Student ID: Student Name:

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

Student ID: Student Name:

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.

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:

一種具階層關係的非線性資料結構，由 Root 出發，與 Binary Tree 最大的區別在於不限制 Children 的數量，即每個 Node 的 Degree 可以是任意值

2. Binary Tree
   Definition:
   規定每個節點最多只能分岔出兩條路（Degree<= 2），它非常講究順序，左邊的 Children 跟右邊的 Children 是完全不同的位置，在 Binary Tree 裡，把左子樹搬到右邊，就會變成一棵完全不同的樹
3. Complete Binary Tree
   Definition:
   除了 Last Level 之外，其餘各層的節點數皆達到最大值，在最後一層，所有節點也必須嚴格遵循 Left-aligned (由左至右依序填入) 的規則，中間不得有空缺
4. Binary Search Tree (BST)
   Definition:
   嚴格規定數值排序，對於樹中的任意節點，其左子樹的所有數值都必須比該節點小，而右子樹的所有數值都必須比該節點大
5. AVL Tree
   Definition:
   樹中任何節點的左右子樹高度差的絕對值不能超過 1，一旦新增或刪除節點導致高度差變成 2 或更多，就會立即透過 Rotation 來重新平衡樹的高度
6. Red-Black Tree
   Definition:
   透過特定規則來確保平衡，紅色節點的子節點必須是黑色（不能有連續的紅色節點）
7. Max Heap
   Definition:
   對於每一個節點，其數值都必須大於或等於其所有子節點的數值，代表 Root 永遠是整棵樹中最大的元素
8. Min Heap
   Definition:
   每個節點的數值必須小於或等於其所有子節點，這確保了 Root 總是包含整棵樹中最小的數值，常用於需要快速存取最小元素的演算法中

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):

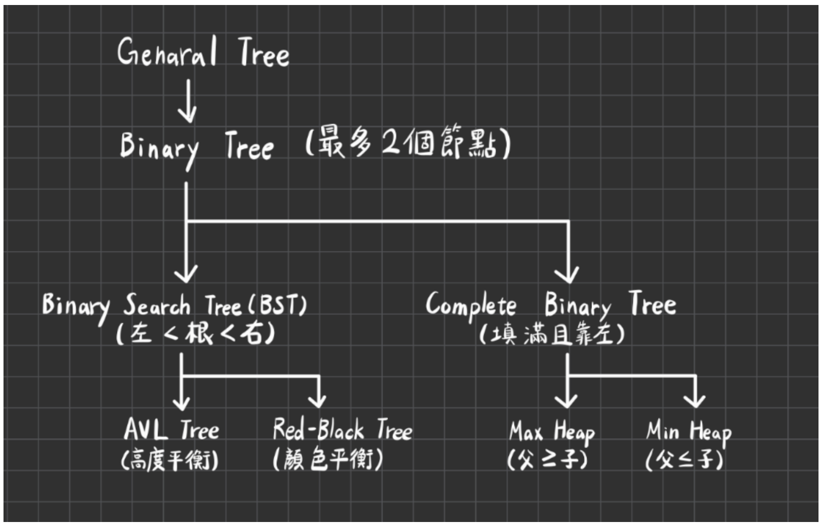
General Tree → Binary Tree → Complete Binary Tree

Student ID: Student Name:

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	最多兩個節點 (Degree<= 2)
Binary Tree	Complete Binary Tree	需填滿並且靠左
Binary Tree	Binary Search Tree	數值排序規則，左 < 根 < 右
BST	AVL Tree	嚴格的高度平衡，左右子樹高度差絕對值<= 1
BST	Red-Black Tree	紅色節點的子節點必為黑色
Binary Tree	Max Heap	父節點數值>=子節點數值
Binary Tree	Min Heap	父節點數值<=子節點數值

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:

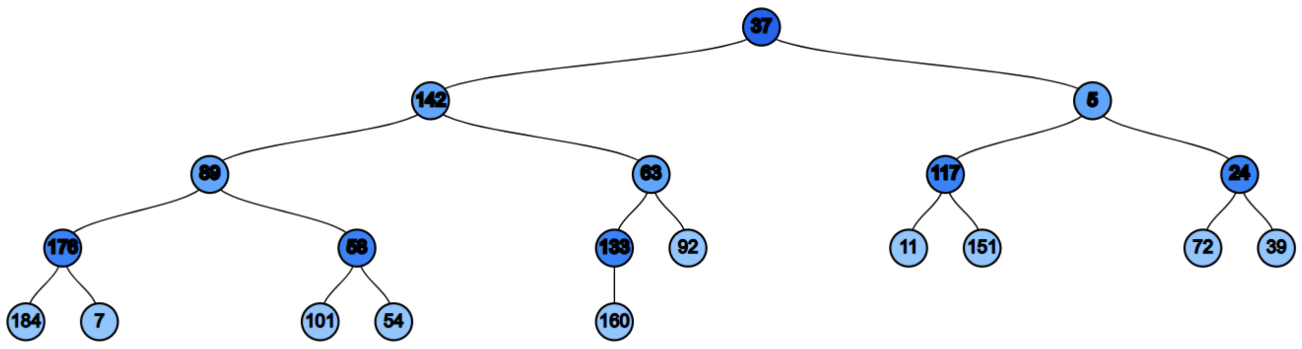
Binary Tree and Graph Visualizer / https://treeconverter.com/

