1-10-2018

Data Structures Lecture Notes (Student Version)

Kyle Burke
Plymouth State University, paithanq@gmail.com

CS 2381: Data Structures*

Lecture Notes - Student Version†

Kyle Burke

January 10, 2018

Abstract

Lecture notes for a data-structures course in computer science with examples in Java. Students in this course should have already taken an intro-programming course in an object-oriented language and have a basic grasp of Java[1]. These notes are not designed to accompany any specific textbook.

Contents

*Kyle would always like to hear about how useful his notes are. If you have any comments about these, please email him at `paithanq@gmail.com`.

†Created with lectureNotes.sty, which is available at: `http://turing.plymouth.edu/~kgb1013/lectureNotesLatexStyle.php` (or, GitHub: `https://github.com/paithan/LaTeX-LectureNotes`). Many or most of the answers to questions are hidden so that some of class will still be a challenge for students.

[1]A Python-to-Java tutorial is available at: `http://turing.plymouth.edu/~kgb1013/pythonToJavaTutorial/start.php`

⟨ Go over syllabus! ⟩

Talk about plan for projects; go over Project 0. (This looked like it would take longer than expected...)

0 Basic OO Java

In order to move forward, I need to expect that you are somewhat comfortable with the following:

- Compiling and running Java programs
- Java types
- Basic Java syntax (statements, methods, recursion, loops)
- Creating Java classes and objects

Some things I'll give an overview of:

- Javadoc
- Generic Types
- Inheritance
- Polymorphism
- Abstract Classes/Interfaces

0.0 Javadoc

Note: I've been skipping this because they do it in the lab.

You only need to Javadoc public members. Members:

- fields (instance variables)
- methods
- classes

Q:	What is the difference between Public and Private methods?
A:	

Q: What about public/private instance variables (fields)?

⟨ Explain Javadoc:

- Field
- Void method with parameters.
- Fruitful method with no params.
- Fruitful method with params.
- Class

⟩

Q: What should you do with private members?

A:

Q: Why?

A:

0.1 Inheritance

Q: What does it mean for class A to inherit from class B?

A:

Q: If I have a class, Fruit, and want to write a new class, Melon, in Java, how do I say that Melon inherits Fruit?

A:

Q: What if I want to add a class, `Cantaloupe`, that extends `Melon`?

A:

Q: What if `Melon` has a `slice` method? How could this make `Cantaloupe` easier to code up?

A:

Q: Would this compile and run? (The `...` signifies some sort of parameters. I don't know what they are off-hand.)

```
Cantaloupe cantaloupe = new Cantaloupe(···);
cantaloupe.slice();
```

A:

Q: Which of these (if any) are legal?

- `Cantaloupe c = new Melon(···);`
- `Melon m = new Cantaloupe(···);`

A:

Q: Can we also implement `slice` in `Cantaloupe`?

A:

0.2 Polymorphism

Q: Why would we do that?

A:

Q: If I do that, which Class's `slice` method is called here?
```
Cantaloupe c = new Cantaloupe(···);
c.slice();
```

A:

Q: What about here? `Melon m = new Melon(···);`
```
m.slice();
```

A:

Q: What about here? `Melon m = new Cantaloupe(···);`
```
m.slice();
```

A:

0.2.1 Polymorphism in Loops

⟨ Draw a class diagram with `Car` as the super class and `Van`, `Sedan`, and `Truck` all as subclasses of `Car`. ⟩

Maybe `Car` has an `openAllDoors()` method:

```
public void openAllDoors() {
    //opens four doors
}
```


Q: Should any of the subclasses override `openAllDoors`?

A:

Let's write some code to create some cars and add them to an `ArrayList`!

```
Van van0 = new Van(...);
Sedan sedan = new Sedan(...);
Truck truck = new Truck(...);
Van van1 = new Van(...);
ArrayList<Car> cars = new ArrayList<Car>();
cars.add(van0);
cars.add(sedan);
cars.add(truck);
cars.add(van1);
for (int i = 0; i < cars.size(); i++) {
    Car car = cars.get(i);
    car.openAllDoors();
}
```

Q: When `i` is 0, which class's `openAllDoors` method gets called in the for loop?

A:

Q: Why?

A:

Q: What about the others?

A:

0.2.2 Application: `equals`

Let's consider the `String` class and the `equals` method.

Q: What would be printed by the following code?

```
int x = 5;
int y = 5;
System.out.println(x == y);
```

A:

Q: What about the following?

```
int x = new String("Hi");
int y = new String("Hi");
System.out.println(x == y);
```

A:

Q: Why?

A:

⟨ Draw picture of the heap with the pointers. ⟩

Q: What about the following?

```
int x = new String("Hi");
int y = x;
System.out.println(x == y);
```

A:

Q: Why?

A:

Q: What should we do instead if we want to test equivalence?

A:

Q: What is the "highest" class that has an equals method?

A:

⟨ Draw class diagram with `Object` and `String` (as subclass) ⟩

Q: What does the body of that look like?

A:

Q: But what happens if I call the following?

```
int x = new String("Hi");
int y = new String("Hi");
System.out.println(x.equals(y));
```

A:

Q: Why does this work?

A:

Q: What must the signature of that method look like?

A:

Q: Why `Object other`?

A:

Q: So how do we compare?

A:

Q: What will that equals look like?

A:

0.3 Abstract classes

Q: CombinatorialGame has the following class signature: `public abstract class CombinatorialGame {`. What does abstract mean?

A:

Could have `Fruit`, from above, be abstract. There might still be multiple subclasses.

⟨ Draw class diagram: Melon, Berry, Banana, all as subclasses of Fruit. ⟩

Q: What else can I do with an abstract class?

A:

```
public abstract void split();
```

Q: What's the point of doing that?

A:

Q: Can you have some non-abstract methods in an abstract class?

A:

Q: Why would you do this?

A:

0.4 Interfaces

(Skip this?)

Q: What if all methods you defined in an abstract class were abstract?

A:

```
public interface Peelable {
    public void peel();
}
...
public class Banana extends Fruit implements
Peelable {
```

Q: What is the benefit of interfaces?

A:

⟨ Draw a picture with Banana, Watermelon, Fruit and Peelable. ⟩

0.5 Generic Types

Q: Why is this useful?

A:


```
 ArrayList<Peelable> fruitBasket = new
ArrayList<Peelable>();
fruitBasket.add(new Banana());
fruitBasket.add(new Orange());
fruitBasket.add(new Banana());
fruitBasket.add(new Cucumber());
for (Peelable food :  fruitBasket) {
    food.peel();
}
```

Q: Why not just peel them after we create them?

A:

```
 ArrayList<Peelable> fruitBasket =
florist.getFruitBasket();
for (Peelable food :  fruitBasket) {
    food.peel();
}
```

Would this compile?

Q:

```
 ArrayList<Fruit> fruitBasket =
florist.getFruitBasket();
for (Fruit fruit :  fruitBasket) {
    fruit.peel();
}
```

A:

Q: What would we have to change to get it to compile?

A:

Q: ⟨ Draw Cheese in class box. ⟩ What are some possible types of Cheese?

⟨ Draw three of their responses as subtypes of cheese. ⟩

Q: Let's say I had a class that allowed me to create a `SnootyMouse` object that liked a specific kind of `Cheese`. What would the statement creating that object look like?

A:

Q: What would the class signature for `SnootyMouse` look like?

A:

1 Arrays

⟨ Play Elephants and Rhinos with the class ⟩

Q: Which data structures do you already know about?

A:

1.0 Array Basics

Q: What's great about arrays?

A:

Q: Let's consider writing a method, `hasMonkey`, with the following signature:
```
public boolean hasMonkey(String[] strings) {
```
What do you think this method does?

A:

Q: What might the Javadoc look like for `hasMonkey`?

A:

Q: Write the body of `hasMonkey`, a method that determines whether `"monkey"` is in `strings`, an array of Strings

A:

Q: Let's say we have an array of n strings. How many statements are executed to run that code?

A:

1.1 Time Complexity

Q: Often, we're going to focus on the worst case. What's the worst case here?

A:

Q: Does this mean that this method will always take $(3n+2)\times k$ where k is the time to execute one statement?

A:

Q: Which factors can affect the running time?

A:

We're going to be a bit more abstract to "measure" running time.

- First, keep only biggest term: $3n + 3 \rightarrow 3n$
- Next, drop coefficients: $3n \rightarrow n$
- Keep: exponents!

This is called "Big-Theta notation". $\Theta(3n + 3) = \Theta(n)$

We say: Takes "Big-Theta" of n steps to run the method. We call this the *Time Complexity* of that algorithm.[2]

Q: How do you add something to the front of an array in Java?

A:

Q: Why do you have to create a new array?

A:

[2]We are using Θ instead of O (Big-O) because we're talking specifically about the running time of code instead of the solvability of the problem. If you hear someone else use the term Big-O, 95% of the time they're talking about the same thing as Big-Theta. The actual difference is covered in an Analysis of Algorithms class, which you can take after this one.

Q: Try to write some code to do that. Signature:
`pub stat int[] appendToFront(int number, int[] numbers)`

A:

Q: How long does it take to run that function?

A:

I have included a figure with a bunch of the different Θ classes in Figure 1.

Q: How did you know that without writing out all the stuff?

A:

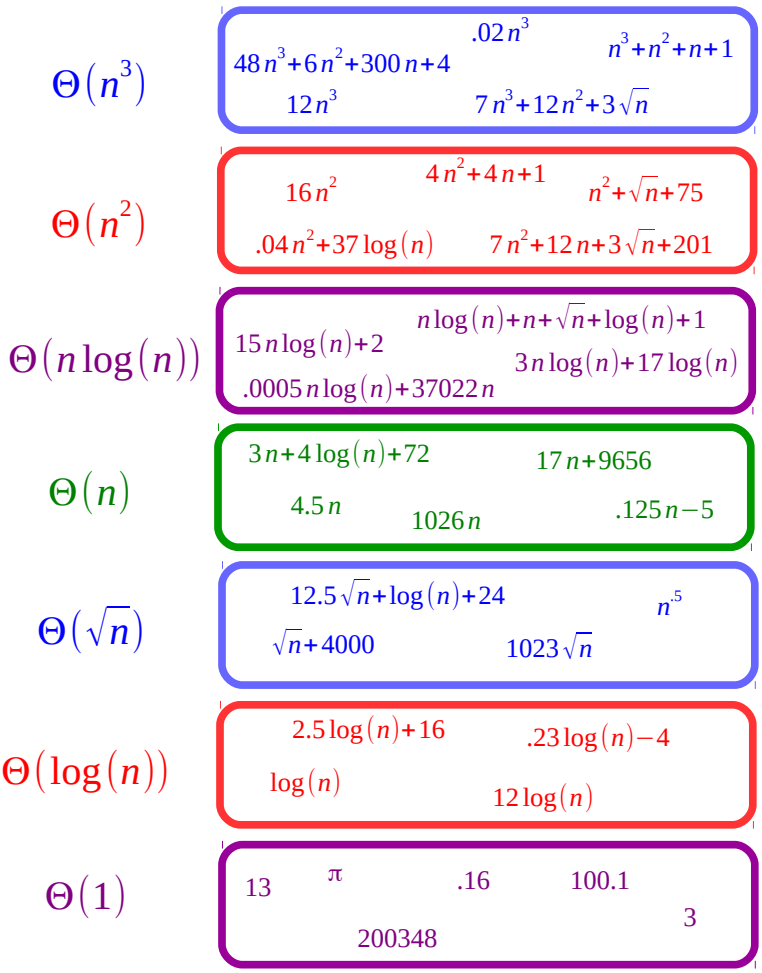


Figure 1: Many common Θ classes and some example functions for each.

Q: What about to remove the first element of an array? (Shift all elements down by 1.) What's the time complexity of that?

A:

Q: What should we do to the last element?

A:

Q: If we just set it to null, how long would it take to add a new value to the end?

