

   e). $E(n) = n + n^2$

2. For the List ADT, what are the tightest worst-case time complexities for the operation $get(i)$ (accessing the element at index i) using a compact array (like a Python list) versus a Singly Linked List (SLL) containing n elements?

   a). Python list: $O(n)$, SLL: $O(1)$

   b). Python list: $O(\log n)$, SLL: $O(n)$

   c). Python list: $O(1)$, SLL: $O(1)$

   d). Python list: $O(1)$, SLL: $O(n)$

   e). Python list: $O(n)$, SLL: $O(n)$

3. We studied several comparison-based sorting algorithms in class. Which algorithm satisfies all of these criteria: 1). Has tightest worst-case time complexity of $O(n \log n)$; 2). Not in-place; and 3). Stable?

   a). Insertion Sort.

   b). Merge Sort.

   c). Randomized Quick Sort.

   d). Heap Sort.

   e). (Balanced BST) Tree Sort.

4. What is the fastest way to enumerate all integers in a Binary Max Heap of n integers with values greater than a given integer x? Suppose that there are k such integers and $k < n$.

   a). Loop through the underlying compact array A from index 1 to n and checking if $A[i] > x$

   b). Perform consecutive $ExtractMax()$ operations until the root is $\leq x$

   c). Create max heap in $O(n)$ time; check all vertices of the max heap if their values $> x$

   d). Run preorder traversal starting from the root, terminating the traversal on any subtree whose root value is $\leq x$

   e). Convert the Binary Max Heap to a Binary Search Tree first, then search for values $> x$

5. In class, we learned how to *uniquely* reconstruct a binary tree T given two tree traversal orderings of T. Which of the following tree reconstructions is **not** possible?

   a). Construct binary tree T from preorder and inorder traversals of T

   b). Construct binary tree T from postorder and inorder traversals of T

   c). Construct binary tree T from preorder and postorder traversals of T

   d). Construct binary search tree T from just preorder traversal of T

   e). Construct binary search tree T from just postorder traversal of T

6. Scenario: You are working on sparse graphs where $m \leq 7 \cdot n$ where $n = |V|$ and $m = |E|$. You need to do two things frequently on this graph: checking the existence of an edge (u, v) and enumerating neighbors of a vertex u. AM/AL are Adjacency Matrix/Adjacency List, respectively. Which of the following statements is TRUE regarding AM vs AL?

   a). AM is faster for checking edge existence ($O(1)$), but AL is faster for enumerating neighbors ($O(k)$).

   b). AM is faster than AL for both operations: $O(1)$ for checking edge existence and $O(1)$ for enumerating neighbors.

   c). AL is faster for checking edge existence ($O(1)$), but AM is faster for enumerating neighbors ($O(n)$).

   d). Both AM and AL require $O(n)$ time for checking edge existence and enumerating neighbors.

   e). AL is faster than AM for both operations because $O(n+m)$ space complexity of AL implies faster queries for both operations.

7. Consider the LeetCode problem rotting-oranges, which can be summarized as follows: Given an $m \times n$ grid where 0, 1, and 2 represent empty, fresh, and rotten cells respectively, each minute every fresh orange that is 4-directionally (N/E/S/W) adjacent to a rotten one becomes rotten. Return the minimum number of minutes required for all oranges to become rotten, or -1 if this is impossible. We solved this unweighted SSSP problem using BFS algorithm from all sources (rotten cells). Which of the following statements is FALSE?

   a). The space requirement if we use an Edge List representation is $O(m \cdot n)$.

   b). The space requirement if we use an Adjacency List representation is $O(m \cdot n)$.

   c). The space requirement if we use an Adjacency Matrix representation is $O(m \cdot n)$.

   d). We can choose not to store the graph explicitly and instead derive the implicit unweighted directed edges from each rotten cell to its 4-directional neighbors as needed during BFS.

   e). None of the above.

