CS 600.226: Data Structures

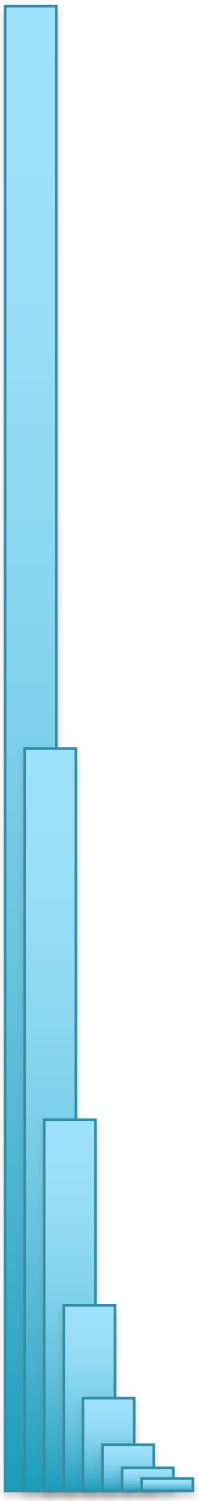
Michael Schatz

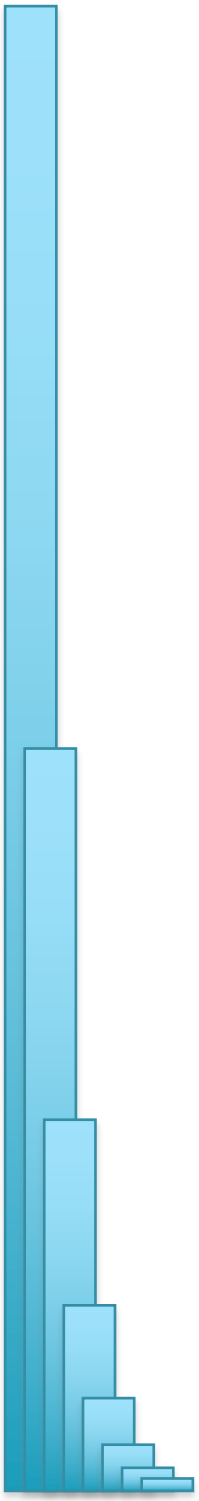
Sept 21 2018

Lecture 10. Stacks and JUnit

Agenda

1. *Review HW2*
2. *Introduce HW3*
3. *Recap on Stacks*
4. *Queues*





Assignment 2: Due Friday Sept 21 @ 10pm

https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment02/README.md

Assignment 2: Arrays of Doom!

Out on: September 14, 2018
Due by: September 21, 2018 before 10:00 pm
Collaboration: None
Grading:

- Functionality 65%
- ADT Solution 20%
- Solution Design and README 5%
- Style 10%

Overview
The second assignment is mostly about arrays, notably our own array specifications and implementations, not just the built-in Java arrays. Of course we also once again snuck a small ADT problem in there...

Note: The grading criteria now include **10% for programming style**. Make sure you use Checkstyle with the correct configuration file from Github!

Assignment 2: Due Friday Sept 21 @ 10pm

https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment02/README.md

Problem 1: Revenge of Unique (30%)

You wrote a small Java program called Unique for Assignment 1. The program accepted any number of command line arguments (each of which was supposed to be an integer) and printed each unique integer it received back out once, eliminating duplicates in the process.

For this problem, you will implement a new version of Unique called ***UniqueRevenge*** with two major changes:

- First, you are no longer allowed to use Java arrays (nor any other advanced data structure), but you can use our Array interface and our SimpleArray implementation from lecture (also available on github)

- Second, you're going to modify the program to read the integers from standard input instead of processing the command line.

Assignment 2: Due Friday Sept 21 @ 10pm

https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment02/README.md

Problem 2: Flexible Arrays (20%)

Develop an algebraic specification for the abstract data type FlexibleArray which works like the existing Array ADT for the most part **except** that both its **lower** and its **upper** index bound are set when the array is created. The lower as well as upper bound can be **any** integer, provided the lower bound is **less than or equal** the upper bound.

Write up the specification for FlexibleArray in the format we used in lecture and **comment** on the design decisions you had to make. Also, tell us what kind of array **you** prefer and why.

Hints

- A FlexibleArray for which the lower bound equals the upper bound has exactly one slot.
- Your FlexibleArray is **not** the Array ADT we did in lecture; it doesn't have to support the exact same set of operations.