A:

Consider the following method:

```
pub stat int biggestDifference(int[] numbers) {
    System.out.println(''Outside of all
loops.'');
    int maxDiff = 0;
    for (int i = 0; i < numbers.length; i++) {
        System.out.println(''Inside one loop.'');
        for (int j = 0; j < numbers.length; j++)
{
            System.out.println(''Inside both
loops.'');
            int diff = Math.abs(numbers[i] -
numbers[j]);
            if (diff > maxDiff) {
                maxDiff = diff;
            }
        }
    }
    return maxDiff;
}
```


Q: What is this code returning?

A:

Q: How many times does the program print each of the print statements? (Assume array size of n, which is common.)

A:

Q: What is the running time of `biggestDifference`?

A:

Q: How long does it take to execute `printSquare(n)`, which prints n lines, each of which is: `1, 2, 3, ..., n-1, n`? (Hint: it might be helpful to write out the body of the method.)

A:

1.2 ArrayLists

Q: How different are these questions for the `ArrayList` class?

⟨ Direct class to visit `http://docs.oracle.com/javase/6/docs/api/java/util/ArrayList.html` Second paragraph, second sentence: "The add operation runs in amortized constant time, that is, adding n elements requires $\Theta(n)$ time." ⟩

Q: How does that work? (Hard!)

Example: Increasing ArrayList length

Say we have an `ArrayList` with 8 elements. This object keeps track of an array and an int for the size. Let's assume that the length of that array is also 8.

Q: A new element is added. What does the `ArrayList` do?

A:

Q: What's the time complexity of that?

A:

Q: I left something out! How big is the new underlying array?

A:

Q: What happens the next time something is added?

A:

Q: What's the time complexity of that?

A:

Q: How big should I make the new array whenever I have to resize?

A:

Although it does take $\Theta(n)$ steps every so often, if I consider that happening only when I reach new powers of 2, it won't take more than $\Theta(n)$ steps to add n elements. Thus, the average time to add one is $\Theta(1)$.

Operation	Array	ArrayList
add	$\Theta(n)$	Often: $\Theta(1)$. Seldom: $\Theta(n)$
get	$\Theta(1)$	$\Theta(1)$
remove	$\Theta(n)$	$\Theta(n)$
set	$\Theta(1)$	$\Theta(1)$

2 Linked Lists

Another way to implement a list is by wrapping each element in it's own object: *linked list*.

2.0 Linked List Basics

Consider a class `LinkedList<T>` which has two fields: `T value` and `LinkedList<T> tail`.

Q: Can I build a data structure out of these?

A:

⟨ Draw a chain of nodes with `String`s from the following code. ⟩

```
LinkedList<String> stringList = new
LinkedList<String>(``Blastoise'');
stringList.add(``Banana'');
stringList.add(``Machamp'');
System.out.println(stringList);
```

Q: What is the `tail` field for the node with value ``Machamp''?

A:

Q: Consider a `setTail` method that resets the tail field to a provided Linked List. What does the list look like after calling: `stringList.setTail(new LinkedList<String>(‘‘Banana’’));`?

A:

Q: Write some code so that another Linked List, `strings` looks like: apple → monkey → banana.

A:

Q: How does the add method work?

A:

Oh no! Recursion! Terrible! Actually, not that bad.

2.1 Recursion on Linked Lists

Let's try to write a method that returns the last element from a linked list. Twist: write it for the actual class!
Header: `public T getLast()`

⟨ Get out the tail-arrows. Have the students line up in a chain and let them choose their own Pokémon names as strings. Demon-

strate `getLast`, with appropriate coaching. ⟩

Reminder: steps to write a recursive method:

- Base Case:
  - What is the recursive case condition?
  - What is the body of the recursive case?
- Recursive Case:
  - What is the recursive call going to be? (Write it down somewhere.)
  - Leap of Faith: Assume it works!
  - How do we use that recursive call? Write the rest of the case.

Q: Which one should we focus on first?

A:

Q: What is the base case here?

A:

Q: How do we know if we're in that case?

A:

Q: What should we do in that case?

A:

Q: Alright, write that case!

A:

Q: Okay, so what is left?

A:

Q: What do we do in a recursive case?

A:

Q: This is one of those nice cases where we don't really have any "little bit" of work to do. What do we need to do?

A:

Q: Write out the rest of the method.

A:

Q: What is the running time of this method?

A:

In lab, you will write `add`, which puts an element on to the end. We're going to write something similar to that: `addSecondToLast(T element)`.

⟨ Get out the tail-arrows again and demonstrate! ⟩

Q: What does this method do?

A:

Q: So, what will be printed out by this code if `mouse` starts as the list $1, 3, 5, 4, 1$?

```
mouse.addSecondToLast(42);
System.out.println(mouse);
```

A:

Q: What's my base case? (In English)

A:

Q: What do I do in that case?

A:

Q: What do I do in the recursive case?

A:

Q: Okay, try to write the base case!

A:

Q: What's a riffle-shuffle?

A:

Q: Let's write a method that creates a new list by riffle-shuffling two linked lists: `public LinkedList<T> riffle(LinkedList<T> list, LinkedList<T> otherList)`. What will this do with two lists: $(1, 2, 3)$ and $(4, 5, 6)$?

A:

Q: What should we do if one list is longer than another? E.g.: $(1, 2, 3)$ and $(4, 5, 6, 7, 8)$? (Try to keep it easy)

A:

Q: Should we modify `list` and `otherList`?

A:

Q: What does the code look like? Hard!

A:

Q: Let's assume that all types can use the less-than operator. How would we write `LinkedList.getMax()`? Hint: go through the recursion steps

A:

But... max could mean different things for different types...

2.2 Sorted Linked Lists and Comparators

Q: How could a linked list of Integers be sorted?

A:

Q: What about Strings?

A:

Q: So if I want to write a `isSorted` method for Linked Lists, what might I give that as a parameter?

A:

Consider the following class:

```
public class LessThan implements
Comparator<Integer> {
    public LessThan() {
        //empty constructor
    }
    public int compare(Integer a, Integer b) {
        if (a < b) return -1;
        else if (a > b) return 1;
        else return 0;
    }
}
```

In Java, each class that implements the `Comparator<T>` interface specifies an *ordering* on `T` elements by implementing the `public in compare(T, T)` method.
`public int compare(T one, T another);` Returns how `one` and `another` should be ordered:

- negative: if `one` comes before `another`
- positive: if `another` comes before `one`
- zero: they can be in either order (usually because they're equal)

Each ordering will have a different class with a `compare` method.

Q: How could we implement the `LessThan.compare` method using only one line?

A:

Q: What would the body of `GreaterThan`'s compare method look like?

A:

Q: How could we write a Comparator for string length?

A:

Q: When is a list sorted?

A:

Q: How would we use an `isSorted` method for the `LinkedList` class?

A:

Q: Write `public boolean isSorted(Comparator<T> comparator)`. (lowest to highest)

A:

Q: Time Complexity?

A:

2.3 Why Linked Lists?

Q: What can you do with a linked list that you can't with an array-based list?

A:

Q: Why did we spend time on this?!?

A:

Q: Consider a list implemented as an ArrayList. What is the time complexity for `remove(1)`? (This removes the oneth element.)

A:

Q: Why?

A:

Q: Can we speed this up by using a linked list instead?

A:

Q: How?

A:

Q: What do I have to do in `remove`?

A:

Q: What's the time complexity of this?

A:

Q: What do I have to do in `add`?

A:

Q: How long does that take?

A:

2.4 Doubly-linked lists

Q: Might it be useful to have linked lists with connections going back and forth?

A:

These are called doubly-linked lists.

Q: Pro to having double-links?

A:

Q: Downsides?

A:

2.5 Empty linked lists

TODO: move this up with intro stuff?

Q: Can you have an empty linked list?

A:

Q: Should we be able to do this?

A:

Q: How can we implement this elegantly?

A:

TODO: add some questions related to this table. Talk about the implementation with the pointer to the last node.

Operation	Array	Array List	Linked List	Linked List w/Last Pointer
add	$\Theta(n)$	Often: $\Theta(1)$ Seldom: $\Theta(n)$	$\Theta(n)$	$\Theta(1)$
get ith	$\Theta(1)$	$\Theta(1)$	$\Theta(i)$	$\Theta(i)$
remove ith	$\Theta(n)$	$\Theta(n-i)$	$\Theta(i)$	$\Theta(i)$
set ith	$\Theta(1)$	$\Theta(1)$	$\Theta(i)$	$\Theta(i)$

3 Stacks

There are many data structures other than arrays and `ArrayLists`! Consider using a word processor.

Q: What happens if write two sentences of an essay, but then decide I don't like the second sentence, so I delete it. After sitting for a while, I realize I want it back. What do I do?

A:

Q: But then I realize that, no, I really don't want that second sentence back. What could I do now?

A:

Q: Then I realize my single sentence doesn't take up much space, so I increase the font size to 24. But... conscience strikes! AND I want that second sentence back. What can I do?

A:

3.0 Stack Operations

I'm using something called a Stack here! A *stack* is a type of data structure where I can only interact with one end of it, called the *top*.

Q: What are the three basic Stack operations?

A:

Operations:

- `push`: put a new element on the top of the stack.
- `pop`: remove (and return) the top element from the stack.
- `peek`: return the top element of the stack without removing it.

Protocol a stack uses is LIFO: Last In, First Out

Q: Which delicious dish is probably the origin of the word "Stack"?

A:

Q: What do you think a Stack should do if you try to `peek` or `pop`, but there are no elements in it?

A:

Q: In Java, what kind of exception should it throw?

A:

Q: How does the Undo-Redoing example use stacks?

A:

Q: What else might you want to answer the next problems I'm going to give you?

A:

- Constructor
- `isEmpty`: returns a boolean. This keeps us from having to catch `NoSuchElementException`s all over the place.

Q: Write some sample code that uses all 5 operations.

A:

```
Stack<String> strings = new
Stack<String>();
System.out.println(strings.isEmpty());
//false
strings.push('Mankey');
System.out.println(strings.peek());
//Mankey
strings.push('Primeape');
System.out.println(strings.pop());
//Primeape
```


3.1 Methods Using Stacks

Q: Consider implementing the following method: `public static <E> void removeSecondFromTop(Stack<E> stack)`, which removes the element just under the top. When might we want to throw an exception?

A:

Q: Write out the Javadoc

A:

```
/**
 * Removes the second element from a
Stack.
 *
 * @param stack The Stack to remove an
element from; it should have at least
two elements.
 * @throws IllegalArgumentException
when stack has less than two elements.
 */
```

Notice the `@throws` tag to describe when the exception is thrown. Let's assume that Stacks only have the three methods descibed above (`push`, `pop`, `peek`) and `isEmpty`. Using these, here's an implementation of the `removeSecondFromTop` method (only the body):


```
if (stack.isEmpty()) {
    throw new IllegalArgumentException("Called
removeSecondFromTop on a stack with zero
elements!");
}
E top = stack.pop();
if (stack.isEmpty()) {
    //this stack only had one element.  Put the
top back on, then throw an exception
    stack.push(top);
    throw new IllegalArgumentException("Called
removeSecondFromTop on a stack with one
element!");
} else {
    stack.pop();
    stack.push(top);
}
```

Q: Assume that each push and pop takes one step. What is the time complexity of `removeSecondFromTop`?

A:

Q: Write a method that adds an element third from the top: `public static <E> void addThirdFromTop(Stack<E> stack, E element)`

Bonus Challenge: write `removeTops(Stack<E> stack, int numToRemove)`

A:

Q: What is the time complexity of `addThirdFromTop`?

A:

Q: What do `addThirdFromTop` and `removeSecondFromTop` have in common?

A: No loops, nor recursion.

Q: Write:

```
public static <E> int getSizeOf(Stack<E> stack)
```

Bonus Challenge:

```
removeEvens(Stack<Integer> integers)
```

A:

Q: Time complexity of `getSizeOf`?

