Student ID:1133322 Student Name:林晉霆

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:

General Tree 是一種從根節點開始，並以父子關係延伸出的階層資料結構。每個節點可以擁有任意多個子節點，且這些子節點的數量不限，子節點之間也沒有固定的順序。

2. Binary Tree

Definition:

二元樹(Binary Tree)是一種以階層方式組成的樹狀結構，每個節點最多只能擁有兩個子節點，分別稱為左子節點及右子節點。

3. Complete Binary Tree

Definition:

Complete Binary Tree 是一種二元樹，除最底層外每一層都被完全填滿，最底層可能不滿且所有節點必須從左至右連續排列。

4. Binary Search Tree (BST)

Definition:

二元搜尋樹（BST）是一種二元樹。其中每個節點的左子樹鍵值皆小於該節點，右子樹鍵值皆大於該節點，且左右子樹本身也符合相同規則。

5. AVL Tree

Definition:

AVL Tree 是一種高度平衡的二元搜尋樹，要求每個節點的左右子樹高度差不超過 1，並在失衡時透過旋轉操作，來維持樹的平衡。

6. Red-Black Tree

Definition:

Red-Black Tree 是一種具有顏色(紅或黑)標記的平衡二元搜尋樹，透過紅黑節點規則與一致的黑高度來維持近似平衡，以確保查詢、插入與刪除等基本操作皆能在 O(log n)時間內完成。紅黑節點規則如下：

(1). 每個節點都是紅色或黑色

(2). 根節點為黑色

(3). 所有葉子（NIL）都是黑色

(4). 若節點是紅色，則其子節點必須是黑色（不能有兩個連續紅）

(5). 從任一節點到其所有葉子的路徑，皆有相同數量的黑色節點（black-height 相同）

7. Max Heap

Definition:

Max Heap 是一種以完全二元樹為結構，並要求每個節點的值不小於其子節點的資料結構，因此根節點必定是整棵樹中的最大值。

8. Min Heap

Definition:

Min Heap 是一種基於完全二元樹的資料結構，其中每個節點的值不大於其子節點，因此根節點必定是整棵樹中的最小值。

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.

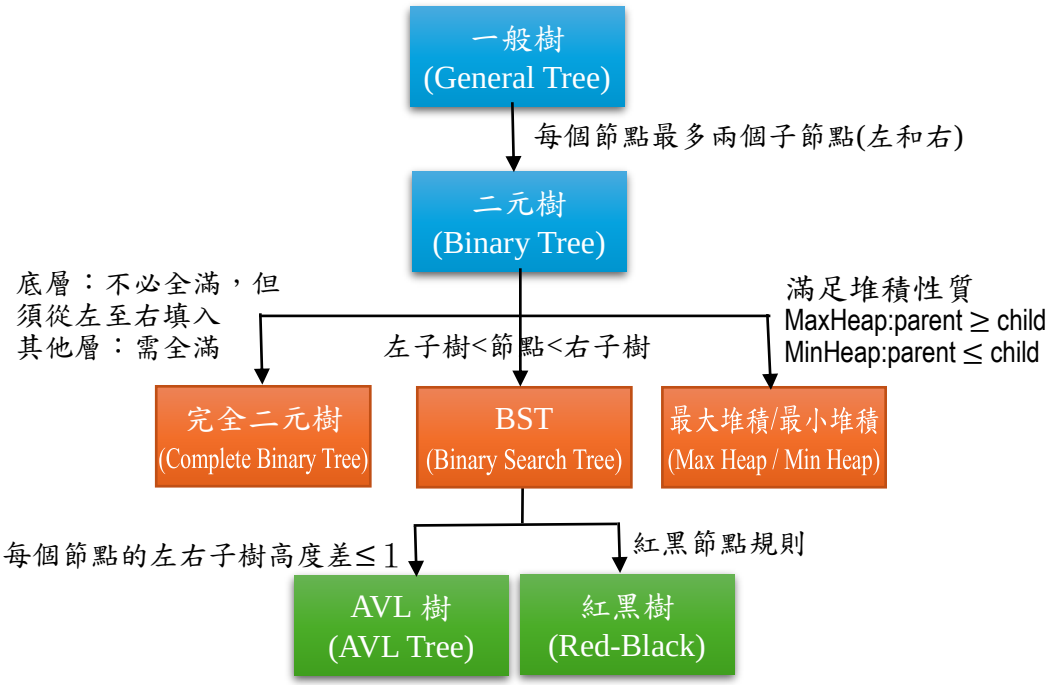
Suggested chain example (you may extend or adjust):

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	限制每個節點最多只能有兩個子節點（左、右）
Binary Tree	Complete Binary Tree	最底層可不必全滿且由左至右填入；其他層填滿
Binary Tree	Binary Search Tree	左子樹 < 節點 < 右子樹，形成可排序的結構