Construction / insertion description:

Student ID: Student Name:

將給定的 20 個數依照順序，由上而下、由左至右依序填入二元樹的每個節點位置，不進行數值排序或結構調整

Screenshot of Binary Tree (paste below):



3.2 Complete Binary Tree

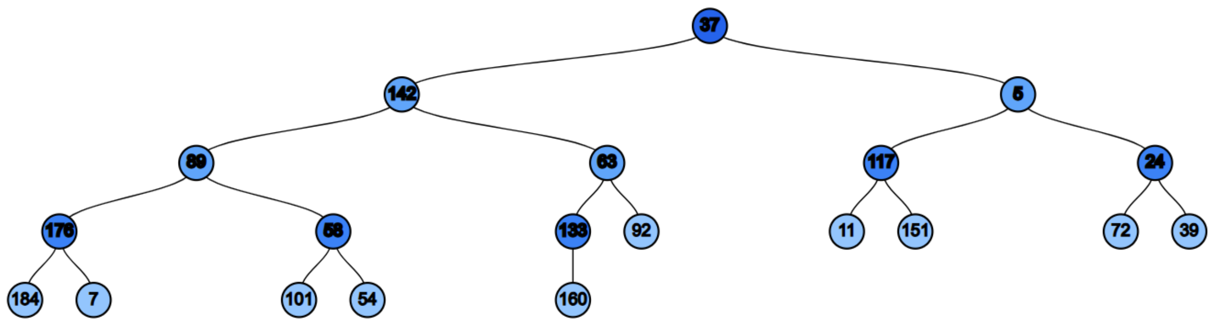
Tool name / URL:

Binary Tree and Graph Visualizer / https://treeconverter.com/

Construction / insertion description:

將給定的 20 個數依照順序，由上而下、由左至右依序填入，確保每一層節點都填滿，最後一層節點靠左對齊，符合 Complete Binary Tree 的結構要求

Screenshot of Complete Binary Tree (paste below):



Student ID: Student Name:

3.3 Binary Search Tree (BST)

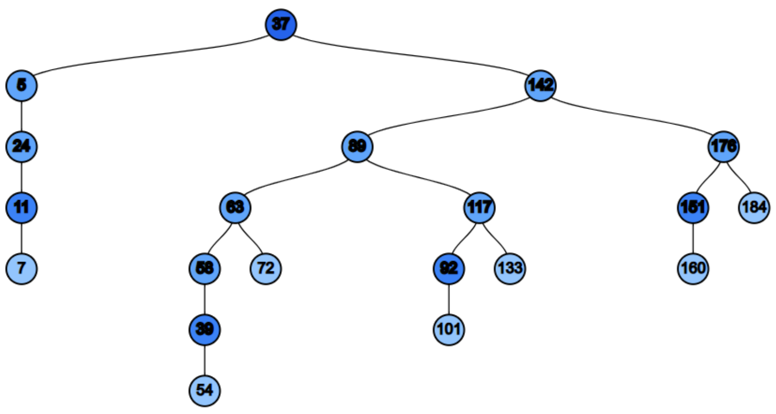
Tool name / URL:

Binary Tree and Graph Visualizer / https://treeconverter.com/

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

將給定的 20 個數依照順序逐一插入 BST，數值較小者放入左子樹，較大者放入右子樹，過程中不執行任何 Rebalancing，保留樹的原始生長結構

Screenshot of BST (paste below):



3.4 AVL Tree

Tool name / URL:

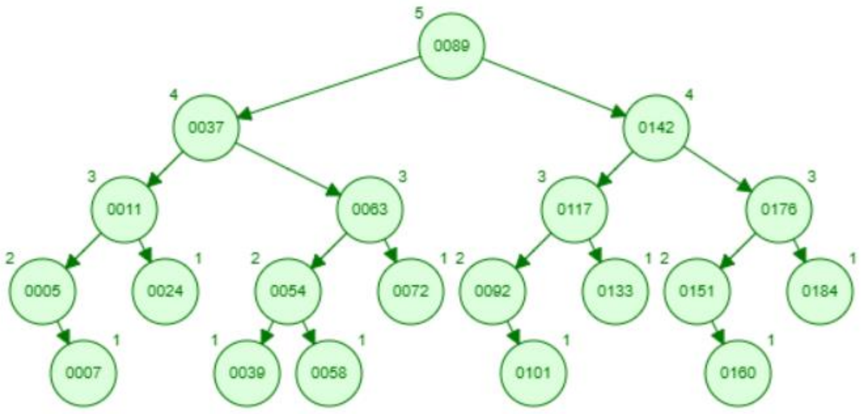
USF CA Data Structure Visualizations/

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

Insertion & balancing description:

將給定的 20 個數依序插入，每次插入後，系統會自動計算每個節點的 Balance Factor，若發現左右子樹高度差的絕對值超過 1，則執行旋轉操作以降低高度並維持全樹平衡

Screenshot of AVL Tree (paste below):



3.5 Red-Black Tree

Tool name / URL:

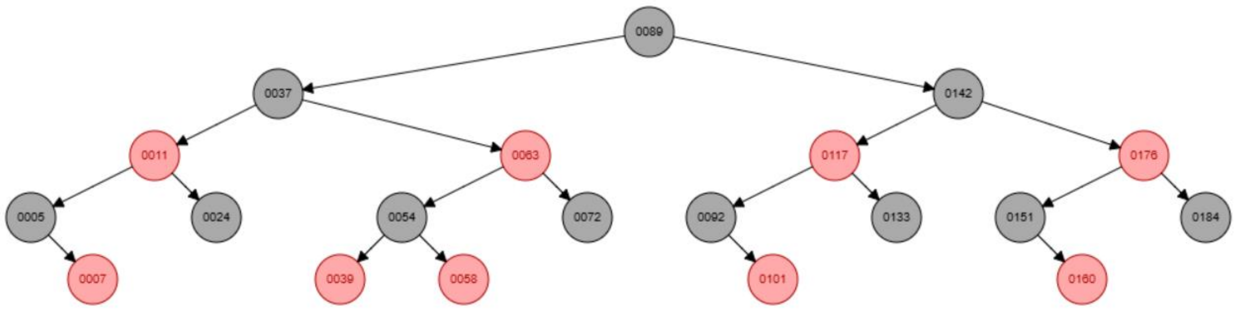
USF CA Data Structure Visualizations /

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

Insertion & balancing description:

將給定的 20 個數插入 Red-Black Tree，新節點一律預設為 Red，並依 BST 規則落位，插入後隨即檢查性質，若偵測到違反規則，則視情況透過 Recoloring 或 Rotation 進行修正，以確保整棵樹的 Black Height 平衡

Screenshot of Red-Black Tree (paste below):



Student ID: Student Name:

3.6 Max Heap

Tool name / URL:

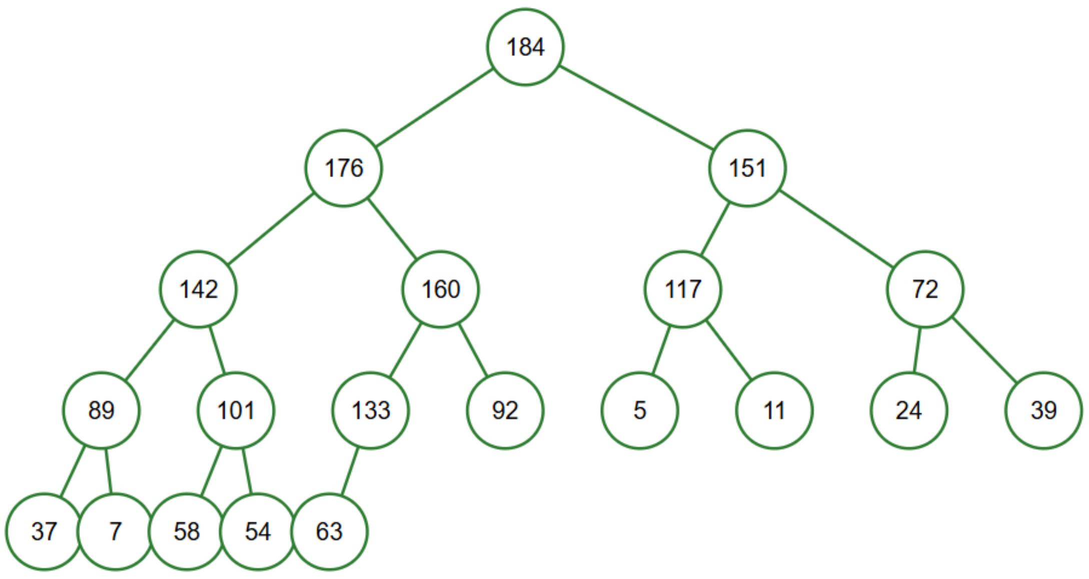
Max Heap Simulator/

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

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

將給定的 20 個數插入，每次插入時，新元素會先放在最底層的最後一個位置，然後執行 Sift Up / Percolate Up，不斷與父節點比較，若比父節點大則交換，直到符合 Max Heap 為止

