Final Exam

- **Friday 12/14/2018 3:30PM-5:30PM Totman Gym**
- Cumulative, but will emphasize more on the second half of the semester (starting from binary tree)
- Same format as midterm 2:

Fill in the Blanks + Programming Sections

#1

The `Node<T>` class below defines a generic binary tree node. All variables are public, so you can access them directly (e.g. you can directly use `node.data`, `node.left`, or `node.right`)

```
class Node<T extends Comparable<T>> {public T data; public Node<T> left, right; }
```

Now write a **recursive** method to return the **maximum** element of a binary tree given its root node. For example, calling `getMax(root)` will return the maximum element in the tree started at `root`.

- Note that **this is not a BST**, so there is no particular ordering of the tree nodes.
- If you declare **any new variables**, they **must be of type `T`** (no other types allowed).
- You must call `getMax` **recursively**. **No more than 12 lines of code**. 1 point deduction for each extra line.

```
public T getMax(Node<T> node) {
if(node==null) return null;
```

#1 Solution

```
public T getMax(Node<T> node) {
if(node==null) return null;
T lmax = getMax(node.left);
  T rmax = getMax(node.right);
  T currmax = node.data;
  if(lmax!=null && lmax.compareTo(currmax)>0)
currmax = lmax;
  if(rmax!=null && rmax.compareTo(currmax)>0)
currmax = rmax;
  return currmax;
```

Summary of What We've Learned

- **Fundamental Storage Data Structures**
  - Arrays
  - Linked Lists
  - Trees
- **High-Level, Abstract Data Structures**
  - List (sorted, unsorted), Stack, Queue
  - BST (Binary Search Tree)
  - Heap, Priority Queue
  - Graph
  - Hash Table

Comparison

data structure	add	search	remove
unsorted array	O(1)	O(N)	O(N)
sorted array	O(N)	O(log N)	O(N)
BST (balanced / worst)	O(log N) / O(N)	O(log N) / O(N)	O(log N) / O(N)
heap	O(log N)	-	O(log N)
hash table	O(1)	O(1)	O(1)

Summary of What We've Learned

- **Fundamental Algorithms**
  - Recursion (three conditions of recursion)
  - Sorting (simple sorting, merge sort, heap sort, quick sort)
  - Graph Search (BFS and DFS)
  - Big-O Notation and Algorithm Analysis
- **Java-Specific Knowledge**
  - Classes, Interfaces, Exceptions, Generics, Iterators
  - Java provides implementations of many data structures we've learned (ArrayList, Stack, Queue…)

How to Prepare

- Study Lecture **Slides**
  - Carefully review all code in lecture slides.
  - Make sure you understand all the **clicker** questions
- Study the two **Midterms** and the **Practice Final** Exams.
- Study lab materials

#1 Advice PRACTICE!!!! Get a friend, do some problems on paper!!!

Second Half of the Semester

- **Binary Tree and BST**
  - Data structure of a binary tree
  - Binary tree terminologies
  - What's a full tree? What's a complete tree?
  - Tree traversals (in-order, post-order, pre-order)
  - What is a BST? What are its properties?
  - BST and in-order traversal.
  - How to perform search, insertion, deletion from BST? What is the cost of each of them? Note that it's `O(h)` and that's not guaranteed to be `O(log N)` yet!

Second Half of the Semester

- **More on Binary Tree and BST**
  - What do (in-order) predecessor and successor mean? Can you write down code to find them.
  - How to create a balanced BST from a sorted array?
  - What is a self-balancing BST? What guarantees does it make?
  - Scapegoat tree: how does it balance the BST? What happens during insertion and deletion?
  - How to store binary tree in an array? Given a node at index i, where are its two children, and its parent?

Second Half of the Semester

- **Heap and Priority Queue**
  - What is priority queue? How is it different from a standard queue?
  - What is a heap? What are its properties? How is it different from a BST? How is a heap stored?
  - How to perform enqueue and dequeue in a heap?
    - Understand bubbleUp and bubbleDown, and be able to write down code for them.
  - What are the costs of enqueue and dequeue with a heap? How does this compare to using sorted array?