A:

Q: Write the following method:
`public static <E> void addToBottom(Stack<E> stack, E bottom)`
Bonus Challenge I: try to write it both iteratively and recursively.
Bonus Challenge II: `swapTopAndBottom(Stack<E> stack)`

A:

Q: What's the time complexity of `addToBottom`? (Tricky!)

A:

Q: Write a method:
`reverse(Stack<E> stack)`
Hint: use two auxilliary stacks!
Bonus challenge:
`double(Stack<E> stack)`, which returns a new stack with the elements repeated. E.g. `[a, b, c]` becomes `[a, b, c, a, b, c]`

A:

Q: Time complexity?

A:

Q: Write: `addToMiddle(Stack<E> stack, E element)`

A:

Q: Time complexity?

A:

Q: Why not $\Theta(\frac{1}{2}n)$ (or $\Theta(\frac{3}{2}n)$)?

A:

3.2 Implementing Stacks

There are lots of different ways to implement a Stack.

3.2.1 With an Arraylist

Q: What do you think is a good way of implementing a Stack?

A:

⟨ Draw a Stack object with an "elements" field that references an `ArrayList`. ⟩

Q: Let's try it so that the front of the `ArrayList` is the top of the Stack? What are the running times for each of: `push`, `pop`, and `peek`?

A:

Q: What about if we have the top be the back of the `ArrayList`? How long do each of these operations take?

A:

Let's summarize this into a table:

Implementing a stack with an ArrayList		
Operation	Top at Front	Top at Back
`peek`	$\Theta(1)$	$\Theta(1)$
`push`	$\Theta(n)$	Often: $\Theta(1)$ Seldom: $\Theta(n)$
`pop`	$\Theta(n)$	Often: $\Theta(1)$

Q: Which one is better?

A:

Q: Can we do better? Can we change that "Often $\Theta(1)$" to "*always* $\Theta(1)$?

A:

3.2.2 With a Linked List

Let's speed things up by implementing our Stack using a Linked List!

Q: What are the fields?

A:

Q: Which end should be the top?

A:

Q: What is the time complexity of pop?

A:

Q: What is the time complexity of push?

A:

Q: What is the time complexity of peek?

A:

Q: Is this better than with an `ArrayList`?

A:

Let's add all of this to our summary table:

Implementing a Stack, using a...			
	ArrayList		LinkedList
Operation	Top at Front	Top at Back	
`peek`	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
`push`	$\Theta(n)$	Often: $\Theta(1)$ Seldom: $\Theta(n)$	$\Theta(1)$
`pop`	$\Theta(n)$	Often: $\Theta(1)$	$\Theta(1)$

4 Queues

Q: What do you think the opposite of a stack might be?

A:

Yes. We call this protocol "First-In-First-Out" or FIFO. This data structure is a *Queue*. Put in the *back*, take from the *front*.

4.0 Queue Operations

Q: Which operations come with a pure Queue?

A:

Q: What do you think happens if you try and `pop` or `peek` when the queue is empty?

A:

4.1 Methods Using Queues

Let's write some methods! Assume the only methods in the `Queue` class are: `enqueue`, `dequeue`, `peek`, and `isEmpty`.

Q: Write
`public static <T> int getSize(Queue<T> queue)`.
Bonus Challenge: write
`public static int numberOfEvens(Queue<Integer> integers)`.

A:

Q: Time Complexity?

A:

Q: Write
`public static <T> boolean hasExactlyOneElement(Queue<T> queue)`
Hint: use the `getSize` method we wrote.
Bonus Challenge:
`boolean hasExactlyNElements(Queue<T> queue, int n)`

A:

Q: Time Complexity?

A:

Q: Why?

A:

Q:

Write

```
public static void removeEvens(Queue<Integer>
integers)
```

Bonus Challenge:

```
public static Queue<Integer>
longestIncreasingSubqueue(Queue<Integer>)
```

A:

Q: Time Complexity for `removeEvens`?

A:

Q: Write
`public static <T> Queue<T> copyQueue(Queue<T> queue)`
Bonus Challenge: write
`boolean isPalindrome(Queue<T>)`

A:

Q: Time Complexity?

A:

Q: Write `public static <T> boolean contains(Queue<T> queue, T element)`
Bonus challenge:
`count(Queue<T> queue, T element)`

A:

Q: Time complexity?

A:

Q: Write `public static <T> void reverse(Queue<T> queue)`
Hint: use a Stack
Bonus Challenge: do this without using another data structure.

A:

Q: Time Complexity of `reverse`?

A:

Q: What if we wanted to swap the first and last elements of a queue? What would the time complexity of that be?

A:

Q: Can anyone describe that code without writing it out?

4.2 Implementing Queues

4.2.1 With an ArrayList

Let's consider implementing a queue. Let's start by using an `ArrayList`.

Q: Assume the Queue class has an ArrayList field, `elements`. If the front of the Queue is the zeroeth element in the list, how long do each of `enqueue`, `dequeue`, and `peek` take?

A:

Q: Well, that's a bit disappointing. What happens if the back of the queue is at the zeroeth element?

A:

That's not that much better. Let's consider some other underlying data structures.

4.2.2 With a Linked List

Let's try using a Linked List instead! Let's try having the front of the Queue be the back of the list.

Q: How long do my three operations take?

A:

Q: What about if the front of the Queue is the front of the list?

A:

Q: That's still disappointing! How can we do better?

A:

Wrap our linked-list nodes in an object that a pointer to the last node as well as the first.

Q: Now how long does it take if the front of the list holds the zeroeth element in the Queue?

A:

4.2.3 With a Stack

We could also try implementing a Queue with a Stack! Let our Queue class have a field, `elements`, that has type `Stack<T>`. Let's try doing it where the top of the stack is the front of the queue.

Q: Where in the Stack do we need to put new elements when we add to the Queue?

A:

Q: Write the body for the `public T remove()` method.

A:

Q: Write the body for `public void add(T element)`.

A:

5 Priority Queues

Sometimes we don't just want to take things out based on the order they came in. Sometimes they have a separate priority. Thus: priority queue! Idea: put things in in any order, but take them out according to highest priority.

5.0 Comparators (Again) and Priority

Q: How can we decide which element has the highest priority?

A:

Recall Comparators, as discussed in Section 2.2. Each Priority Queue will have it's own Comparator which determines the ordering of the elements.

Q: What were the three things a comparator's `compare(a, b)` method can return?

A:

Q: And what do the three categories mean?

A:

Let's rephrase this in terms of priority. When `compare(a, b)` returns:

- Negative: `a` has higher priority
- Positive: `b` has higher priority
- Zero: `a` and `b` have the same priority

`compare(a,b)` return value	Order	Priority
negative	`a`-before-`b`	`a` is higher
positive	`b`-before-`a`	`b` is higher
zero	either way	same

5.1 Priority Queue Operations

```
Comparator<String> stringComparator = new
ShortestLength();
PurePriorityQueue<String> pq = new
PurePriorityQueue<String>();
pq.add("Hello");
pq.add("Root Beer Float");
pq.add("Hi");
pq.add("Cupcake");
```


Q: What does `stringComparator.compare` return on `"Hola"` and `"Hi"`?

A:

Q: What about `compare("Abra", "Kadabra")`?

A:

Q: What about `compare("Jigglypuff", "Wigglytuff")`?

A:

Q: What will be printed out by the following loop? (Put it after the previous code.)

```
while (true) {
    try {
        System.out.println("Next removed:  " +
pq.remove());
    } catch (NoSuchElementException nsee) {
        break;
    }
}
```

A:

Q: What do you think a `AlphabeticalOrder` comparator does?

A:

Q: What will the be printed if we replace the first line with: `Comparator<String> stringComparator = new AlphabeticalOrder();`

A:

Q: Are we going to use this to play a game?

A:

5.2 Implementations

Q: How could we organize our Priority Queue?

A:

Q: What is the difference between the two?

A:

Q: Why would you want to keep it sorted?

A:

Q: If we're using an `ArrayList`, which end do we want to take from?

A:

Q: Example: if our priority queue has these elements: `[-4, -2, 55, 101, 212]` then what has higher priority: bigger or smaller numbers?

A:

Q: How long does it take to call `remove()` on a sorted priority queue?

A:

Q: How long does this take on an unsorted priority queue?

A:

Q: Why?

A:

Q: Why might I choose to use an unsorted priority queue, then?

A:

Q: How hard is it to write the code to add an element to an unsorted priority queue?

A:

Q: What is the time complexity?

A:

Q: How hard is it to write the code to add an element to a sorted priority queue?

A:

Q: Why?

A:

Q: back to our example: if our priority queue has these elements: `[-4, -2, 55, 101, 212]` then where will 42 be inserted (what is the index)?

A:

Q: Let's try to write a helper method: `private int findIndexForNewElement(E element)`. What will that look like? (assume we want the element with highest priority at location 0)

A:

Q: What happens after that?

A:

Q: How long does that process take (worst case)?

A:

Q: Is there a way we can find the index faster?

A:

Q: What does that mean?

A:

Q: Let's write a method to do this: `private int fastFindIndexForNewElement(E element, int lowIndex, int highIndex)`

A:

Q: Time complexity?

A:

Q: Logarithmic? Why?

A:

Q: How many times can you divide n in half?

A:

Q: Does this actually speed up our overall time complexity?

A:

Q: Why not?

A:

Q: Could a linked list speed this up?

A:

Q: Well, do we have to worry about shifting everything down in the array?

A:

Q: Does that make it faster overall?

A:

Q: Is there a way to solve this problem? A way to make adds and removes to priority queue in constant time?

A:

Q: Can we get close?

A:

Q: Sweet! How do we do that?

A:

5.3 (Re)sorting the priority queue

Q: Let's say we have a sorted priority queue, but we have some extra functionality: you can add a new comparator (replacing the old one). What do we have to do when this happens?

A:

Q: How should we do that?

A:

Q: Should we write our own sorting algorithm?

A:

Q: Why not?

A:

Don't reinvent the wheel!

Q: What should you use to sort the priority queue?

A:

Q: How would you use that?

A:

6 Sets & Maps

6.0 Sets and Set Operations

Q: How does a set differ from a list?

Q: What sort of operations make sense for a set?

A:

I'm going to ask you to write these:

- `toString`
- `contains`
- `isEmpty`
- `add`
- `remove`
- `toList`
- `equals`

Q: What does `contains` do?

A:

Q: What would the signature of this method look like?

A:

Q: What does `isEmpty` do?

Q: Signature?

A:

Q: What about `add`?

A:

Q: Which method do you think `add` might call?

A:

Q: Why?

A:

Q: Why might `toList` be important?

A:

Q: Signature of `toList`?

A:

Q: If my underlying data structure is an `ArrayList`, should I just return that?

A:

Q: Can someone show me some code that could cause a problem there?

A:

Q: What happened there?

A:

Q: Why?

A:

Q: What should we do instead?

A:

Q: What would that look like?

A:

Q: Why can I use the foreach loop?

A:

Q: Could there still be a problem here?

A:

Q: How?

A:

Q: What should we do about this?

A:

Q: Let's try writing one of the other operations: `hasSubset`. What will the signature look like?

A:

Q: Javadoc?

A:

Q: Code?

A:

Q: What about `intersectWith`? Signature?

A:

Q: Javadoc?

A:

Q: Code?

A:

Q: What method do the elements in a set need to have implemented?

A:

Q: Why?

A:

6.1 Maps

A `Map` is a Java implementation of a mathematical function from one finite set to another. Each map has two separate types: an Input and Output.


```
Map<Integer, String> idsToNames = new
TreeMap<Integer, String>();
idsToNames.put(1, ‘‘Bill Gates’’);
idsToNames.put(2, ‘‘Steve Ballmer’’);
idsToNames.put(3, ‘‘Steve Wozniak’’);
Integer x = new Integer(3);
System.out.println(x + ‘‘ maps to ’’ +
idsToNames.get(x);
```

Q: What if I want to know whether an ID is used by someone. How do I do that? Hint: look at the API for Maps

A:

Q: What is a multiset?