From	To	New property / constraint added
BST	AVL Tree	每個節點的左右子樹高度差≤1
BST	Red-Black Tree	加上紅黑節點規則
Binary Tree	Max Heap	每個節點的值 ≥ 其子節點的值，根節點是整棵樹最大的節點
Binary Tree	Min Heap	每個節點的值 ≤ 其子節點的值，根節點是整棵樹最小的節點

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

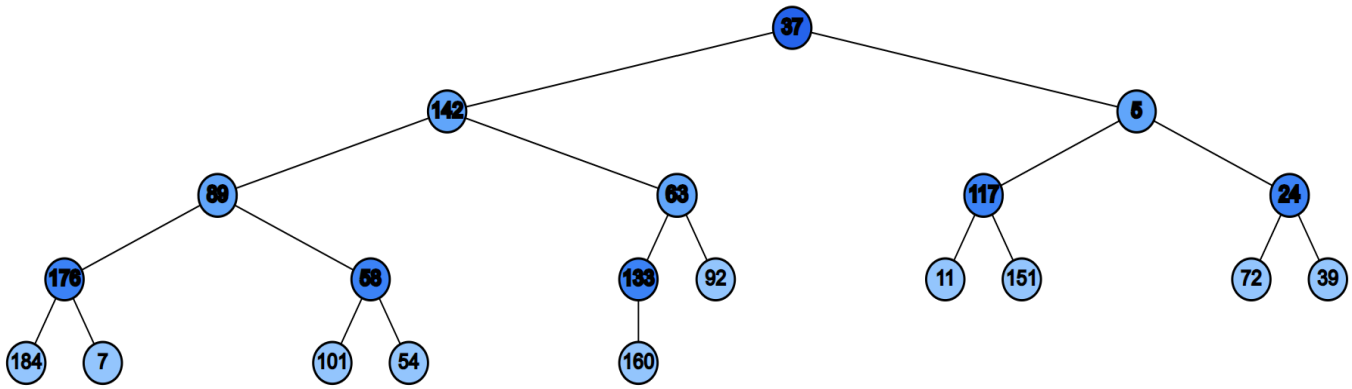
Tool name / URL: https://treeconverter.com/

Construction / insertion description:

①. 插入 37：樹是空的，將 37 作為根節點(root)
②. 插入 142：放到根節點 37 的左子節點
③. 插入 5：放到根節點 37 的右子節點
④. 插入 89：放到 142 的左子節點
⑤. 插入 63：放到 142 的右子節點
⑥. 插入 117：放到 5 的左子節點
⑦. 插入 24：放到 5 的右子節點
⑧. 插入 176：放到 89 的左子節點
⑨. 插入 58：放到 89 的右子節點
⑩. 插入 133：放到 63 的左子節點
⑪. 插入 92：放到 63 的右子節點
⑫. 插入 11：放到 117 的左子節點
⑬. 插入 151：放到 117 的右子節點
⑭. 插入 72：放到 24 的左子節點
⑮. 插入 39：放到 24 的右子節點
⑯. 插入 184：放到 176 的左子節點
⑰. 插入 7：放到 176 的右子節點
⑱. 插入 101：放到 58 的左子節點
⑲. 插入 54：放到 58 的右子節點
⑳. 插入 160：放到 133 的左子節點

Student ID:1133322 Student Name:林晉霆

Screenshot of Binary Tree (paste below):



3.2 Complete Binary Tree

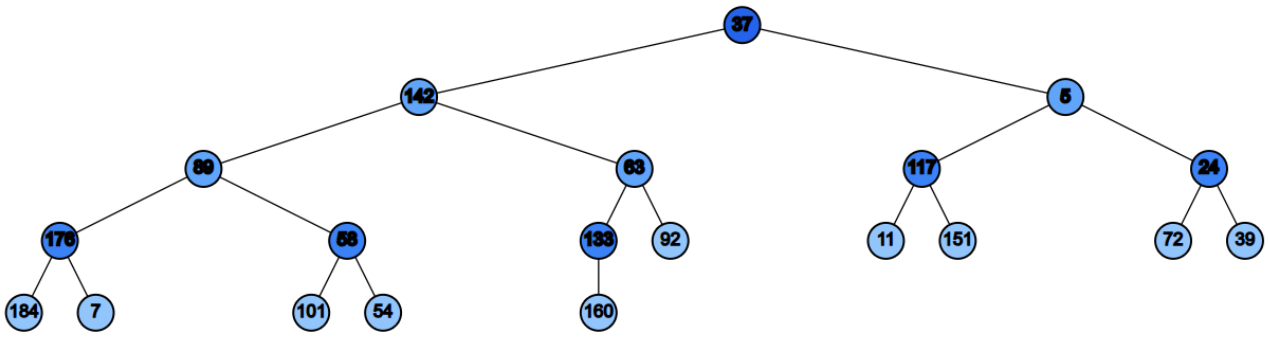
Tool name / URL: https://treeconverter.com/

