Student ID: 1131528 Student Name: 戴光彥

Course: Data Structures (CSE CS203A)

Assignment V: Tree

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: 一個節點可以有多個子節點且沒有數量限制。

2. Binary Tree
   Definition: 一個節點至多只能有兩個的子節點。

3. Complete Binary Tree
   Definition: 是一種 Binary Tree，但除最後一層外，每個節點都必須被填滿，最後一層也需要由左至右填入，中間不得有空缺。

4. Binary Search Tree (BST)
   Definition: 是一種 Binary Tree，但左子節點必須小於父節點、右子節點必須大於父節點。

5. AVL Tree
   Definition: 是一種自平衡 Binary Search Tree，任何節點的左子樹高度與右子節點樹之差的絕對值小於等於 1。

6. Red-Black Tree
   Definition: 是一種自平衡 Binary Search Tree，透過將節點塗成「紅色」或「黑色」並遵守「根是黑色」、「紅節點的子節點必須是黑色」（不能有連續紅節點）等規則。

7. Max Heap
   Definition: 必須是 Complete Binary Tree，其父節點的值大於等於子節點的值，且根節點為最大值。

8. Min Heap
   Definition: 必須是 Complete Binary Tree，其父節點的值小於等於子節點的值，且根節點為最小值。

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.

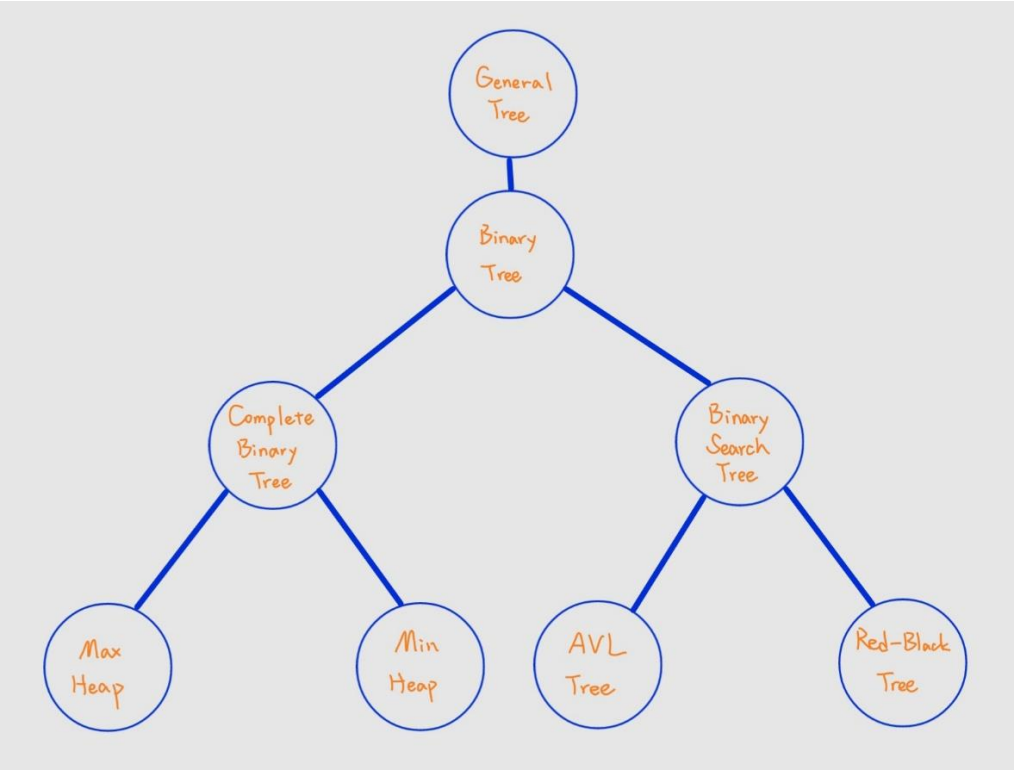
Suggested chain example (you may extend or adjust):

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	一個節點至多只能有兩個的子節點
Binary Tree	Complete Binary Tree	除最後一層外全滿，且最後一層由左至右填入。
Binary Tree	Binary Search Tree	左子節點必須小於父節點、右子節點反之。
BST	AVL Tree	任一節點的左右子樹高度差的絕對值<=1。
BST	Red-Black Tree	節點分為紅黑兩色，且滿足特定著色規則，如紅不相連。

Complete Binary Tree	Max Heap	父節點的值必須大於等於子節點。
Complete Binary Tree	Min Heap	父節點的值必須小於等於子節點。