A:

Q: How might we represent a multiset?

A:

Q: How can I use this to compare equivalence of two multisets represented by `ArrayLists`?

A:

Q: Write the code! Assume `bestStrings` and `evenBetterStrings` have type `List<String>`.

A:

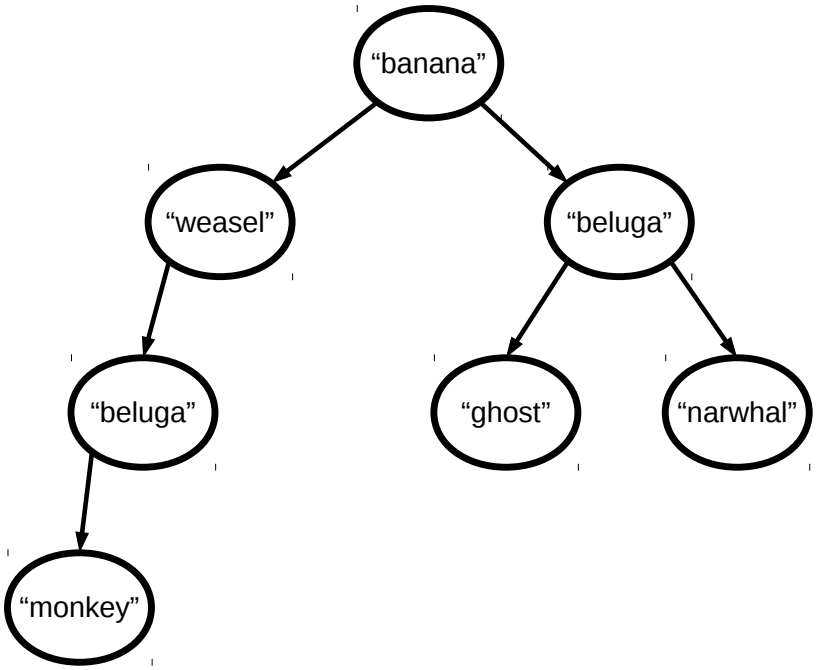
7 Binary Trees

We used `TreeMap`s without really talking about Trees, but Trees are a huge part of Computer Science!

Q: A linked list is a data structure where each node points to another linked list. What do you think a binary tree is?

A:

We draw trees "upside down" from what you're used to.
⟨ Draw this tree: ⟩



⟨ Play Myopic Col on a tree! ⟩

7.0 Tree Fields

The “circles” I’ve drawn are *nodes*. Each node has three fields:

- *value*: object stored at the node
- *left child*
- *right child*

Q: What is the type of each of the children?

A:

Q: Why might you want a fourth field for some trees? Hint: recall doubly-linked lists.

A:

Q: What if we were implementing non-binary trees? What if each node could have any number of children? What might we replace the two child fields with?

A:

For the rest of this, we'll continue considering only Binary trees. Non-binary trees are also very interesting, though!

Q: What is the *root* of the tree?

A:

Q: The *height* of the tree of strings is 4. What's the height of a tree in general?

A:

Q: What do you think it means for a tree to be *balanced*?

A:

⟨ Redraw the tree fully *un*balanced. (No left children) ⟩

Q: What is h, the height of the tree, in this worst-balanced case?

A:

Q: What do you think it means for a tree to be *full*?

A:

Q: Can you draw a full binary tree with 15 nodes?

A:

Q: What about 11 nodes?

A:

Q: Only one of those is a *perfect* binary tree. Which one do you think it is?

A:

Q: What do you think it means to be a perfect binary tree?

A:

Q: How many different full trees can you have with 5 nodes?

A:

Q: How many of them are perfect?

A:

Q: Only one of them is *complete*. Which one?

A:

Q: What do you think a complete tree is?

A:

Q: Can we write h as a function of n for the complete case?

A:

Q: What? How is that?

A:

Q: How many elements on level i? (Root at level 0.)

A:

Q: How many leaves?

A:

Q: How many non-leaves?

A:

Q: Write n in terms of h?

A:

Q: Solve for h?

A:

⟨ Show some examples. ⟩

Q: What does this mean?

A:

[4]

Q: Why is this useful?

A:

7.1 Binary Search Trees

A Binary Search Tree is a sorted collection of elements in a tree.
How a Java `TreeSet` stores elements.

⟨ Draw a BST of integers. ⟩

Q: Worst case: how long does it take to determine whether a BST has a specific element?

A:

Q: How long does that take in the balanced case?

A:

[4] `https://mobile.twitter.com/VitalikButerin/status/794721814815461376`

Q: What about the unbalanced case?

A:

Q: Which is better?

A:

⟨ Example! Add: $[3, 4, 5, 6, 7, 8, 9]$ to a BST. ⟩

Q: Balanced?

A:

Q: How long does it take to add each new element?

A:

Q: What would this look like if it were balanced?

A:

Q: Balanced case: how long to add a new element?

A:

Q: Is there a way to keep the tree balanced?

A:

1. Beforehand: keep track of depth of each subtree. (2 more pieces of information at each node.)
2. First insert the element in the correct place ($\Theta(\log(n))$ steps)
3. Then, go back up the tree and update depths. If subtree unbalanced, rebalance around that node.

Q: How many rebalancings in worst case?

A:

Q: How long does each rebalancing take?

A:

Q: How long do you *hope* each rebalancing takes?

A:

Q: Why?

A:

You: Please show us the AVL rebalancing!
Me: somewhat complicated, but will be covered in future classes

7.2 Traversing entire Tree

Q: Can we use a BST to sort a list of n elements?

A:

Q: How?

A:

Q: How long does that take?

A:

Q: How long to add an element? Both balanced and unbalanced?

A:

Q: So overall?

A:

Q: How do we get the list out of the tree?

A:

Q: There are actually three different ways to traverse trees:

- Inorder: visit left subtree, then value, then right
- Preorder: visit value, then left, then right
- Postorder: visit left, then right, then value

Q: Which one will visit values in sorted order of BST?

A:

Q: Fully balanced tree with values: 1, 2, 3, 4, 5, 6, 7. What is order of visits in pre and post order traversals?

A:

⟨ Show the glove! ⟩

All three of these traversals are *depth-first* traversals. Means, when investigating a child, investigate entire subtree before moving on to the next child.

Q: What is other kind of traversal called?

A:

Q: How does this work?

A:

Q: What is the order vertices will be visited in our $1, \ldots, 7$ tree if we do a left-to-right breadth-first traversal?

A:

7.3 Heaps

A *heap* is a special type of complete binary tree where each element is bigger (or smaller in a min-heap) than all elements below it.

Q: Where is the biggest element in a (max) heap?

A:

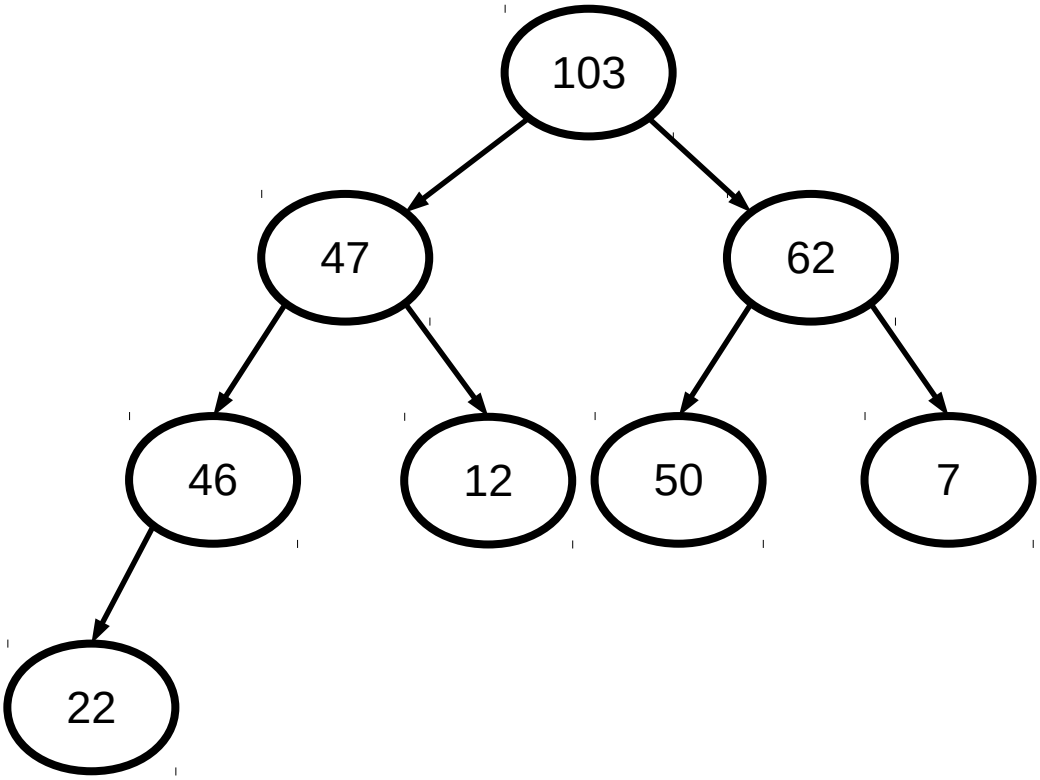
Q: What about the smallest?

A:

Q: How could we describe a (max) heap recursively?

A:

⟨ Show an example: draw this max-heap: ⟩



Notice some odd properties of heaps:

- Elements do not need to be bigger or smaller from left-to-right.
- Elements on higher levels can be smaller than elements on lower levels, so long as they're not directly related.

The main heap operations are:

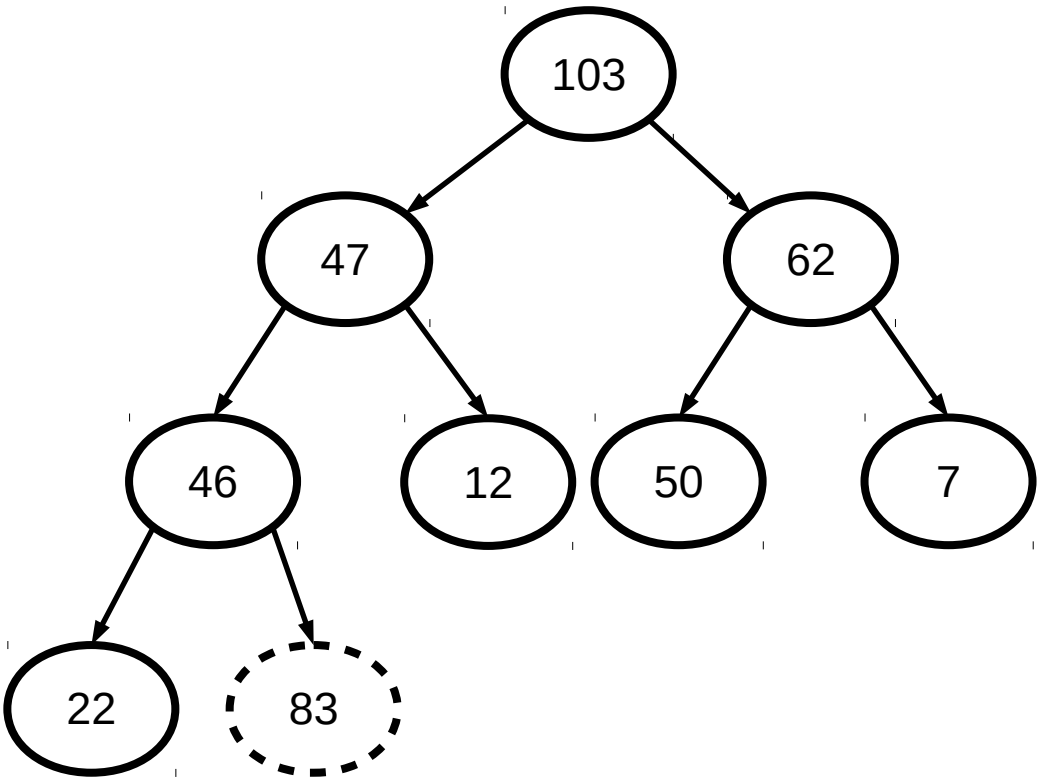
- `add`: add a new element to the heap
- `next`: get the biggest element in the heap
- `remove`: remove the biggest element in the heap

Q: How long does the `next` method take to run?