Second Half of the Semester

- **Graphs**
  - Graph terminologies
  - Directed / Undirected, Connected / Non-connected, Weighted / Unweighted
  - How is a graph typically stored?
  - Graph traversal
    - Depth-First Search (DFS): implementation
    - Breadth-First Search (BFS): implementation
    - Shortest-distance in a weighted graph (uniform cost search)

Second Half of the Semester

- **Simple Sorting (quadratic cost)**
  - Bubble Sort
  - Selection Sort
  - Insertion Sort
- **Advanced Sorting (log linear cost)**
  - Merge Sort and the `merge()` operation
  - Quick Sort and the `partition()` operation
  - Heap Sort and the `heapify()` operation
  - Among them, Quick Sort does not guarantee log linear cost in the worst-case scenario!

Comparison

Sort	Best Case	Average Case	Worst Case
Selection sort	$O(N^2)$	$\mathbf{O(N^2)}$	$O(N^2)$
Bubble Sort	$O(N^2)$	$\mathbf{O(N^2)}$	$O(N^2)$
Insertion sort	$O(N)$	$\mathbf{O(N^2)}$	$O(N^2)$
Merge Sort	$O(N \log_2 N)$	$\mathbf{O(N \log_2 N)}$	$O(N \log_2 N)$
Quick Sort	$O(N \log_2 N)$	$\mathbf{O(N \log_2 N)}$	$\mathbf{O(N^2)}$
Heap Sort	$O(N \log_2 N)$	$\mathbf{O(N \log_2 N)}$	$O(N \log_2 N)$

Second Half of the Semester

- **Hash Table**
  - What is a hashing function and what is a hash table?
  - What are the advantages / disadvantages of hash table compared to other data structures?
  - Properties of the modulo (%) operator. It should be obvious that `(x + k*n) % n = (x % n)`
  - What is collision? How to handle collision?
    - Open addressing: linear, quadratic probing, double hashing
    - Separate chaining

Prepare for Final Exam

- Study the questions you have (clickers, practice exams, midterms). Use these as examples and think about variations / generalizations.

- This is also how we as instructors design exams too: we look at what worked before what didn't work, and make variations accordingly. In a way, you have to learn to predict what's going to appear in the exam.

- The exam will never test you on exactly the same questions as you've seen before. But the basic building blocks remain the same.

Prepare for Final Exam

- For example, you've learned how to find the largest element in an array. How about finding the largest element in a BST, or a tree that's not BST?

- You've learned to implement `merge()` of two sorted arrays, think about what if you are asked to implement `merge()` of two sorted linked list?

- You've done a loop version of algorithm blah, do you know how to write a recursive version of it?

#2

Complete the following class. Do **NOT** add new class or instance variables. The code you write must be self-contained without relying on any method or object that isn't shown here.

You should not assume that elements in the heap array beyond the size are set to null. Please use the size instead.

```
public class MaxQueue<T extends Comparable<T>> {
  private int size = 0;
  private T heap[];
```

Write a **recursive** reheapDown method. Repeat: do not use any other method from typical heap implementations.

```
protected void reheapDown(int i) {


}
```