Assignment 2: Due Friday Sept 21 @ 10pm

https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment02/README.md

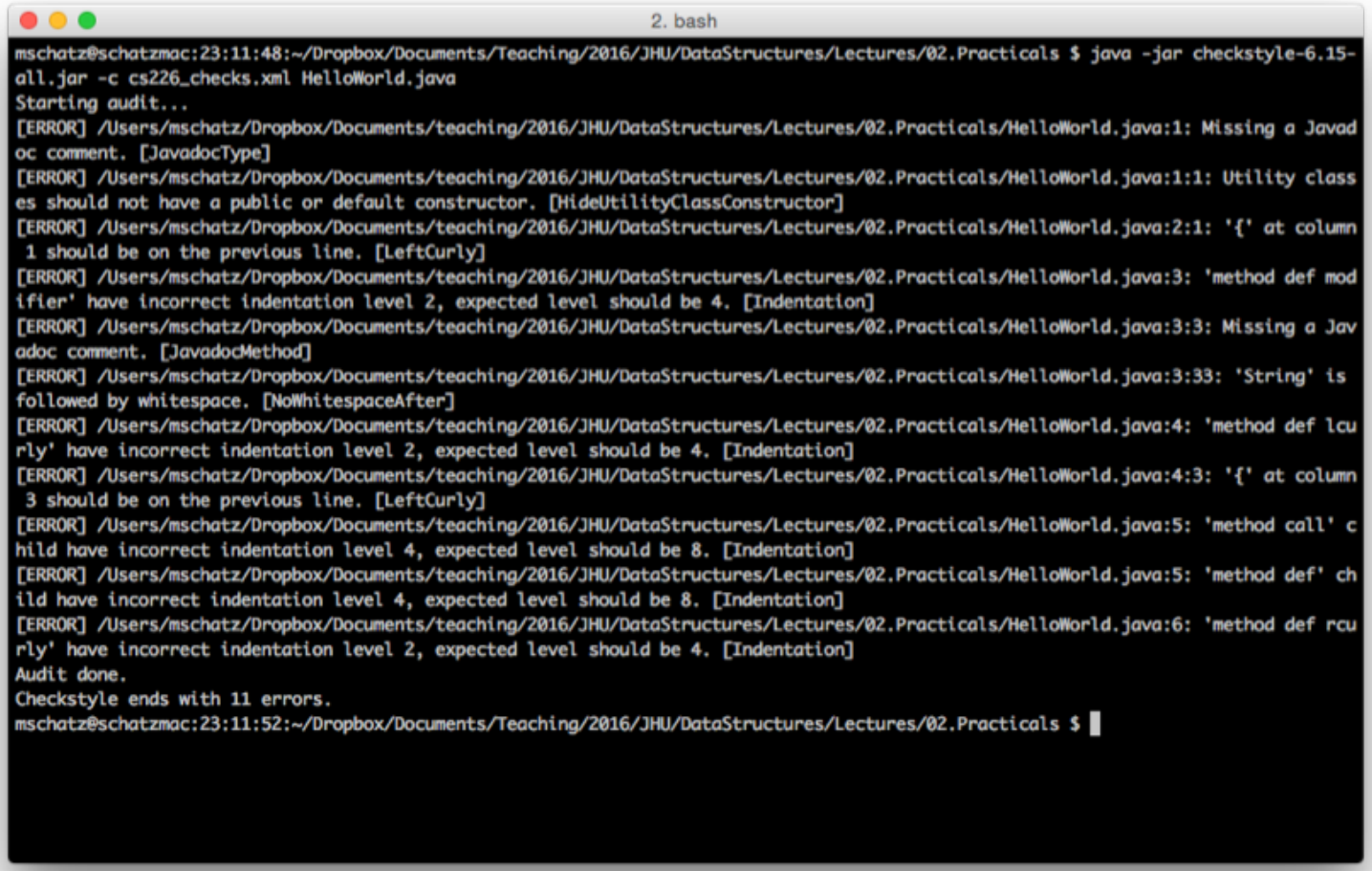
Problem 3: Sparse Arrays (35%)

A **sparse** array is an array in which **relatively few** positions have values that differ from the initial value set when the array was created. For sparse arrays, it is wasteful to store the value of **all** positions explicitly since **most of them never change** and take the default value of the array. Instead, we want to store positions that **have actually been changed**.