A:

7.3.1 Heap `add`

`add` is a different story. Let's say we can add a new value in at the last place. (83 here:)



Q: Is this currently a legal max-heap?

A:

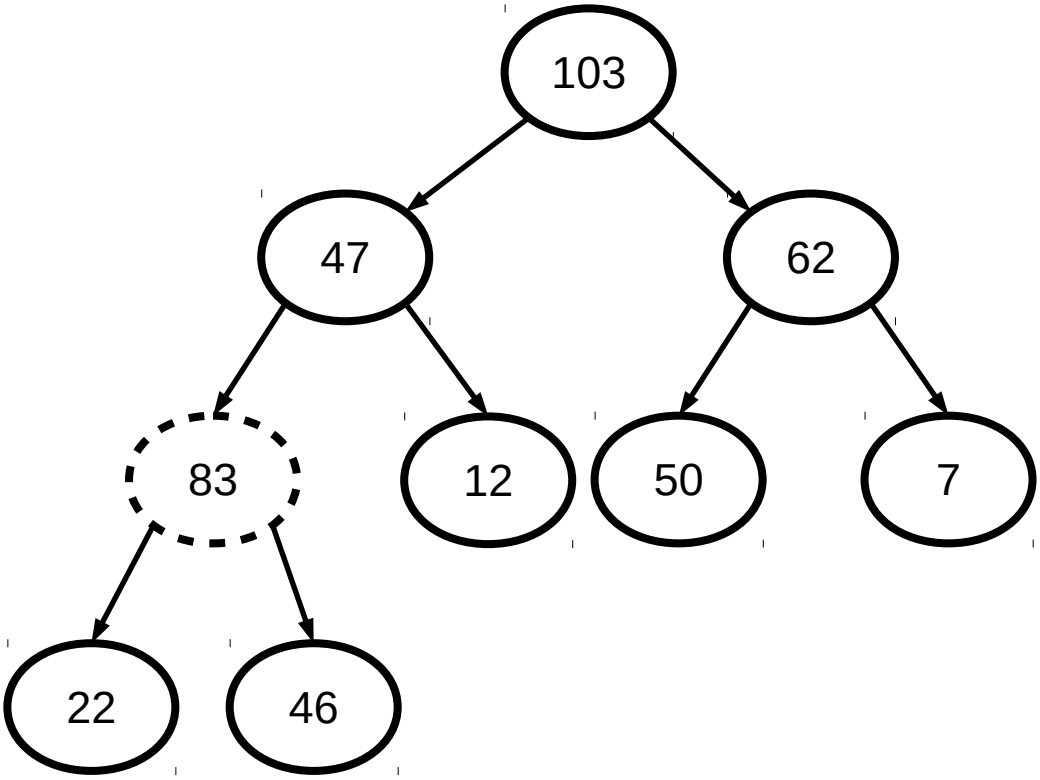
Q: Why not?

A:

Q: How can I fix the relationship between those two nodes?

A:

The result is below:



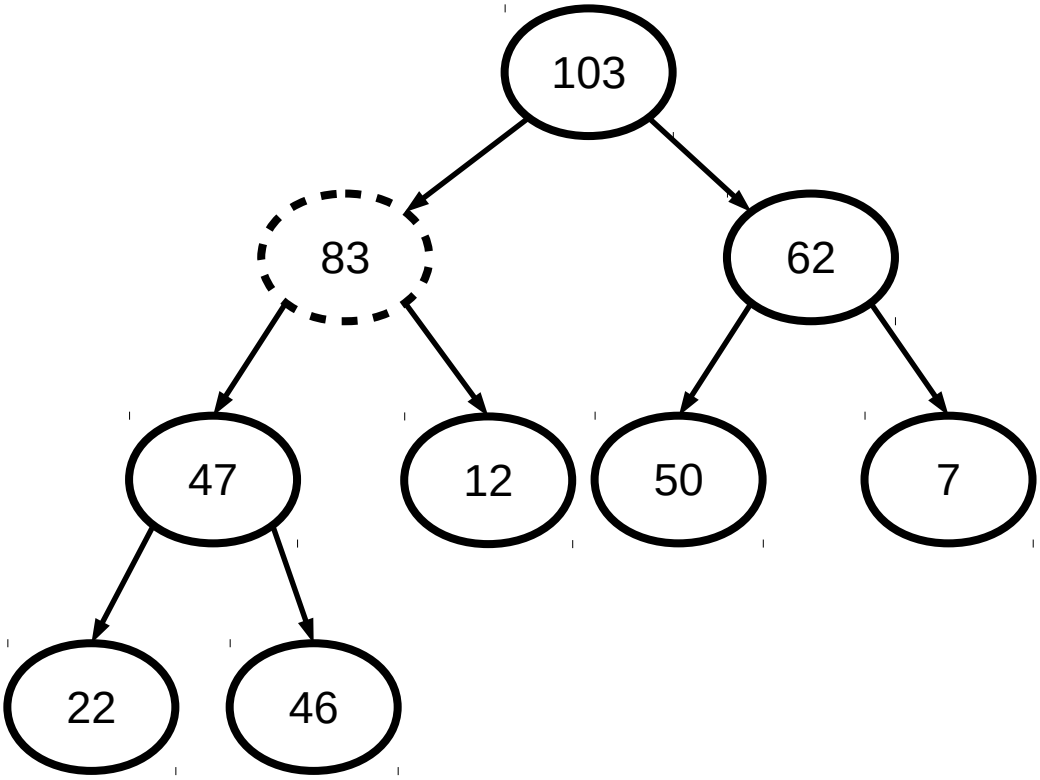
Q: Why is this still not a heap?

A:

Q: How can we fix this?

A:

Result:



Q: Is this a heap?

A:

Q: What might we have to do if the number we added was bigger?

A:

This process of moving the new element upward is known as *heapify*ing.

Q: If we have n elements in our heap, what's the maximum number of swaps we might have to do?

A:

Q: Why logarithmic?

A:

Q: So how long does this reheapifying process take?

A:

In order to reheapify, we need:

- To be able to add the new node on to the first free spot on the lowest level (in constant time) and
- We need nodes to point to their parent as well as their children to work our way back up the tree.

Luckily, there's a trick we can use with arrays. (We'll use an Ar-

rayList for simplicity.)

7.3.2 Modelling a Heap with an Array

To use an ArrayList to implement the heap:

- Leave the zeroeth position empty.
- Put the root of the heap in the space with index 1.
- For any node at index i, the children are at indices $2i$ and $2i+1$.

Q: What are the children of the root (index 1)?

A:

Q: What are the children of index 2?

A:

Q: What are the children of index 3?

A:

Q: Write out the ArrayList that contains the heap we got from adding the 83.

A:

Q: How do I get the parent of index i?

A:

Q: Does that work for both odd and even integers?

A:

Q: Does the ArrayList solve our two needs?

- Be able to get to the last position in constant time, and
- Be able to get from nodes to their parents

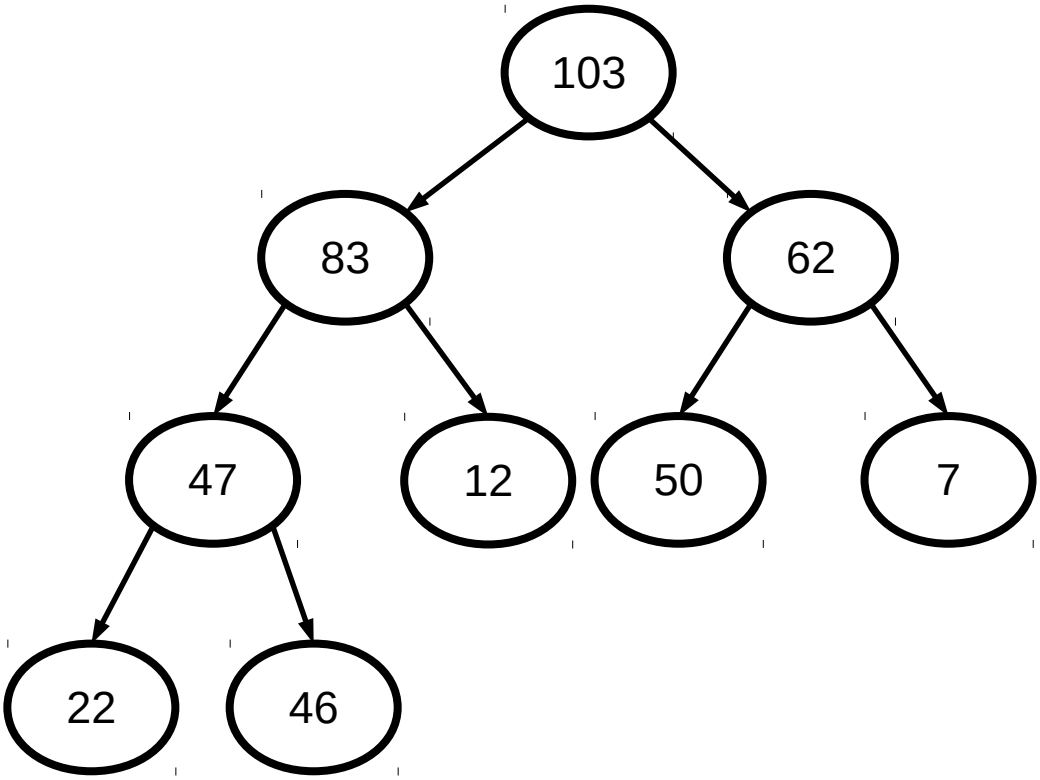
A:

Q: Assume our `Heap` object of `Integer`s has one field: `contents`, the `ArrayList<Integer>`. How can we write `public void add(Integer i)`?

A:

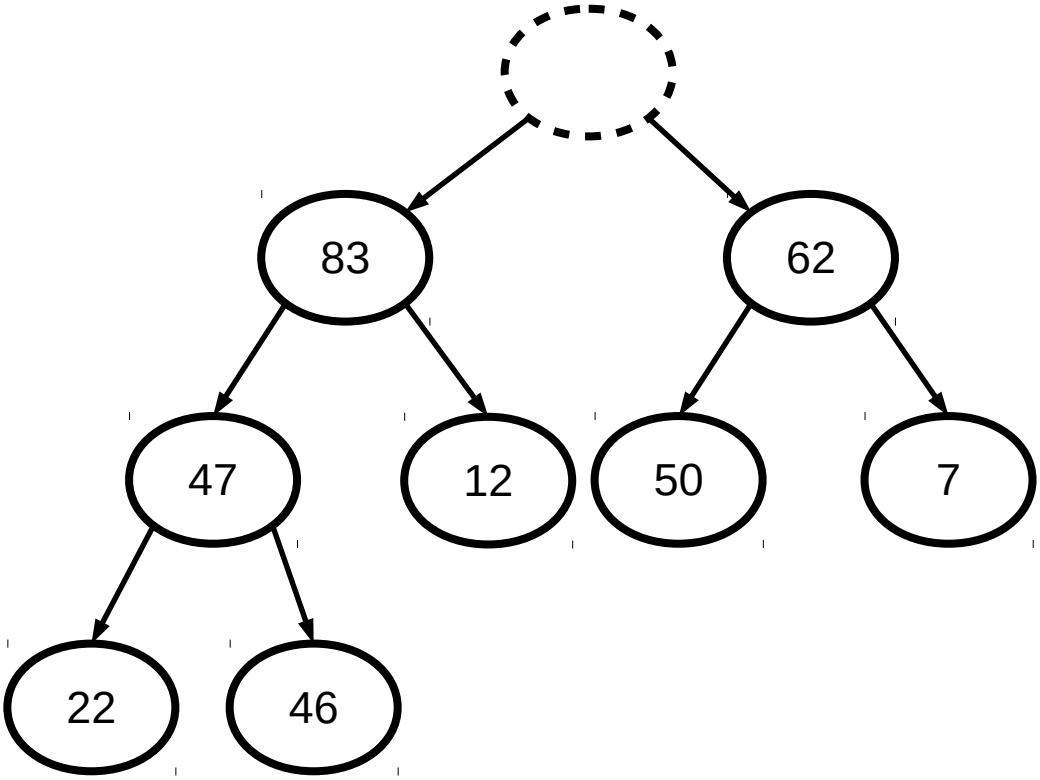
7.3.3 Heap `remove`

Let's see how quickly we can remove elements! Here's a heap before removing the biggest element:



Q: How hard is it to find the element to remove?

