More About Heaps
Complete Binary Tree As \textit{Array}

- To use an \textit{Array} to represent a complete binary tree, you need more information:
  - An \texttt{int} index of the root of the tree of interest
    - There might be many subtrees inside the tree rooted at index \texttt{0}, and you might want to do something to these subtrees as well as to the whole tree
  - An \texttt{int} index of the last node of the whole tree
    - You need to know which entries in the \textit{Array} contain tree node labels; entries at index \texttt{0} through the last index are useful, but others are “junk”
Note that the **Array** might be larger than the tree *now*, even if at one point they were the same size. How could this happen?
For the gray tree:
\[ \text{top} = 0 \]
\[ \text{last} = 4 \]
For the pink tree:
\[ \text{top} = 1 \]
\[ \text{last} = 4 \]
For the blue tree:

top = 2

last = 4
Note that all the entries in a given subtree (e.g., blue here) are in positions *top* through *last* of the *Array*—but not all entries of the *Array* in that range are in the subtree!
private Comparator<T> machineOrder;
private boolean insertionMode;
private Queue<T> entries;
private Array<T> heap;
private int heapSize;
private Comparator<T> machineOrder;
private boolean insertionMode;
private Queue<T> entries;
private Array<T> heap;
private int heapSize;

This instance variable is set by the constructor, and does not change; it determines the ordering of T values.
private Comparator<T> machineOrder;
private boolean insertionMode;
private Queue<T> entries;
private Array<T> heap;
private int heapSize;

This instance variable records whether this is in insertion mode.
private Comparator<T> machineOrder;
private boolean insertionMode;
private Queue<T> entries;
private Array<T> heap;
private int heapSize;

This instance variable holds the entries of this while it is in insertion mode.
SortingMachine5a Representation

```java
private Comparator<T> machineOrder;
private boolean insertionMode;
private Queue<T> entries;
private Array<T> heap;
private int heapSize;
```

This instance variable holds the entries of this while it is in extraction mode.
private Comparator<T> machineOrder;
private boolean insertionMode;
private Queue<T> entries;
private Array<T> heap;
private int heapSize;

This instance variable is 0 while this is in insertion mode; it holds the number of entries of this while it is in extraction mode.
/**
 * @correspondence
 * if $this.insertionMode then
 * this = (true, $this.machineOrder,
 *      multiset_entries($this.entries))
 * else
 * this = (false, $this.machineOrder,
 *      multiset_entries(
 *      $this.heap.entries[0, $this.heapSize]))
 */
/**
 * @convention
 * if $this.insertionMode then
 *   $this.heapSize = 0
 * else
 *   $this.entries = <> and
 *   |$this.heap.examinableIndices| =
 *     |$this.heap.entries| and
 *   SUBTREE_IS_HEAP($this.heap, 0,
 *     $this.heapSize - 1, [relation computed by
 *     $this.machineOrder.compare method]) and
 *   0 <= $this.heapSize <= |$this.heap.entries|
 */
/**
 * @convention
 * if $this.insertionMode then
 *   $this.heapSize = 0
 * else
 *   $this.entries = <> and
 *   |$this.heap.examinableIndices| =
 *       |$this.heap.entries| and
 *   SUBTREE_IS_HEAP($this.heap, 0,
 *   $this.heapSize - 1, [relation computed by
 *   $this.machineOrder.compare method]) and
 *   0 <= $this.heapSize <= |$this.heap.entries|
 */

This mathematical definition (not shown on slides) means what you should think it does.
Commutative Diagram

\[(\leq, \text{true}, <3, 1>, ?, 0)\]
Commutative Diagram

Does this concrete value satisfy the representation invariant?

\((\leq, \text{true}, <3, 1>, ?, 0)\)
Commutative Diagram

What abstract value does it represent?

(≤, true, <3, 1>, ?, 0)
What does the contract say the abstract result of this method call will be?

\[(\leq, \text{true}, 3, 1, ?, 0)\]
What concrete value represents that abstract value?

$(\leq, \text{true}, <3, 1>, ?, 0)$
Commutative Diagram

What method body would do this (for any valid starting point)?

($\leq$, true, <3, 1>, ?, 0)
Another Useful Pseudo-Contract

/**
 * Reports whether a complete binary tree is a heap.
 * @requires [t is a complete binary tree]
 * @ensures isHeap = [t is a heap]
 */

public static boolean isHeap(BinaryTree<T> t)
{
...}
Implementing `isHeap`

- If $|t| \leq 1$ then $t$ is a heap
- If $|t| > 1$ then $t$ is a heap iff:
  - The root is $\leq$ the root of the left subtree
  - The left subtree is a heap
  - The root is $\leq$ the root of the right subtree (if any)
  - The right subtree (if any) is a heap
Implementing isHeap

- If $|t| \leq 1$ then $t$ is a heap
- If $|t| > 1$ then $t$ is a heap iff:
  - The root is $\leq$ the root of the left subtree
  - The left subtree is a heap
  - The root is $\leq$ the root of the right subtree (if any)
  - The right subtree (if any) is a heap

A smaller subproblem of the same kind; a hint to use recursion.
Example: $|t| > 1$
Example: $|t| > 1$

Is $x \leq y$?
If not, then $t$ is not a heap.
If so, then check whether this subtree is a heap.
Example: $|t| > 1$
Example: $t$

Is $x \leq y$? If not, then $t$ is not a heap. If so, then check whether this subtree is a heap.