Chapter 3, Case Studies:
  ▶ Linear-time sorting algorithms (last week Monday and Wednesday)
  ▶ Disjoint sets and array forests (last week Friday)
  ▶ Priority queues (Monday and Wednesday)
  ▶ $N$-sets and bit vectors (Thursday in lab)
  ▶ (Start graphs Friday)

Today:
  ▶ Recent HW problems
  ▶ Worklist algorithms
  ▶ Priority queue ADT (problem statement)
  ▶ Inefficient solutions
  ▶ Abstractions for the heap data structure
  ▶ Heap implementation details, part 1
  ▶ Excursion: heap sort
  ▶ Heap implementation details, part 2
  ▶ Analysis and optimization
public class ArrayList<E> implements List<E> {
    private E[] internal;
    private int size;
    ...
}

Informal: The positions of internal in range [0, size) contain the elements in the conceptual list in order.

Formal:
(a) $size \leq \text{internal.length}$
(b) $size$ equals the number of calls to insert() and add() minus the number of (non-empty) calls to remove().

Informal: size is the number of items in the conceptual stack, which appear in order of arrival in range [0, size)

Formal:
(a) $size \leq \text{internal.length}$
(b) $size$ equals the number of calls to push() minus the number of (non-empty) calls to pop().
(c) For all $i \in [0, size - 1)$, the item in internal[$i$] was pushed before the item in internal[$i + 1$].
Let $p_{\text{orig}}$ be the original value of $p$. Let $i$ be the number of iterations completed.

(a) $0 \leq p < N$.

(b) $\text{leader}(p) = \text{leader}(p_{\text{orig}})$

(c) $p = p[p_{\text{orig}}] i$ times
An array forest is an example of what kind of data structure?
It contains both array-based and linked concepts.

For each, is it best considered a data structure, abstraction, or abstract data type?
- **Array**: Data structure (simple)
- **Array Forest**: Data structures (hybrid/advanced)
- **Disjoint set**: Abstract data type
- **Forest**: Abstraction

What design pattern was (explicitly) used in the implementation of disjoint sets?
**Strategy**
<table>
<thead>
<tr>
<th></th>
<th>ListPriorityQueue</th>
<th>SortedListPriorityQueue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize empty</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Initialize populated</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n^2)$</td>
</tr>
<tr>
<td>insert</td>
<td>$\Theta(1)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>max</td>
<td>$\Theta(n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>extractMax</td>
<td>$\Theta(n)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>contains</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>increaseKey</td>
<td>$\Theta(1)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>decreaseKey</td>
<td>$\Theta(1)$</td>
<td>$\Theta(n)$</td>
</tr>
</tbody>
</table>
\[ \sum_{i=0}^{h-1} 2^i(h - 1 - i) = (h - 1) \sum_{i=0}^{h-1} 2^i - \sum_{i=0}^{h-1} i2^i \]

\[ = (h - 1)(2^h - 1) - 2 - (h - 2)2^h \]

\[ = h2^h - 2^h - h + 1 - 2 - h2^h + 2 \cdot 2^h \]

\[ = 2^h - h - 1 \]

\[ = 2^{\lg(n+1)} - \lg(n + 1) - 1 \]

\[ = n + 1 - \lg(n + 1) - 1 \]

\[ = n - \lg(n + 1) \]
HeapPriorityQueue
- internal: E[]
- heapSize: int
- compy: Comparator<E>
- add(E)
- decreaseKeyAt(int)
- decrementHeapSize()
- findKey(E)
- get(int)
- heapSize()
- increaseKeyAt(int)
- isEmpty()
- isFull()
- set(int, E)
- swap(int, int)

Heap
- isEmpty()
- insert(E)
- max()
- extractMax()
- contains(E)
- increaseKey(E)
- decreaseKey(E)

PriorityQueue
- <<interface>>
- isEmpty()
- insert(E)
- max()
- extractMax()
- contains(E)
- increaseKey(E)
- decreaseKey(E)

ListPriorityQueue

SortedListPriorityQueue

HeapPriorityQueue
<table>
<thead>
<tr>
<th>Operation</th>
<th>ListPriorityQueue</th>
<th>SortedPriorityQueue</th>
<th>HeapPriorityQueue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize empty</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Initialize populated</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>insert</td>
<td>$\Theta(1)$</td>
<td>$\Theta(n)$</td>
<td>$\Theta(lg\ n)$</td>
</tr>
<tr>
<td>max</td>
<td>$\Theta(n)$</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>extractMax</td>
<td>$\Theta(n)$</td>
<td>$\Theta(1)$</td>
<td>$\Theta(lg\ n)$</td>
</tr>
<tr>
<td>contains</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>increaseKey</td>
<td>$\Theta(1)$</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>decreaseKey</td>
<td>$\Theta(1)$</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
</tr>
</tbody>
</table>
Coming up: (all end-of-day)

Do linear sorting project (due this past Mon, Feb 5)
Do heaps and priority queue project (due next Mon, Feb 12)

Due Wednesday:
Read Section 3.3 (heaps and priority queues)
Take heap/pq quiz

Due Thursday:
Read Section 3.4
Do Exercises 3.(26 & 27).
Take N-sets quiz