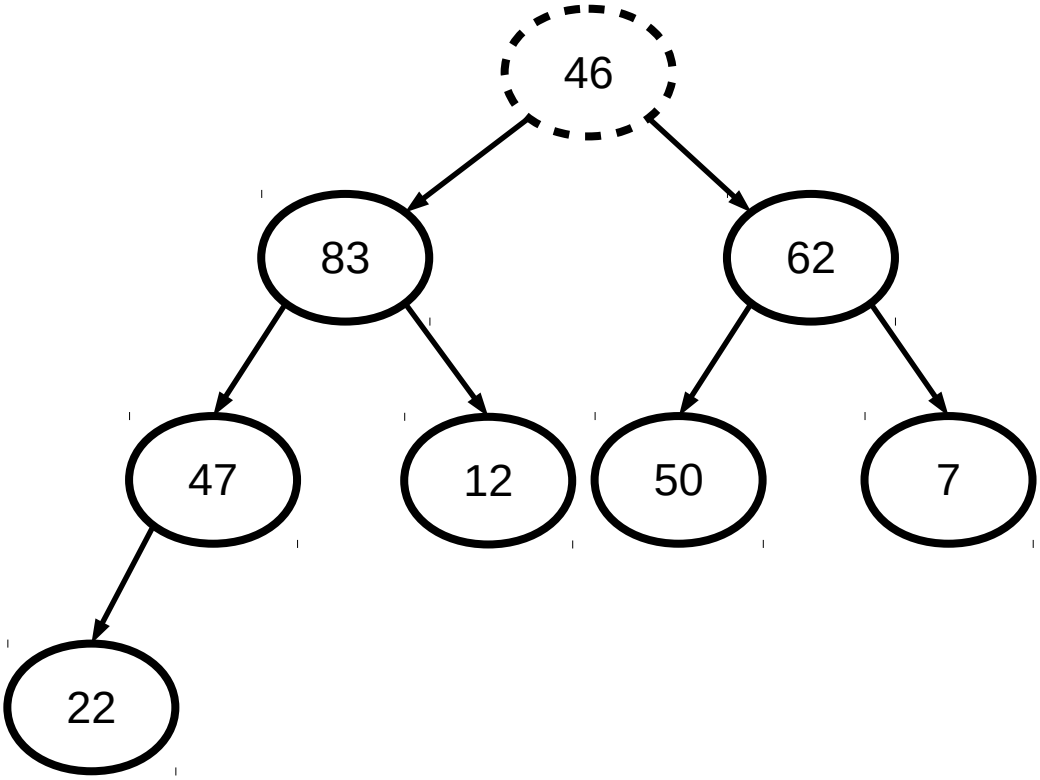
A:



Q: But when we take it out, we're going to have an empty space up top. How are we going to fill that? One option: recursively move the larger child up. What would be the problem with this?

A:

Instead, let's move the last element up top, then move things to the right place. Here's the heap with the last element up top:



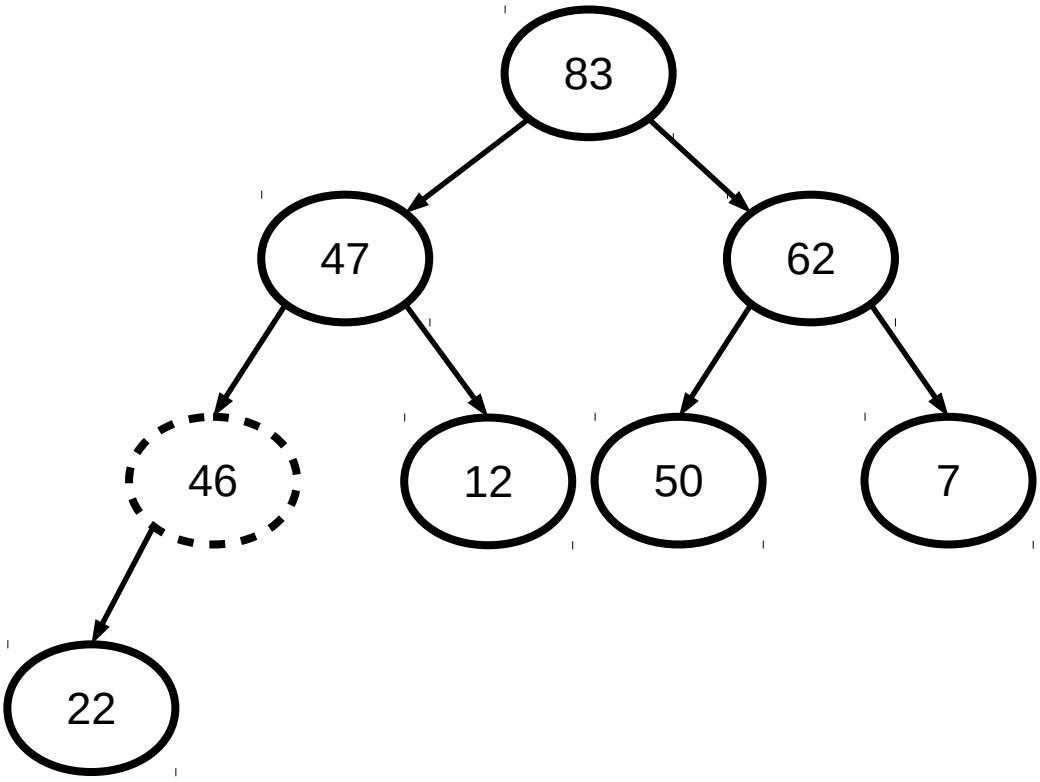
Q:	We need to swap that with one of the two children. Which child should we swap with?
A:	

Q:	What will we do after that?
A:	

This operation is called *reheapify*ing.

Q: What will our example heap look like after it's done reheapifying?

⟨ Swap the element down until it's above the 22. ⟩



Q: What's the maximum amount of swaps the reheapify process could make?

A:

7.3.4 Heap as a Priority Queue

Q: Does this heap perform better than the other Priority Queue implementations we learned?

A:

8 Graphs

⟨ Play Col, Node-Kayles, and/or Neighboring Nim. ⟩

8.0 Graph Definitions

Trees are actually a special case of a more general data structure: graphs. A *graph* is a collection of vertices with edges connecting those vertices.

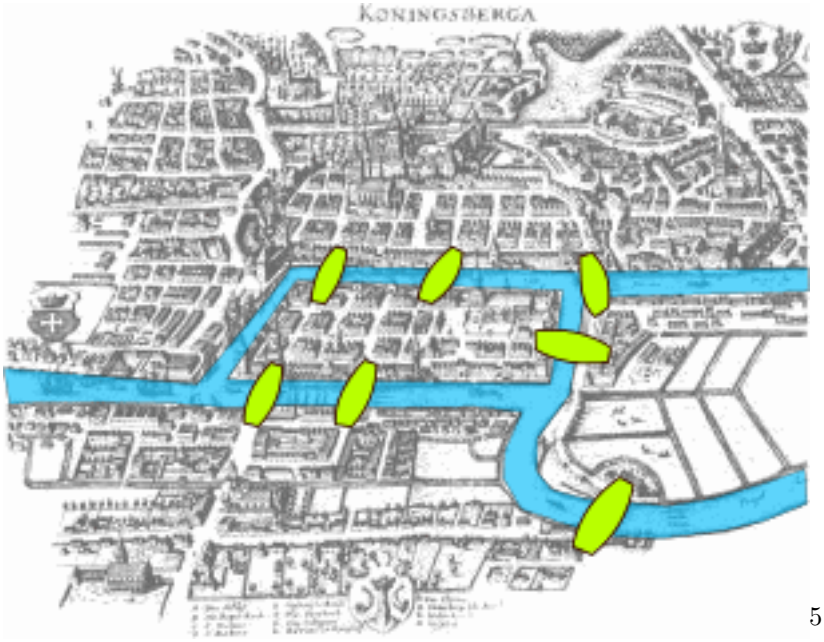
⟨ Draw a picture. ⟩

Q: Where is the data stored?

A:

Q: What could we use a graph for?

A:

⟨ Seven bridges of Königsberg over Pregel River! Draw the picture: 

[5] ⟩

Q: Can you walk around and cross each bridge exactly once?

A:

[5] Image by Bogdan Giuşcă, available at `https://en.wikipedia.org/wiki/File:Konigsberg_bridges.png`.

Q: How can you think about this problem?

A:

Q: If I have n vertices in my graph, what's the smallest number of edges I can have?

A:

Q: What if I want my graph to be *connected*?

A:

⟨ Draw 4 vertices and no edges. ⟩

Q: How can I have a connected graph with 3 edges?

A:

A connected graph with exactly $n - 1$ edges is a *tree*.
⟨ Show "pulling out" tree. ⟩

Q: What's the maximum number of edges I can have if I allow self-loops, but no repeated edges?

A:

Q: What if I don't allow self-loops?

A:

Q: What happens if I have n or more edges?

A:

8.1 Graph Properties

Lots of different questions we can ask:

- Is the graph connected?
- What is the shortest path between two vertices?
- What is the length of the shortest path between two vertices? (*distance*)
- What is the longest distance?
- How many edges are *incident* to a vertex? (*degree*)
- What is the highest degree in a graph?

We can ask some questions about cycles:

- What is the length of the smallest cycle? (*girth*)
- What is the length of the longest cycle?

Q: Euler interested in solving the Koenigsberg problem. How could he have used an answer to the last question?

A:

This sort of cycle is known as an *Eulerian Cycle*: each edge is visited exactly once and you wind up where you started. *Eulerian Path*: each edge is visited exactly once and don't have to start where you end. (If there's an Eulerian Cycle, that cycle is an Eulerian path.)

⟨ Draw a few connected graphs. Ask whether there's an Eulerian Cycle. ⟩

Q: If I have a connected graph, G, when is there an Eulerian Cycle?

A:

Q: When is there an Eulerian Path?

A:

Another type of cycle is a *Hamiltonian* cycle: cycle that visits each vertex exactly once. (Except for source.)

⟨ Draw some (connected) graphs. Which ones have Hamiltonian Paths? ⟩

Q: How hard is it to find a Hamiltonian path?

A:

8.2 NP-Hardness

This video is awesome: `https://www.youtube.com/watch?v=YX40hbAHx3s`

TODO: add stuff here!

8.3 Graph Traversal

Q: If I want to search a graph for a specific value, how can I do that?

A:

Q: What is Depth-First search? (Example: finding the maximum integer in a Graph of integers)

A:

Q: What if all other neighbors have already been visited?

A:

⟨ Draw a graph with IDs as letters of the alphabet and integer values. ⟩

Q: What is the order that vertices are visited using DFS if we break ties using the lower of IDs?

Q: How do we keep track of what's been visited?

A:

Q: How does BFS work?

A:

Q: (Using the same graph) What is the order that vertices are visited using BFS? (Also break ties using ID.)

Q: Do we always break ties this way?

A:

Q: What can we use to implement the BFS?

A:

Q: Which search method is better?

