Student ID: 1131417 Student Name:魏允鴻

Course: Data Structures (CSE CS203A)

Assignment V: Tree

Student Worksheet Companion

Due date: 2025.12.30 23:59:59

Academic Integrity and AI Usage Statement

In this assignment, you must use AI tools (such as ChatGPT, Gemini, Claude, Grok, M365 Copilot, etc.) as learning assistants, but you must also take full responsibility for understanding and organizing your own work.

1. Permitted Use of AI Tools

   You may use AI to:

   - Review or clarify definitions and concepts.
   - Compare different tree data structures.
   - Get suggestions for report layout or examples.
   - Ask for explanations of algorithms (e.g., BST insertion, AVL rotation, heapify process).

   You should read, think about, and rewrite the content in your own words.

2. Not Permitted
   - Do not copy/paste AI-generated content directly as your final answer.
   - Do not ask AI to draw the final diagrams or directly produce the final tree screenshots.
   - Do not ask AI to complete the whole assignment report for you.
3. Your Responsibility
   - You are responsible for understanding the definitions and algorithms.
   - You are responsible for verifying whether AI answers are correct or not.
   - You must produce your own original explanations and diagrams.
4. AI Usage Log
   - You must record all AI queries related to this assignment.
   - At the end of your report, include an AI Usage Log table with: Index, Prompt, AI service name.

   By submitting this assignment, you acknowledge that you have used AI tools only as study aids, and that the final content of this assignment represents your own understanding and work.

Section 1. Definitions of Tree Variants

Task: Write your own definitions for each tree type. You may use AI for learning, but rewrite in your own words.

1. General Tree

   Definition: 一般樹(General Tree)是一種階層式的資料結構，具備以下特質：

   (1)由多個節點(node)組成。

   (2)子節點(child)的數量無限制。

   (3)整棵樹只有一個根節點(root)

   (4)除了根節點之外，每個節點都只有一個父節點(parent)。

2. Binary Tree

   Definition: 二元樹（Binary Tree）是一種樹狀資料結構，具備以下特質：

   (1) 每個節點最多只能擁有兩個子節點，分別為左子節點（left child）與右子節點（right child）。

3. Complete Binary Tree

   Definition: 完全二元樹（Complete Binary Tree）是一種有形狀規則的二元樹，具備以下特質：

   (1) 除了最後一層之外，其餘各層的節點皆被完全填滿。

   (2) 最後一層的節點必須由左至右依序排列。

   (3) 適合以陣列(array)方式儲存。

4. Binary Search Tree (BST)

   Definition: 二元搜尋樹（Binary Search Tree, BST）是一種具有排序規則的二元樹，具備以下特質：

   (1) 任一節點的左子樹中所有節點值皆小於該節點值。

   (2) 任一節點的右子樹中所有節點值皆大於該節點值。

   (3) 利用中序遍歷(Inorder)可以列出由小到大的數字排列

5. AVL Tree

   Definition: AVL Tree 是一種自我平衡的二元搜尋樹(BST)，具備以下特質：

   (1) 任一節點的左右子樹高度差最多為 1。

   (2) 當插入或刪除節點造成失衡時，會透過旋轉操作進行調整。

   (3) 樹高被限制在 O(logn)

6. Red-Black Tree

   Definition: 紅黑樹（Red-Black Tree）是一種自我平衡的二元搜尋樹(BST)，具備以下特質：

   (1) 每個節點會被標示為紅色或黑色。

   (2) 透過顏色規則限制樹的高度，避免嚴重失衡。

   * 根節點必為黑色
   * 兩個紅色節點不可為父子關係
   * 根節點到每個葉節點的路徑中，經過的黑節點數相同

   (3) 插入與刪除操作時所需的調整次數較少，效能穩定

7. Max Heap

   Definition: 最大堆積樹（Max Heap）是一種完全二元樹，具備以下特質：

   (1) 每個父節點的值皆大於或等於其子節點的值。

(2) 整棵樹中的最大值必定位於根節點。

(3) 常用於實作優先權佇列。

8. Min Heap

Definition: 最小堆積樹（Min Heap）是一種完全二元樹，具備以下特質：

(1) 每個父節點的值皆小於或等於其子節點的值。

(2) 整棵樹中的最小值必定位於根節點。

(3) 適合用於需要快速取得最小值的應用。

Section 2. Tree Family Hierarchy and Transformations

Task: Show how these structures are related (general → specialized). Use a simple diagram and explanations of what constraints are added at each step.