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

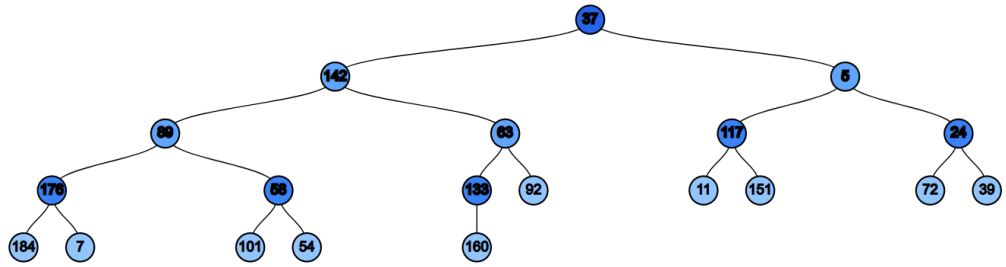
Tool name / URL:

https://treeconverter.com/

Construction / insertion description:

將數字逐一輸入，再把數字依序由上而下、由左而右填滿。

Screenshot of Binary Tree (paste below):



3.2 Complete Binary Tree

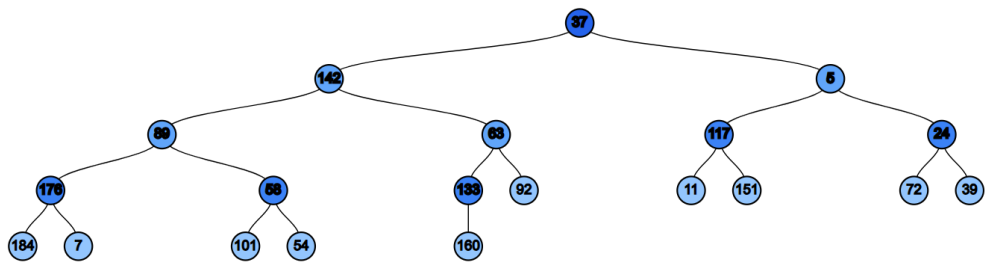
Tool name / URL:

https://treeconverter.com/

Construction / insertion description:

將數字逐一輸入，再把數字依序由上而下、由左而右填滿，並滿足一個節點最多兩的子節點的條件。

Screenshot of Complete Binary Tree (paste below):



3.3 Binary Search Tree (BST)

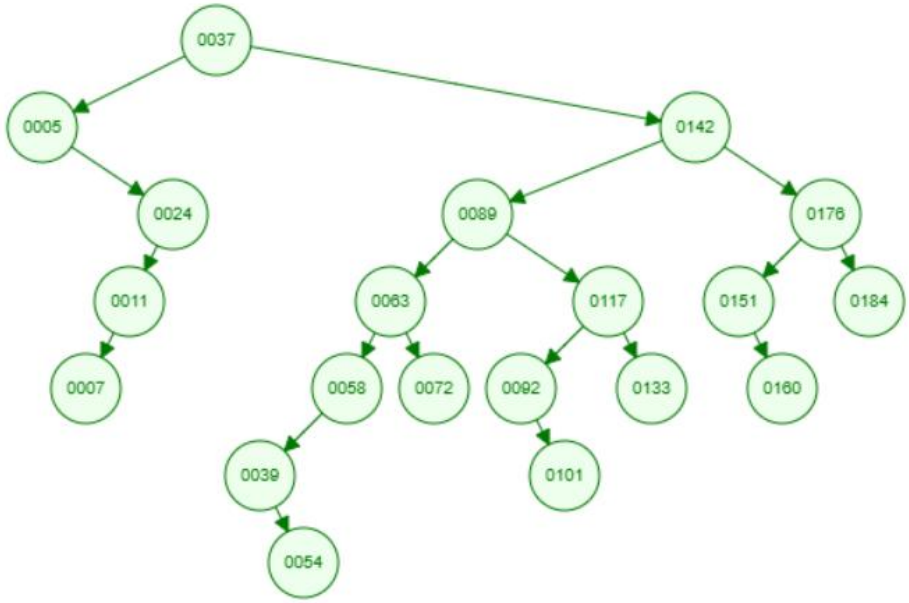
Tool name / URL:

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

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

將數字逐一輸入，接著逐個節點進行比大小。

Screenshot of BST (paste below):