A:

8.4 Spanning Trees

TODO: add stuff about BFS, DFS, SPT trees, Dijkstra's Algo, etc.

9 Sorting

Q: If I give you a bunch of cards with different numbers on them, how will you sort them?

A:

9.0 Sorting

In general, we could use a Comparator[6] to choose the order we want to sort. All of the algorithms given here could be generalized (and therefore improved) by including a Comparator. Here we will always assume we are sorting numbers from the lowest to highest.

Let's look at lots of different sorting algorithms and measure their running times.

⟨ Either follow the sections here in order or let the students suggest different algorithms and hop to the appropriate sections. ⟩

[6] See Section 2.2

9.1 Selection Sort

Q: What if I choose the smallest value and move it to the front? How many elements do I have to swap?

A:

Q: How long does it take?

A:

Q: How could we turn this into a recursive algorithm?

A:

Q: What's our base case?

A:

Q: Let's write it out! Method: `selectionSort(int[] integers)`

A:

Q: Yes, you're right! First let's write `selectionSortHelper(int[] integers, int lowIndex)`. `Do it!`

A:

Q: So, what does the code for `selectionSort(int[] integers)` look like?

A:

Q: So, what is the running time to sort the entire list?

A:

Q: Could we write this non-recursively?

A:

Q: Try it out!

A:

Q: Can we do better?

A:

Q: Sweet! Let's learn the better one now!

A:

9.2 Bubble Sort

Let's look at a sort-of improvement: Bubble Sort. Here's the idea:

- Each time through, we will select the Biggest and move it to the back.
- We'll also keep track of the biggest we've seen so far, and "bubble" it up by swapping with the next element.
- But! We won't bubble if the next element is bigger.

Q: What does the code look like for one run through the algorithm?

A:

Q: How long does that single run take?

A:

Q: How many times do we have to do it?

A:

Q: What does the code for that look like?

A:

Q: What's the overall running time?

A:

⟨ If possible, do a demonstration! Have a student do the steps! I use playing cards. ⟩

Q: Can we improve things?

A:

Q: How?

A:

Q: If the outer loop variable is `j`, what will the condition of the inner loop look like now?

A:

Q: Can we improve further?

A:

Q: How could we do this with n processors?

A:

⟨ Set up the demonstration using $n-1$ students. In each round, $\frac{n}{2}$ of them (first even indexed, then odd-indexed) are active, comparing the two elements next to them. Bubble if necessary. Then go to the next round. ⟩

9.3 Insertion Sort

Selection Sort is sort of a brute-force method. Bubble Sort is another. Let's check out a third: Insertion Sort.

Q: How does Insertion sort work (high level)?

A:

Q: What happens to insert just one element?

A:

Q: Code for that shifting?

A:

Q: How many times do we need to do it?

A:

Q: How do we write that?

A:

Q: How long does this take?

A:

Q: Can we do better than $\Theta(n^2)$?

A:

Q: Can we learn that now?

A:

9.4 Merge Sort

Let's divide and conquer!

Q: If I cut the list in half and sort each half, how can I merge those two halves together?

A:

Q: How long does that merging take? (Assume the two lists sum to size n.)

A:

⟨ Draw two columns of long rectangles. Left column, top row: one rectangle (whole array). Below is arrow pointing down labelled "Divide". Second row: array-rectangle split into half. Second column has the same rectangle, but with the arrow going up and saying "Merge". Write $\Theta(n)$ nearby. ⟩

Q: So... how should I sort each of those halves before merging?

A:

Q: How long is each of those arrays? (That've been cut in half twice?)

A:

⟨ Add another row to the Left side of the diagram: 4 subarrays. ⟩

Q: Once they're each sorted, how long will it take to merge two of them?

A:

Q: So how long to do all the merging at that level?

A:

⟨ Add another row to the right side and add the $\Theta(n)$ ⟩

Q: How long will each array be one-level deeper?

A:

Q: And how long total to perform the four merges?

A:

⟨ Add another row to the diagram. Then draw an ellipsis. ⟩

Q: When do we stop the recursion?

A:

⟨ Draw singleton boxes. ⟩

Q: How long does it take in total to merge each pair of singleton boxes?

A:

Q: So it takes $\Theta(n)$ time to do all the merging at one level. How many levels are there? (Hint: how many times can we divide n by 2?)

A:

Q: So what is our overall running time?

A:

Q: Code?

A:

Q: This is very complicated to code! Is there an easier way?

A:

9.5 Quick Sort

Q: What if I could get the median element of a list in constant time? (Hint: think divide and conquer.)

A:

TODO: code the shifting around the median

Q: How long does it take to shift around the median?

A:

Q: Then what?

A:

Q: How long to shift total at each level?

A:

Q: And how far down do we have to recurse?

A:

Q: So what's the overall running time?

A:

Q: Wait! I lied to you! Where?

A:

Q: What should we do instead?

A:

Q: Does that work?

A:

Q: What could go wrong in the worst case?

A:

Q: If we kept doing that, how long would the algorithm take?

A:

Q: Ohno! This algorithm sucks! Should we use it?

A:

Q: Don't those $\Theta(n^2)$ worst cases hurt us?

A:

9.6 HeapSort

There is another algorithm we can use that can start working even if we don't have the whole list to start! Sweet! How do we do that? With Heaps![7]

[7] Heaps were introduced in Section 7.3.

Q: How does this one work?

A:

Q: If we're sorting lowest-to-highest, what kind of heap do we want to use?

A:

Q: How long does it take to put all the elements into the heap?

A:

Q: How long does it take to take them all out?

A:

Q: So how long in total?

A:

9.7 Radix Sort

All the sorting algorithms we've seen so far compare elements to determine which elements should go where. This last algorithm, Radix Sort (a.k.a. Bucket Sort) uses no such comparisons!

Q: What's the idea behind this one?

A:

Q: How many buckets will we have?

A:

Q: How many rounds will we have?

A:

This doesn't seem like it will work at all, but let's try it out! List of 3-digit numbers: $[312, 128, 442, 319, 825, 445, 378, 198]$. Now, set up the buckets.

Q: Which digit will we sort by in the first round?

A:

Q: Wait, really?!?

A:

Q: Which bucket does 312 go into?

A:

Q: What does the situation look like now?

A:

Q: What about after the 128?

A:

Q: Does the 442 go before or after the 312?

A:

Q: Because it's larger than the 312?

A:

Q: What does our situation look like?

A:

Q: Okay, now add the rest. What's going on after that?

A:

Q: Now what?

A:

Q: In what order?

A:

Q: What is the list now?

A:

Q: Now we do it again, with one little difference. What's that difference?

A:

Q: Which digit are we using in this round?

A:

Q: Sort them all into buckets again. What do the buckets look like?

A:

Q: How long does it take to do one round?

A:

Q: Now put them back into the array. What does the array look like?

A:

Q: What next?

A:

Q: Which digit are we checking?

A:

Q: Put them into buckets. Then what's it look like?

A:

Q: What happens when we put them into the list now?

A:

Q: How long does that take?

A:

Q: How many rounds?

A:

Q: Yes. And no. Hmmmmmm. Why is it not really a constant?

A:

Q: Then what's the running time?

A:

Oh. So how do we write the running time of this down in general? First, remember something very important!

Q: If I have a number, N, what is the number of digits of N?

A:

Q: If I have a list with n integers, and the largest number if x, what is my overall running time?

A:

It could easily be the case that $x > n$. So... not necessarily any faster!

9.8 Summary

In summary, you should watch these visualizations of sorting algorithms: `https://www.youtube.com/playlist?list=PL0J1qhm02c3ZUW8iF9L8LfUA`

[8]

A Java Sockets

Let's learn how to transmit data using Java across networks!

[8] XKCD #1185: `https://xkcd.com/1185/`. The alt-text on this is awesome: "StackSort connects to StackOverflow, searches for 'sort a list', and downloads and runs code snippets until the list is sorted." So awesome, that someone implemented it! `http://gkoberger.github.io/stacksort/`

A.0 Network Basics

Q: What is a port?

A:

We have ports 8000 - 8050 available on turing for our use.

Q: How do you use a port?

A:

Q: What is a socket?

A:

Things needed to create a socket:

- Accessible server with address.
- Free port on that server.

A.1 Client and Server code

Let's see how to write some code to talk to a remote server! Plymouth State students can use this to talk to a Server-side Java program I have running by first VPNing into campus (or using a lab machine.) If you are not a Plymouth State student, you can still use the following code, you just won't be able to connect a client-side program to my server-side player.

First, we'll need to import some packages:


```
import java.net.*;
import java.io.*;
```

Q: Will the code be different for the server and client?

A:

```
(Server-side)
int port = 8000;
ServerSocket serverSocket = new
ServerSocket(port);
Socket socket = serverSocket.accept();
```

Last line will block (wait) until the socket is created on the client side.

```
(Client side)
int port = 8000;
String serverAddress = ‘‘gnirut.plymouth.edu’’;
Socket socket = new Socket(serverAddress, port);
```

Now the two pieces of code can send and receive messages! Let's send a message from the client to the server.

```
(Server-side)
InputStream input = socket.getInputStream();
InputStreamReader streamReader = new
InputStreamReader(input);
BufferedReader reader = new
BufferedReader(streamReader);
String line = reader.readLine();
```

This last line will block until the client sends a message that includes a line break. Let's write that!


```
(Client-side)
OutputStream output = socket.getOutputStream();
OutputStreamWriter streamWriter = new
OutputStreamWriter(output);
BufferedWriter writer = new
BufferedWriter(streamWriter);
writer.write("monkey!");
writer.newLine();
writer.flush();
```

Q: What code should I add to the server to see the message?

A:

Q: Let's say the server handles that line, then responds. What code should I add to the client to see the response?

A:

Q: Have I "swept anything under the rug"?

A:

⟨ Let's all try to send a message to the server. Send the string: "new" followed by a line break! ⟩

A.2 Port Congestion

Q: Can we use that initial socket for all of our communication?

A:

Q: Why not?

A:

Q: What can we do instead?

A:

Q: Which are we doing in the project?

A:

Q: How do you write that code?

A:

In our implementation, after you connect on the new port, send the string `"new"` to the server. It will give you back a new game.

⟨ Try to get the string for that new game! ⟩

More useful `String` methods:

- `trim`
- `indexOf`
- `charAt`
- `substring`
- `replace`
- `split`

B Graphics

B.0 Java Swing Graphics

Java Swing is a package for GUIs that is becoming obsolete due to Java FX. Still, useful for describing graphical interactions. Important players:

- `JFrame`: holds the graphics in a window displayed by the OS.
- `JComponent` or `JPanel`: "canvas" inside the JFrame.
- `Graphics`: class that does the actual drawing.

To use these, import `javax.swing.*` and `java.awt.*`.

I'm going to run the following code line-by-line in interactive mode in Dr. Java. You should *not* do the same. Instead, create a new class, `MyPanel.java`, and put this all in the main method there.


```
JFrame frame = new JFrame("Hello!");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_
CLOSE);
int windowWidth = 1000;
int windowHeight = 500;
frame.setPreferredSize(new Dimension(windowWidth,
windowHeight));
frame.setBackground(Color.GRAY);
frame.pack();
frame.setVisible(true);
```

In Java, a Graphics object does the drawing onto different "surfaces", such as JPanels. Whenever Java wants to display a JPanel (or, more generally, a JComponent) it calls that component's `paintComponent` method, which takes a Graphics object as a parameter. When some user's code wants to redraw a JPanel, it calls `revalidate`:

```
JComponent panel = frame.getContentPane();
panel.revalidate();
```

That method probably looks something like this:

```
public void revalidate() {
    Graphics g = this.getGraphics();
    this.paintComponent(g);
}
```

(I'm sure it's actually a bit more complicated than that.) You shouldn't call the `paintComponent` method yourself, but that is what you should override if you want to draw something. Let's draw something! We can create our own `JPanel` subclass and implement the `paintComponent` method to do the drawing for us:


```
public class MyPanel extends JPanel {
    public MyPanel() {
        super();
    }
    public void paintComponent(Graphics g) {
        //code to draw stuff goes in here.
    }
}
```

Checking out the Graphics API reveals a bunch of `drawX`-type methods. For example, we can draw a string:

```
public void paintComponent(Graphics g) {
    g.drawString("Hi there!", 100, 5);
}
```

We can test this out by writing some code to use our `MyPanel` class:

```
JFrame frame = new JFrame("Hello!");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_
CLOSE);
int windowWidth = 1000;
int windowHeight = 500;
frame.setPreferredSize(new Dimension(windowWidth,
windowHeight));
//set the content pane to our new object!
frame.setContentPane(new MyPanel());
frame.setBackground(Color.GRAY);
frame.pack();
frame.setVisible(true);
```


Q: `drawString` takes two location arguments describing the position of the string to draw. Which one describes the horizontal position and which describes the vertical?

