Course: Data Structures (CSE CS203A, 114-1)

Final Exam (Answer Key)

December 23, 2025, 15:10~17:00

Language: English or Chinese (please use one language consistently)

Student ID: Student Name:

Q1: **Comprehensive Multiple-Choice / 綜合選擇題** (10 pts, 1 pt each)

1. The primary goal of a hash function in a hash table is to:

   A. Sort keys in ascending order

   B. Map keys to table indices efficiently

   C. Guarantee zero collisions

   D. Convert integers to strings

   雜湊表中的雜湊函式其主要目標是：

   A. 將鍵以遞增順序排序

   B. 將鍵高效率地映射到表格索引位置

   C. 保證零碰撞

   D. 把整數轉換成字串

2. Compared to an adjacency matrix, an adjacency list is usually better for:

   A. Dense graphs

   B. Sparse graphs

   C. Sorting vertices

   D. Guaranteeing O(1) BFS

   相較於鄰接矩陣（adjacency matrix），鄰接串列（adjacency list）通常較適合：

   A. 稠密圖

   B. 稀疏圖

   C. 排序頂點

   D. 保證 O(1) 的 BFS

3. If a graph has V vertices and is represented by an adjacency matrix, checking whether edge (u, v) exists is:

   A. $O(\log V)$

   B. $O(V)$

   C. $O(1)$

   D. $O(E)$

   若一個圖有 V 個頂點，並以鄰接矩陣表示，檢查邊 (u, v) 是否存在的時間複雜度是：

A. O(log V)

B. O(V)

C. O(1)

D. O(E)

4. A leaf node is a node with:

A. Exactly one child

B. Exactly two children

C. A parent pointer

D. No children

葉節點是具有下列哪種特性的節點：

A. 正好一個子節點

B. 正好兩個子節點

C. 有父節點指標

D. 沒有子節點

5. In a max-heap, every node must satisfy the following property:

A. Parent ≥ children

B. Parent ≤ children

C. Left child ≥ right child always

D. The tree is a binary search tree (BST)

在最大堆積(max-heap)中，每個節點必須滿足下列性質：

A. 父節點 ≥ 子節點

B. 父節點 ≤ 子節點

C. 左子節點必定 ≥ 右子節點

D. 該樹是一棵二元搜尋樹(BST)

6. A BST becomes degenerate (behaving like a linked list) when:

A. Keys are inserted in random order

B. Keys are inserted in already sorted order (ascending or descending)

C. The tree is complete

D. The tree is full

一棵二元搜尋樹(BST)在以下何種情況會退化(行為像鏈結串列)：

A. 鍵以隨機順序插入

B. 鍵以已排序(遞增或遞減)的順序插入

C. 該樹是完全二元樹

D. 該樹是滿二元樹

7. When deleting a BST node with two children, a standard strategy is to replace it with:

A. Any leaf node

B. Its inorder predecessor or inorder successor

C. The root always

D. The deepest node in the tree

刪除一個擁有兩個子節點的 BST 節點時，標準策略是用下列哪一個替代它：

A. 任意葉節點

B. 其中序前驅或中序後繼

C. 一律用根節點

D. 樹中最深的節點

8. In separate chaining, collisions are handled by:

A. Searching the next empty slot

B. Maintaining a list (or similar structure) at each table index

C. Rebuilding the table on every collision

D. Using only prime table sizes

在分離鏈結（separate chaining）中，碰撞的處理方式是：

A. 尋找下一個空槽

B. 在每個表格索引位置維護一個串列（或類似結構）

C. 每一次碰撞都重建表格

D. 只使用質數大小的表格

9. If you observe many collisions for string keys with a naive hash = “sum of ASCII”. The best immediate improvement is:

A. Stop using strings as keys

B. Limit table size to 2^k only

C. Use a polynomial rolling hash (e.g., base * accumulation)

D. Only store lowercase characters

若你對字串鍵使用單純的雜湊＝「ASCII 碼總和」而觀察到大量碰撞，最好的即時改進是：

A. 停止使用字串作為鍵

B. 將表格大小只限制為 2^k

C. 改用多項式滾動雜湊(例如以基底進行累乘累加)

D. 只儲存小寫字元

10. The Left-Child Right-Sibling (LCRS) representation is primarily used to:

A. Represent a general tree using a binary tree structure

B. Convert a graph into a tree

C. Convert a heap into a BST

D. Eliminate the need for recursion

左孩子-右兄弟(LCRS)表示法主要用於：

A. 用二元樹結構表示一般樹

B. 把圖轉換成樹

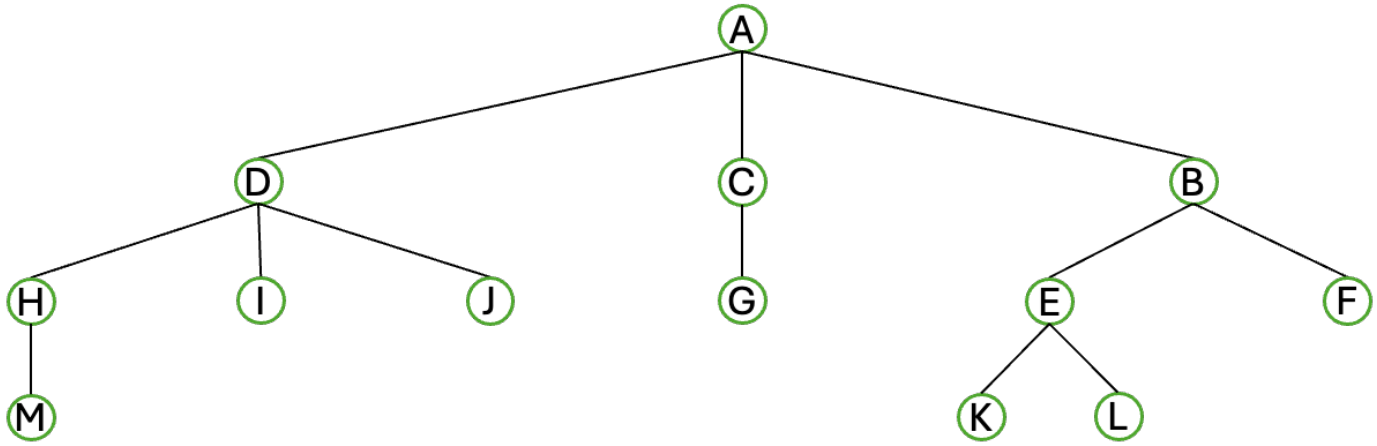
C. 把堆轉換成二元搜尋樹

D. 消除遞迴的需求

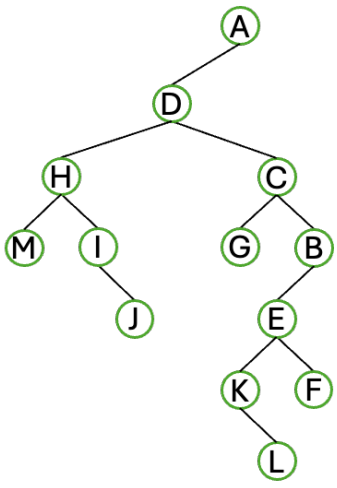
Q2: **Tree Transformation / 樹狀圖轉換** (10 pts)

Convert the tree in the figure below into a binary tree by Left-Child Right-Sibling (LCRS) approach.

將下圖中的樹轉換為左孩子-右兄弟(LCRS)架構的二元樹。



A:



Q3: **Two Sum /** 兩數之和 (21 pts)

Given an array of integers nums and an integer target. Return indices of the two different elements in nums such that: nums[i] + nums[j] = target

Rules:

- Exactly one valid answer exists.
- You cannot use the same element twice (i ≠ j).
- You may return the indices in any order.

Tasks: Provide two solutions

Solution 1: Brute Force Method (Straightforward approach)

Solution 2: Efficient Method (Your time complexity must be lower than Solution 1)

Deliverables

a. The data structure(s) you use in Solution 1 and Solution 2, respectively

b. Time and space complexity (and a short comparison between the two solutions)

c. A brief explanation (2-4 sentences) of why Solution 2 is faster than Solution 1

	Solution 1	Solution 2
Data Structure	(3 pts)	(3 pts)
Time complexity	(3 pts)	(3 pts)
Space complexity	(3 pts)	(3 pts)
Brief explanation	(3 pts)	

給定一個整數陣列 nums 與一個整數 target。請回傳 nums 中兩個不同元素的索引，使得：nums[i] + nums[j] = target。

規則

- 恰有一組有效解。
- 不可重複使用同一元素 (i ≠ j)。
- 可以用任意順序回傳索引。

任務：請提供兩種解法

解法一：暴力法 (直接做法)

解法二：高效法 (其時間複雜度必須低於解法一)

繳交項目

a. 你在解法一與解法二分別使用的資料結構

b. 時間與空間複雜度(並對兩種解法做簡短比較)

c. 為何解法二比解法一更快的簡要說明(1-3 句)

A:

	Solution 1	Solution 2

Data Structure	Array	Hash Table (Hashing)
Time complexity	$O(n^2)$	O(n)
Space complexity	O(n)	O(n)
Brief explanation	Solution 2 is faster because the Hash Table allows for O(1) average-time lookups of the "complement" value (target−current), replacing the inner O(n) loop with a constant-time check.	

Q4: **Real-world Application: ER Registration & Triage (應用：急診室登記與檢傷分類)** (24 pts)
Scenario: In the emergency room, patients first line up at the counter for registration. After preliminary assessment, each patient is assigned a triage level from Level 1 to Level 5:
Level 1 – Resuscitation
Level 2 – Emergent
Level 3 – Urgent
Level 4 – Less Urgent
Level 5 – Non-urgent
You are a developer building a medical information system.

Tasks

Choose appropriate data structures to manage:

1. Registration line at the counter (patients arrive and are served in arrival order)
2. Triage process (patients are treated according to priority level)

Deliverables

For each part (1) and (2), provide:

1. Data structure name
2. Core operations you will use
3. Why it fits the workflow (1–3 sentences)
4. Time complexity for the main operations you listed (big-O)

情境：在急診室，病患會先到櫃檯排隊完成掛號。經初步評估後，每位病患會被分派從第 1 級到第 5 級的檢傷分類：
第 1 級 - 復甦 (Resuscitation)
第 2 級 - 緊急 (Emergent)
第 3 級 - 急迫 (Urgent)
第 4 級 - 較不緊急 (Less Urgent)
第 5 級 - 非緊急 (Non-urgent)
你是一位正在建置醫療資訊系統的開發者。

任務

選擇合適的資料結構來管理：

1. 櫃檯掛號排隊 (病患按到達順序受理)

2. 檢傷流程 (病患依優先等級接受處置)

繳交項目

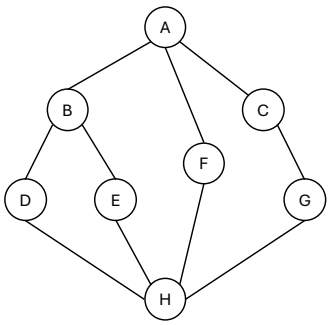
對 1 與 2 各自提供：

1. 資料結構名稱
2. 將使用的核心操作
3. 為何符合該工作流程 (1–3 句)
4. 主要操作的時間複雜度 (Big-O)

A:

Task	Registration Line	Triage process
Data Structure	Queue	Min Heap
Operations	enqueue(), dequeue()	insert(), extract_min()
Reason	Matches arrival-order service	treatment order (highest-priority patient first)
Time Complexity	O(1) for both operations	O(log n) for insertion and extraction

Q5: **Graph Traversal: Breadth-First Search (BFS), Depth-First Search (DFS) / 圖形走訪：BFS 與 DFS** (12 pts)



You are given an unweighted graph and will traverse it starting from vertex A using:

- Breadth-First Search (BFS)
- Depth-First Search (DFS)

Assume the graph may contain cycles.

Traversal rule:

- Alphabetical Order: When a vertex has multiple unvisited neighbors, visit neighbors in alphabetical order.
- Check Before Push: To prevent redundant entries and cycles, you must check if a neighbor is already visited before pushing it onto the stack or queue.
- Immediate Marking: Once a vertex is pushed, it must be marked as "visited" immediately to ensure it is not added to the data structure again by a different path.

Tasks

1. Non- recursion (iterative) implementation

   Without using recursion, which data structure would you use to implement:

   - BFS?
   - DFS?

2. Avoiding infinite loops

   What mechanism can be used to avoid infinite loops when traversing a graph?

3. Traversal order

Start from vertex A, write the visiting order (sequence of vertices) for:

- BFS
- DFS

4. BFS/DFS and shortest path (concept question)

In an unweighted graph, explain why BFS can be viewed as a shortest-path algorithm from the start vertex. Then contrast this with DFS: why doesn't DFS guarantee shortest paths?

Deliverables

For each question, provide:

1. Data structure choice
   - BFS uses: (2 pts)
   - DFS uses: (2 pts)
2. Cycle prevention (2 pts)
   - Mechanism:
   - Brief explanation of how it works (1–2 sentences).
3. Traversal result
   - BFS order from A: A → … (2 pts)
   - DFS order from A: A → … (2 pts)
4. Shortest path connection (2 pts)
   - BFS as shortest path (2–4 sentences):
   - DFS contrast (1–3 sentences):

	BFS	DFS
Data Structure	(2 pts)	(2 pts)
Cycle prevention	(2 pts)	
Traversal Order	(2 pts)	(2 pts)
Shortest Path	(2 pts)	

給定一張無權重圖 (unweighted graph)，並從頂點 A 開始進行走訪，使用：

- 廣度優先搜尋 (BFS)
- 深度優先搜尋 (DFS)

假設圖中可能包含環 (cycle)。

走訪規則：

- **字母順序（Alphabetical Order）**：當某頂點有多個尚未造訪的鄰居時，依字母順序造訪這些鄰居。
- **推入前檢查（Check Before Push）**：為避免重複加入與形成循環，將鄰居推入堆疊（stack）或佇列（queue）之前，必須先檢查該鄰居是否已被造訪。
- **立即標記（Immediate Marking）**：一旦將頂點推入資料結構，就必須立即將其標記為「已造

訪」，以避免由其他路徑再次被加入。

任務

1. 非遞迴 (迭代實作)

   在不使用遞迴的情況下，分別會用哪一種資料結構來實作：

   - BFS？
   - DFS？

2. 避免無限迴圈

   走訪圖時，可使用何種機制避免無限迴圈？

3. 走訪順序

   從頂點 A 出發，寫出造訪順序(頂點序列)：

   - BFS
   - DFS

4. BFS／DFS 與最短路徑(概念題)

   在無權重圖中，說明為何 BFS 可視為從起點出發的最短路徑演算法。並與 DFS 對照：為何 DFS 不保證得到最短路徑？

繳交項目

1. 資料結構選擇
   - BFS 使用：
   - DFS 使用：
   - 各自的一句話理由。
2. 循環防止
   - 機制：
   - 簡要說明該機制如何運作 (1–2 句)。
3. 走訪結果
   - 從 A 的 BFS 順序：A → …
   - 從 A 的 DFS 順序：A → …
4. 最短路徑關聯
   - BFS 作為最短路徑（2–4 句）：
   - DFS 的對比（1–3 句）：

A:

	BFS	DFS
Data Structure	Queue	Stack
Cycle prevention	A "Visited" set or boolean array. It tracks vertices already processed to prevent re-entry.	
Traversal Order	A → B → C → F → D → E → G → H	A → B → D → H → E → F → G → C

Shortest Path	BFS explores level-by-level, ensuring the first time a node is reached, it is via the fewest edges. DFS explores paths as deeply as possible, potentially finding a long path before a short one.

Q6: **Binary Search Tree (BST) /** 二元搜尋樹 (9 pts)

The insertion order of keys affects the final shape (height and balance) of a binary search tree.
Given the input integers (40, 35, 30, 25, 15, 16, 18, 20, 21), insert them one by one into an initially empty BST to construct the tree.
Rule for duplicates: If a key equals the current node's key, insert it into the right subtree.

Tasks

1. Construct the BST

   Draw (or clearly represent) the resulting BST after all insertions.

2. Describe the shape of the BST

   Use correct terminology (e.g. degenerate, unbalanced, complete) to describe the resulting tree

3. Rebuild into a more balanced BST

   Propose the idea that take the resulting BST and rebuild it into a more balanced BST.

Deliverables

1. BST construction (3 pts)

   Your final BST drawing

2. Shape description (3 pts)

   Terminology used

3. Rebalancing algorithm (3 pts)

   Key idea

已知鍵的插入順序會影響二元搜尋樹的最終形狀（高度與平衡）。
給定輸入整數 (40, 35, 30, 25, 15, 16, 18, 20, 21)，從一棵初始為空的二元搜尋樹（BST）開始，依序插入以建構該樹。
重複值規則：若鍵值等於目前節點的鍵值，則將其插入右子樹。

任務

1. 建立 BST

   將所有插入完成後的 BST 畫出（或以清楚的方式表示）。

2. 描述 BST 的形狀

   使用正確術語（例如：退化、不平衡、完全（二元樹））來描述結果樹。

3. 重建為更平衡的 BST

   提出將結果 BST 重建成更平衡 BST 的構想。

繳交項目

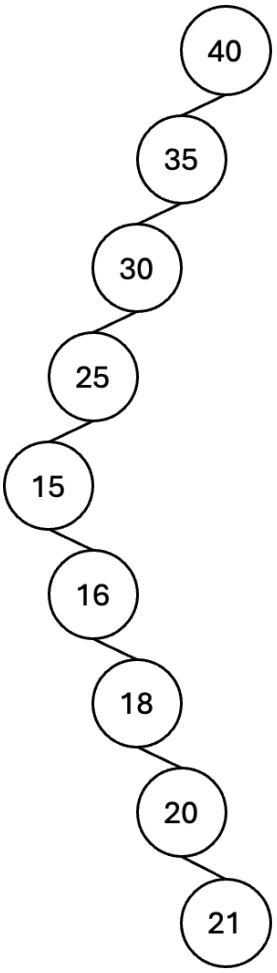
1. BST 建構

最終的 BST 圖示。

2. 形狀描述
   使用之術語。
3. 重新平衡演算法
   核心構想。

A:

1. Shape



2. Terminology: Unbalanced, Degenerate/Skewed
3. Rebalancing: Store keys in a sorted array and pick the middle element as the root recursively.

Q7: **Time complexity** (14 pts, 1 pt each)
Based on your understanding of array and pointer, the data structures we learned include linked lists, stacks, queues, trees, heaps, graphs, and hash tables.
根據你對陣列與指標的理解，我們所學的資料結構包含：鏈結串列、堆疊、佇列、樹、堆、圖、以及雜湊表。

1. Given the definition of graph G(V, E), the worst-case cost of graph-processing operations (Hints: You can use degree(u) to represent the number of connections to vertex u, and V or E for total

vertices/edges)

依據圖的定義 $G(V,E)$，說明圖處理操作的最壞情況成本。(提示：你可以使用degree(u)表示與頂點 u相連的邊數，並以V與E分別代表頂點／邊的總數。)

Operation 操作	Adjacency Matrix 鄰接矩陣	Adjacency List 鄰接串列
Space complexity 空間複雜度	①	②
Find if edge (*u*, *v*) exists 檢查邊 (*u*, *v*) 是否存在	③	④
Determine if vertex v is isolated? 判斷頂點 *v* 是否為孤立點？	⑤	⑥
Path from *u* to *v*? *u* 到 *v* 是否有路徑？	⑦	⑧

2. Compare the average-case time complexity for a Binary Search Tree (BST) and a Hash Table.

比較二元搜尋樹（BST）與雜湊表在平均情況下的時間複雜度

Operation 操作	Binary Search Tree 二元搜尋樹	Hash Table 雜湊表
Search/Insert 搜尋／插入	①	②
Find Minimum key 找出最小鍵	③	④
Print all keys sorted 按順序列印所有鍵	⑤	⑥

A:

1. Graph vs. Tree

Operation	Adjacency Matrix	Adjacency List
Space complexity	$O(V^2)$	O(V+E)
Find if edge (*u*, *v*) exists	O(1)	O(degree(u))
Determine if vertex v is isolated?	O(V)	O(1)
Path from *u* to *v*?	$O(V^2)$	O(V+E)

2. BST vs. Hash Table

Operation	Binary Search Tree	Hash Table
Search/Insert	O(log n)	O(1)

Find Minimum key	O(log n)	O(n)
Print all keys sorted	O(n)	O(nlog n) or $O(n^2)$