

Course: Data Structures (CSE CS203A)

Assignment V: Tree

Due date: 2025.12.30 23:59:59

Important Notice – Use of AI Tools

In this assignment, you must use at least one AI assistant (e.g. ChatGPT, Gemini, Claude, Grok, M365 Copilot) as a learning tool to help you:

- review definitions,
- compare tree variants, and
- organize your report.

You are not allowed to let the AI directly produce your final diagrams or final report content without your own understanding and rewriting.

You must log all AI prompts and services used (see “AI Usage Log” section below).

1. Goal of This Assignment

In the lectures, we introduced the concept of the tree as a data structure, starting from the general tree and then moving to more specialized forms.

In this assignment, you will:

- a. Understand and clearly define:
 1. General tree
 2. Binary tree
 3. Complete binary tree
 4. Binary search tree (BST)
 5. AVL tree
 6. Red-Black tree
 7. Max heap
 8. Min heap
- b. Build a hierarchy and transformation path from the general tree to these variants, and explain how each variant adds more structure or constraints.
- c. Use a fixed list of integers to construct multiple tree variants and visualize them.
- d. Choose one real-world application for each tree type and explain why that data structure fits the application.
- e. Practice using AI tools as study companions and keep a simple Q&A log.

2. Given Data

Use the following 20 integers as the input data for all your tree constructions:

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

You will reuse this same sequence for every tree type (binary tree, complete binary tree, BST, AVL, Red-Black, max heap, min heap).

3. Deliverables

Please complete your work in the Student Worksheet Companion and upload it to the YZU Portal System.

Your report should include the following parts:

a. Definitions (Concept Review)

Provide clear, concise definitions for each of the following:

1. General tree
2. Binary tree
3. Complete binary tree
4. Binary search tree (BST)
5. AVL tree
6. Red-Black tree
7. Max heap
8. Min heap

You are encouraged to use AI tools to help you understand these concepts, but you must rewrite the definitions in your own words.

b. Hierarchy and Transformation of Tree Variants

Based on the definitions above, build a “tree family hierarchy” that shows how these structures are related. For example:

- General tree → Binary tree
- Binary tree → Complete binary tree / Binary search tree
- BST → AVL tree / Red-Black tree
- Binary tree → Max heap / Min heap

Tasks:

- Draw a diagram or flow chart that shows the transformation or specialization path:
general tree → binary tree → complete binary tree → BST → AVL/Red-Black, etc.
- For each arrow (transformation), briefly explain:
 - What new constraint or property is added?
 - e.g., “Binary tree = tree with at most 2 children per node”,
“BST = binary tree with left < root < right”,
“AVL = BST with strict height-balance rule”, etc.

c. Tree Construction with the Given Integers

Using the given 20 integers, construct the following tree variants:

1. Binary tree
2. Complete binary tree
3. Binary search tree (BST)

4. AVL tree
5. Red-Black tree
6. Max heap
7. Min heap

Important Hint / Restriction:

- For these trees, you must use tree visualization tools (e.g., online visualizers or software) to build and display the tree.
- You may not ask AI tools to directly generate the final tree pictures for you.
- Instead:
 - Use AI only to help you understand algorithms,
 - Then apply those algorithms in a visualizer (or your own implementation).

What to submit for this part:

For each tree type:

- A snapshot (image) of the constructed tree.
- The URL / name of the visualization tool you used.
- A short note on how you inserted the integers (e.g., “insert in the given order as BST”, “build max heap using heapify”, etc.).

d. Application Example for Each Tree

For each of the following:

1. Binary tree
2. Complete binary tree
3. Binary search tree (BST)
4. AVL tree
5. Red-Black tree
6. Max heap
7. Min heap

Choose one application (real-world or system-level) and explain:

1. Application description
 - e.g., priority scheduling, dictionary lookup, memory allocation, database indexing, etc.
2. Why this tree structure fits
 - What property of this data structure makes it suitable?
 - Example:
 - Max heap → good for priority queue because the largest element is always at the root, so extracting max is efficient.
 - Red-Black tree → good for standard library maps/sets because it guarantees $O(\log n)$ operations even under many insertions/deletions.

Your explanation should show that you understand the link between the data structure and its use case.

e. Report Layout and Organization

You are free to design the layout of your report, but it should:

- Be well-structured (use sections, headings, tables, and diagrams).
- Have a clear flow from:
 - definitions →
 - hierarchy/transformation →
 - constructed trees →
 - applications →
 - AI usage log.
- Be easy for another student to read and learn from.

