Slide 1

Administrivia

- (Review minute essay from last time.)
- Reminder: Homework 6 code due Thursday. Also Quiz 6 (topic — recursion).
- Original plan was to have eight homeworks, with Homework 7 asking you to do an alternate implementation of the priority queue and Homework 8 asking you to finish anything about your game that's not already done. Probably not enough to time to do both, so: I'll require one, but not both; do both for extra credit. Due date will be during finals period.

Slide 2

Trees — Mathematical Definition

- One definition —
  - Set of nodes, one called root.
  - Set of edges (directed connections between nodes).
  - Root has no incoming edges; all other nodes have exactly one (from parent).
  - Each node can have 0 or more outgoing edges (to children — if none, leaf node).
- Another, recursive definition — tree is one node connected by edges to 0 or more subtrees.
- This is a general tree — e.g., to represent hierarchy such as filesystem.

Slide 3

Implementing Trees

- Define `Node` data structure, analogous to linked list, with reference to data and references to children (linked list or `Vector` or . . .).
- Easier if number of children is limited to two, and this turns out to be sufficiently useful in practice — "binary tree". Then `Node` consists of pointers to data and left and right subtrees.

Slide 4

Tree Traversals

- For linked lists we defined a way to visit all elements — "iterator". Is there something analogous for trees?
- Well — three orders that are easy to define and implement:
  - Preorder — root first.
  - Postorder — root last.
  - Inorder — leftmost subtree first, then root, then remaining subtrees. (Admittedly a little weird for non-binary trees.)
- Sketch some code for at least one of these.

Slide 5

Sorted Binary Trees (Binary Search Trees)

- Key property — everything in the left subtree is smaller than the root, and everything in the right is bigger.
- Why is this useful? If you want a data structure to hold a collection that will be searched frequently, what are the choices? and how fast is each to search? to modify (insert/remove)? Compare approximate times for arrays (sorted and unsorted), linked lists (sorted and unsorted), sorted binary tree.
- Sketch some code for `add` and `find`. (`remove` is trickier, so we'll just talk about general idea.)

Slide 6

Heaps

- Heap is another tree-based data structure, with two properties:
  - A node is always "bigger than" both its children.
  - Tree is "complete".
- For a priority queue, we want to retrieve the "biggest" thing (for game problem, smallest update time). Does this seem useful?
- Note also that we can store a complete binary tree in an array.
- How to insert and remove? Compare running times.

Slide 7

Minute Essay

- None — quiz.