(p1) Hello, everyone. I'm Hyunjin Lee from Team 7, and I'll be presenting our progress to date. This presentation covers what we've achieved up through the preparation of this PPT.

(p2) Here is the agenda. We will go through each item in order.

(p3) First, the roles we each took on. I set up the initial files based on our agreed -upon class design. Nam-kyung handled code reviews and prepared the PPT materials. Tae-hwan implemented the Find function and created our .clang-format file for consistent formatting. Si-young took charge of the Empty, Height, and Size functions and wrote the license.

(p4) Before building our class structure, we reviewed AVL trees. An AVL tree is a self-balancing binary search tree that limits the height difference between subtrees to ensure that insert, delete, and search operations run in O(log N) time.

(p5) In a normal BST, a skewed tree can degrade search to O(N) time. In an AVL tree, rotations restore balance—keeping the height difference to at most 1—and guarantee O(log N) search time.

(p6) To implement this, we defined two types:

Node type, to store each node's properties (key, height, and pointers to parent, left, and right).

AVL type, to manage the tree structure via its root and provide required operations.

We chose class over struct so we could encapsulate members and expose only getters and setters. The Node class is considered part of the AVL class (a part-whole relationship), not a standalone type.

(p7) Here is the Node class: it includes a constructor, getters and setters, and member variables. (You can ignore the friend class AVL line for now.)

(p8) Here is the AVL class: we defined functions for Empty, Size, Height, Find, and part of Insert.

(p9) In our implementation file, we use direct member access to achieve O(1) time for Size, Empty, and Height operations.

(p10) Next, the Find function. Starting from the root, FindNode recursively compares the target key: if the node is found, it returns its pointer; if not, it returns nullptr, and we interpret that as 0. Then CalculateDepth walks up via parent pointers to compute depth, adding stored height and depth to return the final result.

(p11) (Not shown)

(p12) To maintain consistency, we adopted clang-format with Google's fallback style, which aligns with our CSE3210 style guide for tab size and other basics.

(p13) We formalized our comment style too: function input/output descriptions go at the top of each file so reviewers can quickly grasp logic flow. We chose the MIT license—a permissive license—since this is an academic project exploring AVL behavior, not a commercial product.

(p14) During code reviews, we encountered naming inconsistencies (e.g. current_node) and comment/formatting gaps. After each review, we updated our style guide. We also configured clang-format to align asterisks in pointer declarations on the left for uniformity.
(p15) We established PR rules to streamline reviews:
No PR should exceed 200 lines of changes.
PR titles must specify the code type, the change made, and the responsible author.
(p16) Initially, the PR opener assigned two reviewers: one focused on functionality and comments, the other on style compliance. Now, each author selects one relevant reviewer, and I, as team lead, manage merge-conflict resolution before final merge.
(p17) For testing, we will use WSL and CMake. Each member will test their assigned functions. Detailed plans are still under discussion.
(p18) Remaining schedule:
By July 7: Complete implementation and code reviews.
July 8-14: Run and finalize tests.
July 8-15: Draft the final report.
(p19) Through this project, we've learned the importance of teamwork and collaboration. Although our pace slowed while aligning on style and structure, we now appreciate how critical these processes are for larger projects. We also gained hands-on experience with GitHub version control and peer reviews. Moving forward, we will conduct reviews and tests more systematically to continuously improve our code.
(p20) Thank you for your attention. I'm happy to take any questions.