BST Complexity Analysis

Every BST operation involves one find() call, up to one successorDescendant() call, and O(1) other work

Helper methods both do O(1) work per level so run in O(H) time
↖ height

⇒ All BST operations have worst case O(H) runtimes

Our recursive definitions require O(H) space for call frames, but iterative variants use O(1) space

Balanced BSTs

For a fixed height H, at most how many nodes can a binary tree have?



Lecture 18: Heaps and Priority Queues

CS 2110

October 27, 2026

Today's Learning Outcomes

69. Describe the invariants of a heap and determine whether they are satisfied by a given binary tree.

70. Translate between the tree and array representations of a heap.

71. Implement operations on a heap and determine their time/space complexities.

72. Use a heap to implement a priority queue and analyze its performance.

Motivation: Emergency Room Triage



Patients arrive arbitrarily with a variety of medical issues of different urgencies.

Hospitals have limited bed spaces, and patients must wait for one to become available to be admitted.

Triage = prioritize the most critical patients

The `PriorityQueue<T>` ADT

Like all "ordered collections", `PriorityQueues` support four operations:

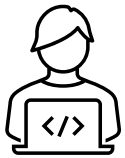
1. **Add another element**

2. **Remove/return "next" element**
   (ADT determines which is "next")

3. **Return "next" element**
   (without removing it)

4. **Check if there are more elements**

Approach 1: Compose with `List`



- Auxiliary `Entry` type to associate priorities with elements
- `add()` at the most efficient index
- Perform linear scan to find highest priority element

Runtime Complexities:

`add()`:

`remove()`:

`peek()`:

`isEmpty()`:

Toward a Better Approach

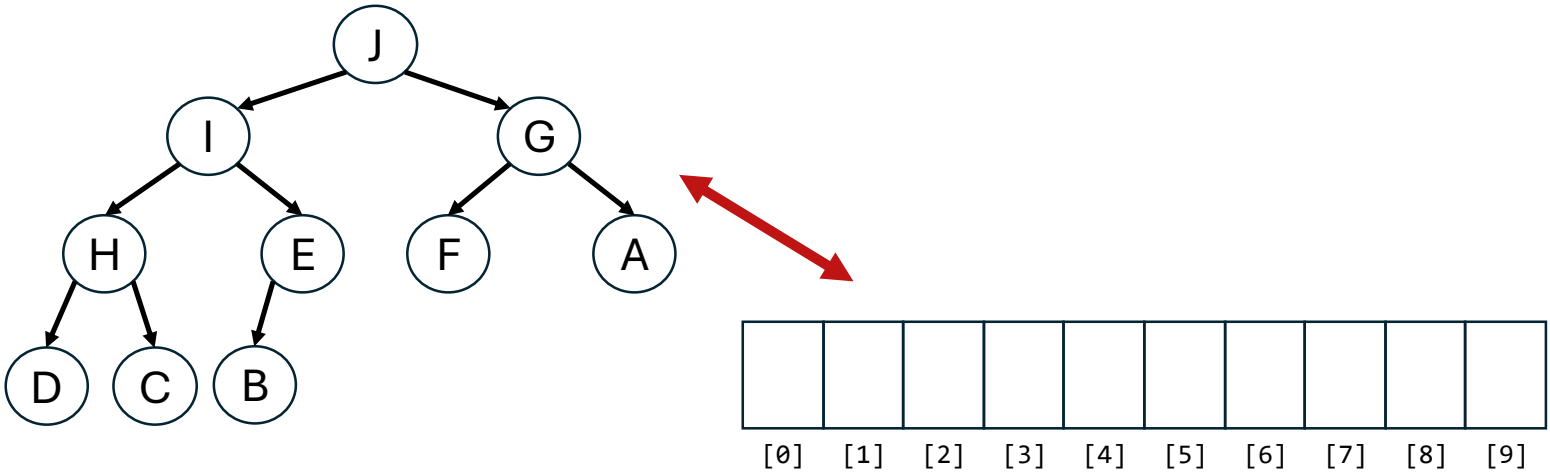
Max Heaps

= Binary trees with `Comparable` elements and two additional invariants:

1. Shape Invariant

2. Heap Order Invariant

Array Representation



For a node at array index `i`:

Left child at _____ Right child at _____ Parent at _____



Coding Demo: `MaxHeap` State



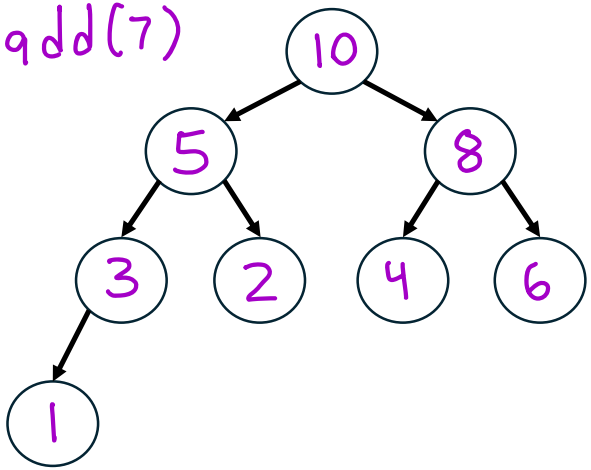
Key Ideas:

- Generic Type Bound
- `ArrayList` backing storage
- `parent()`, `left()`, and `right()` `static` helper methods
- `invariantSatisfied()`
- `size()` and `peek()`

add()ing to a Max Heap

Simplest approach: `add()` to the backing `ArrayList`

Issue: This may violate the heap order invariant



The `bubbleUp()` Method

Exercise: Work with a partner to write the `bubbleUp()` method *iteratively*.

MaxHeap.java

```
/** Swaps the `i`th and `j`th entries in this heap. Requires `0 <= i,j < heap.size()`. */
private void swap(int i, int j) { ... }
/** Returns the index of the parent of the heap entry at index `i`. Requires `i > 0`, */
private static int parent(int i) { ... }

/** Relocates the element initially at index `i` to restore the heap invariant. */
private void bubbleUp(int i) {







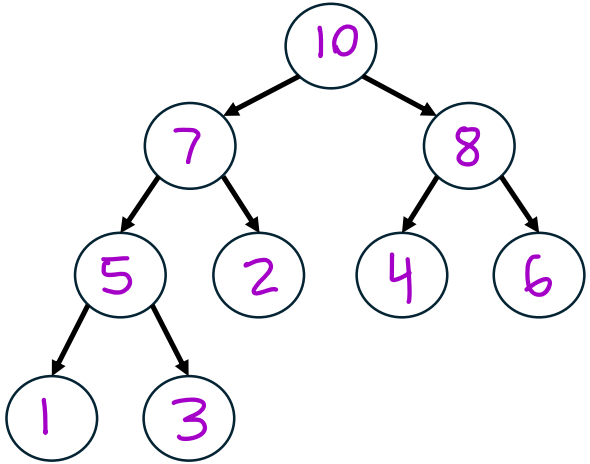


}
```

Max Heap `remove()`

We need to remove/return the first element of the backing `ArrayList`

Issue: Doing this with a `remove(0)` call can "catastrophically break the heap order invariant





Coding Demo: `remove()` and `bubbleDown()`



Max Heap Complexity Analysis

Space Complexity:

Worst-case Runtime Complexity:

`size()`:

`peek()`:

`add()`:

`remove()`:



Coding Demo: MaxHeapPriorityQueue



Key Ideas:

- `Comparable Entry` auxiliary class
  - Define `compareTo()` using priorities
- Composition relationship with `MaxHeap<Entry>`