2.1 Tree Family Diagram

You may draw this by hand and paste a photo, or use drawing tools.

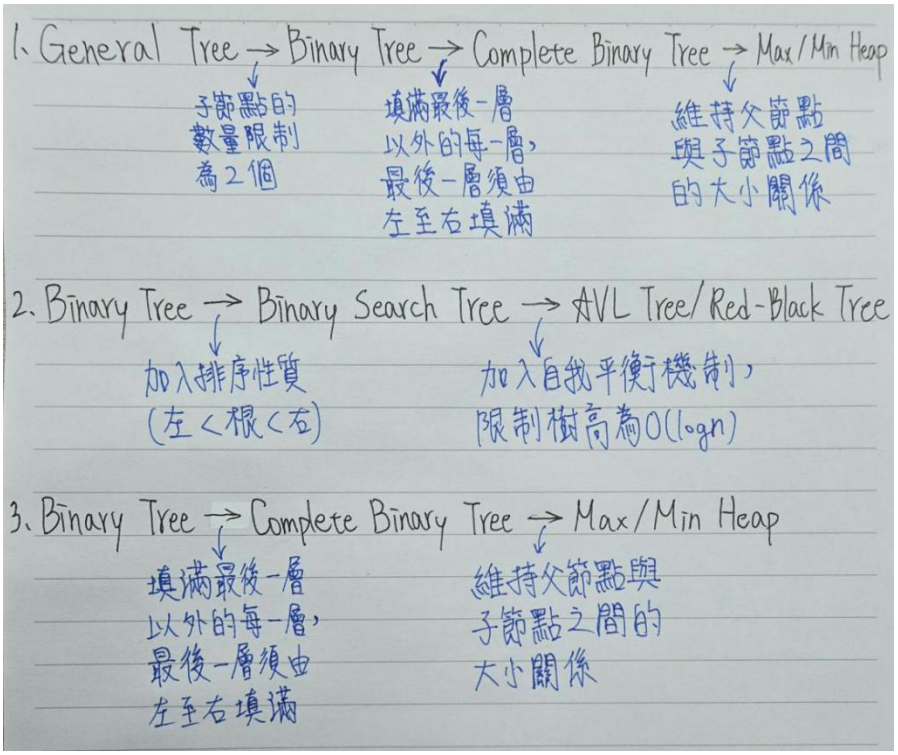
Suggested chain example (you may extend or adjust):

General Tree → Binary Tree → Complete Binary Tree

Binary Tree → Binary Search Tree → AVL / Red-Black

Binary Tree → Max Heap / Min Heap

Your Diagram:



2.2 Explanation of Transformations

Fill in what new property or constraint is added at each step.

From	To	New property / constraint added

General Tree	Binary Tree	限制每個節點最多只能擁有兩個子節點。
Binary Tree	Complete Binary Tree	除最後一層外，每一層節點皆須完全填滿，且最後一層節點必須由左至右依序排列。
Binary Tree	Binary Search Tree	加入排序性質，使左子樹所有節點值小於根節點，右子樹所有節點值大於根節點。
BST	AVL Tree	加入高度平衡機制，於插入或刪除節點時，透過旋轉操作使任一節點左右子樹高度差不超過 1，以限制樹高維持在 O(log n)。
BST	Red-Black Tree	加入顏色標記與平衡規則，於插入或刪除節點時，透過顏色調整與旋轉操作限制樹的高度，維持樹高為 O(log n)。
Binary Tree	Max Heap	在完全二元樹結構下，加入父節點與子節點之間的大小關係限制，使父節點的值大於或等於其子節點的值。
Binary Tree	Min Heap	在完全二元樹結構下，加入父節點與子節點之間的大小關係限制，使父節點的值小於或等於其子節點的值。

Section 3. Tree Constructions Using Given Integers

Given integers (fixed for all parts):

37, 142, 5, 89, 63, 117, 24, 176, 58, 133, 92, 11, 151, 72, 39, 184, 7, 101, 54, 160

Task: For each tree type below, construct the tree using these integers, take a screenshot of the tree from your chosen tool, record the tool name/URL, and describe the insertion / heap-building procedure.