Construction / insertion description:

使用 Binary Tree 視覺化工具建構 Complete Binary Tree，每個節點最多兩個子節點，依插入順序逐層、從左到右依序填入空位，建立完整樹形。

① 插入 37：樹是空的，將 37 作為根節點(root)
② 插入 142：放到根節點 37 的左子節點
③ 插入 5：放到根節點 37 的右子節點
④ 插入 89：放到 142 的左子節點
⑤ 插入 63：放到 142 的右子節點
⑥ 插入 117：放到 5 的左子節點
⑦ 插入 24：放到 5 的右子節點
⑧ 插入 176：放到 89 的左子節點
⑨ 插入 58：放到 89 的右子節點
⑩ 插入 133：放到 63 的左子節點
⑪ 插入 92：放到 63 的右子節點
⑫ 插入 11：放到 117 的左子節點
⑬ 插入 151：放到 117 的右子節點
⑭ 插入 72：放到 24 的左子節點
⑮ 插入 39：放到 24 的右子節點
⑯ 插入 184：放到 176 的左子節點
⑰ 插入 7：放到 176 的右子節點
⑱ 插入 101：放到 58 的左子節點
⑲ 插入 54：放到 58 的右子節點
⑳ 插入 160：放到 133 的左子節點

Screenshot of Complete Binary Tree (paste below):



3.3 Binary Search Tree (BST)

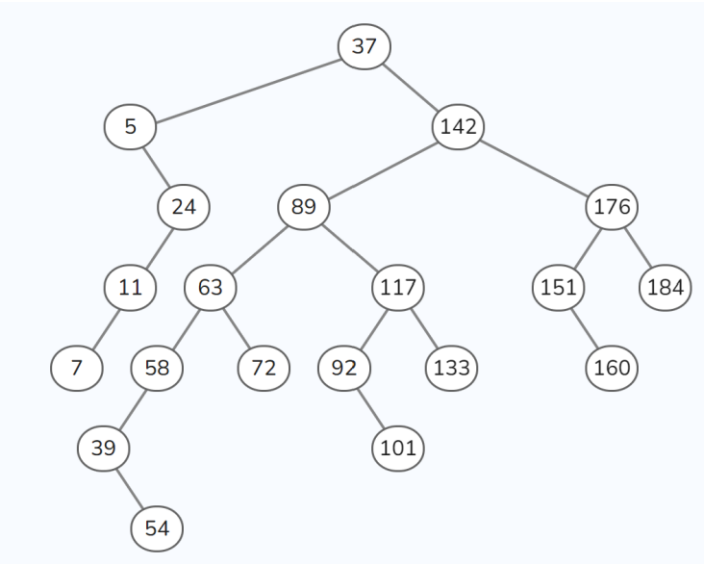
Tool name / URL: https://see-algorithms.com/data-structures/BST

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

①. 插入 37：樹是空的，將 37 作為根節點(root)
②. 插入 142：142 > 37，放到根節點 37 的右子節點
③. 插入 5：5 < 37，放到根節點 37 的左子節點
④. 插入 89：89 > 37→89 < 142，放到節點 142 的左子節點
⑤. 插入 63：63 > 37→63 < 142→63 < 89，放到節點 89 的左子節點
⑥. 插入 117：117 > 37→117 < 142→117 > 89，放到節點 89 的右子節點
⑦. 插入 24：24 < 37→24 > 5，放到 5 的右子節點
⑧. 插入 176：176 > 37→176 > 142，放到節點 142 的右子節點
⑨. 插入 58：58 > 37→58 < 142→58 < 89→58 < 63，放到節點 63 的左子節點
⑩. 插入 133：133 > 37→133 < 142→133> 89→133> 117，放到節點 117 的右子節點
⑪. 插入 92：92 > 37→92 < 142→92 > 89→92 < 117，放到節點 117 的左子節點
⑫. 插入 11：11 < 37→11 > 5→11 < 24，放到 24 的左子節點
⑬. 插入 151：151> 37→151 > 142→151 < 176，放到節點 176 的左子節點
⑭. 插入 72：72> 37→72< 142→72 < 89→72> 63，放到節點 63 的右子節點
⑮. 插入 39：39> 37→39< 142→39< 89→39< 63→39< 58，放到節點 58 的左子節點
⑯. 插入 184：184> 37→184 > 142→184 > 176，放到節點 176 的右子節點
⑰. 插入 7：7 < 37→7 > 5，放到 5 的左子節點
⑱. 插入 101：101> 37→101<142→101 > 89→101 < 117→101 > 92，放到節點 92 的右子節點
⑲. 插入 54：54> 37→54< 142→54< 89→54< 63→54< 58→54> 39，放到節點 39 的右子節點
⑳ 插入 160：160> 37→160>142→160 < 176→160 > 151，放到節點 151 的右子節點

Screenshot of BST (paste below):