3.4 AVL Tree

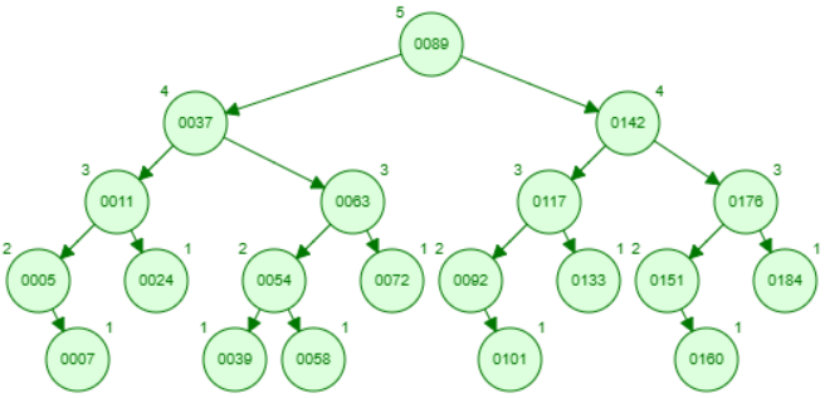
Tool name / URL:

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

Insertion & balancing description:

將數字逐一輸入，接著逐個節點進行比大小，若出現左右子樹高度差的絕對值<=1 時，調整位置並檢查是否符合 BST。

Screenshot of AVL Tree (paste below):



3.5 Red-Black Tree

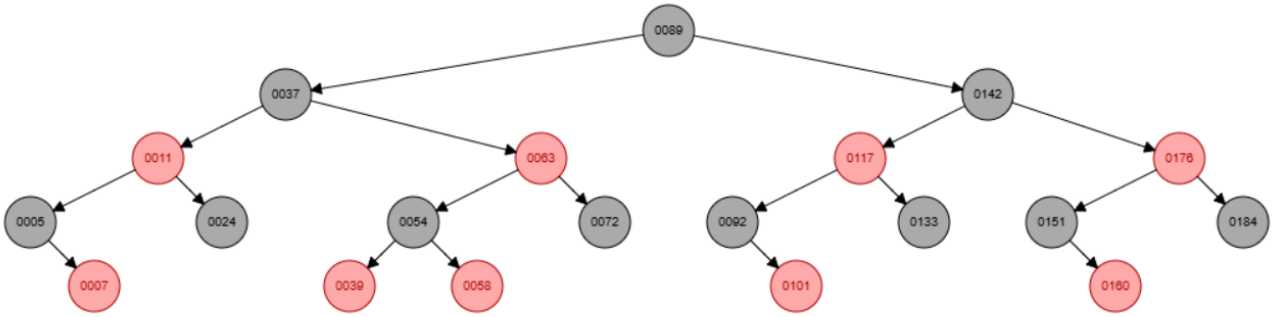
Tool name / URL:

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

Insertion & balancing description:

將數字逐一輸入，接著逐個節點進行比大小，若出現兩個節點連須出現紅色時，調整位置或顏色，並檢查是否符合 BST。

Screenshot of Red-Black Tree (paste below):



3.6 Max Heap

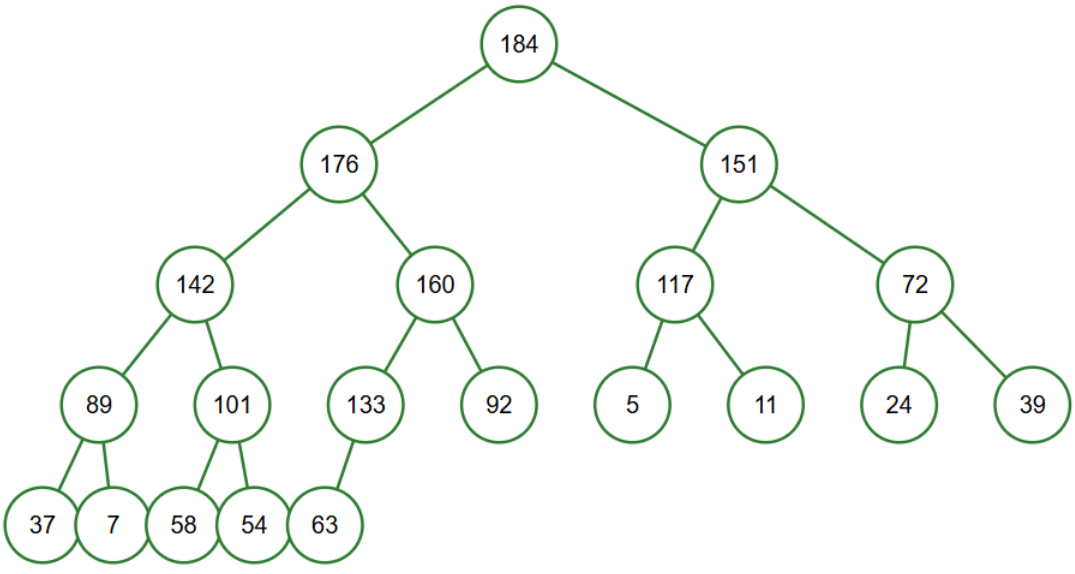
Tool name / URL:

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

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

將數字逐一輸入，以 Complete Binary Tree 的方式填入，再與父節點比大小，比父節點大的與其交換位置。

Screenshot of Max Heap (paste below):



Student ID: 1131528 Student Name: 戴光彥

3.7 Min Heap

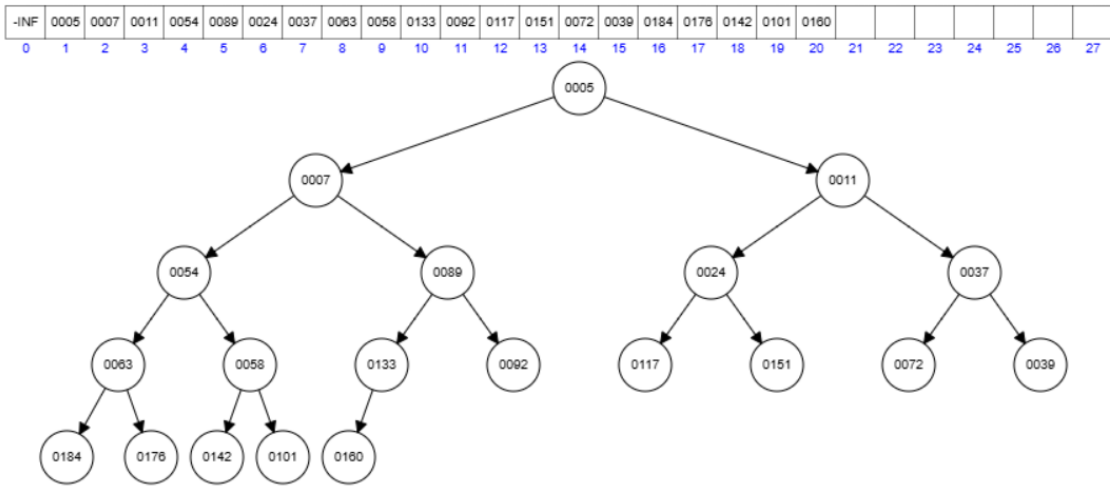
Tool name / URL:

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

Construction / heap-building description:

將數字逐一輸入，以 Complete Binary Tree 的方式填入，再與父節點比大小，比父節點小的與其交換位置。

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	運算式樹	如 3+5。運算子+是父節點，3 和 5 則為是左右子節點。
Complete Binary Tree	Heap 的實作基礎	它可以用陣列來存儲，不需要用指標，節省記憶體空間，且陣列在 CPU Cache 的存取效率極高。
Binary Search Tree	基本的資料查找	利用「左小右大」的特性，讓搜尋像「終極密碼」遊戲一樣，每次都能排除一半的資料，讓查找資料較有效率。
AVL Tree	字典查詢	樹非常扁平，查詢速度最快。
Red-Black Tree	C++ STL std::map	查詢速度略慢於 AVL，但插入和刪除時需要的移動次數少得多。對於通用的程式庫，因為無法預期使用者是多讀或多寫，它提供穩定的綜合效能。
Max Heap	作業系統的工作排程	當 CPU 需要決定下一個執行哪個程式時，它需要選「優先權最高」的那個。

Min Heap	Dijkstra 最短路徑演算法	在導航軟體算路徑時,我們需要從目前距離最近的點開始延伸。

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，AVL Tree 較為扁平，搜尋時不需要經過太多節點，讀取效能較好，在很少更新的情況下，更新資料成本高的問題也可忽略。

Frequent insertions and deletions? Why?

Red-Black Tree，雖然它的搜尋速度略慢於 AVL，但在頻繁寫入的情況下，它省下的時間讓整體效率更高(因其平衡條件較寬鬆)。

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，在資料只需靜態搜尋且不會更新資料的情況下，這樣更能節省空間，因為不需要額外儲存指標占用記憶體空間，且因 Array 是儲存於記憶體的連續空間，單純查找資料更快速。

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	20251215	Gemini	告訴我 general tree, binary tree, complete binary tree, binary search tree, AVL tree, red-black tree, max heap and min heap 的定義，協助我更加熟悉
2	20251215	Gemini	告訴我 binary tree, complete binary tree, binary search tree, AVL tree, red-black tree, max heap and min heap 的應用實例，及為何適合