Student ID: Student Name:

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:最基本的 tree，沒有任何限制，每個節點可以有任意的子節點。

Student ID: Student Name:

2. Binary Tree
Definition:在 General Tree 上加上限制，每個節點[最多]只能有兩個子節點。演算法中最常使用的基礎。

3. Complete Binary Tree
Definition: Binary Tree 再更嚴格。除了最後一層外，每一層都被填滿，而最後一層必須靠左。因為都填滿，適合用 array 去做，不會浪費空間。

4. Binary Search Tree (BST)
Definition: Binary Tree 的延伸，有排序的規則，優化搜尋時的效率。左子樹的所有值 < 根節點的值，右子樹相反。

5. AVL Tree
Definition: 嚴格平衡的 BST。要確保左子樹和右子樹不會相差超過 1。避免像 1234 等這種已經排序的資料，導致樹變成直直一條線。透過旋轉調整。

6. Red-Black Tree
Definition:比 AVL 寬鬆一些。將節點塗紅色和黑色:紅節點的子節點是黑色，從根到葉子的黑色節點數量要相同。比 AVL 效率高，因為旋轉次數比較少。

7. Max Heap
Definition: Complete Binary Tree 再嚴格，確保父節點的值>=子節點。左右節點沒有大小關係，只有上下有。用於找最大值

8. Min Heap
Definition:和 Max Heap 相反，父節點的值<=子節點。常用於優先權佇列。

**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):

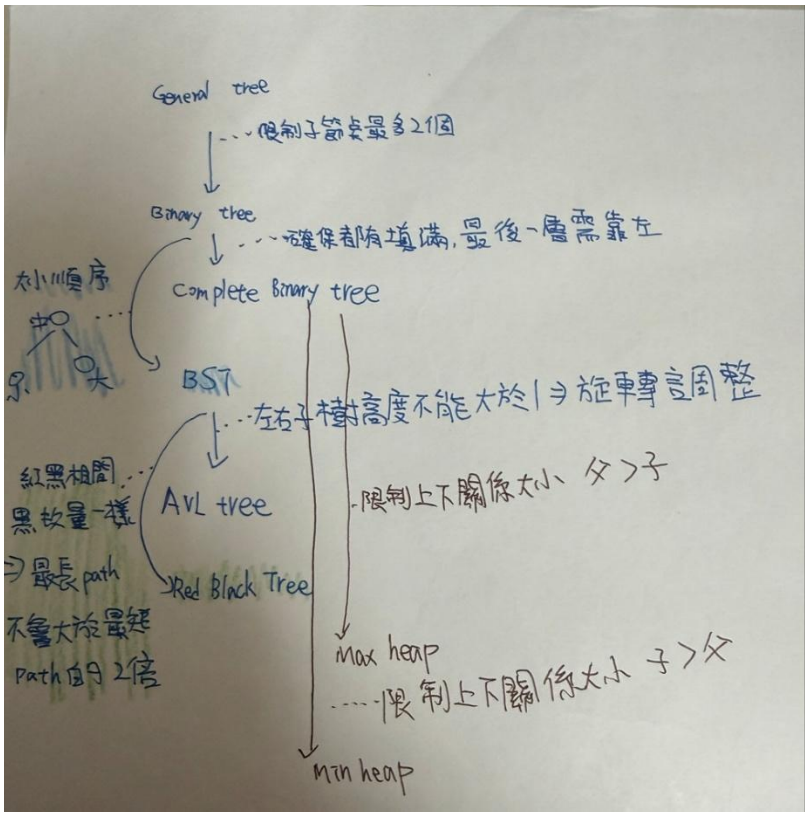
Student ID: Student Name:

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 | 限制最多兩個子節點 |
| Binary Tree | Complete Binary Tree | 每一層都要填滿，而且最後一層需要靠左排列 |
| Binary Tree | Binary Search Tree | 按照大小順序:左節點>根>右節點 |
| BST | AVL Tree | 確保左右高度相差不會大於 1，大於 1 的話，作轉處理 |
| BST | Red-Black Tree | 紅黑相間，因為黑色數目要一樣，可以確保路徑差距不超過一倍 |
| Binary Tree | Max Heap | 限制上下關係:父>子 |
| Binary Tree | Min Heap | 限制上下關係:父<子 |

**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

Student ID: Student Name:

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

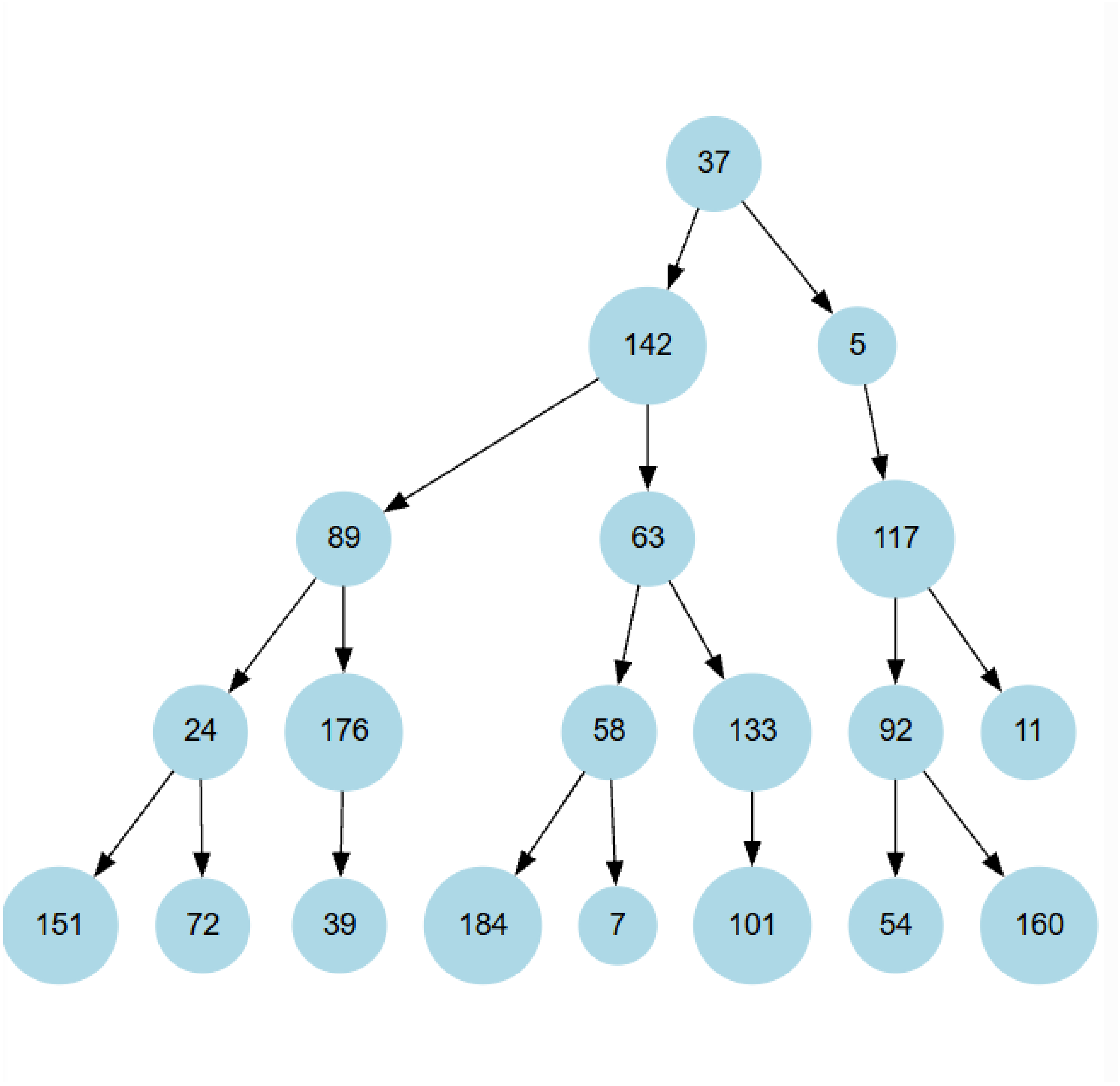
Tool name / URL: Graphviz Online

URL

Construction / insertion description:

依照給定順序插入，有空位就放，維持每個節點最多兩個子節點的規則。

Screenshot of Binary Tree (paste below):

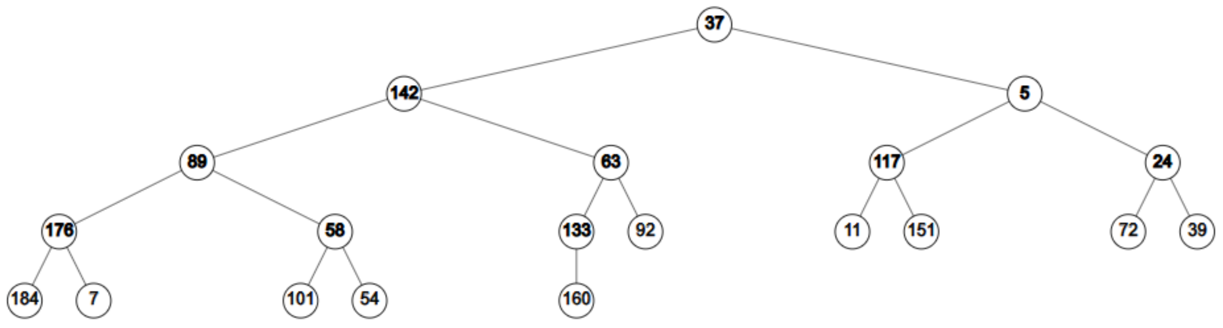


Student ID: Student Name:

3.2 Complete Binary Tree
Tool name / URL:
treeconverter,https://treeconverter.com/

Construction / insertion description:
嚴格按照從上到下、由左至右的順序填滿每一層，確保中間沒有空缺。

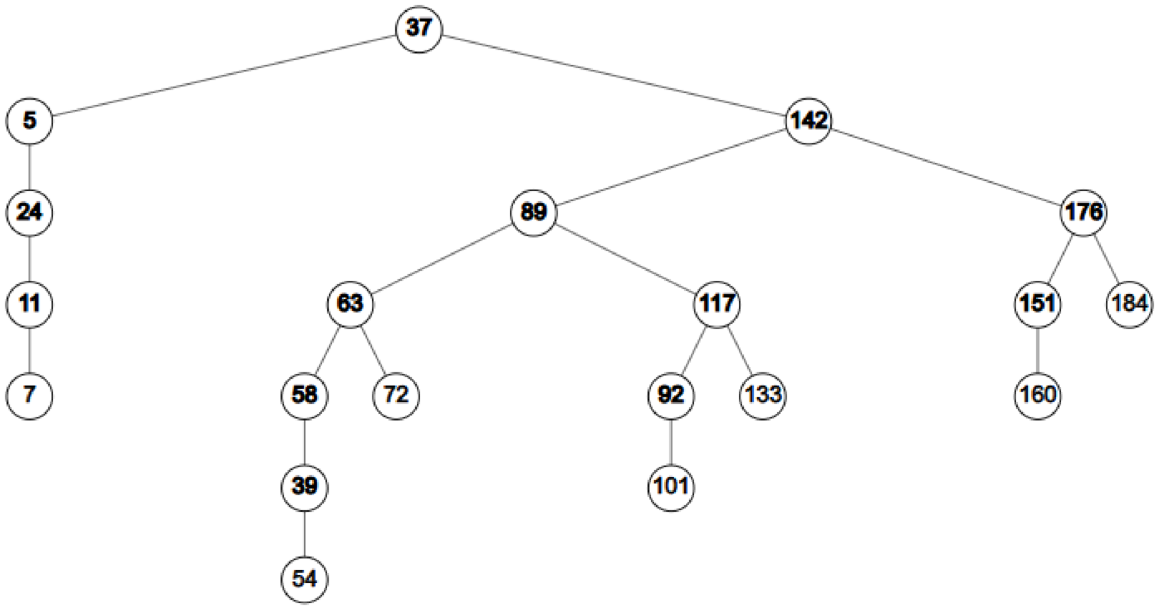
Screenshot of Complete Binary Tree (paste below):



