Midterm Exam 1
- TUESDAY FEBRUARY 28th, 5:00 pm
- Lecture 1: Room 272 of Bascom Hall
- Lecture 2: Room 125 of Agriculture Hall
- Lecture 3: Room 204 of Educational Sciences Building
- UW ID required
- See posted exam information

Program 2: due 10 pm Sunday, March 5th, TIME IS RUNNING OUT!
Program 3: available next week
Homework 6: no quiz this week!

Last Week: StackADT, QueueADT, array vs chain of nodes implementations, circular array data structure, Tree terms, PriorityQueueADT
   Read: Priority Queue ADT
   - concept
   - operations
   - implementation options

This Week:
(Monday and Tuesday) Exam mechanics and sample questions solution (bring a copy)
   Read: Priority Queue
   PriorityQueueADT review and implementation options
   Java’s Comparable Interface
   Heap Data Structure
   - insert
   - removeMax
   Java’s Stack, Queue, PriorityQueue
   Call Stack Tracing

Next Week
   Read: start Recursion
   Recursion
   - recursion vs. iteration
   - constructing recursive code
   - practice writing recursive code
Recall the PriorityQueue ADT

Concept

A Priority Queue is a general container that stores a collection of items comparable by their priorities with fast access to the item with the highest priority. Priorities are typically integer values where the highest priority can be either the largest or smallest number, and duplicate priorities are allowed.

Operations

```java
// assume largest is highest priority
void insert(Comparable item)
Comparable getMax()
Comparable removeMax()
boolean isEmpty()
```

Issues

Null item – detect then signal with IllegalArgumentException
Empty – detect then signal with EmptyPriorityQueueException

Options for Implementing a Priority Queue ADT (repeat from week 6)

<table>
<thead>
<tr>
<th>data structure</th>
<th>insert</th>
<th>removeMax</th>
</tr>
</thead>
<tbody>
<tr>
<td>unordered array</td>
<td>(O(1)) – add at end and shadow array</td>
<td>(O(N)) – linear search and shift</td>
</tr>
<tr>
<td>ordered array</td>
<td>(O(N)) – linear search and shift or binary search and shift</td>
<td>(O(1)) – remove from end if highest priority at end</td>
</tr>
<tr>
<td>unordered chain of nodes</td>
<td>(O(1)) – insert at head</td>
<td>(O(N)) – linear search</td>
</tr>
<tr>
<td>ordered chain of nodes (SLL w/ head)</td>
<td>(O(N)) – max at end max at start</td>
<td>(O(N)) – must traverse max at start</td>
</tr>
<tr>
<td>HEAP (it’s a tree!)</td>
<td>(O(\log N))</td>
<td>(O(\log N))</td>
</tr>
</tbody>
</table>
Java's Comparable Interface
Implementing a Priority Queue ADT using a Heap

Heap

- min heap
- max heap

Shape Constraint

Ordering Constraint (max)

Implementing Heaps

Max Heap Example:

|   | 56 | 42 | 37 | 38 | 14 | 12 | 26 | 29 | 16 | 8 |

→ Draw the corresponding binary tree:
Inserting into a Max Heap

Algorithm

Given the following max heap:

```
64  52  35  46  17  15  34  12  23  14
```

→ Show the heap after inserting 36:

```
64  52  35  46  17  15  34  12  23  14  36
```

→ Show the heap after inserting 57:

```
64  52  35  46  17  15  34  12  23  14  57
```

Complexity
Inserting into a Max Heap (cont.)

PriorityQueue Class Instance Variables:

```java
private Comparable[] queue;
private int numItems;
```

Pseudo-code

```java
public void insert(Comparable item) {
```
Removing from a Max Heap

Algorithm

Given the following max heap:

\[
\begin{array}{cccccccccccc}
64 & 52 & 57 & 46 & 36 & 35 & 34 & 12 & 23 & 14 & 17 & 15 \\
\end{array}
\]

→ What will the heap look like after doing a removeMax?

\[
\begin{array}{cccccccccccc}
 & & & & & & & & & & & \\
\end{array}
\]

→ What will the heap look like after doing another removeMax?

\[
\begin{array}{cccccccccccc}
 & & & & & & & & & & & \\
\end{array}
\]

Complexity
Java’s Stacks, Queues, PriorityQueues
Call Stack Tracing - Displaying a Singly-Linked Chain of Nodes

Method Call:
   print(head);

Iterative Implementation:
   void print (Listnode<String> curr) {
      while (curr != null) {
         System.out.println(curr.getData());
         curr = curr.getNext();
      }
   }

Recursive Implementation:
   void print(Listnode<String> curr) {
      if (curr == null) return;
      System.out.println(curr.getData());
      print(curr.getNext());
   }

How do these work?