For this problem, write a class SparseArray that implements the Array interface we developed in lecture (the same interface you used for Problem 1 above). **Do not modify the Array interface in any way!** Instead of using a plain Java array like we did for SimpleArray, your SparseArray should use a **linked list** of Node objects to store values, similar to the ListArray from lecture (and available in github). However, your nodes no longer store just the **data** at a certain position, they also store **the position itself**!

Introduction to Checkstyle

http://checkstyle.sourceforge.net/

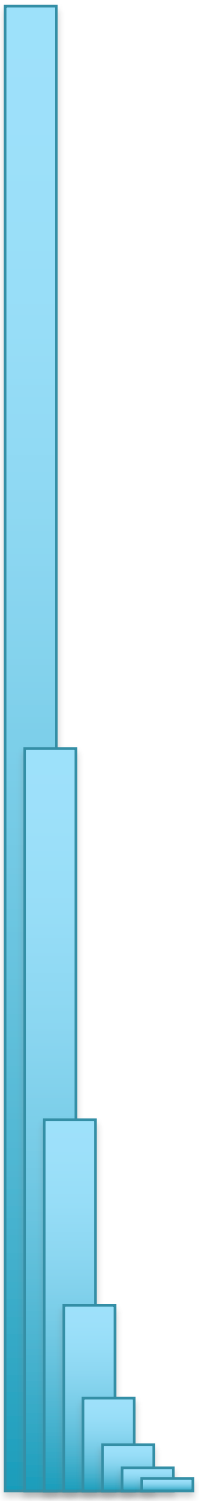



```
$ java -jar datastructures2018/resources/checkstyle-8.12-all.jar \
       -c datastructures2018/resources/cs226_checks.xml HelloWorld.java
```

```
# Add to bashrc:
alias check='java -jar datastructures2018/resources/checkstyle-8.12-all.jar \
       -c datastructures2018/resources/cs226_checks.xml'
```

Agenda

1. *Review HW2*
2. *Introduce HW3*
3. *Recap on Stacks*
4. *Queues*



Assignment 3: Due Friday Sept 28 @ 10pm

https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment03/README.md

Assignment 3: Assorted Complexities

Out on: September 21, 2018
Due by: September 28, 2018 before 10:00 pm
Collaboration: None
Grading:

Functionality 60% (where applicable)
Solution Design and README 10% (where applicable)
Style 10% (where applicable)
Testing 10% (where applicable)

Overview

The third assignment is mostly about sorting and how fast things go. You will also write yet another implementation of the Array interface to help you analyze how many array operations various sorting algorithms perform.

Note: The grading criteria now include 10% for unit testing. This refers to JUnit 4 test drivers, not some custom test program you hacked. The problems (on this and future assignments) will state whether you are expected to produce/improve test drivers or not.

Assignment 3: Due Friday Sept 28 @ 10pm

https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment03/README.md

Problem 1: Arrays with Statistics (30%)

Your first task for this assignment is to develop a new kind of `Array` implementation that keeps track of how many read and write operations have been performed on it. Check out the `Statable` interface first, reproduced here in compressed form (be sure to use and read the full interface available in github):

```
public interface Statable {
    void resetStatistics();
    int numberOfReads();
    int numberOfWrites();
}
```

This describes what we expect of an object that can collect statistics about itself. After a `Statable` object has been "in use" for a while, we can check how many read and write operations it has been asked to perform. We can also tell it to "forget" what has happened before and start counting both kinds of operations from zero again.

Assignment 3: Due Friday Sept 28 @ 10pm

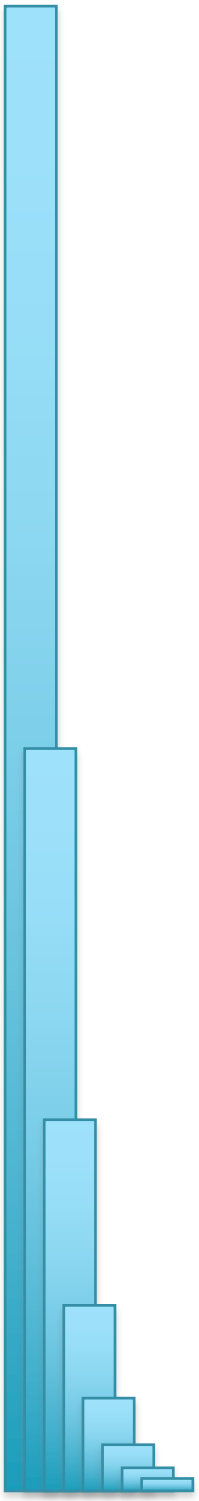
https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment03/README.md

