IT415 – Week 6 Binary Trees & BST Toolkit

Reflection

How traversal order changes your view of the tree

Traversal is how the program walks the branches. Inorder (L, Root, R) reads BST values in sorted order, so the structure feels linear. Preorder (Root, L, R) surfaces the shape from the top—great for rebuilds/serialization because you see parent intent first. Postorder (L, R, Root) is bottom-up: children finish before parents, which fits teardown and expression evaluation. Same data, different walk, different story.

What I learned about height and efficiency

Using the edges convention (empty = −1, leaf = 0) keeps the math honest: each parent adds +1 along the longest path. Height caps recursion depth and guides search expectations. Traversals run in O(n) time with O(h) extra space; taller trees mean deeper stacks. For BSTs, average time tracks height: near O(log n) when height stays low, drifting to O(n) as it stretches.

Why balanced vs. skewed insertions matter

Insertion order sets performance. Balanced-ish inserts keep height near $\log_2(n)$, so Insert/Contains stay quick. Sorted inserts make a stick (height ≈ n−1), so work turns linear and the stack deepens. The demo showed it clearly: same numbers, same code,

very different height and speed. If I can’t control order, use a self-balancing tree; if I can, seed a solid root and spread inserts to avoid the linked-list shape.