Student ID:1133322 Student Name:林晉霆



3.4 AVL Tree

Tool name / URL: https://see-algorithms.com/data-structures/AVL

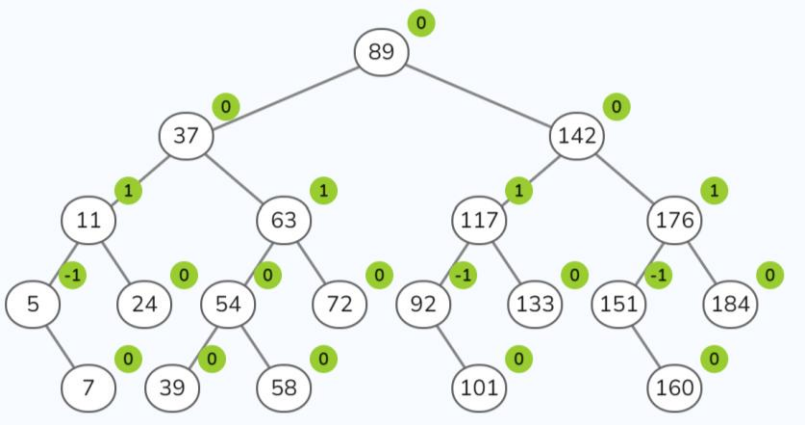
Insertion & balancing description:

	插入	加入位置	高度差(|左 – 右|)	失衡否	處理方式
①.	37	樹是空的，將 37 作為根節點	0	平衡	
②.	142	142 > 37，放到根節點 37 的右子節點	37:|0 – 1|=1	平衡	
③.	5	5 < 37，放到根節點 37 的左子節點	37: |1 – 1|=0	平衡	
④.	89	89 > 37→89 < 142，放到節點 142 的左子節點	37: |1 – 2|=1 142: |1 – 0|=1	平衡	
⑤.	63	63 > 37→63 < 142→63 < 89，放到節點 89 的左子節點	89: |1 – 0|=1 142: |2 – 0|=2>1	LL	右旋 142: 節點 89 上升成新子樹根、節點 142 下降成節點 89 的右子樹
⑥.	117	117 > 37→117 > 89→117 < 142，放到節點 142 的左子節點	142: |1 – 0|=1 89: |1 – 2|=1 37: |1 – 3|=2 >1	RR	左旋 37: 節點 89 上升成新的根節點、節點 37 下降成 89 的左子節點
⑦.	24	24 < 89→24 < 37→24 > 5，放到 5 的右子節點	5: |0 – 1|=-1 89: |3 – 2|=1 37: |2 – 1|=1	平衡	
⑧.	176	176 > 89→176 > 142，放到節點 142 的右子節點	142: |1 – 1|=0	平衡	
⑨.	58	58 < 89→58 > 37→58 < 63，放到節點 63 的左子節點	63: |1 – 0|=1 37: |2 – 2|=0	平衡	
⑩.	133	133> 89 → 133 < 142 → 133> 117，放到節點 117 的右子節點	117: |0 – 1|=1 142: |2 – 1|=1 89: |3 – 3|=0	平衡	
⑪	92	92 > 89→92 < 142→92 < 117，放到節點 117 的左子節點	117: |1 – 1|=0	平衡	

Student ID:1133322 Student Name:林晉霆

	插入	加入位置	高度差(|左－右|)	失衡否	處理方式
⑫	11	11 < 89→11 < 37→11> 5→11 < 24，放到 24 的左子節點	5: |0 – 2|=2 >1 24: |1 – 0|=1	RL	左旋 24: 節點 11 上升、節點 24 下降成節點 11 的右子樹 右旋 5: 節點 11 上升成新子樹根、節點 24 下降成節點 11 的右子樹
⑬	151	151> 89→151 > 142→151 < 176，放到節點 176 的左子節點	176: |1 – 0|=1 142: |1 – 1|=0	平衡	
⑭	72	72 < 89→72> 37→72> 63，放到節點 63 的右子節點	63: |1 – 1|=0	平衡	
⑮	39	39< 89→39> 37→39< 63→39< 58，放到節點 58 的左子節點	58: |1 – 0|=1 63: |2 – 1|=1 37: |2 – 3|=1 89: |4 – 3|=1	平衡	
⑯	184	184> 89→184 > 142→184 > 176，放到節點 176 的右子節點	176: |1 – 1|=0 142: |2 – 2|=0	平衡	
⑰	7	7 < 89→7 < 37→7 < 11→7 > 5，放到 5 的右子節點	5: |0 – 1|=1 11: |2 – 1|=1 37: |3 – 3|=0	平衡	
⑱	101	101> 89→101<142→101 < 117→101 > 92，放到節點 92 的右子節點	92: |0 – 1|=1 117: |2 – 1|=1 142: |3 – 2|=1 89: |4 – 4|=0	平衡	
⑲	54	54< 89→54> 37→54< 63→54< 58→54> 39，放到節點 39 的右子節點	39: |0 – 1|=1 58: |2 – 0|=2 >1	LR	左旋 39: 節點 54 上升、節點 39 下降成節點 54 的左子樹 右旋 58: 節點 54 上升成新子樹根、節點 58 下降成節點 54 的右子樹
⑳	160	160> 89→160>142→160 < 176→160 > 151，放到節點 151 的右子節點	151: |0 – 1|=1 176: |2 – 1|=1 142: |2 – 2|=0	平衡	

Screenshot of AVL Tree (paste below):



3.5 Red-Black Tree

Tool name / URL:

https://ds2-iiith.vlabs.ac.in/exp/red-black-tree/red-black-tree-oprations/simulation/redblack.html

Insertion & balancing description:

①. 插入 37：樹是空的，將 37 作為根節點、黑色

②. 插入 142：BST 規則放 37 右子節點，插入紅色，父 37(黑)→無衝突

③. 插入 5：BST 規則放 37 左子節點，插入紅色，父 37(黑)→無衝突

④. 插入 89：BST 規則放 142 左子節點，插入紅色，父 142(紅)→紅紅衝突→uncle 節點 5(紅)→改色

- 父節點 142(紅)→黑色
- uncle 節點 5(紅)→黑色
- 祖父節點 37(黑)→紅色→黑色(根節點需黑色)

⑤. 插入 63：BST 規則放 89 左子節點，插入紅色，父 89(紅)→紅紅衝突→uncle 節點 NIL(黑) →旋轉+改色

- 節點 63(紅)左、父 89(紅)左→右旋祖父 142(黑)→節點 89 上升成新子樹根、節點 142 下降成節點 89 的右子樹
- 節點 89(紅)→黑色
- 節點 142(黑)→紅色

⑥. 插入 117：BST 規則放 142 左子節點，插入紅色，父 142(紅)→紅紅衝突→uncle 節點 63(紅)→改色

- 父節點 142(紅)→黑色
- uncle 節點 63(紅)→黑色
- 祖父節點 89(黑)→紅色

⑦. 插入 24：BST 規則放 5 右子節點，插入紅色，父 5(黑)→無衝突

⑧. 插入 176：BST 規則放 142 右子節點，插入紅色，父 142(黑)→無衝突

⑨. 插入 58：BST 規則放 63 左子節點，插入紅色，父 63(黑)→無衝突

⑩. 插入 133：BST 規則放 117 右子節點，插入紅色，父 117(紅) →紅紅衝突→uncle 節點 176(紅) →改色

- 父節點 117(紅)→黑色
- uncle 節點 176(紅)→黑色
- 祖父節點 142(黑)→紅色
- 節點 142(紅)右、父節點 89(紅)右→左旋祖父 37(黑)→節點 89 上升成新子樹根、節點 37 下降成節點 89 的左子樹
- 節點 37 (黑)→紅色
- 節點 89(紅)→黑色

⑪. 插入 92：BST 規則放 117 左子節點，插入紅色，父 117(黑)→無衝突

⑫. 插入 11：BST 規則放 24 左子節點，插入紅色，父 24(紅)→紅紅衝突→uncle 節點 NIL(黑) →旋轉+改色

- 節點 11(紅)左、父 24(紅)右
  →父 24(紅)右旋：節點 11 上升、父節點 24 下降成節點 11 的右子樹
  →祖父 5(黑)左旋：節點 11 上升成新子樹根
- 節點 5(黑)→紅色
- 節點 11(紅)→黑色

⑬. 插入 151：BST 規則放 176 左子節點，插入紅色，父 176(黑)→無衝突

⑭. 插入 72：BST 規則放 63 左子節點，插入紅色，父 63(黑)→無衝突

⑮. 插入 39：BST 規則放 58 左子節點，插入紅色，父 58(紅)→紅紅衝突→uncle 節點 72(紅)→改色

- 父節點 58(紅)→黑色
- uncle 節點 72(紅)→黑色
- 祖父節點 63(黑)→紅色
- 節點 63(紅)右、父節點 37(紅)左→紅紅衝突→改色
  - ➲ 節點 37(紅)→黑色
  - ➲ 節點 89(黑)→紅色
  - ➲ 節點 142(紅)→黑色