8. LeetCode problem 'number-of-provinces' can be abridged as follows: Count the number of Connected Components (CCs) in a given undirected graph of n vertices (given as a symmetric Adjacency Matrix of size $n \times n$). We solved this problem by calling $BFS(s)$ from *the lowest vertex number s of EACH CC*, possibly up to n times. Suppose there are m edges in the graph and you are told that $m < n^{1.5}$ (the graph is sparse enough). What is the resulting tightest worst-case time complexity?

   a). $O(n + m)$.

   b). $O(n^2)$.

   c). $O(n \log n)$.

   d). $O(n^2 + nm)$.

   e). $O(n^3)$.

9. Consider the Depth-First Search (DFS) algorithm with ternary-states shown in class for detecting cycles in a directed graph $G = (V, E)$. Recall that in this DFS version, the three states per vertex are: unvisited, (currently) explored, and (fully) visited. Which of the statements below is FALSE?

   a). With three states, DFS can identify a back (cyclic) edge $(u \rightarrow v)$ if the current vertex u encounters a neighbor v already marked as (currently) explored but not (fully) visited.

   b). With only two states: unvisited versus already visited, DFS cannot identify a back (cyclic) edge versus other non-spanning tree edge types.

   c). The time complexity of this DFS version is $O(|V|+|E|)$ if the graph is stored in an Adjacency List (AL).

   d). The time complexity of this DFS version is $O(|V|^2)$ if the graph is stored in an Adjacency Matrix (AM).

   e). None of the above.

10. You are given a positive-weighted Directed Acyclic Graph (DAG) $G = (V, E)$. Which among the five algorithms below is the fastest for correctly solving the SSSP problem on that DAG?

    a). Bellman-Ford in $O(|V| \times |E|)$ as the faster $O((|V| + |E|) \log |V|)$ Dijkstra's algorithm would fail because it cannot handle general weighted graphs.

    b). Dynamic Programming (DP) after Topological Sort in $O(|V| + |E|)$.

    c). Modified Dijkstra's in $O((|V| + |E|) \log |V|)$ because it can handle negative edges (even though none exist).

    d). BFS in $O(|V| + |E|)$ because the graph is acyclic.

    e). Original Dijkstra's in $O(|V| \log |V|)$ because $|E| \in O(|V|)$ in a DAG.

B Application Questions ($5 \times 15 = 75$ marks)

B.1 ICPC Problem (15 marks)

You are given a string s containing only the characters `'c'`, `'i'`, and `'p'`. The length of the string s is n characters ($n \geq 1$). Your task is to determine whether the string s is special (output `True` or `False`).

A string s is considered special if, starting with an empty string $w =$ `""`, you can transform w into s by performing the following operation any number of times:

> Insert the 4-character string `"icpc"` into any position in w.

Below are some examples where the underlined "icpc" shows each insertion step:

No	Input string s	Output	Explanation
1	`"icicpcpc"`	`True`	`""` → `"`icpc`"` → `"ic`icpc`pc"`
2	`"icpicpcc"`	`True`	`""` → `"`icpc`"` → `"icp`icpc`c"`
3	`"icpicpccicpc"`	`True`	`""` → `"`icpc`"` → `"icpc`icpc`"` → `"icp`icpcc`icpc"`
4	`"icpccpic"`	`False`	Cannot form s using only the insert operation
5	`"icicicic"`	`False`	No 'p'; cannot form s using only the insert operation
6	`"icpci"`	`False`	Length of s is not a multiple of 4

Table 1: A Few Examples for the ICPC Problem

For 7 marks, design any correct algorithm that solves this task in $O(n^2)$ time complexity.
For 11 marks, the correct algorithm must run in $O(n \log n)$ time complexity.
For full 15 marks, the correct algorithm must run in $O(n)$ time complexity or faster.

B.2 Special List (15 marks)

You are given a list L of n positive integers ($n \geq 1$ and each integer is $\in [1..100]$). An integer i is called special if:

- i appears exactly once in the list, and
- Neither $i - 1$ nor $i + 1$ appears in the list.

You are asked to output the list of all special integers in L in increasing order.