Problem 1: Arrays with Statistics (30%)

You need to develop a class `StatableArray` that extends our dear old `SimpleArray` and also implements the `Statable` interface; yes, both at the same time. When a `StatableArray` is created, you initialize internal counters to keep track of the number of read and write operations it has been asked to perform so far; obviously both counts start at zero. ***Each time an operation is performed on the `StatableArray` object, you need to increment the relevant counter by one and invoke the actual operation in the super class using Java's super keyword.***

Don't forget that your constructor for `StatableArray` will also have to invoke the `SimpleArray` constructor!

Consider a freshly constructed `StatableArray` object. It would return 0 for both `numberOfReads` and `numberOfWrites`. Now imagine we call the length operation followed by three calls to the get operation. At this point, our object would return 4 for `numberOfReads` but still 0 for `numberOfWrites`. If we now call the put operation twice, the object would return 2 for `numberOfWrites` but still 4 for `numberOfReads`.



Assignment 3: Due Friday Sept 28 @ 10pm

https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment03/README.md

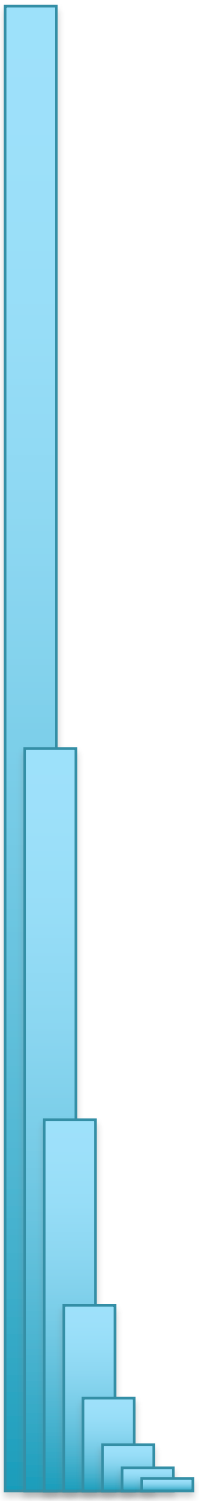
Problem 1: Arrays with Statistics (30%)

You need to write JUnit 4 test cases for StatableArray. Your focus should be on the Statable aspect of the class, but you will need to call Array methods to trigger the various possible outcomes. Call the file with your test cases StatableArrayTest.java please.

Hints

- You can get by with the basic @Before and @Test annotations provided by JUnit, nothing fancier than that is required.

- Since the Statable interface doesn't use exceptions, you don't have to test any preconditions either; remember that your focus is on Statable and not on Array for which (presumably) some other test exists.



Assignment 3: Due Friday Sept 28 @ 10pm

https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment03/README.md

Problem 2: All Sorts of Sorts (50%)

Your second task for this assignment is to explore some of the basic sorting algorithms and their analysis. All of these algorithms are quadratic in terms of their asymptotic performance, but they nevertheless differ in their actual performance.

We'll focus on the following three algorithms:

- Bubble Sort (with the "stop early if no swaps" extension)
- Selection Sort
- Insertion Sort

The github repo contains a basic framework for evaluating sorting algorithms.

Assignment 3: Due Friday Sept 28 @ 10pm

https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment03/README.md

Problem 2: All Sorts of Sorts (50%)

The github repo contains a basic framework for evaluating sorting algorithms. You'll need a working `StatableArray` class from Problem 1, and you'll need to understand the following interface as well (again compressed, be sure to to use and read the full interface:

```
public interface SortingAlgorithm<T extends Comparable<T>> {
     void sort(Array<T> array);
     String name();
 }
```

Assignment 3: Due Friday Sept 28 @ 10pm

https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment03/README.md

Problem 2: All Sorts of Sorts (50%)

Let's look at the simple stuff first:

- ***An object is considered an algorithm suitable for sorting*** in this framework if (a) we can ask it to sort a given Array and (b) we can ask it for its name (e.g. "Insertion Sort").