⑯. 插入 184：BST 規則放 176 右子節點，插入紅色，父 176(黑)→無衝突

⑰. 插入 7：BST 規則放 5 右子節點，插入紅色，父 5(紅)→紅紅衝突→uncle 節點 24(紅)→改色

- 父節點 5(紅)→黑色
- uncle 節點 24(紅)→黑色
- 祖父節點 11(黑)→紅色

⑱. 插入 101：BST 規則放 92 右子節點，插入紅色，父 92(紅)→紅紅衝突→uncle 節點 133(紅)→改色

- 父節點 92(紅)→黑色
- uncle 節點 133(紅)→黑色
- 祖父節點 117(黑)→紅色

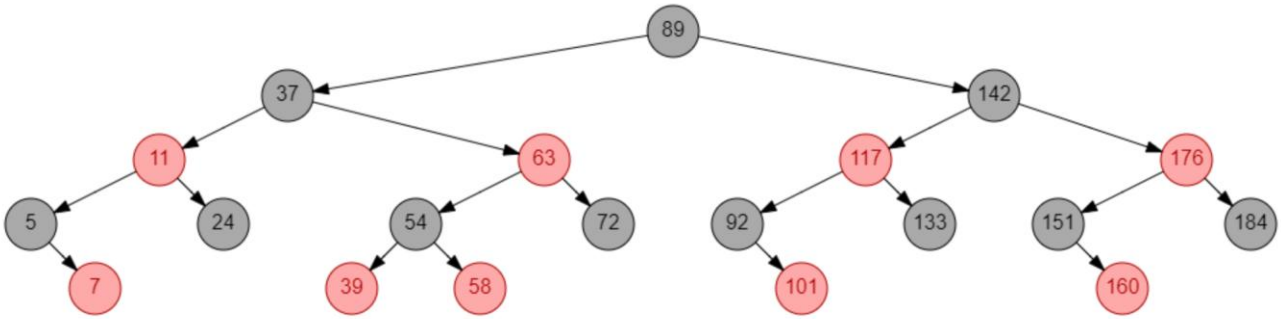
⑲. 插入 54：BST 規則放 28 右子節點，插入紅色，父 39(紅)→紅紅衝突→uncle 節點 NIL(黑) →旋轉+改色

- 節點 54(紅)右、父節點 39(紅)左
  →父 39(紅)左旋：節點 54 上升、父節點 39 下降成節點 54 的左子樹
  →祖父 58(黑)右旋：節點 54 上升成新子樹根
- 節點 54(紅)→黑色
- 節點 58(黑)→紅色

⑳. 插入 160：BST 規則放 151 右子節點，插入紅色，父 151(紅)→紅紅衝突→uncle 節點 184(紅) →改色

- 父節點 151(紅)→黑色
- uncle 節點 184(紅)→黑色
- 祖父節點 176(黑)→紅色

Screenshot of Red-Black Tree (paste below):



3.6 Max Heap

Tool name / URL:

https://see-algorithms.com/data-structures/BinaryHeap

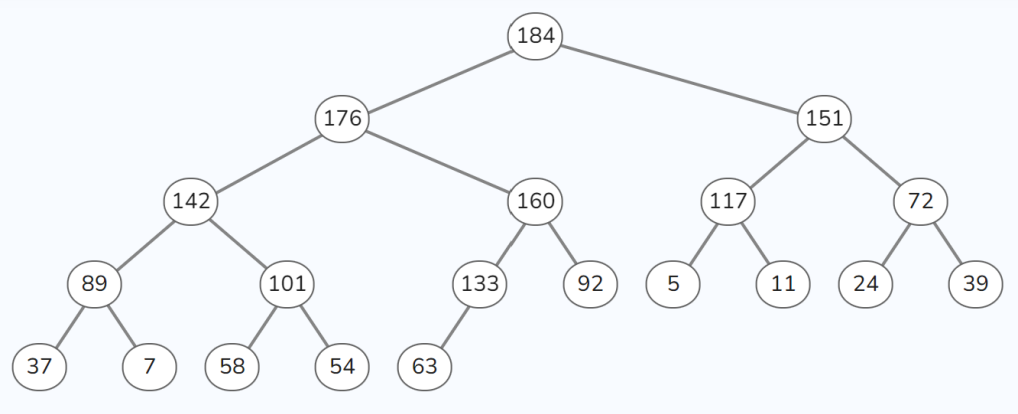
Construction / heap-building description (e.g. heapify, insert-and-sift-up):

	插入	加入位置	Bubble Up	處理方式
①.	37	樹是空的，將 37 作為根節點		
②.	142	放到根節點 37 的左子節點	142>37	交換:根節點變成 142
③.	5	5 放到根節點 142 的右子節點	5<142	不交換
④.	89	89 放到節點 37 的右子節點	89>37 89<142	89 與 37 交換
⑤.	63	63 放到節點 89 的右子節點	63<89	不交換
⑥.	117	117 放到節點 5 的左子節點	117>5 117<142	117 與 5 交換
⑦.	24	24 放到節點 117 的右子節點	24<117	不交換
⑧.	176	176 放到節點 37 的左子節點	176>37 176>89 176>142	176 與 37 交換 176 與 89 交換 176 與 142 交換 根節點變成 176
⑨.	58	58 放到節點 89 的右子節點	58<89	不交換
⑩.	133	133 放到節點 63 的左子節點	133>63 133<142	133 與 63 交換
⑪	92	92 放到節點 133 的右子節點	92<133	不交換
⑫	11	11 放到 5 的左子節點	11>5 11<117	11 與 5 交換