Example 1. If $L = [3, 7, 1, 12, 5, 10, 8]$, then the output is $[1, 3, 5, 10, 12]$.

Example 2. If $L = [3, 7, 1, 12, 5, 10, 8, 3, 10]$, then the output is $[1, 5, 12]$.

Example 3. If $L = [3, 7, 1, 12, 5, 10, 8, 3, 10, 6]$, then the output is $[1, 12]$.

For 7 marks, design any correct algorithm that solves this task in $O(n^2)$ time complexity.
For 11 marks, the correct algorithm must run in $O(n \log n)$ time complexity.
For full 15 marks, the correct algorithm must run in $O(n)$ time complexity or faster.

B.3 Count Number of Vertices with Special Grandparent (15 marks)

You are given the root of a Binary Search Tree (BST) containing n ($n \geq 1$) distinct integers, where each integer lies in the range $[1, 10^9]$. You are asked to output the number of vertices that have a grandparent (i.e., the parent of their parents, if it exists) whose value is divisible by 7 — that is, values such as 7, 14, 21, and so on.

As discussed in class, each BST vertex has the following attributes available to you:

- `value` (an integer)
- `left` (reference to the left child)
- `right` (reference to the right child)

Assume that each vertex does not have a reference to its parent, i.e., from a vertex, we cannot go to its parent (or grandparent) directly.

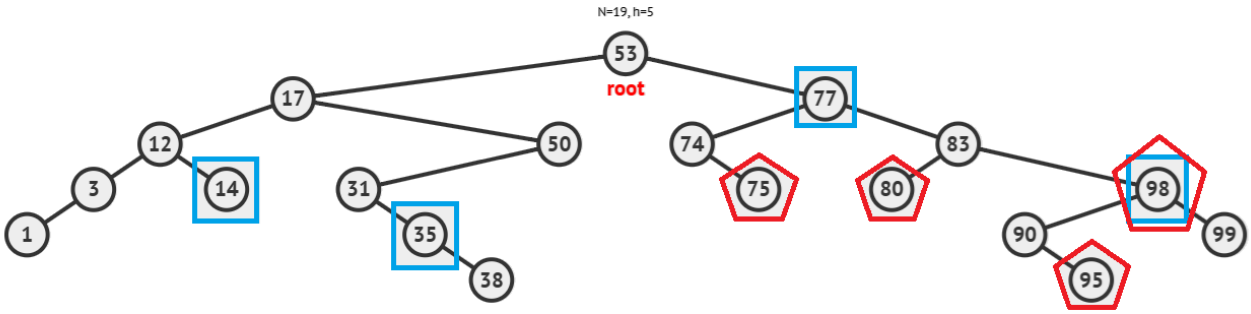


Figure 1: An example BST T with $n = 19$ vertices.

For example, if the given BST T is as shown in Figure 1, the output should be 4 (vertices with values 75, 80, 95, and also 98). The vertices with values 75 and 80 each have a grandparent with value 77, which is divisible by 7. The vertex with value 95 has a grandparent with value 98, which is divisible by 7. Additionally, the vertex with value 98 (which itself is divisible by 7) has a grandparent with value 77, which is divisible by 7.

Note that there are two other vertices whose own values are divisible by 7 — those with values 14 (a leaf) and 35 (a vertex with only one right child) — but they do not contribute to the answer, since we are only counting vertices whose **grandparent's** value is divisible by 7.

For answering this problem, please complete the skeleton Python code shown in the answer sheet (Page 12). There are 15 small boxes worth 1 mark each and each one must be filled with the correct **short Python code** (pseudo-code is not allowed for this question).

B.4 Ordering in a Photo (15 marks)

There are n ($n \geq 2$) friends who lined up for a photo in a single long row, from left (index 0) to right (index $n-1$). You have forgotten the exact ordering of them, but somehow (for the sake of this problem), you remember some distinct pairs of two friends who were standing next to each other in the photo.

You are given m ($1 \leq m \leq n-1$) pieces of distinct pairing information in the form of an $m \times 2$ list of string pairs, where $pair[i] = [u_i, v_i]$ means that the two friends named u_i and v_i (each a short string of at most 10 characters) stood next to each other in the photo.

Your task is to output the ordering of your friends (as a list of n names). If there are exactly two possible orderings, output the one where the leftmost friend (index 0 of the output) has the lexicographically smallest name – see Example 1 and 2. If the given *pair* information cannot be used to produce exactly two possible orderings, output `["??"]` – see Example 3, 4, and 5.

Example 1. If *pair* = `[["steven","grace"],["arief","erlyn"],["arief","steven"]]`,
then there are two possible orderings:
`["grace","steven","arief","erlyn"]` or `["erlyn","arief","steven","grace"]`.
The correct output is `["erlyn","arief","steven","grace"]` as `"erlyn" < "grace"`.

Example 2. If *pair* = `[["steven","grace"]]`,
then there are two possible orderings:
`["grace","steven"]` or `["steven","grace"]`.
The correct output is `["grace","steven"]` as `"grace" < "steven"`.

Example 3. If *pair* = `[["steven","grace"],["arief","grace"],["arief","steven"]]`,
then the information forms a complete graph K_3, not a row, so the output is `["??"]`.

Example 4. If *pair* = `[["steven","grace"],["arief","steven"],["steven","erlyn"]]`,
then the information forms a star graph $K_{1,3}$, not a row, so the output is `["??"]`.

Example 5. If *pair* = `[["steven","grace"],["arief","erlyn"]]`,
then the information is insufficient (two disconnected edges), so the output is `["??"]`.

For 7 marks, design any correct algorithm that solves this task in $O(n^2)$ time complexity.
For 11 marks, the correct algorithm must run in $O(n \log n)$ time complexity.
For full 15 marks, the correct algorithm must run in $O(n)$ time complexity or faster.

B.5 Shortest Swim Route (15 marks)

You are given an $m \times n$ topographical `grid` where a cell is 1/0 if it is a land/ocean cell, respectively. There are exactly two distinct islands shown in the grid. An island is a 4-directionally (N/E/S/W) connected group of 1s not connected to the other group of 1s. You are now on the first island (it doesn't matter which one, just call it island A). Your task is to first identify the perimeter (the edge cells) of the first island, e.g., island A, and the perimeter of the second/the other island, e.g., island B. Then, find the shortest distance between any cell on the first perimeter and any cell on the second perimeter, where distance is measured by the number of ocean (0) cells that must be traversed. This will be your shortest swim route from island A to island B. Return the shortest distance.

Example 1: $m = 7$, $n = 14$, and `grid` is as follows:

```
[[0,0,0,0,0,0,0,0,0,0,0,0,0,0],
 [0,1,1,1,1,0,0,0,0,0,1,1,0,0],
 [1,1,1,1,1,0,0,0,0,0,0,1,1,1],
 [1,1,1,1,0,0,0,0,0,0,1,1,1,1],
 [1,1,1,1,0,0,0,0,0,0,0,1,1,1],
 [0,1,1,0,0,0,0,0,0,0,0,0,1,0],
 [0,0,0,0,0,0,0,0,0,0,0,0,0,0]]
```

Then the perimeter of one island/the other is highlighted with A's/B's, respectively (with 0s hidden). The shortest swim route involves five 0 cells, marked with 'X's. We return 5.

```
[[ , , , , , , , , , , , , , , ],
 [ ,A,A,A,A,X,X,X,X,X,B,B, , ],
 [A,1,1,1,A, , , , , , ,B,B,B],
 [A,1,1,A, , , , , , , ,B,1,1,B],
 [A,1,1,A, , , , , , , , ,B,1,B],
 [ ,A,A, , , , , , , , , ,B, ],
 [ , , , , , , , , , , , , , ]]
```

Example 2: $m = 3$, $n = 14$, and `grid` is as follows:

```
[[0,0,0,0,0,0,0,0,0,0,0,0,0,0],
 [0,1,0,0,0,0,0,0,0,0,0,0,0,0],
 [0,0,0,0,0,0,0,0,1,0,0,0,0,0]]
```