- ***The more complicated stuff is at the top:*** The use of extends inside the angle brackets means that any type T we want to sort must implement the interface Comparable as well. It obviously can't just be any old type, it must be a type for which the expression "a is less than b" actually makes sense. Using Comparable in this form is Java's way of saying that we can order the objects; you should definitely read up on the details here!

- ***As an example for all this, we have provided an implementation*** of SelectionSort on Piazza already. (Actually, there are also two other algorithms, NullSort and GnomeSort, just so you start out with a few to run right away.)

Assignment 3: Due Friday Sept 28 @ 10pm

https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment03/README.md

Problem 2: All Sorts of Sorts (50%)

You need to write classes implementing BubbleSort and InsertionSort for this problem. Just like our example algorithms, your classes have to implement the SortingAlgorithm interface.

All of this should be fairly straightforward once you get used to the framework. Speaking of the framework, the way you actually "run" the various algorithms is by using the PolySort.java program we've provided as well. You should be able to compile and run it without yet having written any sorting code yourself.

Here's how:

```
$ java PolySort 4000 <random.data

 Algorithm            Sorted?  Size        Reads        Writes       Seconds

 Null Sort            false    4,000       0            0            0.000007
 Gnome Sort           true     4,000       32,195,307   8,045,828    0.243852
 Selection Sort       true     4,000       24,009,991   7,992        0.252085
```

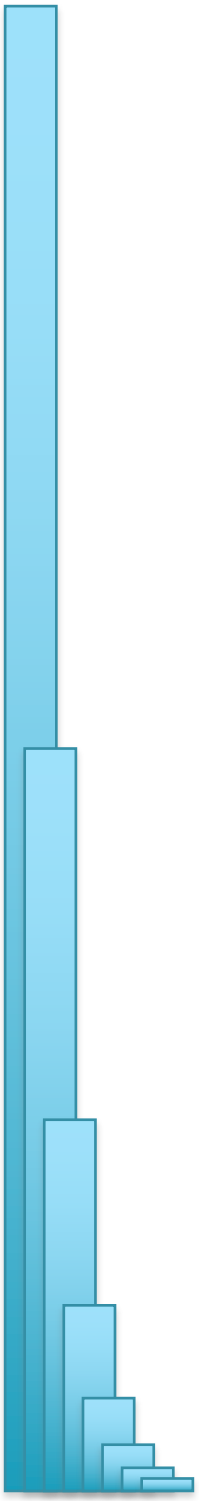
Assignment 3: Due Friday Sept 28 @ 10pm

https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment03/README.md

Problem 2: All Sorts of Sorts (50%)

This will read the first 4000 strings from the file random.data and sort them using all available algorithms. As you can see, the program checks if the algorithm actually worked (Sorted?) and reports how many operations of the underlying StatableArray were used in order to perform the sort (Reads, Writes). Finally, the program also prints out long it took to sort the array (Seconds) but that number will vary widely across machines so you can really only use it for relative comparisons on the machine actually running the experiment.

However, the main point of all this is not the coding work. Instead, the main point is to evaluate and compare the sorting algorithms on different sets of data. We've provided three sets of useful test data on github and you can use the command line argument to vary how much of it is used (thereby changing the size of the problem). You should try to quantify how the various algorithms differ and explain why they differ as well (i.e. what about a given algorithm makes it better or worse than another one for a given data set).



Assignment 3: Due Friday Sept 28 @ 10pm

https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment03/README.md

Problem 2: All Sorts of Sorts (50%)

In your README file you should describe the series of experiments you ran, what data you collected, and what your conclusions about the performance of these algorithms are.

Some ideas for what to address:

- Does the actual running time correspond to the asymptotic complexity as you would expect?
- What explains the practical differences between these algorithms?
- Does it matter what kind of data (random, already sorted in ascending order, sorted in descending order) you are sorting?
- ***Just to be clear: Yes, we'll need the code, and it should be up to the usual standards. But the "report" you put in your README is just as important as the code!***

Assignment 3: Due Friday Sept 28 @ 10pm

https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment03/README.md

Problem 3: Analysis of Selection Sort (20%)

Your final task for this assignment is to analyze the following selection sort algorithm theoretically (without running it) in detail (without using O-notation).

