CS 367 - Introduction to Data Structures
Thursday, March 3, 2016

Program 2 due 10 pm Sunday, March 6th, TIME IS RUNNING OUT!
Program 3 assigned Monday?

Last Time

Exam mechanics
Sample questions solution

Today

PriorityQueue Review
Java's Comparable Interface
Heap Data Structure

- insert
- removeMax

Java's Stack, Queues, PriorityQueues
Call Stack Tracing

Next Time

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

```
//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`

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:

→ Show the heap after inserting 57:

Complexity

Inserting into a Max Heap (cont.)

PriorityQueue Class Instance Variables:

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

Pseudo-code

```
public void insert(Comparable item) {
```

Removing from a Max Heap

Algorithm

Given the following max heap:

	64	52	57	46	36	35	34	12	23	14	17	15

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

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

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?