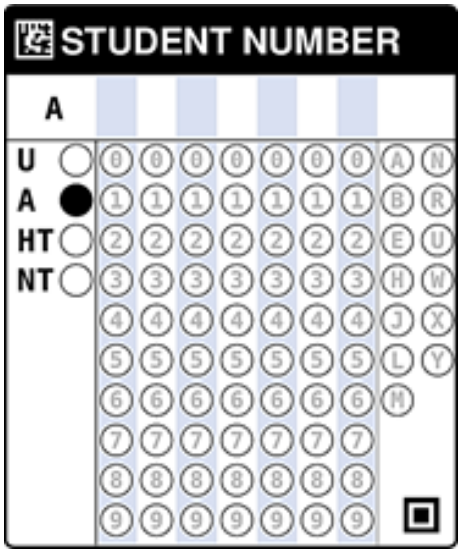
The shortest swim route involves seven 0 cells, marked with 'X's. We return 7.

```
[[ , , , , , , , , , , , , , , ],
 [ ,A,X,X,X,X, , , , , , , , ],
 [ , , , , ,X,X,X,B, , , , , ]]
```

Design any correct algorithm that solves a special case of this task for 7 marks: both islands have size $1 \cdot 1 = 1$ cell only, like Example 2 above.
For full 15 marks, the correct algorithm must run in $O(m \cdot n)$ time complexity or faster for the full version of this task, like Example 1 above.

The Answer Sheet for Semester 1 AY2025/26

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



This portion is for examiner's use only

Section	Maximum Marks	Your Marks	Grading Remarks
A	25		
B	75		
Total	100		

Write your MCQ answers in the special MCQ answer box below for automatic grading.
We do not manually check your answer.
Shade your answer properly (use (2B) pencil, fully enclose the circle; select just one circle).

No	1	2	3	4	5	6	7	8	9	10
A	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯
B	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯
C	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯
D	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯
E	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯

Box B.1. ICPC Problem

I claim that my solution runs in $O($_____$)$ worst-case time complexity.

Box B.2. Special List

I claim that my solution runs in $O($_____$)$ worst-case time complexity.

Box B.3. Count Number of Vertices with Special Grandparent

```
# Definition for a BSTvertex.
# class BSTVertex:
#     def __init__(self, value=0, left=None, right=None):
#         self.value = value
#         self.left = left
#         self.right = right
class Solution:
    def countVertices(self, root: Optional[BSTVertex]) -> int:
        def traverse(root):

                        ----------------    ----------------
            if root == [01.             ]: [02.             ]
                        ----------------    ----------------
                ------------------------------------------------
            if [03.                                            ]:
                ------------------------------------------------
                    ---------------------------------
                if [04.                             ] != None:
                    ---------------------------------
                                        ---------------------------------
                    self.ans += 0 if [05.                               ] == None else 1
                                        ---------------------------------
                                    ----------------
                    self.ans += [06.             ] if root.left.left == None else 1
                                    ----------------
                                     ----------------
                if root.right != [07.             ]:
                                     ----------------
                                    ----------------
                    self.ans += [08.             ] if root.right.left != None else 0
                                    ----------------
                                              ----------------
                    self.ans += 1 if root.[09.             ].right != None else 0
                                              ----------------
                      ----------------
            traverse([10.             ].right)
                      ----------------
             ----------------        ----------------
            [11.             ](root.[12.             ])
             ----------------        ----------------

        self.ans = 0

                  ----------------
        traverse([13.             ])
                  ----------------
                  ----------------------                                  ----------------
        return [14.                   ] # Overall time complexity is O([15.             ])
                  ----------------------                                  ----------------
```

Box B.4. Ordering in a Photo

I claim that my solution runs in $O($_____$)$ worst-case time complexity.

Box B.5. Shortest Swim Route

I claim that my solution runs in $O($_____$)$ worst-case time complexity.

– END OF PAPER; All the Best –