Here's the code, and you must analyze exactly this code (the line numbers are given so you can refer to them in your writeup for this problem):

```
 1: public static void selectionSort(int[] a) {
 2:     for (int i = 0; i < a.length - 1; i++) {
 3:         int min = i;
 4:         for (int j = i + 1; j < a.length; j++) {
 5:             if (a[j] < a[min]) {
 6:                 min = j;
 7:             }
 8:         }
 9:         int t = a[i]; a[i] = a[min]; a[min] = t;
10:     }
11: }
```

Assignment 3: Due Friday Sept 28 @ 10pm

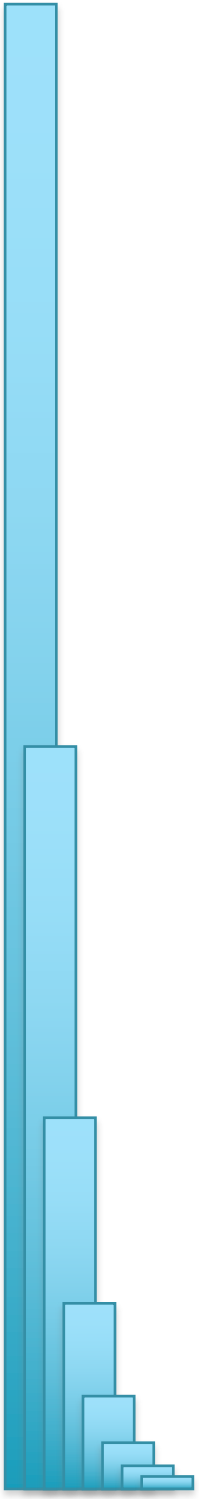
https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment03/README.md

Problem 3: Analysis of Selection Sort (20%)

You need to determine exactly how many comparisons C(n) and assignments A(n) are performed by this implementation of selection sort in the worst case. Both of those should be polynomials of degree 2 since you know that the asymptotic complexity of selection sort is O(n^2). (As usual we refer to the size of the problem, which is the length of the array to be sorted here, as "n" above.)

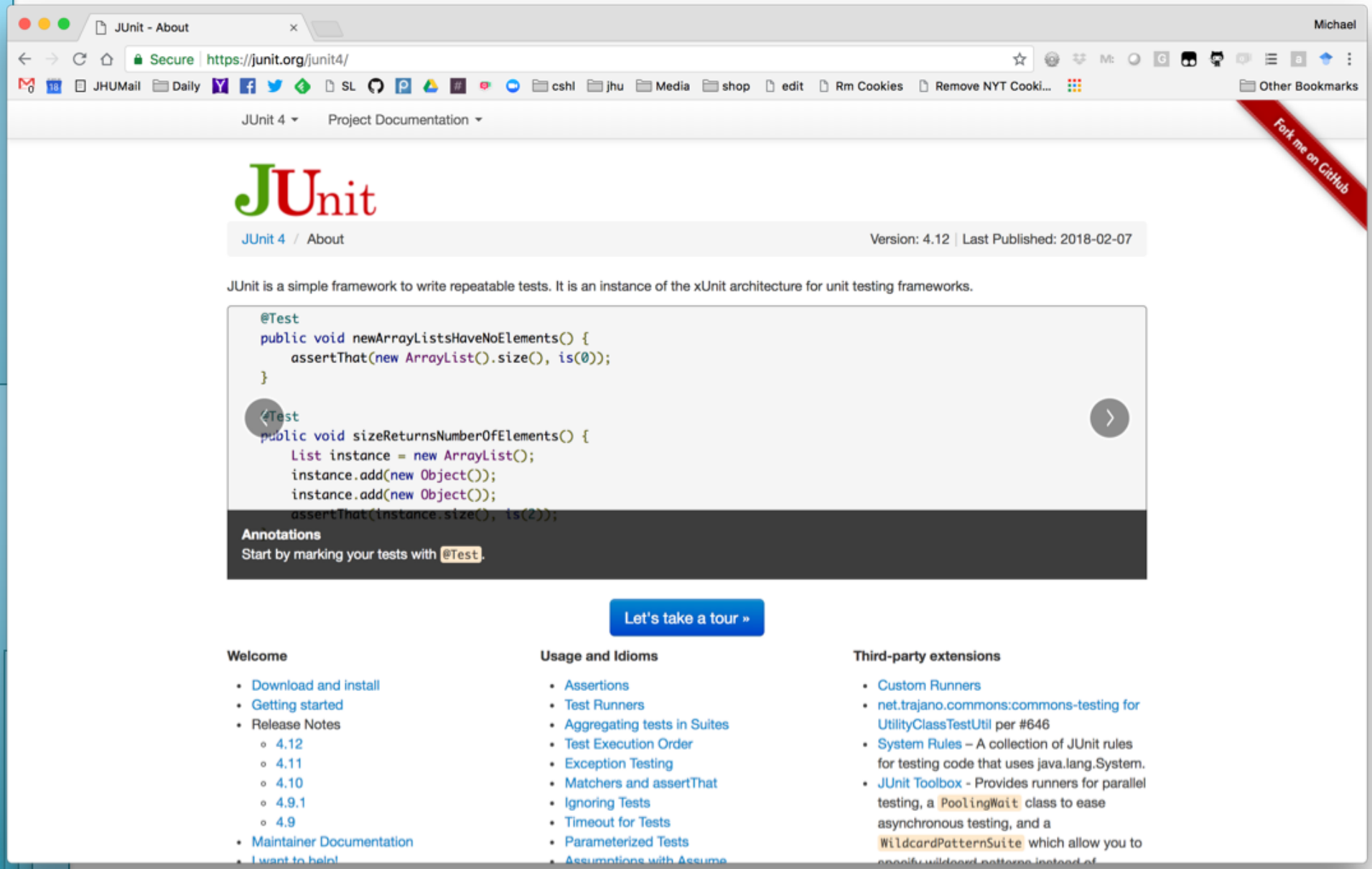
Important:

- Don't just state the polynomials, your writeup has to explain how you derived them! Anyone can google for the answer, but you need to convince us that you actually did the work!



Introducing JUnit

Introducing JUnit



JUnit According to Peter ☺

So why use a testing framework like JUnit instead of writing the tests like we did so far, using Java's assert instruction and a main method with the test code in it?

- ***For one thing, JUnit allows you to modularize your tests better.*** It's not uncommon for large software projects to have just as much testing code as actual program code, and so the principles you use to make regular code easier to read (splitting things into methods and classes, etc.) should also apply to test code.

- ***Also, JUnit allows you to run all your test cases every time***, it doesn't stop at the first failing test case like assert does. This way you can get feedback about multiple failed tests all at once.

- ***Finally, lots of companies expect graduates to have some experience with testing frameworks***, so why not pick it up now? Note that testing is not just for software developers anymore, increasingly people working with software developers but who are themselves not software developers will be asked to contribute to testing a certain application being developed by their company.

So it's a really useful skill to have on your list.

JUnit According to Peter ☺

So why use a testing framework like JUnit instead of writing the tests like we did so far, using Java's assert instruction and a main method with the test code in it?

- ***For one thing, JUnit allows you to modularize your tests better.*** It's not uncommon for large software projects to have just as much testing c
r
s
tc.)

From HW3 on out, when we say "write test cases" as part of an assignment, we mean "write JUnit 4 test cases" as described here.

- A
s
f
't

- I
ce
With testing frameworks, so why not pick it up now? Note that testing is not just for software developers anymore, increasingly people working with software developers but who are themselves not software developers will be asked to contribute to testing a certain application being developed by their company.

So it's a really useful skill to have on your list.

Use it!

For JUnit 4 we import the "Test" annotation from the framework with:

```
import org.junit.Test;
import static org.junit.Assert.assertEquals;
```

and then write our test methods using that annotation:

```
@Test
public void testThisAndThat() {
    //set up thisAndThat
    assertEquals(thisAndThat.method() == expectedResult);
}
```

- ***Any methods tagged with @Test annotation will be run by JUnit as a test case***
- ***Use assertEquals and other keywords to check the returned results match expected results. Make sure to test both positive (correct value, correct length, etc) and negative results (exceptions correctly thrown, other error conditions detected)***
- ***Write code, compile, and then run the special driver program to generate a report***

TestSimpleArray.java

```
import org.junit.Test;
import org.junit.BeforeClass;
import static org.junit.Assert.assertEquals;

public class TestSimpleArray {
    static Array<String> shortArray;

    @BeforeClass
    public static void setupArray() throws LengthException {
       shortArray = new SimpleArray<String>(10, "Bla");
    }

    @Test
    public void newArrayLengthGood() throws LengthException {
        assertEquals(10, shortArray.length());
    }

    @Test
    public void newArrayInitialized() throws LengthException, IndexException {
        for (int i = 0; i < shortArray.length(); i++) {
            assertEquals("Bla", shortArray.get(i));
        }
    }

    @Test(expected=IndexException.class)
    public void IndexDetected() throws IndexException {
        shortArray.put(shortArray.length(), "Paul");
    }
}
```