#2 Solution

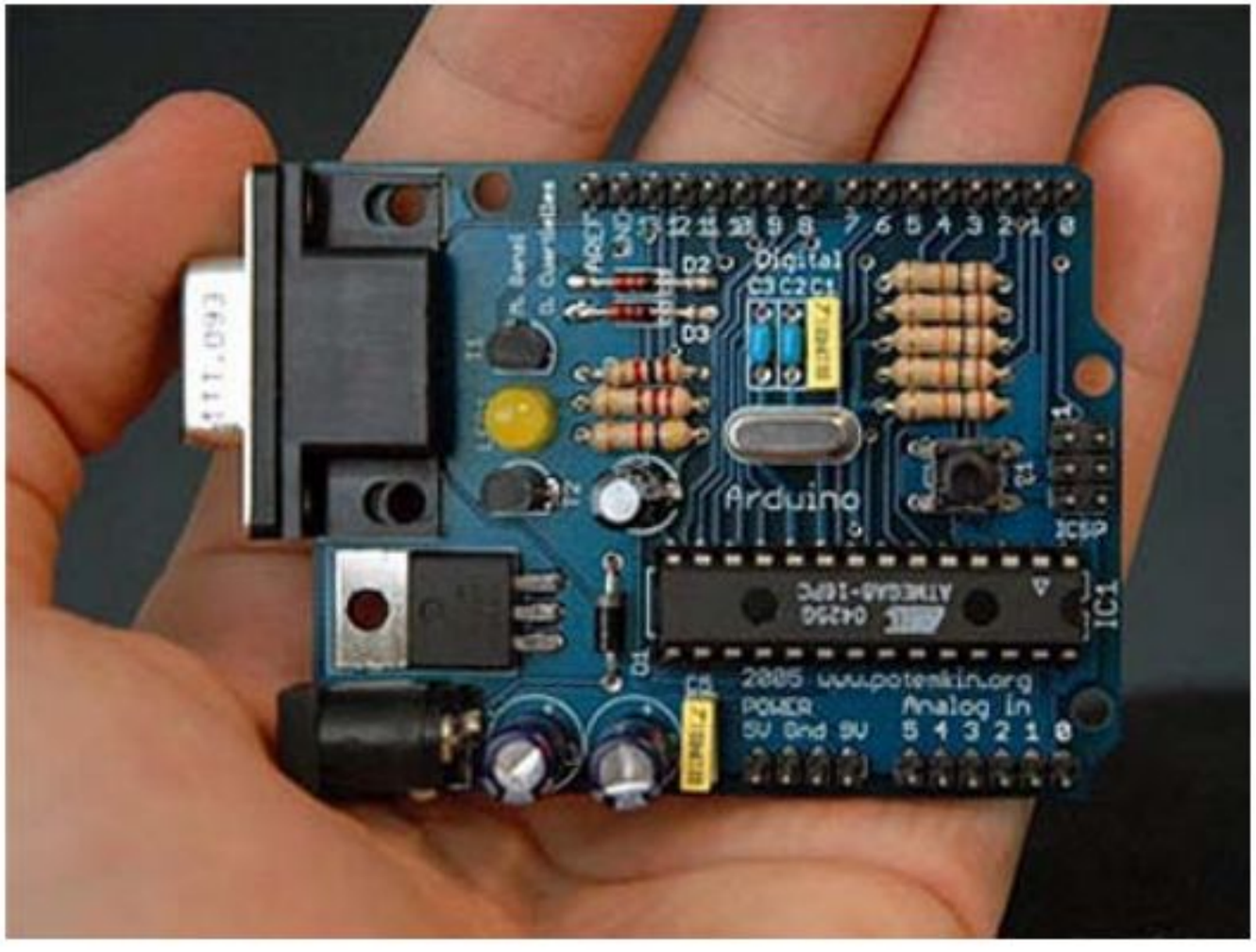
```
protected void reheapDown(int i) {

  int leftChild = 2 * i + 1;
  int rightChild = 2 * i + 2;

  if (leftChild >= size)
    return;

  int larger = leftChild;

  if ((rightChild < size) && heap[rightChild].compareTo(heap[leftChild]) > 0)
    larger = rightChild;

  if (heap[i].compareTo(heap[larger]) < 0) {
    T temp = heap[larger];
    heap[larger] = heap[i];
    heap[i] = temp;
    reheapDown(larger);
  }
}
```

Sample Questions

- Implement binary search in a sorted array (both loop version and recursive version).
- Print out binary tree nodes in in-, post-, pre- order traversal.
- Build a balanced BST from sorted array.
- Search / Insertion in a BST
- Find the predecessor / successor of any node in a BST
- Implement bubbleUp / bubbleDown
- Check if a binary tree is a heap or not; is a BST or not.
- Print out vertices in a graph in DFS and BFS order
- Implement Bubble / Selection / Insertion sort
- Implement `merge()`, `partition()`, `heapify()` methods
- *Important to understand them, instead of just memorize them!*

Now a Fun Topic

- **Physical Computing and Microcontroller Programming**



What's Physical Computing?

- Computation that involves **sensing and responding** to the physical world — in other words, involving physical objects.
- Why is this relevant?
  - Increasingly, computing is not all about the digital world alone, but it's also about the interactions between the digital world and the physical world.
  - You can get to apply your programming skills to make something tangible, useful, and beautiful!

What’s Physical Computing?

- To give you a few examples:
  - Traveler’s Umbrella Interactions
  - Lego Rubik’s Cube Solver
  - Wooden Mirror

What Does Physical Computing Involve?

- Electronic Components
  - Visual/Audio: LEDs, buzzers, speaker.
  - Motion: Servos, motors
  - Sensors
- Physical Components
- Microcontrollers (the brain)
- Most important of all: your **programming** skills!

What is a Microcontroller?

- A tiny, single-chip computer
- Input/output pins to directly talk to electronic components (LED, servos, sensors).
- Serial communication (transfer data between it and a computer through USB port)
- Low power consumption
- Microcontrollers are everywhere in our life:
  - Calculators, printers, electronic toys, all kinds of home appliances
  - They are the brains of modern gadgets.