Student ID:1133322 Student Name:林晉霆

	插入	加入位置	Bubble Up	處理方式
⑬	151	151 放到節點 11 的右子節點	151>11 151>117 151<176	151 與 11 交換 117 與 151 交換
⑭	72	72 放到節點 24 的左子節點	72>24 72<151	72 與 24 交換
⑮	39	39 放到節點 72 的右子節點	39<72	不交換
⑯	184	184 放到節點 37 的左子節點	184>37 184>89 184>142 184>176	184 與 37 交換 184 與 89 交換 184 與 142 交換 184 與 176 交換 根節點變成 184
⑰	7	7 放到 89 的右子節點	7<89	不交換
⑱	101	101 放到節點 58 的左子節點	101>58 101<142	101 與 58 交換
⑲	54	54 放到節點 101 的右子節點	54<101	不交換
⑳	160	160 放到節點 63 的左子節點	160>63 160>133 160<176	160 與 63 交換 160 與 133 交換

Screenshot of Max Heap (paste below):



3.7 Min Heap

Tool name / URL:

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

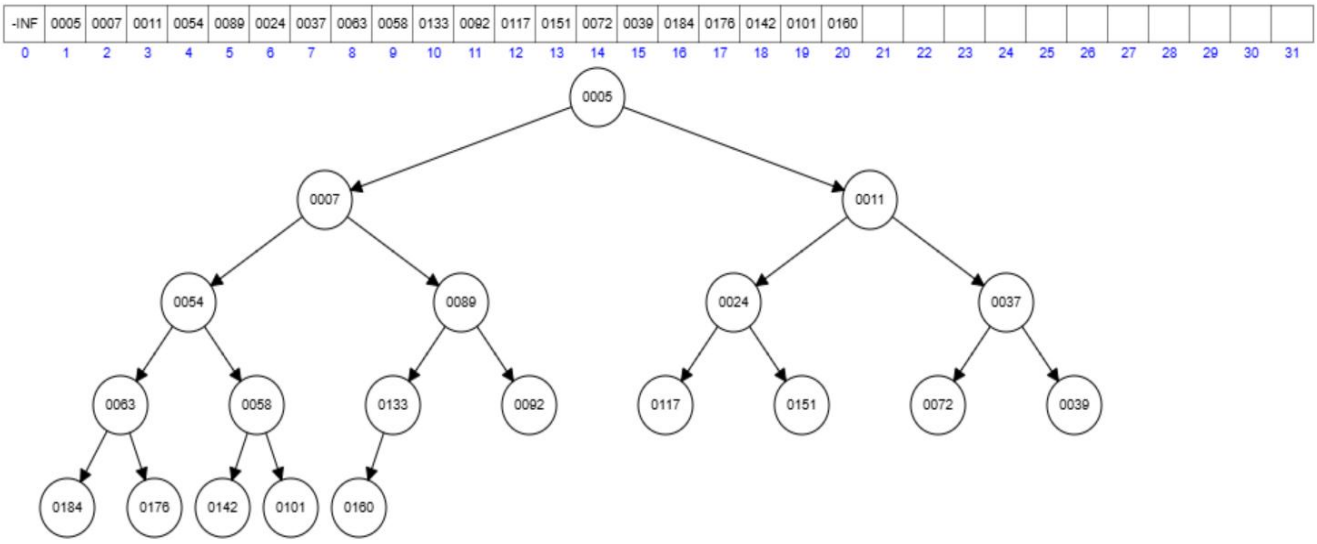
Construction / heap-building description:

	插入	加入位置	Bubble Down	處理方式
①.	37	樹是空的，將 37 作為根節點		
②.	142	放到根節點 37 的左子節點	142>37	不交換
③.	5	5 放到根節點 37 的右子節點	5<37	5 與 7 交換；根節點→5
④.	89	89 放到節點 142 的左子節點	89<142; 89>5	89 與 142 交換
⑤.	63	63 放到節點 89 的右子節點	63<89; 63>5	89 與 63 交換

Student ID:1133322 Student Name:林晉霆

	插入	加入位置	Bubble Down	處理方式
⑥.	117	117 放到節點 37 的左子節點	117>37	不交換
⑦.	24	24 放到節點 37 的右子節點	24<37 ; 24>5	24 與 37 交換
⑧.	176	176 放到節點 142 的左子節點	176>142	不交換
⑨.	58	58 放到節點 142 的右子節點	58<142 ; 58<63 ; 58>5	58 與 142 交換 58 與 63 交換
⑩.	133	133 放到節點 89 的左子節點	133>89	不交換
⑪	92	92 放到節點 89 的右子節點	92>89	不交換
⑫	11	11 放到 117 的左子節點	11<117 ; 11<24 ; 11>5	11 與 117 交換 11 與 24 交換
⑬	151	151 放到節點 24 的右子節點	151>24	不交換
⑭	72	72 放到節點 37 的左子節點	72>37	不交換
⑮	39	39 放到節點 37 的右子節點	39>37	不交換
⑯	184	184 放到節點 176 的左子節點	184>176	不交換
⑰	7	7 放到 176 的右子節點	7<176 ; 7<63 ; 7<58 ; 7>5	7 與 176 交換 7 與 63 交換 7 與 58 交換
⑱	101	101 放到節點 142 的左子節點	101<142 ; 101>58	101 與 142 交換
⑲	54	54 放到節點 101 的右子節點	54<101 ; 54<58 ; 54>7	54 與 101 交換 54 與 58 交換
⑳	160	160 放到節點 133 的左子節點	160>133	不交換