3.3 Binary Search Tree (BST)
Tool name / URL:
: treeconverter,https://treeconverter.com/

Insertion rule (e.g., “insert in given order using BST rules”):
依照順序一個個插入。遵守 BST 規則（小放左、大放右）。
Screenshot of BST (paste below):

Student ID: Student Name:



3.4 AVL Tree

Tool name / URL:

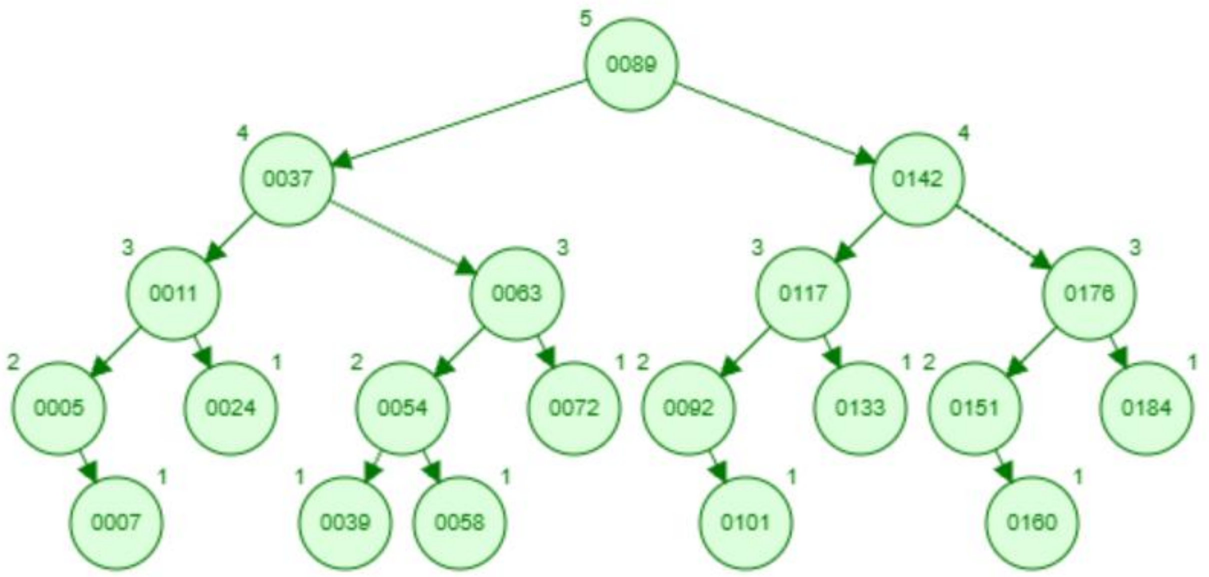
USF Data Structure Visualizations

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

Insertion & balancing description:

按照順序插入，每次插入後檢查有沒有平衡，無則作旋轉

Screenshot of AVL Tree (paste below):



Student ID: Student Name:

3.5 Red-Black Tree

Tool name / URL:

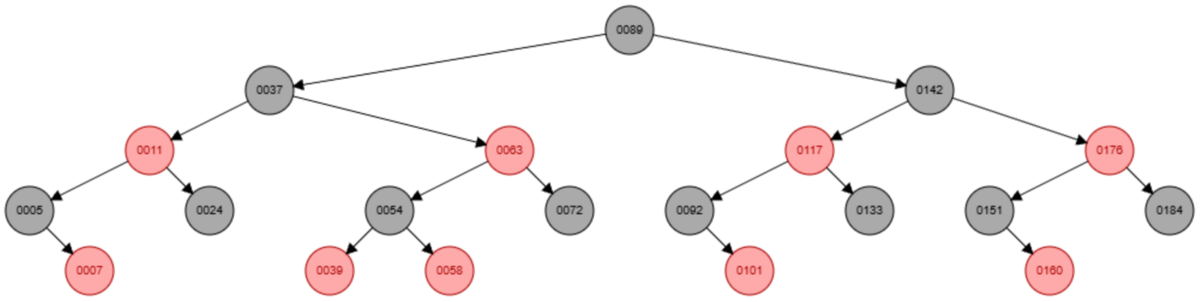
USF Data Structure Visualizations

https://www.cs.usfca.edu/~galles/visualization/RedBlackTree.html

Insertion & balancing description:

依照順序插入，新節點預設為紅色。隨後透過變色或旋轉來修正。

Screenshot of Red-Black Tree (paste below):



3.6 Max Heap

Tool name / URL:

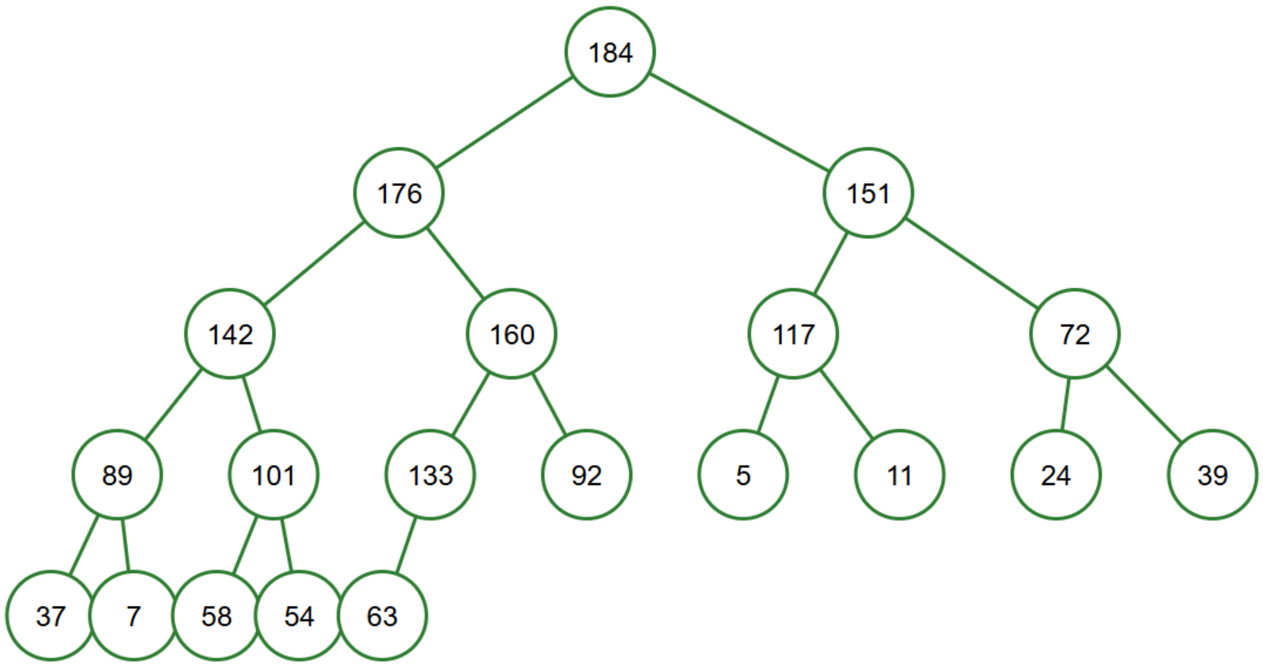
Max Heap Simulator

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

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

Screenshot of Max Heap (paste below):

Student ID: Student Name:



3.7 Min Heap

Tool name / URL:

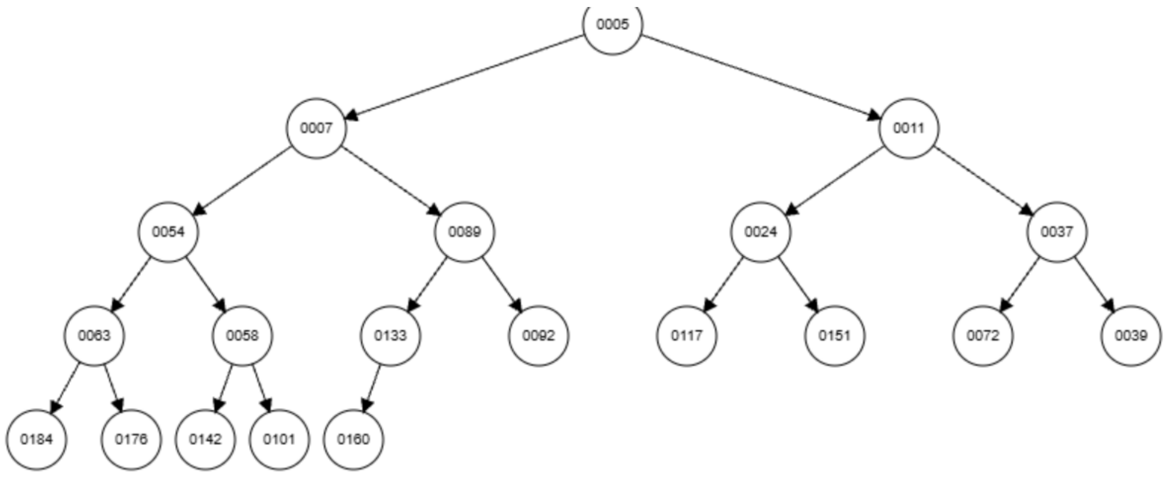
USF Data Structure Visualizations

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

Construction / heap-building description:

一個個插入元素。每插入一個，就執行 Sift-Up，確保父節點永遠小於或等於子節點。

Screenshot of Min Heap (paste below):



**Section 4. Application Examples**

Student ID:                                                  Student Name:

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 | Yes no 決策 | 每個問題只有兩個答案，剛好對應左右分支。 |
| Complete Binary Tree | 淘汰賽賽程表(倒過來的樹) | 因為中間沒有空缺，非常適合用來儲存這種一層一層填滿的結構，或者用來 heap |
| Binary Search Tree | 猜終極密碼 | BST 的結構為了砍半搜尋設計。猜 50 說太大，就往左找。 |
| AVL Tree | 字典 | 字典通常不會需要再去改動，但很需要快速搜尋。 |
| Red-Black Tree | 遊戲即時排行榜 | 和 AVL 比，更需要去常常變動 |
| Max Heap | 急診室檢傷分類 | 永遠把最大的放在最上面，醫生只要抓走最上就可以馬上處理最危急的病人 |
| Min Heap | 導航系統 | 和 max heap 相反，最小的永遠在最上面，適合隨時決定最小路徑。 |

**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?

AVL tree。在實作中有寫到，他適合多搜尋少更新的狀態。因為嚴格平衡的關係，所以可以把樹壓到最矮，相對降低了許多搜尋時的成本。因為少更新的關係，不用擔心為了平衡不斷旋轉所造成的成本。

Frequent insertions and deletions? Why?

Red-Black Tree。紅黑樹相較沒有那麼嚴格，可以讓兩條路徑的高度差距到將近兩倍。所以成本會比 AVL 來的低很多。

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?

Sorted Array + Binary Search。為了省空間和保持查找資料的快速性，既然資料是固定的，那就不需要樹的 pointer，使用 array 儲存就夠了。雖然理論上 BST 和 Binary Search 的讀取速度一樣，但因為 array 是連續的，讀取時會比 tree 的分散節點快速。

Student ID: Student Name:

**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 | 12/10 22:20 | gemini | 解釋這些樹的定義與差異 |
| 2 | 12/10 23:20 | gemini | 提供我適合的工具去建立樹 |
| 3 | 12/11 00:25 | gemini | 這些樹有什麼適合的應用場景 |
| 4 | 12/11 01:00 | gemini | 檢查有無漏寫的題目 |

You may extend this table as needed.