3.1 Binary Tree

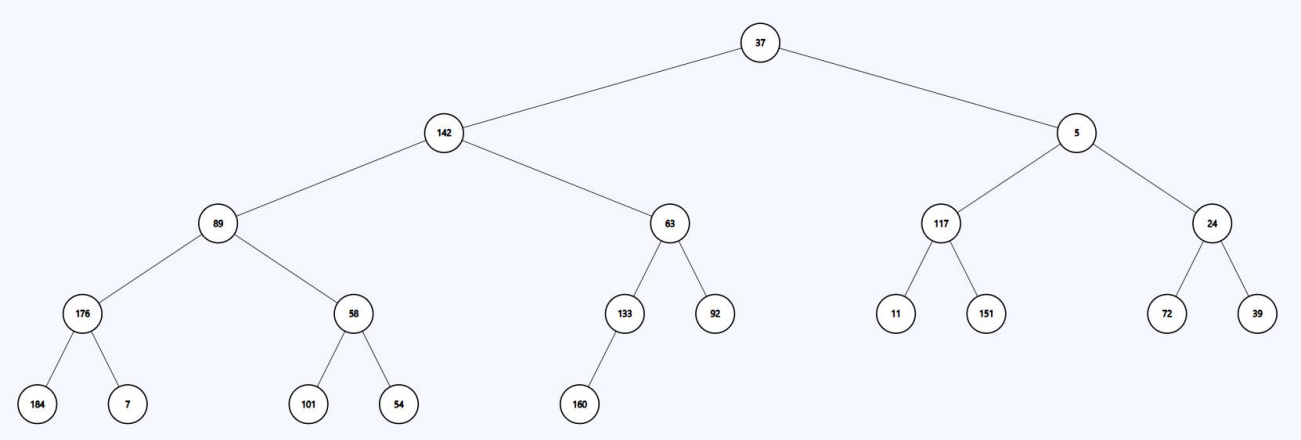
Tool name / URL:

Tree Visualizer

Construction / insertion description:

Binary Tree 的節點沒有固定的插入順序或排序規則，每個節點最多只能擁有兩個子節點（左子節點與右子節點）。其結構僅受「子節點數量上限」限制，因此樹的形狀不固定，可能是不平衡的。

Screenshot of Binary Tree (paste below):



3.2 Complete Binary Tree

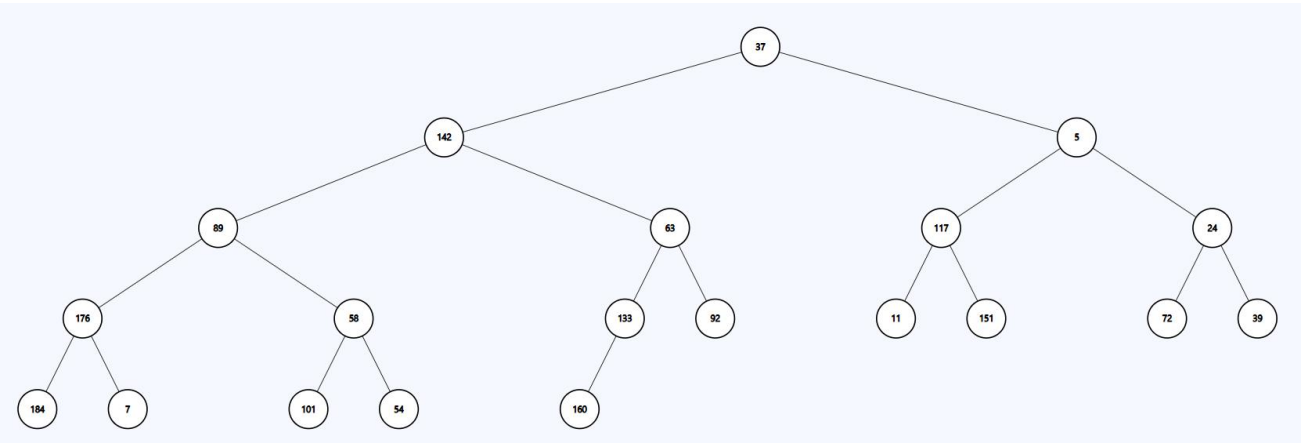
Tool name / URL:

Tree Visualizer

Construction / insertion description:

將節點依層級從左至右依序插入，每個節點最多只能有兩個子節點。除了最後一層之外，每一層的節點都必須完全填滿，而最後一層的節點必須由左至右排列。

Screenshot of Complete Binary Tree (paste below):



3.3 Binary Search Tree (BST)

Tool name / URL:

Tree Visualizer

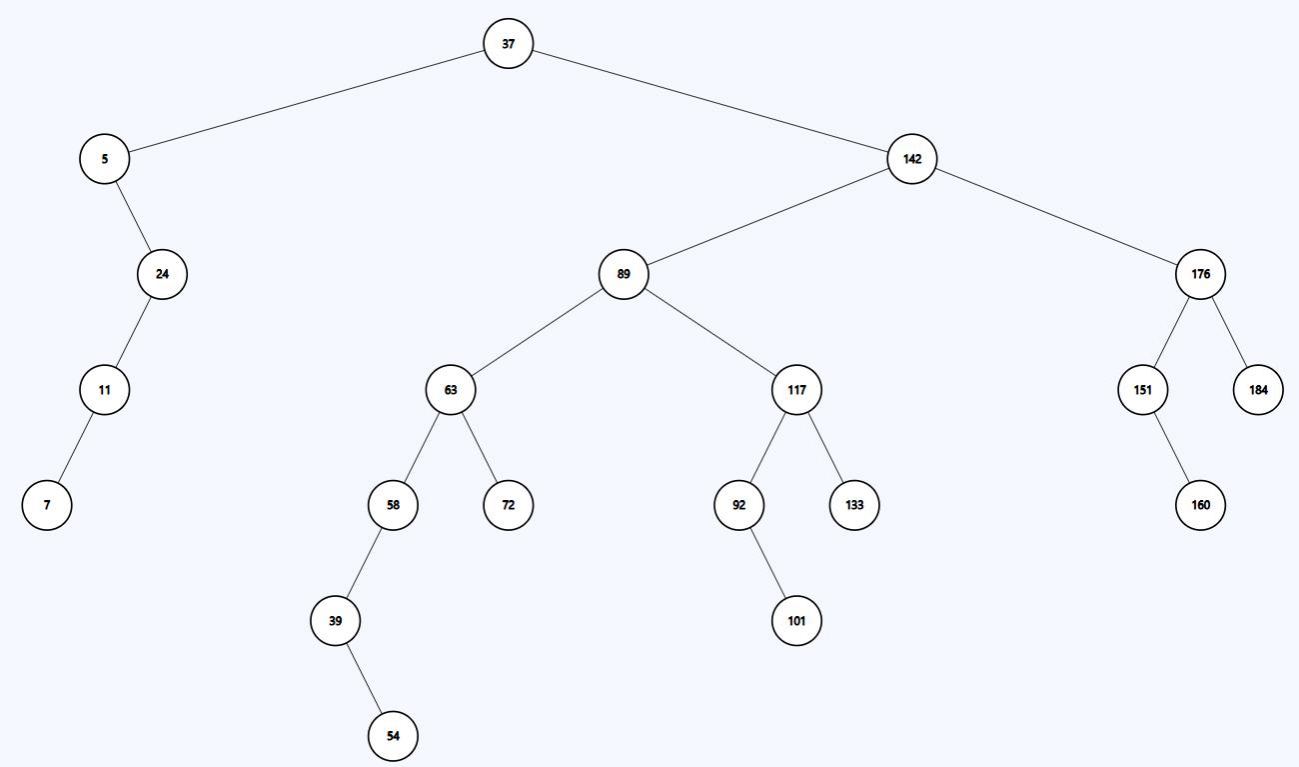
Insertion rule (e.g., “insert in given order using BST rules”):

規則：任一節點的左子樹中所有節點值皆小於該節點值，任一節點的右子樹中所有節點值皆大於該節點值。

插入（Insertion）步驟：

1. 第一個數字作為根節點。
2. 依序插入剩餘數字：比當前節點小 → 插入左子樹，比當前節點大 → 插入右子樹
3. 重複步驟 2 直到所有數字都插入完畢。

Screenshot of BST (paste below):



3.4 AVL Tree

Tool name / URL:

Tree Visualizer

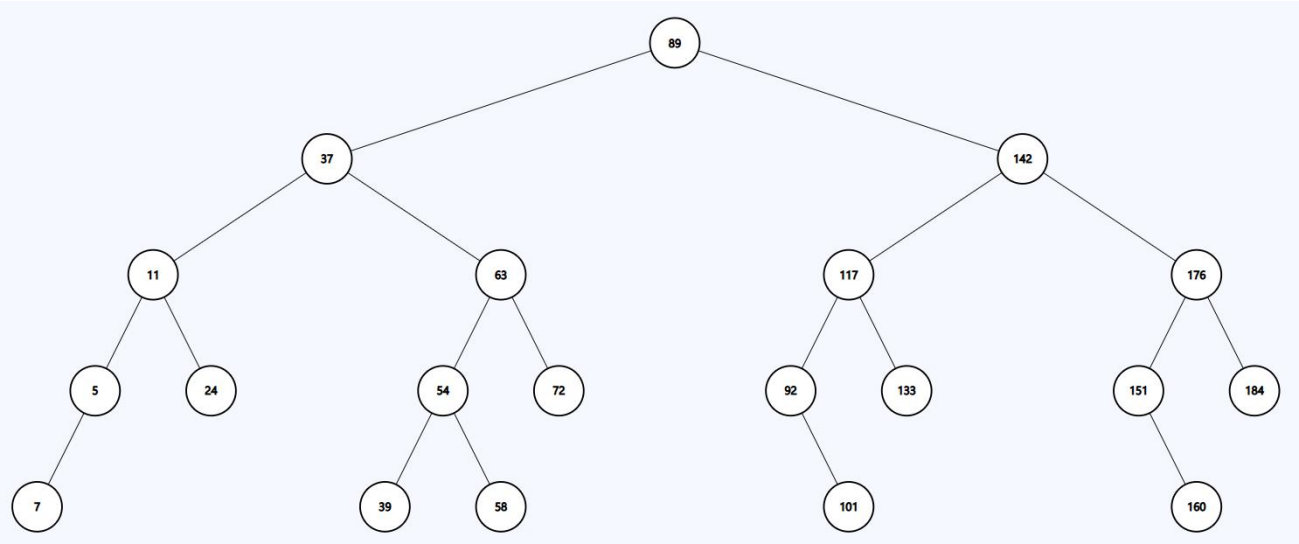
Insertion & balancing description:

目的：讓任意節點的左右子樹高度差小於 1，方法是利用遞迴選取中間值構建新樹。

插入（Insertion）步驟：

1. 中序遍歷（Inorder Traversal）原 BST 取得由小到大的排序序列：按「左子樹→根→右子樹」順序遍歷整棵 BST，把所有節點的鍵值依次排成遞增序列。
2. 選取序列中點為根：整個序列的中間值作為新樹的根，將左側子序列分配給左子樹，右側子序列分配給右子樹。
3. 遞迴構建子樹：對左側和右側子序列重複上述取中點的步驟，遞迴產生左右子樹，直到子序列為空為止。
4. 連接子樹：遞迴返回時把構建好的左右子樹連接到當前節點，這樣每個節點的左右子樹高度差不超過 1，保證 AVL 樹的平衡性。

Screenshot of AVL Tree (paste below):



3.5 Red-Black Tree

Tool name / URL:

Red/Black Tree Visualization

Insertion & balancing description:

插入（insertion）步驟：

1. 將新節點著色為紅。

2. 若父節點為黑，插入結束。

3. 若父節點和叔父節點皆為紅，則將父節點和叔父節點著色為黑，將祖父節點著色為紅，然後把祖父節點當成新插入點繼續向上檢查。

4. 若父節點為紅、叔父節點為黑（或不存在），則根據新節點在父節點的相對位置決定旋轉方式：

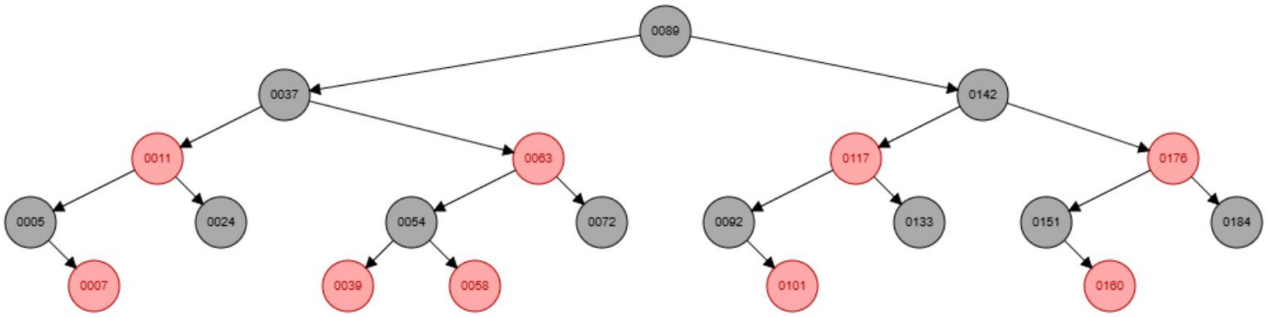
4-1. 左左（LL）或右右（RR）：新節點和父節點在同一側時，對祖父節點做一次單旋（LL 用右旋，RR 用左旋），並在旋轉前將父節點著色為黑、祖父節點著色為紅。

4-2. 左右（LR）或右左（RL）：新節點和父節點在不同側時，先對父節點做一次旋轉將其變成同側，再回到 LL/RR 情況處理。

5. 完成所有插入後的紅黑樹：**遵照上面的修正規則，最終得到的紅黑樹根節點會是黑色的 89，整棵樹高度約為 4 層

Student ID: 1131417 Student Name:魏允鴻

Screenshot of Red-Black Tree (paste below):



3.6 Max Heap

Tool name / URL:

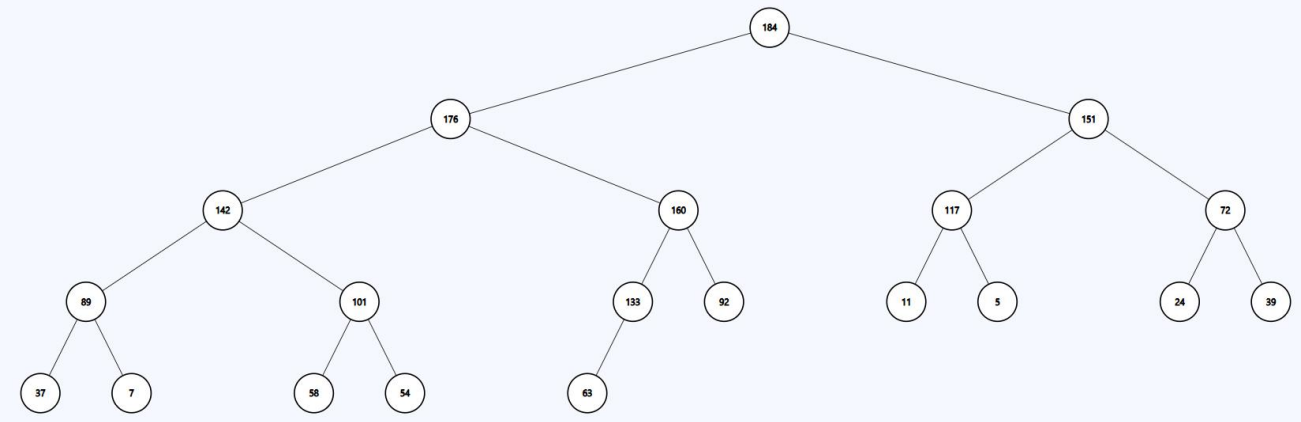
Tree Visualizer

Construction / heap-building description (e.g. heapify, insert-and-sift-up):

heap-building 步驟：

1. 先把二十筆資料寫成完全二元樹（Complete Binary Tree）：根節點位於索引位置 0(值:37)，左子節點是 2*i+1，右子節點是 2*i+2，父節點是 floor((i-1)/2)，i 為索引值。
2. 從最後一個非葉節點開始 heapify（bottom-up）：最後一個非葉節點是索引 9（值 133）
3. 依次向上處理：依序處理索引 8、7、6…0，總共有 10 個非葉節點。每一步都比較父節點與子節點；若有子節點比父節點大，就跟較大的子節點互換，然後對交換後的位置再次 heapify，直到該子樹滿足父 ≥ 子的性質

Screenshot of Max Heap (paste below):



3.7 Min Heap

Tool name / URL:

Tree Visualizer

Student ID: 1131417 Student Name:魏允鴻

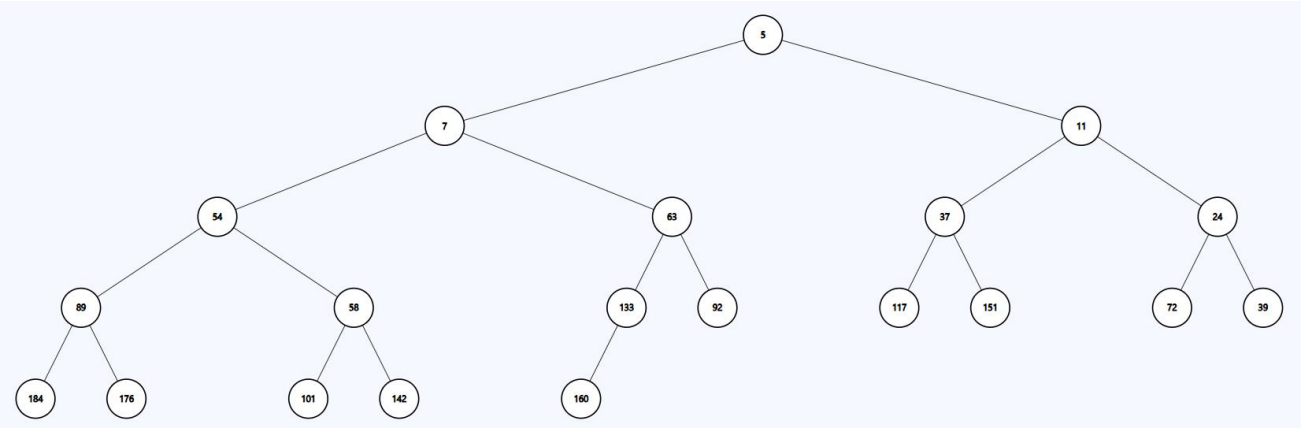
Construction / heap-building description:

heap-building 步驟：

1. 同樣先將資料放入完全二元樹（Complete Binary Tree）：根節點位於索引位置 0(值:37)，左子節點是 2*i+1，右子節點是 2*i+2，父節點是 floor((i-1)/2)，i 為索引值。

2. 從最後一個非葉節點向上做 heapify（bottom-up），依次向上處理：依序處理索引 8、7、6…0，總共有 10 個非葉節點，只是比較方向反過來：每個節點若大於任一子節點，則與較小的子節點交換，並繼續對交換後的位置做 heapify，直到該子樹滿足父 ≤ 子

Screenshot of Min Heap (paste below):



Section 4. Application Examples

Task: For each tree type, choose one application and explain why this tree is suitable.

Tree Type	Application Example (name / context)	Why this tree fits (properties that matter)
Binary Tree	Expression Tree（運算式樹）	二元樹可自然表示運算子與左右運算元的關係，結構清楚，適合表示數學或邏輯運算。
Complete Binary Tree	Binary heap storage（堆積樹儲存）	完全二元樹結構緊密，節點排列連續，適合以陣列儲存，空間利用率高。
Binary Search Tree	Dictionary / Symbol Table（字典、符號表）	BST 具有排序性質，可依鍵值快速搜尋、插入與刪除。
AVL Tree	Database indexing（資料庫索引）	AVL 樹高度嚴格平衡，能確保搜尋時間維持在 O(log n)，適合大量查詢的情境。
Red-Black Tree	Programming language library（標準函式庫 Map / Set）	紅黑樹透過顏色規則維持平衡，插入與刪除調整較少，效能穩定。
Max Heap	Priority scheduling（優先權排程）	Max Heap 能在 O(1) 時間內取得最大值，並在 O(log n) 時間內調整結構
Min Heap	Task scheduling / Dijkstra's	Min Heap 可快速取得最小值，適合用於

	algorithm（排程、最短路徑）	排程或最短路徑演算法。

Section 5. Reflection on Tree Family and Performance (Optional but recommended)

Among BST, AVL, and Red-Black trees, which one would you pick for:

Mostly search (few updates)? Why?

1. 適合使用 AVL 樹，因為 AVL 樹有嚴格的高度平衡規則：每個節點的左右子樹高度差最多為 1。這種嚴格的平衡確保樹的高度最小化，使搜尋操作通常比一般的 BST 或紅黑樹更快，能穩定地在 O(log n) 時間內完成查詢。

Frequent insertions and deletions? Why?

2. 適合使用 紅黑樹 (Red-Black Tree)，它透過節點顏色與旋轉規則維持平衡。紅黑樹的高度限制較寬鬆，因此在插入或刪除時需要的旋轉次數較少，相較於 AVL 樹更節省調整成本，能在大量更新操作下保持穩定的效能。

補充：AVL 樹規則

* 每個節點左右子樹高度差 ≤ 1。

If you must store these 20 integers for static search only (no updates), which structure or representation would you prefer (sorted array + binary search, BST, AVL, etc.)? Why?

3. 若資料在建立後不再進行插入或刪除，使用**排序陣列搭配二分搜尋**是最佳選擇。將資料由小到大排序後存放於陣列中，可利用二分搜尋在 O(log n) 時間內找到目標值。相比樹狀結構，排序陣列的實作簡單，不需要指標或旋轉操作，記憶體使用也更有效率，特別適合純搜尋、無更新的情境。

補充：紅黑樹（Red-Black Tree）規則

* 節點顏色為紅或黑，根節點為黑
* 紅節點不能連續（紅色節點不可為父子關係）
* 根到每個葉節點的黑色節點數相同
* 插入或刪除後透過旋轉與變色維持平衡。

Section 6. AI Usage Log (Required)

Task: Record every time you ask an AI assistant about this assignment.

Index	Date / Time	AI Service (ChatGPT, Gemini, etc.)	Your Full Prompt / Question
1	12/29;17:26	ChatGPT	根據我傳給你的這張手寫圖(2.1)，告訴我該如何把”new property or constraint is added at each step(藍筆部分)”寫得更完整且嚴謹
2	12/29;17:47	ChatGPT	(Section 4)”我自己透過學習資料

			結構後所學到的知識，整理了一段筆記，內容是針對不同種類的樹狀資料結構，各選擇一個常見的應用情境，並說明為什麼該樹結構適合用在該用途上，請你根據我的筆記內容，幫我修改成更嚴謹的文字敘述。” 筆記內容：Binary Tree 能清楚表示運算子與左右運算元的關係，因此常用於運算式樹。Complete Binary Tree 節點排列緊密，適合用陣列儲存，常作為堆積樹基礎。Binary Search Tree 有排序性質，搜尋、插入、刪除都快，常用於字典或符號表。AVL Tree 與 Red-Black Tree 屬自我平衡樹，避免退化，搜尋效率維持在對數時間，常用於資料庫索引或標準函式庫。Max Heap 與 Min Heap 可快速找最大或最小值，適合優先權排程或最短路徑演算法。
3	12/29;18:20	ChatGPT	“以下是我自己透過學習資料結構後的所學到的知識，來回答這三題，想請你幫我把我的文筆修正的更正確且嚴謹” (Section5)Among BST, AVL, and Red-Black trees, which one would you pick for: Mostly search (few updates)? Why? 適合使用 AVL 樹，因為其擁有嚴格的高度限制規則，左右子樹高度差需小於 1，搜尋效率通常優於一般 BST 與紅黑樹。 Frequent insertions and deletions? Why? 適合使用紅黑樹(Red-Black Tree)，其透過顏色規則與較少的旋轉次

			數來維持整體平衡，因為旋轉次數較少，所以在大量插入與刪除的情況下的效能更穩定。 補充：AVL 樹規則 * 每個節點左右子樹高度差 ≤ 1。 If you must store these 20 integers for static search only (no updates), which structure or representation would you prefer (sorted array + binary search, BST, AVL, etc.)? Why? 若資料不再新增或刪除，無需自我平衡樹。將資料排序存陣列，使用二分搜尋可在 O(log n) 找到目標。相比 tree，排序陣列實作簡單、記憶體效率高。 補充：紅黑樹(Red-Black Tree)規則 * 節點顏色為紅或黑，根節點為黑 * 紅節點不能連續 * 所有葉節點到根的黑色節點數相同 * 插入或刪除後透過旋轉與變色維持平衡。
4	12/30;12:21	Genimi	(Section3) Binary Tree 沒有規定由左至右插入，所以有可能會退化成 Linkeded list 嗎？
5	12/30;12:58	Genimi	(Section3) 我沒記錯的話，遞迴取中間值方法可以保證左右子樹高度差小於 1，以達到 AVL 的條件對嗎？
6	12/30;13:40	Genimi	(Section3) 我沒記錯的話，heapify 除了可以同時用在 Max heap 跟 Min heap 之外，也是最有效率的方法，對吧？
7	12/30;14:35	ChatGPT	(Section3) 以下 5 點是我在學習完

			程式設計和資料結構後，對於紅黑樹的理解所寫出的筆記，請幫我優化我的筆記： (1)每個新節點先當作一般 BST 插入。 (2)新插入的節點一律設為「紅色」（根節點例外，最後一定是黑色） (3)若父節點是黑色 → 不需調整，紅黑樹性質未被破壞，結束此次插入。 (4)若父節點是紅色 → 必須修正（違反紅紅規則） 分兩種情況處理： (4-1) 叔叔節點是紅色 父、叔 → 染黑 祖父 → 染紅 把「祖父」當成新插入節點，繼續往上檢查 (4-2) 叔叔節點是黑色或不存在 依節點位置做旋轉 LL / RR → 單旋 LR / RL → 先轉一次再單旋 旋轉後： 新父節點染黑 原祖父染紅

			(5)最後確保根節點為黑色

You may extend this table as needed.