Feel free to use AI to suggest a good outline, but you must decide and finalize the layout yourself.

f. AI Usage Log (Q&A Table)

Every time you use an AI copilot service for this assignment, record:

- Index (1, 2, 3, ...)
- Prompt (what you asked)
- Service (e.g., ChatGPT, Gemini, Copilot, ...)

Example log table:

Index	Prompt	Service
1	Assist me to have the definition of general tree, binary tree, complete binary tree, binary search tree, AVL tree, red-black tree, max heap and min heap for self-learning.	ChatGPT
2	Explain the difference between AVL tree and Red-Black tree in terms of balancing strategy and use cases.	Gemini
...

Place this table at the end of your report.

4. Evaluation (100 pts)

A possible breakdown (you can adjust if needed):

- a. Concept definitions (20 pts)
 - Correctness and clarity of all 8 tree type definitions.
- b. Hierarchy & transformation explanation (20 pts)
 - Clear diagram / explanation of how each tree variant evolves from the general tree.
 - Correct identification of constraints/invariants.
- c. Tree constructions & visualizations (25 pts)
 - Correct constructions for each tree type using the given integers.
 - Proper screenshots and tool URLs.

- Consistent insertion / heap-building strategy descriptions.
- d. Applications & explanations (20 pts)
 - One application per tree type.
 - Clear explanation linking data structure properties to the application.
- e. Report organization & AI usage log (15 pts)
 - Logical report structure and readability.
 - AI log completeness (all prompts listed with service names).
 - Thoughtful use of AI as a learning assistant, not as a copy-paste generator.

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: 一種最基礎的非線性階層結構，由一組有限的節點組成（至少包含一個根節點）。在此結構中，每個節點可以延伸出任意數量的子節點，且子節點之間沒有強制的排序規則。

2. Binary Tree

Definition: 一種受限的樹狀結構，每個節點最多只能有兩個子節點（0,1 或 2 個）。子節點被嚴格區分為「左子節點」和「右子節點」。

3. Complete Binary Tree

Definition: 一棵高度為 h 的 Binary tree，除第 h 層外，其他各層節點數都達到最大值，且第 h 層的節點都連續集中在最左邊。

4. Binary Search Tree (BST)

Definition: 一棵 Binary tree，對任意節點皆滿足：左子樹所有節點的值 $<$ 根節點的值，右子樹所有節點的值 $>$ 根節點的值。

5. AVL Tree

Definition: 一種自我平衡的 Binary search tree (BST)，對任一節點的左子樹與右子樹的高度差，其絕對值不能超過 1（即只能是 -1, 0, 1）。

6. Red-Black Tree

Definition: 一種自我平衡的 Binary search tree (BST)，節點帶有顏色（紅或黑），需滿足：根是黑色、不能有兩個連續的紅色節點、從任一節點到其所有後代 Null 節點的黑色節點數量必須相同。

7. Max Heap

Definition: 一棵 Complete Binary Tree，滿足 Heap Property：父節點的值 \geq 子節點的值，因此根節點永遠是最大的。

8. Min Heap

Definition: 一棵 Complete Binary Tree，滿足 Heap Property：父節點的值 \leq 子節點的值，因此根節點永遠是最小的。

Section 2. Tree Family Hierarchy and Transformations

Task: Show how these structures are related (general \rightarrow 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.