Screenshot of Max Heap (paste below):



3.7 Min Heap

Tool name / URL:

USF CA Data Structure Visualizations /

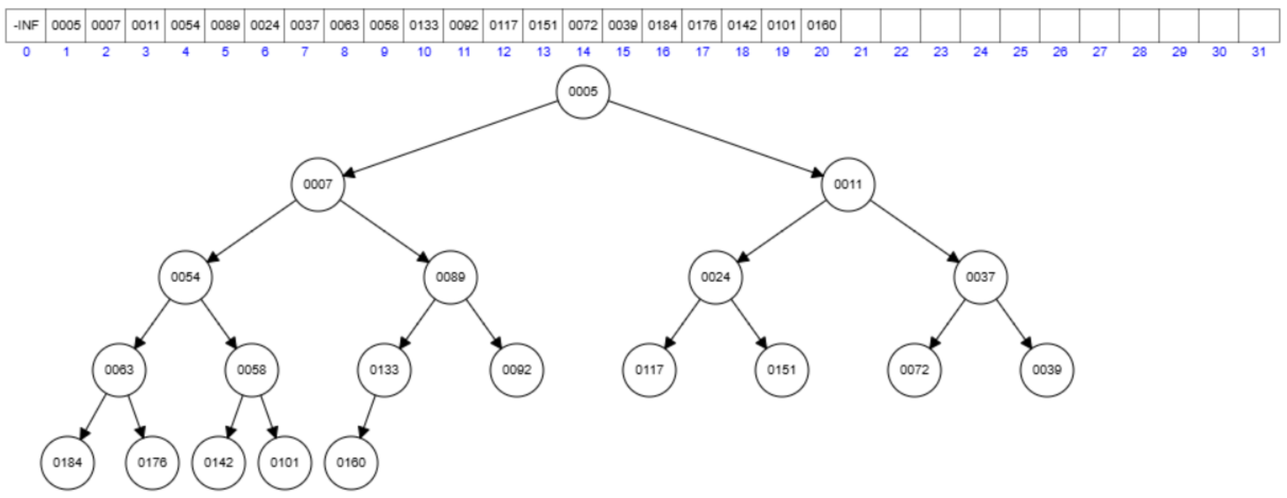
https://www.cs.usfca.edu/~galles/visualization/Heap.html

Construction / heap-building description:

將給定的 20 個數插入，每次插入新元素時，先放置在堆積的最末端，然後執行 Sift Up / Percolate Up，若該元素比父節點小，則與父節點交換位置，直到滿足 Min Heap 規定

Screenshot of Min Heap (paste below):

Student ID: Student Name:



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	樹狀結構能正確保存運算的優先順序，確保計算結果正確
Complete Binary Tree	Binary Heap Data Structure	因為節點緊密排列沒有空隙，可以完美映射到 Array 中儲存，節省記憶體並提升存取效率
Binary Search Tree	Simple Symbol Table	透過「左小右大」的規則，能將搜尋時間縮短，大幅提升效率
AVL Tree	Spell Checkers	AVL Tree 嚴格的高度平衡，確保了樹的高度最低。這提供了比 Red-Black Tree 更快的查詢速度，非常適合用於查詢頻繁但極少修改內容的系統
Red-Black Tree	C++ STL std::map / std::set	在插入與刪除時需要的旋轉次數較少，因此在資料頻繁變動的情況下效能較佳，且能保持資料排序
Max Heap	Emergency Room Triage	系統必須隨時優先處理數值最大病情最嚴重，優先權最高的項目
Min Heap	Navigation Systems	導航需要不斷選取距離最近的節點來計算路徑，Min Heap 最高效

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 的平衡限制較嚴格，樹的高度較低，所以能提供較快的查詢速度，對於 Mostly search (few updates)系統來說，是最有效率的選擇

Frequent insertions and deletions? Why?
Red-Black Tree；相較於 AVL，Red-Black Tree Insert and Delete Nodes 所需的旋轉次數較少，雖然查詢速度較慢，但在資料頻繁變動的系統中，它的綜合效能優於 AVL

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；因為資料是靜態的不需要做 Insert or delete，sorted array 不僅節省空間，且記憶體連續性佳，搭配 binary search 就可提供極高的查詢效率

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 13:20	Gemini	用引導式教學的方式告訴我 section2 是需要完成什麼並逐步教學我幫助我完成
2	12/14 14:34	Gemini	幫我比較 BST, AVL, and Red-Black trees 三種使用情境的差別
3			
4			

You may extend this table as needed.