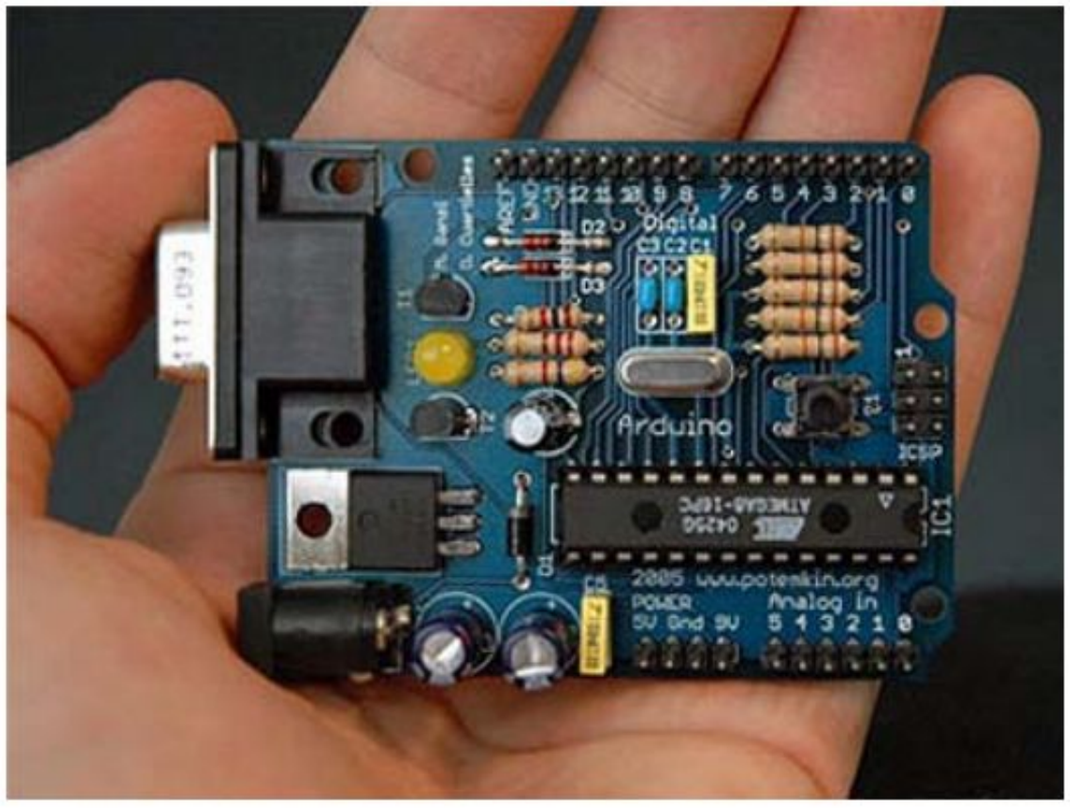
What is a Microcontroller?

- Modern micro controllers can be programmed with high-level programming language, like Java or C++!
- Toaster oven (or generally thermostat) example: heat at a desired temperature for a desired amount of time.

```
  start_time = get_time();
  while (get_time() – start_time < heating_time) {
  if (get_temperature() < desired_temperature - 15)
    turn_on_heating_element();
  else if (get_temperature() > desired_temperature + 15)
    turn_off_heating element();
}
```

What is a Microcontroller?

- Most popular in the hobby / open-source community: **Arduino** (arduino.cc)
  - 16MHz CPU
  - 32KB flash memory
  - 2KB RAM (main memory)
  - 14 input / output pins
  - USB port
  - Java/C++ language
- Also check out Raspberry Pi



Microcontroller Demos

- Blink LED
- Blink an LED pattern
- Control LED using a button
- Put **a lot of LEDs** and buttons together, what do you get?
    - A game controller!
- Now see a bigger LED array
- A more colorful LED array
- Control wirelessly using a cell phone

Microcontroller Demos

- This summer, think about making a cool project
  - A simple line chasing robot
  - Remote controlled door lock, garage door opener, sprinkler controller, thermostat
  - A temperature logger
  - Automatic cat feeder, chicken coop
  - What do you want to build?
- **arduino.cc**

Concluding Remarks

- With microcontrollers, you can apply the programming skills you've learned to make a gadget that's tangible, useful, practical, and beautiful!

- The whole point is to provide you with a broader and refreshing view of **what computing and programming are about,** and what they can do.

- Computing is more than just staring at a computer screen, it can be artistic, entertaining, and fun. It is and will continue to be an integral part of our lives.