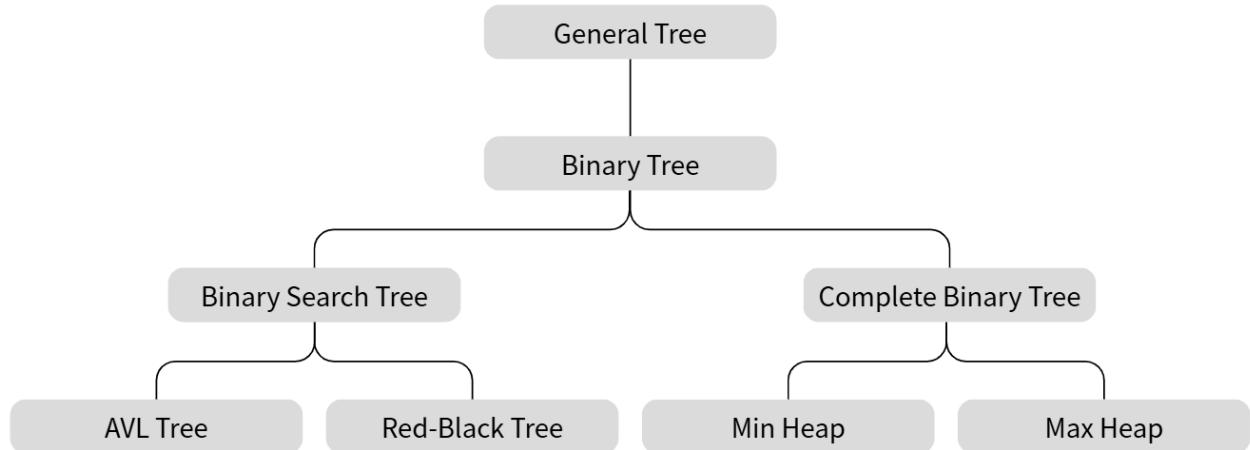
Suggested chain example (you may extend or adjust):

General Tree \rightarrow Binary Tree \rightarrow Complete Binary Tree

Binary Tree \rightarrow Binary Search Tree \rightarrow 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	限制子節點數量：每個節點最多只能有 2 個子節點。
Binary Tree	Complete Binary Tree	結構形狀限制：樹必須從上到下、從左到右填滿。除了最後一層外，每一層都必須是滿的，且最後一層節點靠左排列。
Complete Binary Tree	Max Heap	父子節點大小限制：父節點的值必須大於或等於子節點。
Complete Binary Tree	Min Heap	父子節點大小限制：父節點的值必須小於或等於子節點。
Binary Tree	Binary Search Tree	數值順序限制：對於任一節點，左子樹的所有值都小於根節點，右子樹的所有值都大於根節點。
Binary Search Tree	AVL Tree	嚴格高度平衡：增加平衡因子限制。任一節點的左右子樹高度差絕對值不能超過 1(即-1,0,1)。
Binary Search Tree	Red-Black Tree	著色與規則限制：節點分為紅或黑，透過顏色規則（根是黑、無連續紅節點、路徑黑節點數相同）來維持大致平衡。

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:

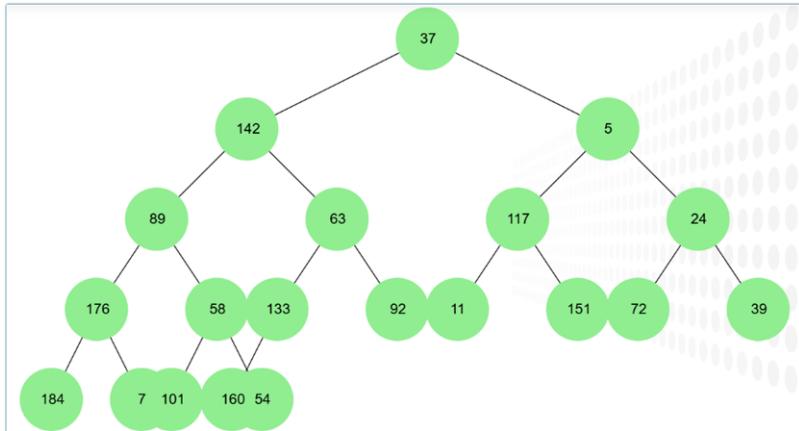
Tree Visualizer

<https://trees-visualizer.netlify.app/trees>

Construction / insertion description:

我使用模擬 **Binary Tree** 的線上工具依序插入給定的 20 個整數。該工具將這些數字建立為一般的二元樹結構，每個節點最多擁有兩個子節點，符合二元樹的基本定義。

Screenshot of Binary Tree (paste below):



3.2 Complete Binary Tree

Tool name / URL:

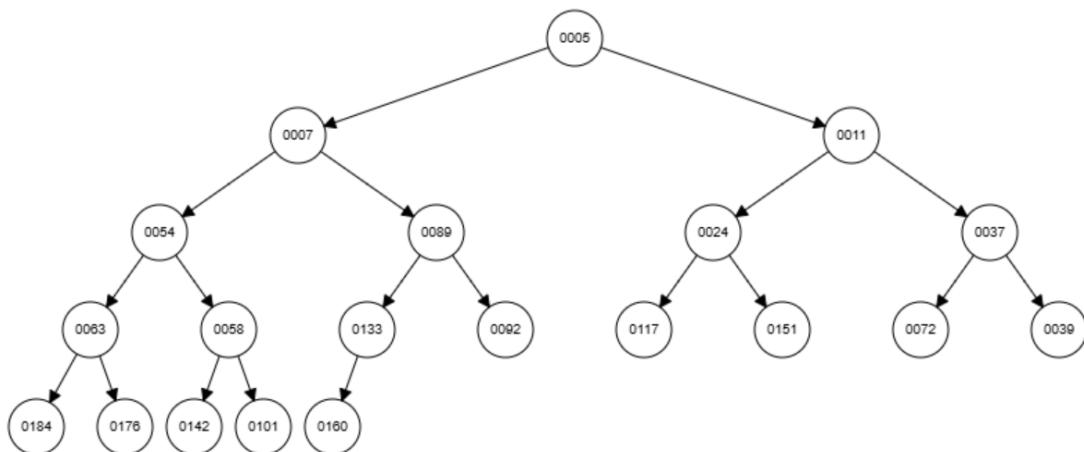
USF CAES Data Structure Visualizations

<https://www.cs.usfca.edu/~galles/visualization/Heap.html>

Construction / insertion description:

為了展示完全二元樹，我使用了 Min Heap 工具，因為 Heap 定義上必須是完全二元樹，工具自動將樹由左至右層層填滿，可以展示了其結構特性。

Screenshot of Complete Binary Tree (paste below):



3.3 Binary Search Tree (BST)

Tool name / URL:

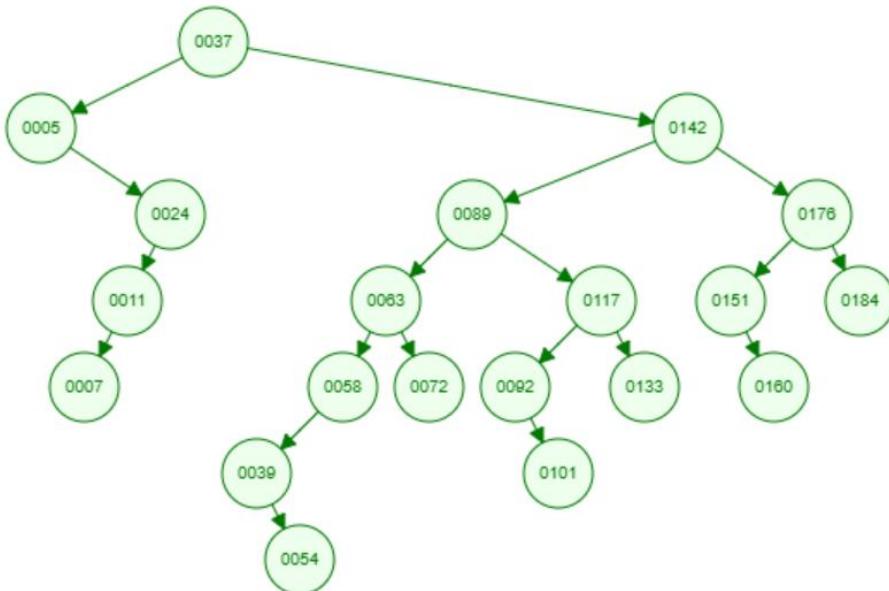
USF CAES Data Structure Visualizations

<https://www.cs.usfca.edu/~galles/visualization/BST.html>

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

我依照順序插入數據，工具每次都會根據 BST 屬性放置數值：小的在左，大的在右，進形排列，沒有使用平衡演算法，樹的形狀完全取決於輸入順序。

Screenshot of BST (paste below):



3.4 AVL Tree

Tool name / URL:

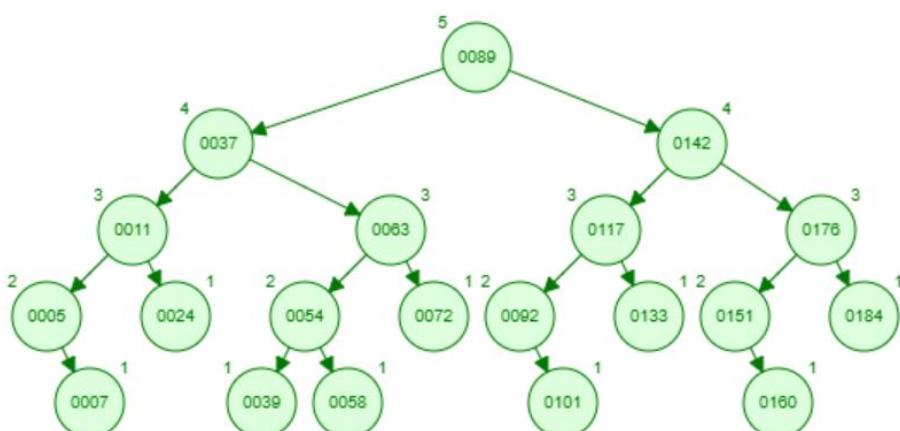
USF CAES Data Structure Visualizations

<https://www.cs.usfca.edu/~galles/visualization/AVLtree.html>

Insertion & balancing description:

我依序插入數據，作為 AVL 樹，工具在每次插入後都會檢查平衡因子，一旦超出範圍，工具會自動執行旋轉以維持平衡。

Screenshot of AVL Tree (paste below):



3.5 Red-Black Tree

Tool name / URL:

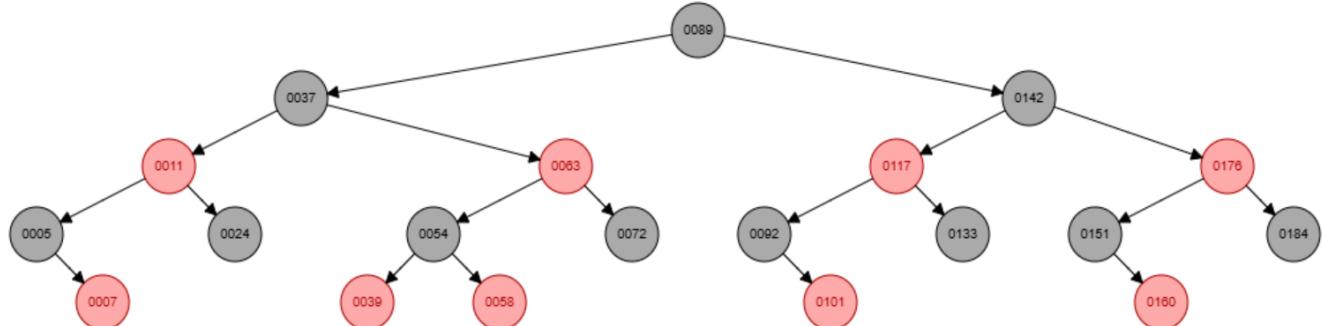
USF CAES Data Structure Visualizations

<https://www.cs.usfca.edu/~galles/visualization/RedBlack.html>

Insertion & balancing description:

我將序列插入紅黑樹視覺化工具。工具為節點分配顏色並強制執行紅黑樹規則，並在必要時自動變色或旋轉，維持平衡。

Screenshot of Red-Black Tree (paste below):



3.6 Max Heap

Tool name / URL:

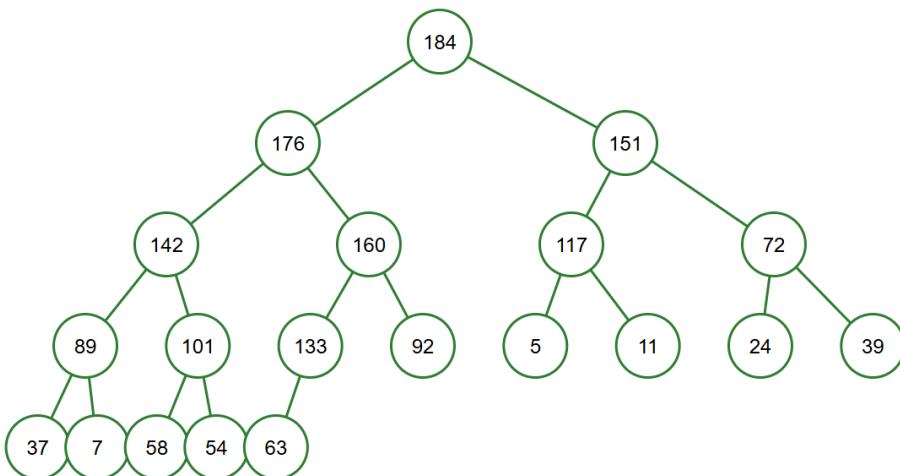
Max Heap Simulator

https://sercankulcu.github.io/files/data_structures/slides/Bolum_08_Heap.html

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

我將整數逐一插入 Max Heap 視覺化工具，每次插入後工具會先以完全二元樹的規則為主將新元素排列在最後一層且由左至右緊密排列，再確保父節點大於等於子節點進行向上交換 (sift up) 直到正確位置。

Screenshot of Max Heap (paste below):



3.7 Min Heap

Tool name / URL:

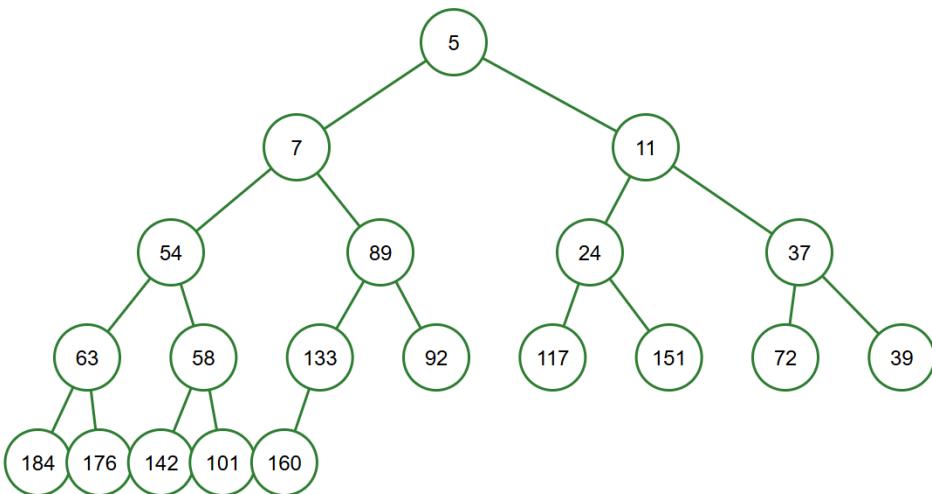
Min Heap Simulator

https://sercankulcu.github.io/files/data_structures/slides/Bolum_08_Min_Heap.html

Construction / heap-building description:

我將整數逐一插入 Min Heap 視覺化工具，每次插入後工具會先以完全二元樹的規則為主將新元素排列在最後一層且由左至右緊密排列，再確保父節點小於等於子節點的屬性，插入時工具會比較並交換數值，確保最小的元素(5)最終位於根部。

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	哈夫曼編碼檔案壓縮 (Huffman Coding)	ZIP 壓縮或 JPEG 圖檔背後的關鍵技術。將常用的字元用較短的二進位碼表示，不常用的字元用較長的碼，而葉子節點儲存資料，路徑代表編碼。為了讓電腦能快速解碼且不產生歧義，二元樹的左分支代表 0，右分支代表 1。這種結構確保了解碼速度，電腦只需沿著 bit 流一直走，直到撞到葉子節點就能印出字元，不需要回頭掃描，達到極高的解壓縮吞吐量。

Complete Binary Tree	固定長度通訊協議中的「區塊定址」	在某些嵌入式系統或硬體底層，若需要儲存一堆感應器資料，但記憶體空間非常有限，甚至不允許使用指標時，因為完全二元樹結構緊湊、沒有空隙，可以映射到連續陣列來儲存。這消除了儲存「左/右子節點記憶體地址」所需的額外開銷，對於空間極端受限的系統，這種「隱含式結構」能將記憶體利用率提升到近乎 100%，且計算父子關係只需位移運算，處理速度極快。
Binary Search Tree	動態字典或符號表 (Dynamic Dictionary / Symbol Table)	適用於資料隨機性較高、且不常遇到極端排序資料的場景，透過數值大小的排序特性 (左<根<右)，BST 能夠在平均 $O(\log n)$ 的時間內完成搜尋、插入和刪除操作，比使用陣列或鏈結串列更有效率。
AVL Tree	讀取頻繁的資料庫索引 (Read-heavy Database Indexing)	AVL 樹維持「嚴格」的高度平衡，雖然插入時旋轉次數較多，但它保證了樹的高度最低。這對於「查詢多、修改少」的系統非常有利，能提供最快的搜尋速度。
Red-Black Tree	C++ STL 的 std::map	紅黑樹在平衡規則上比 AVL 稍微寬鬆，因此在插入和刪除時所需的旋轉次數較少。這使它在搜尋、插入、刪除的綜合效能上表現更平均，適合做為通用的標準函式庫實作。
Max Heap	作業系統的行程排程 (Process Scheduling in OS)	在優先權排程中，我們需要隨時取出「優先權最高」的任務。Max Heap 的根節點永遠是最大值，因此能在 $O(1)$ 時間內確認最高優先權任務，並以 $O(\log n)$ 效率移除它。
Min Heap	Dijkstra 最短路徑演算法 (Dijkstra's Shortest Path Algorithm)	此演算法需要不斷從未訪問的節點中找出「距離最小」的一個。Min Heap 的根節點永遠是最小值，能讓我們最有效地取得當前最短路徑的節點來進行路徑擴展。

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) AVL Tree。

(2) 因為 AVL 樹採用「嚴格」的高度平衡策略：高度差不超過 1，這保證了樹的高度維持在最小可能範圍。相比於紅黑樹，AVL 樹的平均查詢路徑更短，因此在「讀多寫少」的情境下，搜尋效率是最高的。

Frequent insertions and deletions? Why?

Red-Black Tree (紅黑樹)。

因為紅黑樹的平衡條件比 AVL 樹寬鬆，只要求黑高度一致。這意味著在進行插入或刪除節點時，紅黑樹觸發「旋轉 (Rotation)」來重新平衡的頻率比 AVL 低很多。對於需要頻繁變動資料的場景，紅黑樹能提供更佳的綜合寫入效能。

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?

(1) Sorted Array (排序陣列) 搭配 Binary Search (二分搜尋法)。

(2) 既然資料是靜態的，不會發生新增或刪除的事件，我們就不需要承擔樹狀結構的指標 (Pointers) 所帶來的額外記憶體開銷。使用陣列不僅節省空間，而且記憶體配置是連續的，具有極佳的 Cache Locality (快取局部性)，因此實際執行速度通常會比遍歷樹狀結構更快。

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	12/18	Gemini	請協助我翻譯作業說明，並歸納出本次作業內容以及評分標準。
2	12/19	Gemini	針對 General tree, Binary tree, BST, AVL, Red-Black tree, Max/Min heap 這八種樹，請以初學者能理解的方式，詳細比較它們的差異與核心概念。請先用淺顯易懂的比喻解釋，再協助我修正出符合學術規範的準確定義。
3	12/19	Gemini	我撰寫了這些樹的定義，請幫我檢查目前的定義是否準確且精簡？是否有遺漏關鍵資訊？如果有，請給予我修正建議。

4	12/19	Gemini	請解釋從 General Tree 演變到 AVL、Heap 等特殊樹種的「演化路徑」與層級關係。我想了解每一步驟具體增加了哪些限制才導致了結構的變化，以便我繪製層級圖。
5	12/20	Gemini	請針對 Binary Tree, Complete Binary Tree, BST, AVL, Red-Black Tree, Max/Min Heap，分別舉出一些具代表性的現實世界或系統層級應用範例。對於每個範例，請深入解析該樹狀結構的特定屬性是如何滿足該應用的效能需求，讓我能清楚理解資料結構特性與實際應用之間的關聯性。
6	12/20	Gemini	我撰寫了這些樹的應用範例，請幫我確認我的內容是否邏輯通順且正確？若有觀念上的誤解請指正。
7	12/20	Gemini	針對報告的反思題，請協助我比較 AVL 與紅黑樹在「搜尋優先」與「頻繁寫入」場景下的效能差異，並分析若資料是靜態的，為何使用陣列結構可能優於樹狀結構。
8	12/20	Gemini	針對反思題 (Reflection)，我已嘗試回答關於 AVL 與紅黑樹的效能取捨。請幫我檢查我的論點是否準確，特別是在比較「嚴格平衡」與「寬鬆平衡」對搜尋/插入效能影響的解釋上，是否夠清晰且符合理論？
9	12/20	Gemini	請幫我比對作業說明的要求，確認整份報告是否有遺漏任何關鍵項目，且內容是否符合作業的繳交標準？