A:

Q: The first value is Java's x-coordinate. The second is the y-coordinate. The values for these are in pixels. Which corner is the text positioned 100 pixels from in the x-direction and 5 pixels in the y-direction?

A:

Q: How are these coordinates different from standard cartesian coordinates?

A:

There may not be anything actually drawn at point $(100, 5)$, but that point is at the upper-left-hand corner of the *bounding box* that surrounds the text that is drawn. Different programming languages draw things differently, but many of them use bounding boxes to describe the locations of objects. In this case, you don't specify the entire bounding box, just one corner of it.

What if we want to, say, change the color of the text? Let's add some more text in a different color:


```
public void paintComponent(Graphics g) {
    g.drawString("Hi there!", 100, 5);
    g.setColor(Color.RED);
    g.drawString("Awesome as I wanna be!", 10,
100);
}
```

Text is still pretty boring. Let's add some shapes! Here's some code that adds an oval:

```
public void paintComponent(Graphics g) {
    g.drawString("Hi there!", 100, 5);
    g.setColor(Color.RED);
    g.drawString("Awesome as I wanna be!", 10,
100);
    g.setColor(Color.WHITE);
    int mysteryA = 5;
    int mysteryB = 200;
    int mysteryC = 300;
    int mysteryD = 150;
    g.drawOval(mysteryA, mysteryB, mysteryC,
mysteryD);
}
```


Q: Test this code out. Which of the four mystery variables cooresponds to each of the following?

- Height of the bounding box of the circle?
- Width of the bounding box of the circle?
- x coordinate of the bounding box's top left corner?
- y coordinate of the bounding box's top left corner?

Hint: try changing `drawOval` to `drawRect` to see the bounding box.

A:

Q: What happens if I replace `drawOval` with `fillOval`?

A:

Q: What are the coordinates of the center of that oval?

A:

Q: What if we want to put a circle with height (diameter) of 130 in the center of that oval? Which method should we use? (Go ahead and check out the Graphics class if you haven't already.)

A:

Q: What's the radius of that circle going to be?

A:

Q: What should we put for the values of these?

- x offset? (x coordinate of the upper left corner)
- y offset? (y coordinate of the upper left corner)
- width?
- height?

A:

Q: So what's that call going to look like?

A:

Add that to your code, right above drawing the bigger oval: (I've removed the initial text-drawing stuff.)

```
public void paintComponent(Graphics g) {
    g.setColor(Color.GREEN);
    g.fillOval(90, 210, 130, 130);
    g.setColor(Color.WHITE);
    g.fillOval(5, 200, 300, 150);
}
```

Q: Run it! Why didn't that do anything?

A:

Q: What do we need to do to fix it?

A:

Q: Change your code to draw the green part second. Does that work?

A:

Q: Let's put a smaller black circle inside of the green one, with diameter 60. What is the new `fillOval` line going to look like?

A:

Q: Add that to your code (don't forget to change the color to black first). What does the picture look like?

A:

Here's my code so far. I added some comments to make it clear what's being drawn.

```
public void paintComponent(Graphics g) {
    //draw the eyeball
    g.setColor(Color.WHITE);
    g.drawOval(5, 200, 300, 150);
    //draw the iris
    g.setColor(Color.GREEN);
    g.drawOval(90, 210, 130, 130);
    //draw the pupil
    g.setColor(Color.BLACK);
    g.drawOval(110, 230, 90, 90);
}
```

Q: How easy is it to tell that that's what the code is drawing?

A:

Q: What if I want to change the location of the eye? How much code do I have to change?

A:

Q: Would I have to do arithmetic to recalculate all of the new values?

A:

Why do math? Let's make the computer do it for us! Also, we'll make the code *way* more readable in the process.

First things first: let's add variables for:

- The center of the eye.
- The eye ball's bounding box size.

```
public void paintComponent(Graphics g) {
    int eyeCenterX = 155;
    int eyeCenterY = 275;
    //draw the eyeball
    int eyeWidth = 300;
    int eyeHeight = 150;
    ...  there's still more to add...
}
```


Q: Let's add variables `eyeBallTopLeftX` and `eyeBallTopLeftY` for the top left corner of the bounding box. How can we calculate these?

A:

Q: What should our line of code look like that actually draws the eye ball now?

A:

Note that only two of those are hard-coded. The other are dependent on my other variables. Here's my code now:

```
public void paintComponent(Graphics g) {
    int eyeCenterX = 155;
    int eyeCenterY = 275;
    //draw the eyeball
    int eyeWidth = 300;
    int eyeHeight = 150;
    int eyeBallTopLeftX = eyeCenter - eyeWidth /
2;
    int eyeBallTopLeftY = eyeCenter - eyeHeight /
2;
    g.setColor(Color.WHITE);
    g.fillOval(eyeBallTopLeftX, eyeBallTopLeftY,
eyeWidth, eyeHeight);
    g.setColor(Color.GREEN);
    ...
}
```

Q: You might not like some of these long variable names. Let's scale it back for the iris and pupil. For the iris, I used `diameter`, `x`, and `y`. How can we define these three? (Hint: only one needs to be hard-coded.)

A:

Q: Then what's the `fillOval` command?

A:

Here's the code again:


```
public void paintComponent(Graphics g) {
    int eyeCenterX = 155;
    int eyeCenterY = 275;
    //draw the eyeball
    int eyeWidth = 300;
    int eyeHeight = 150;
    int eyeBallTopLeftX = eyeCenter - eyeWidth /
2;
    int eyeBallTopLeftY = eyeCenter - eyeHeight /
2;
    g.setColor(Color.WHITE);
    g.fillOval(eyeBallTopLeftX, eyeBallTopLeftY,
eyeWidth, eyeHeight);

    //draw the iris
    g.setColor(Color.GREEN);
    int diameter = 130;
    int x = eyeCenterX - diameter/2;
    int y = eyeCenterY - diameter/2;
    g.fillOval(x, y, diameter, diameter);

    //draw the pupil
    ...  }
```

Q: Let's reassign values to `diameter`, `x`, and `y`. What needs to change in these three lines?

A:

Q: What about the `fillOval` line?

A:

Here's the code again:

```
public void paintComponent(Graphics g) {
    int eyeCenterX = 155;
    int eyeCenterY = 275;
    //draw the eyeball
    int eyeWidth = 300;
    int eyeHeight = 150;
    int eyeBallTopLeftX = eyeCenter - eyeWidth /
2;
    int eyeBallTopLeftY = eyeCenter - eyeHeight /
2;
    g.setColor(Color.WHITE);
    g.fillOval(eyeBallTopLeftX, eyeBallTopLeftY,
eyeWidth, eyeHeight);

    //draw the iris
    g.setColor(Color.GREEN);
    int diameter = 130;
    int x = eyeCenterX - diameter/2;
    int y = eyeCenterY - diameter/2;
    g.fillOval(x, y, diameter, diameter);

    //draw the pupil
    g.setColor(Color.BLACK);
    diameter = 60;
    x = eyeCenterX - diameter/2;
    y = eyeCenterY - diameter/2;
    g.fillOval(x, y, diameter, diameter);
}
```


Q: What happens if we change the `eyeCenterX` and/or `eyeCenterY` values? Hint: Try changing them to 800 and 100, respectively.

A:

Change these values a few more times until you're comfortable with moving the coordinate system.

B.1 Interactive Components

Let's play around with some clickable buttons! I created a new file: `Buttons.java`:


```
(Buttons.java)
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Buttons {

    public static void main(String[] args) {
        //set up the JFrame
        JFrame frame = new JFrame("Hello!");
        frame.setDefaultCloseOperation(JFrame.EXIT_
ON_ CLOSE);
        int windowWidth = 1000;
        int windowHeight = 500;
        frame.setPreferredSize(new
Dimension(windowWidth, windowHeight));

        //create the button
        JButton button = new JButton(``Click
me!'');
        frame.setContentPane(button);

        //pack and display the fram
        frame.pack();
        frame.setVisible(true);
    }
}
```

B.2 GUI Events

Q: Now let's make something happen when we click the button! How do we do that?

A:

Q: Which method do we need to implement for the `ActionListener` interface?

A:

We'll need to import another package: `import java.awt.event.*;`

```
...
JButton button = ...
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent
event) {
        System.out.println(``Clicked the
button!'');
    }
});
frame.setContentPane(button);
...
```

Q: Try this out! What happens when the button is clicked?

A:

Q: What happens if the button is clicked multiple times?

A:

Q: Why? Why not just once?

A:

Q: What happens if the button is clicked while other code is running?

A:

Let's see if we can change the button itself when it's clicked! To do this, we'll need to use the `getSource` method.

```
...
    public void actionPerformed(ActionEvent
event) {
        System.out.println(``Clicked the
button!'');
        JButton button = (JButton)
event.getSource();
        button.setForeground(Color.RED);
    }
...
```

Q: What happens now when the button is clicked?

A:

You can also set the background. This will do different things based on the operating system you're using!

Let's include a second button! We'll need to change the way we add them to the `JFrame` by first putting both buttons into a `JPanel`:

```
  ...
});
JButton anotherButton = new JButton(``Click me
too!'');
anotherButton.addActionListener(new
ActionListener() {
    ...  students fill in details here...
});
JComponent buttonPanel = new JPanel();
buttonPanel.setLayout(new GridLayout(1,2));
buttonPanel.add(button);
buttonPanel.add(anotherButton);
frame.setContentPane(buttonPanel);
frame.pack();
...
```

Q: What do you think a `GridLayout` is?

A:

Q: What would we have to do to add a third button?

A:

Q: Bonus challenge: use a `BorderLayout` to arrange three or more buttons. You'll need to take a look at the API to see how to use this.

C Event-Driven Programming

C.0 Java Events

Clicking a button triggers an event which starts a new thread of computation. Let's play around with events and see if we can get two threads running simultaneously! First, let's rename the first button and make it count down from 10.

```
...
JButton button = new JButton(``Countdown'');
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent
event) {
        for (int i = 10; i > 0; i--) {
            System.out.println(i + ``...'');
            try {
                Thread.sleep(1000);
            } catch (Exception e) {
                System.out.println(e.getMessage());
            }
        }
        System.out.println(``Blastoff!'');
    }
});
...
```

Q: If this creates two simultaneous threads, what should happen when we click the button twice?

A:

Q: Is that what happens here?

A:

Q: What does happen?

A:

Q: Why?

A:

Q: Let's try triggering a different event with the other button. Rename that one `Countup` and have it count upwards.

A:

Q: Now can we create simultaneous threads? Try clicking one button, then the other.

A:

Q: Why not?

A:

Q: Time to give up?

A:

Let's try explicitly creating and dispatching new threads! To do this, we're going to create anonymous objects that implement the `Runnable` interface.

Q: Which method do we need to implement?

A:

Q: Once we have a `Runnable` object, say `monkey`, how do we run it in a new thread?

A:

Q: Let's use it! Inside the `actionPerformed` method of `button`, encapsulate the counting down into a new `Runnable` object and start it.

A:

Q: Does this (finally) work? Try clicking the button twice.

A:

Q: Why?

A:

