

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: Composed of multiple nodes with a designated root node. Nodes are connected arbitrarily, but every node must have a path that leads back to the root.

2. Binary Tree

Definition: An extension of the general tree structure where each node is restricted to having at most two child nodes.

3. Complete Binary Tree

Definition: An extension of a binary tree where every level, except possibly the last, is completely filled with the maximum number of nodes, and all nodes in the last level are positioned as far left as possible.

4. Binary Search Tree (BST)

Definition: An extension of a binary tree that satisfies the following property: for any given node, its value must be greater than all node values in its left subtree and less than all node values in its right subtree.

5. AVL Tree

Definition: An extension of a binary tree where the height difference between the left and right subtrees of any node is at most one. It maintains this balance automatically through rotation operations.

6. Red-Black Tree

Definition: An extension of a binary search tree where each node is assigned a color (red or black), and the tree must follow these rules: (1) Each node is either red or black. (2) The root node is always black. (3) All leaf nodes (NIL nodes) are black. (4) Red nodes cannot have red children. (5) Every path from a given node to any of its descendant NIL nodes must contain the same number of black nodes.

7. Max Heap

Definition: An extension of a complete binary tree where the value of each node is greater than or equal to the values of its child nodes, ensuring the root holds the maximum value.

8. Min Heap

Definition: An extension of a complete binary tree where the value of each node is less than or equal to the values of its child nodes, ensuring the root holds the minimum value.

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.

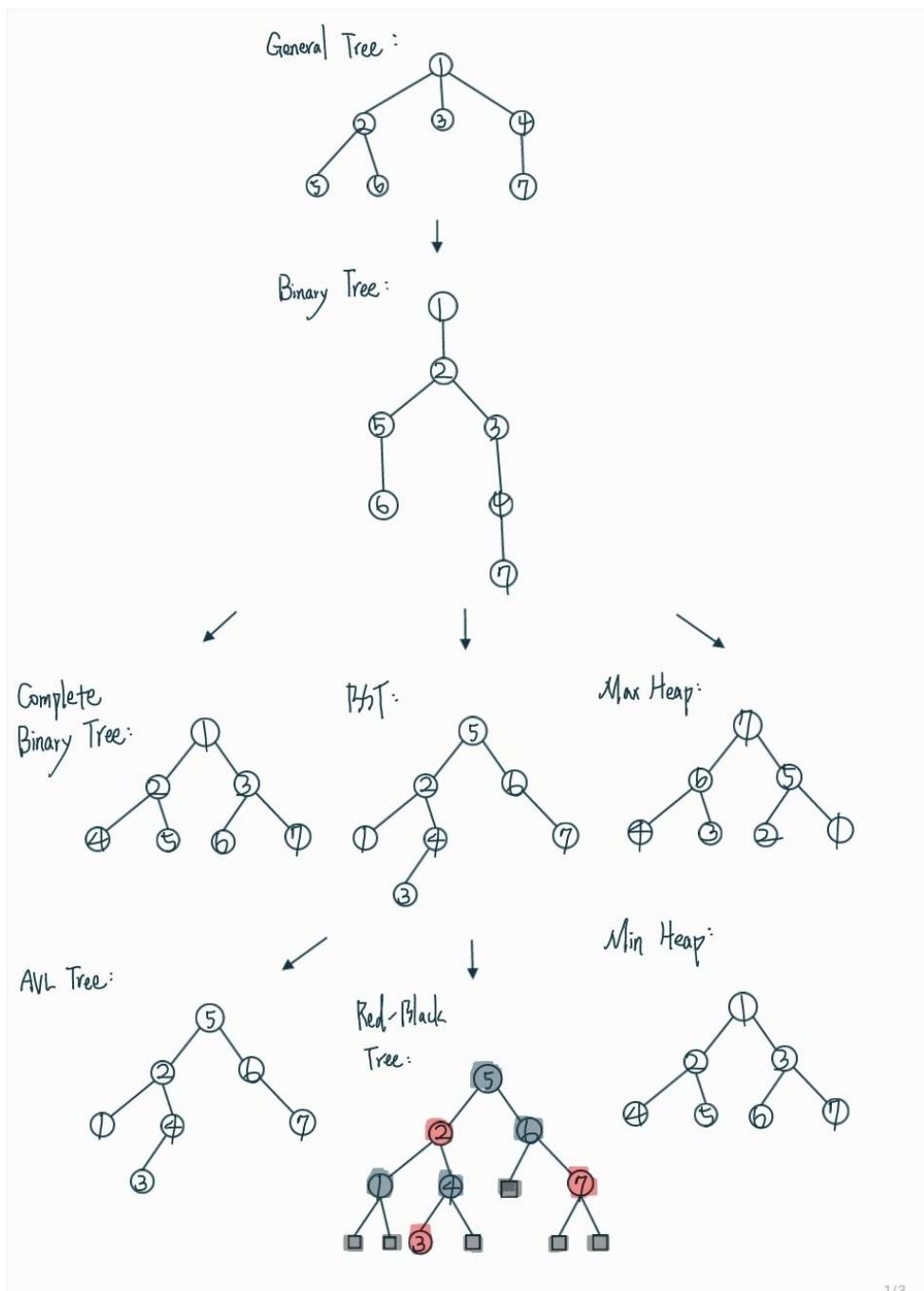
Suggested chain example (you may extend or adjust):

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	Each node can have at most two children.
Binary Tree	Complete Binary Tree	Nodes must be filled level by level from top to bottom, and within each level from left to right; all levels except the last must be completely filled.
Binary Tree	Binary Search Tree	For any node, all values in its left subtree are less than the node's value, and all values in its right subtree are greater than the node's value.
BST	AVL Tree	For every node, the heights of the left and right subtrees differ by at most 1.
BST	Red-Black Tree	Nodes are colored red or black, and a set of coloring rules (e.g., the root is black and no red node has a red child) helps keep the tree approximately balanced.
Binary Tree	Max Heap	Each parent node's value must be greater than or equal to the values of its children, and the tree must maintain the complete binary tree (shape) property.
Binary Tree	Min Heap	Each parent node's value must be less than or equal to the values of its children, and the tree must maintain the complete binary tree (shape) property.

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

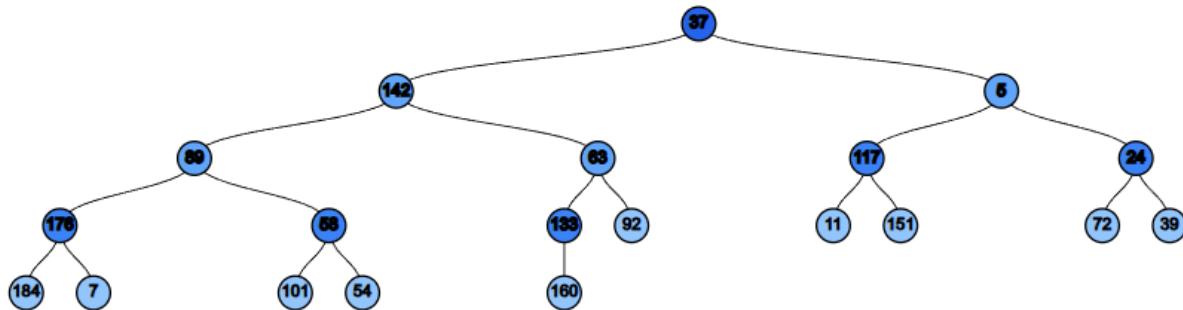
Tool name / URL:

Binary Tree and Graph Visualizer (<https://treeconverter.com>)

Construction / insertion description:

Insert the integers as nodes from left to right in level order. When the current level is full, continue inserting at the next level. Each node can have at most two children.

Screenshot of Binary Tree (paste below):



3.2 Complete Binary Tree

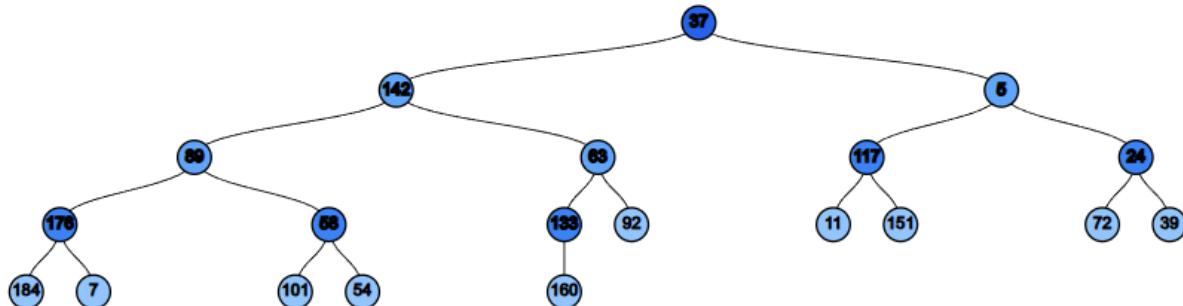
Tool name / URL:

Binary Tree and Graph Visualizer (<https://treeconverter.com>)

Construction / insertion description:

Same as Binary Tree. Because the insertion rule fills nodes from left to right and only moves to the next level after the current level is full, the resulting structure matches the definition of a complete binary tree.

Screenshot of Complete Binary Tree (paste below):



3.3 Binary Search Tree (BST)

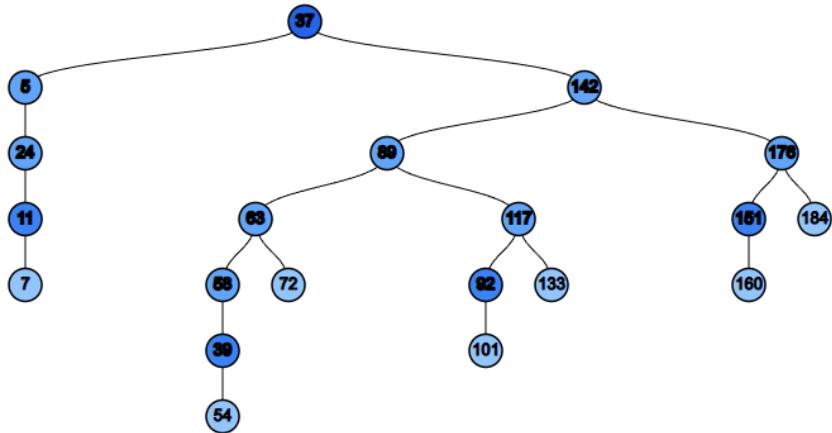
Tool name / URL:

Binary Tree and Graph Visualizer (<https://treeconverter.com>)

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

When inserting a node, start at the root and compare the key with the current node’s value. If the key is greater, move to the right child; if it is smaller, move to the left child. Repeat until reaching a null child position, then insert the new node there as the left or right child according to the comparison.

Screenshot of BST (paste below):



3.4 AVL Tree

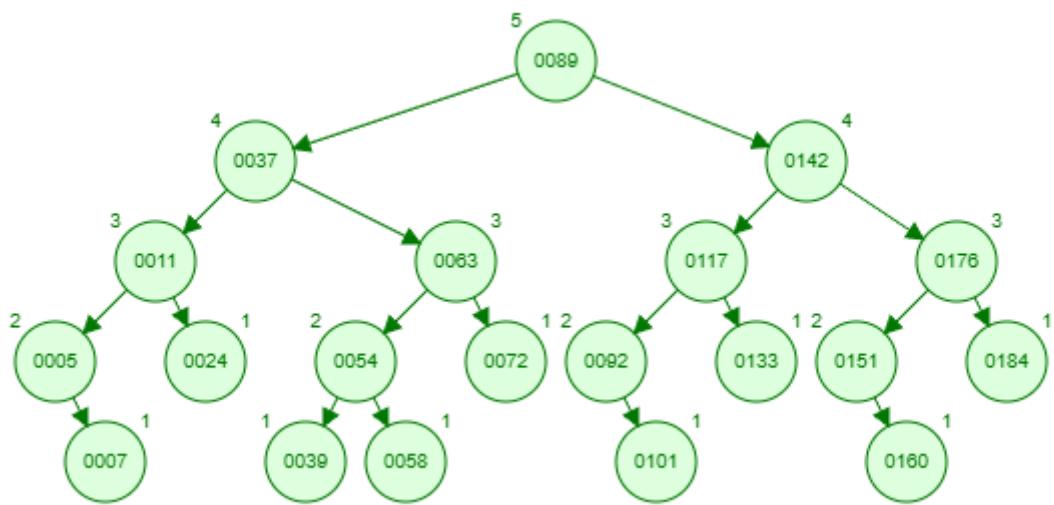
Tool name / URL:

University of San Francisco (<https://www.cs.usfca.edu/~galles/visualization/AVLtree.html>)

Insertion & balancing description:

1. Each integer is initially inserted following standard Binary Search Tree rules
2. After insertion, the algorithm retraces the path back to the root, updating the height of each node and calculating its Balance Factor (BF = Height_Left - Height_Right).
3. If a node's balance factor violates the AVL property, a rotation is triggered immediately to restore balance: Single Rotations (LL/RR) or Double Rotations (LR/RL).

Screenshot of AVL Tree (paste below):



3.5 Red-Black Tree

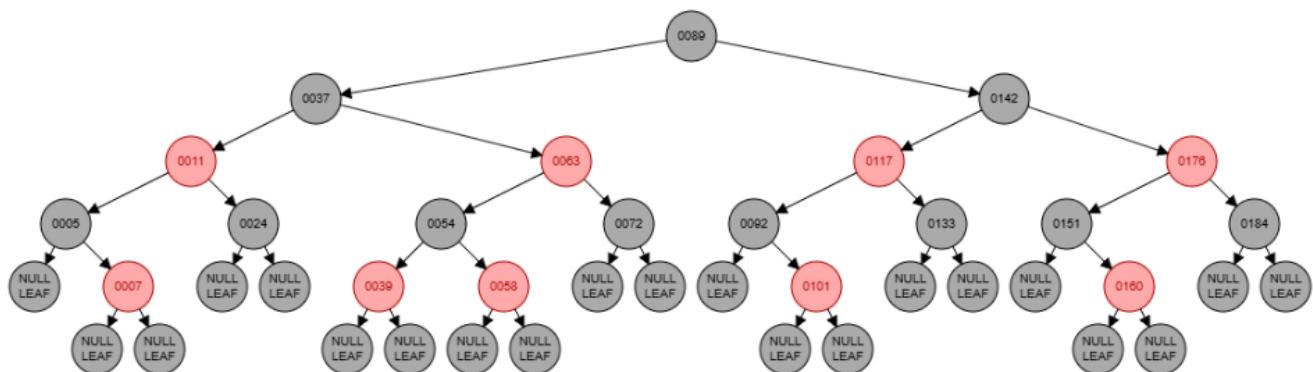
Tool name / URL:

University of San Francisco (<https://www.cs.usfca.edu/~galles/visualization/RedBlack.html>)

Insertion & balancing description:

1. Every new node is inserted following standard BST rules and is initially colored Red.
2. If the parent of the new node is also Red, the algorithm checks the color of the Uncle node (parent's sibling) to decide the fix: (1) Uncle is Red: Perform Recoloring. Flip the colors of the parent, uncle, and grandparent, then propagate the check upwards. (2) Uncle is Black (or NIL): Perform Rotation (and specific recoloring). This structurally rearranges the nodes to resolve the violation immediately.
3. The root of the tree is always re-colored Black at the end of the process.

Screenshot of Red-Black Tree (paste below):



3.6 Max Heap

Tool name / URL:

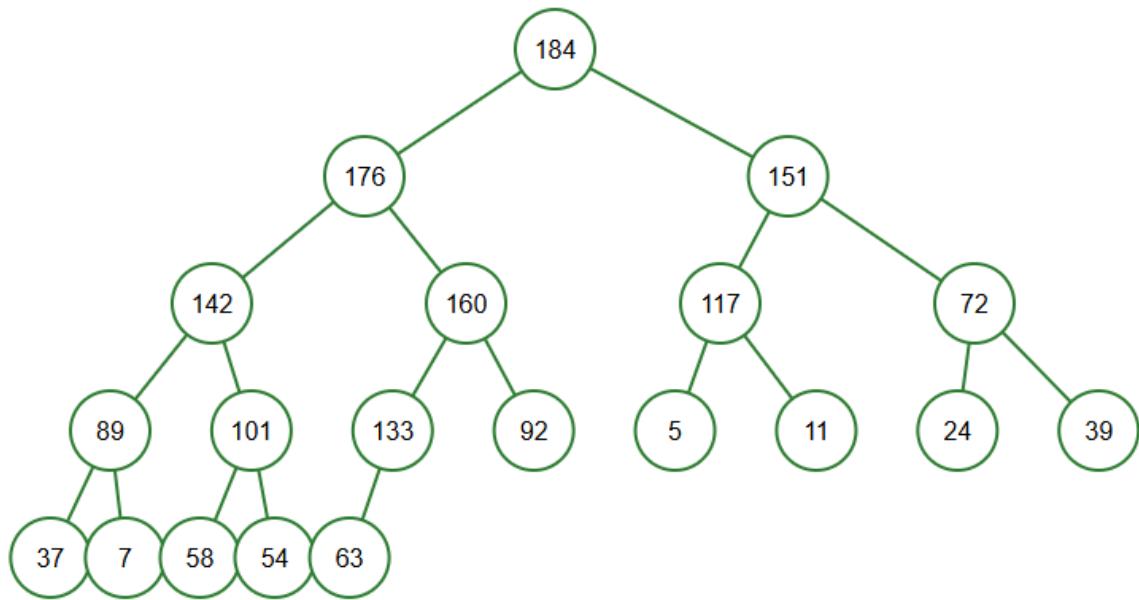
Heap Simulator - Interactive Max Heap Visualization

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

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

Insert the 20 given integers into the heap one by one. For each insertion, first place the new node at the last position of the complete binary tree. Then compare the node with its parent; if the child is greater than the parent, swap them. Repeat this “sift-up” process until the max-heap property is satisfied or the node reaches the root.

Screenshot of Max Heap (paste below):



3.7 Min Heap

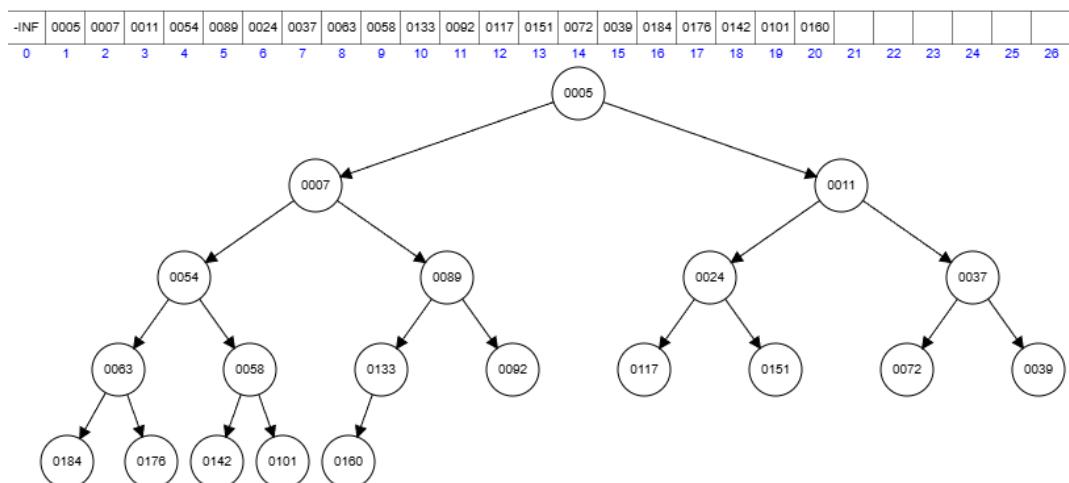
Tool name / URL:

University of San Francisco (<https://www.cs.usfca.edu/~galles/visualization/Heap.html>)

Construction / heap-building description:

Insert the 20 given integers into the heap one by one. For each insertion, first place the new node at the last position of the complete binary tree. Then compare the node with its parent; if the child is smaller than the parent, swap them. Repeat this “sift-up” process until the min-heap property is satisfied or the node reaches the root.

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	Calculator (expression tree)	The parent node stores an operator, and the left/right child nodes store operands, which matches the binary structure of expressions and the evaluation process.
Complete Binary Tree	Array Representation	Because the structure is compact and “complete” (no gaps in level-order filling), implementing it with an array can greatly save memory space.
Binary Search Tree	Search system	It can achieve much faster searching performance than simple linear search.
AVL Tree	Read-only Database	AVL trees enforce strict height balance, which is ideal for scenarios with many searches and few updates.
Red-Black Tree	C++ built-in library: map	Red-black trees have looser balancing requirements, so even with frequent insertions and deletions, operations can be completed in $O(\log n)$.
Max Heap	Task Scheduling	The root node is always the maximum value, so an operating system can retrieve the highest-priority task very quickly in $O(1)$.
Min Heap	Graph algorithms (e.g., Dijkstra)	The algorithm needs to repeatedly find the node with the current shortest distance; in a min-heap, the root is always the minimum, which improves the efficiency of retrieving the minimum-weight node.

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

Since all of them are derived from a BST, a search operation will always follow only one path from the root to a leaf; an AVL tree can guarantee that the height is evenly distributed, so its search efficiency is higher.

Frequent insertions and deletions? Why?

BST

Compared with the other two, it has fewer constraints, so when the tree is updated (e.g., after insertions or deletions), it can be rebalanced in less time.

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

An array only needs to store the data itself and does not require extra pointers like a tree.

Moreover, using a sorted array with binary search can achieve $O(\log n)$ search time, without the need to maintain the structure with complex balancing algorithms.

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	2025/12/30 18:42	Gemini	給我以下這些樹的定義，越豐富越好 (Binary Tree, Complete Binary Tree, Binary Search Tree, AVL Tree, Red-Black Tree, Max Heap, Min Heap)
2	2025/12/30 20:11	Gemini	這二元樹如何變成完全二元 二元搜尋 最大 最小 heap (上傳手繪二元樹圖片)
3	2025/12/30 20:43	Gemini	在以下樹的變化有哪些 New property / constraint added 我會以 (from, to) 的形式告訴你 (General Tree,) (Binary Tree, Complete Binary Tree) (Binary Tree, BST) (BST, AVL Tree) (BST, Red-Black Tree) (Binary Tree, Max Heap) (Binary Tree, Min Heap)
4	2025/12/30 22:06	Gemini	AVL Tree 的 Insertion & balancing description
5	2025/12/30 22:34	Gemini	red black tree 的 Insertion & balancing description

6	2025/12/30 23:02	Gemini	Application Examples and Why (Binary Tree, Complete Binary Tree, Binary Search Tree, AVL Tree, Red-Black Tree, Max Heap, Min Heap)
7	2025/12/31 00:00	Gemini	Among BST, AVL, and Red-Black trees, which one would you pick for: Mostly search (few updates)? Why?
8	2025/12/31 00:11	Gemini	Among BST, AVL, and Red-Black trees, which one would you pick for: Frequent insertions and deletions? Why?
9	2025/12/31 00:24	Gemini	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?