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(Expression Tree) • 用二元樹表示運算式，程式可透過樹結構計算結果或調整運算順序。	每個節點最多兩個子節點，父節點是運算子、子節點是運算元或子運算式，清楚表達運算順序；程式可用前序、中序或後序遍歷計算結果或轉換表示法。
Complete Binary Tree	• Priority Queue(程式資料結構) • 管理需要依優先順序處理的任務，如印表機工作佇列)	節點按層次填滿，每個節點能快速找到父節點和子節點，插入或刪除元素效率高且穩定，適合需快速取得最大或最小元素的優先佇列，如印表機工作佇列或最短路徑。
Binary Search Tree	• 資料庫索引 • 將資料欄位（如學號、訂單編號）建立成資料庫索引，採用二元搜尋樹實作，方便程式快速查找特定紀錄	二元搜尋樹的有序特性讓程式能依鍵值快速查找特定紀錄，即使新增或刪除資料，樹仍保持有序，有助於提升查詢效率，因此適合用來建立資料庫索引。
AVL Tree	• 路由表（routing table） • 透過 AVL Tree 幫助路由器快速決定封包該往哪邊走。	AVL Tree 樹不會太高，因此查詢效率快且穩定。尤其路由表大多查詢多、修改少，旋轉成本不高，可以讓封包迅速找到該走的路。
Red-Black Tree	• C++ STL 的 map / set • STL 的 map 和 set 底層使用紅黑樹，程式可快速查詢、插入或刪除元素，且元素會自動維持排序	元素依鍵值自動排序，有助於程式高效率地查詢、插入或刪除資料，同時方便維護順序，因此適合用於 STL 的 map 和 set。
Max Heap	• 作業系統排程（CPU Scheduling） • OS 從等待執行的程序中，選出優先權最高的程序讓 CPU 執行	樹根節點即最大值，CPU 可以直接取得優先權最高的程序；節點排列密集，節省記憶體且方便存取，適合排程中快速挑出高優先級程序的需求。
Min Heap	• GPS 導航系統 • 用 Dijkstra 最短路徑演算法，找從起點到目的地的最短路徑	樹根節點是最小值，因此程式可以直接快速取得目前最短的路徑節點，適合演算法中需要頻繁選取最短距離節點的情況。

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

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

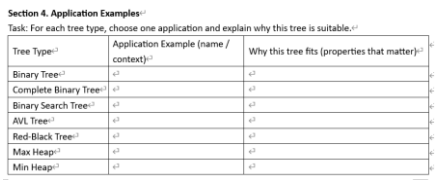
Mostly search (few updates)? Why?
AVL 會把樹維持得非常平衡，所以搜尋速度總是很快，不會因資料偏斜而變慢。如果更新不頻繁，維持平衡所需的旋轉成本也很低。

Frequent insertions and deletions? Why?
紅黑樹採用較寬鬆的平衡規則，因此在插入與刪除時所需的旋轉遠少於 AVL 樹。雖然其高度略高於 AVL，但在更新階段多半只需重新著色而不必旋轉，使得在高更新頻率的系統中效率更佳。

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?
資料不會再更動，那麼將其排序後以陣列的方式儲存最佳。不用擔心插入或刪除慢，也不用浪費空間額外儲存指標或其他資訊。用二元搜尋資料不但速度快，且因資料連續存放，透過 CPU 快取機制順便把附近資料讀進快取，查找速度更快。

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	12/8 23:15	ChatGPT	提供 general trees, binary trees, complete binary trees, binary search trees, AVL trees, red-black trees, max-heaps, and min-heaps 的定義
2	12/9 14:30	ChatGPT	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 幫我生成 binary trees, complete binary trees, binary search trees, AVL trees, red-black trees, max-heaps, and min-heaps 的 Construction / insertion description
3	12/9 19:30	ChatGPT	 給點意見跟寫的方向
4	12/9 19:39	ChatGPT	Application Example 實際點
5	12/9 20:39	ChatGPT	Among BST, AVL, and Red-Black trees, which one would you pick for: Mostly search (few updates)? Why? Frequent insertions and deletions? Why? 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?具體點

You may extend this table as needed.

https://chatgpt.com/share/69381966-1b74-800f-8750-45913e0a720f