TestSimpleArray.java

```
import org.junit.Test;
import org.junit.BeforeClass;
import static org.junit.Assert.assertEquals;

public class TestSimpleArray {
    static Array<String> shortArray;

    @BeforeClass
    public static void setupArray() throws LengthException {
       shortArray = new SimpleArray<String>(10, "Bla");
    }

    @Test
    public void newArrayLengthGood() throws LengthException {
        assertEquals(10, shortArray.length());
    }

    @Test
    public void newArrayInitialized() throws LengthException, IndexException {
        for (int i = 0; i < shortArray.length(); i++) {
            assertEquals("Bla", shortArray.get(i));
        }
    }

    @Test(expected=IndexException.class)
    public void IndexDetected() throws IndexException {
        shortArray.put(shortArray.length(), "Paul");
    }
}
```

@BeforeClass causes the method to be run once before any of the test methods in the class

Check the results with assertEquals, or listing the expected exception

Running JUnit

```
// Step 0: Download junit-4.12.jar and hamcrest-core-1.3.jar
// Jar files are bundles of java classes ready to run

// Step 1: Compile your code as usual and checkstyle
$ javac –Xlint:all SimpleArray.java
$ check SimpleArray.java

// Step 2: Compile tests, but not checkstyle for these :)
$ javac -cp .:junit-4.12.jar -Xlint:all TestSimpleArray.java

// Step 3: Run Junit on your TestProgram. Notice that
org.junit.runner.JUnitCore is the main code we run, and
TestSimpleArray is just a parameter to it
$ java -cp .:junit-4.12.jar:hamcrest-core-1.3.jar \
     org.junit.runner.JUnitCore TestSimpleArray
JUnit version 4.12
...
Time: 0.011

OK (3 tests)

// Hooray, everything is okay!
```

Hint: save commands to a file!
chmod +x tester.sh
./tester.sh

-cp sets the class path. This tells Java where to find the relevant code needed for compiling and running

TestSimpleArray.java

```
import org.junit.Test;
import org.junit.BeforeClass;
import static org.junit.Assert.assertEquals;

public class TestSimpleArray {
    static Array<String> shortArray;

    @BeforeClass
    public static void setupArray() throws LengthException {
       shortArray = new SimpleArray<String>(10, "Bla");
    }

    @Test
    public void newArrayLengthGood() throws LengthException {
        assertEquals(10, shortArray.length());
    }

    @Test
    public void newArrayInitialized() throws LengthException, IndexException {
        for (int i = 0; i < shortArray.length(); i++) {
            assertEquals("Bla", shortArray.get(i));
        }
    }

    @Test(expected=IndexException.class)
    public void IndexDetected() throws IndexException {
        shortArray.put(shortArray.length(), "Paul");
    }
}
```

What other tests should we add?

Advanced Testing

Example tests:

```
assertEquals(expectedValue, actualValue);
assertTrue(booleanExpression);  // also assertFalse
assertNotNull(someObject);      // also assertNull
```

The assertion methods are also overloaded with a version that has a first parameter string to detail the cause of an error in the case that the assertion fails. This is strongly recommended to give valuable feedback to the junit user when running the tests.

```
assertEquals("detailed message if fails", expectedValue, actualValue);
assertFalse("something going on", booleanExpression);
```

All of the test annotation types (Test, Before, BeforeClass, etc.) and assertion types (assertEquals, assertTrue, assertNotNull, etc.) must be imported before they can be used.

More Advanced Testing

If you prefer to put the jar files in a parent folder (and then actual project folders in subdirectories), then from the project folders you would list the classpaths for the jars with "../" prepended to them, like this:

```
$ javac -cp junit-4.12.jar:.  TestArray.java
$ java -cp junit-4.12.jar:hamcrest-core-1.3.jar:. \
    org.junit.runner.JUnitCore TestArray
```

Note: if you are doing this via command-line on a windows machine, you must use a ';' instead of a ':' to separate the paths:

```
$ javac -cp junit-4.12.jar;. -Xlint:all TestArray.java
$ java -cp junit-4.12.jar;. org.junit.runner.JUnitCore TestArray
```

If you're using an IDE (Eclipse/jGRASP), doing JUnit testing is even easier. You can import a test file into your project, or create a test file. Then select that file, and choose Run As JUnit Test.

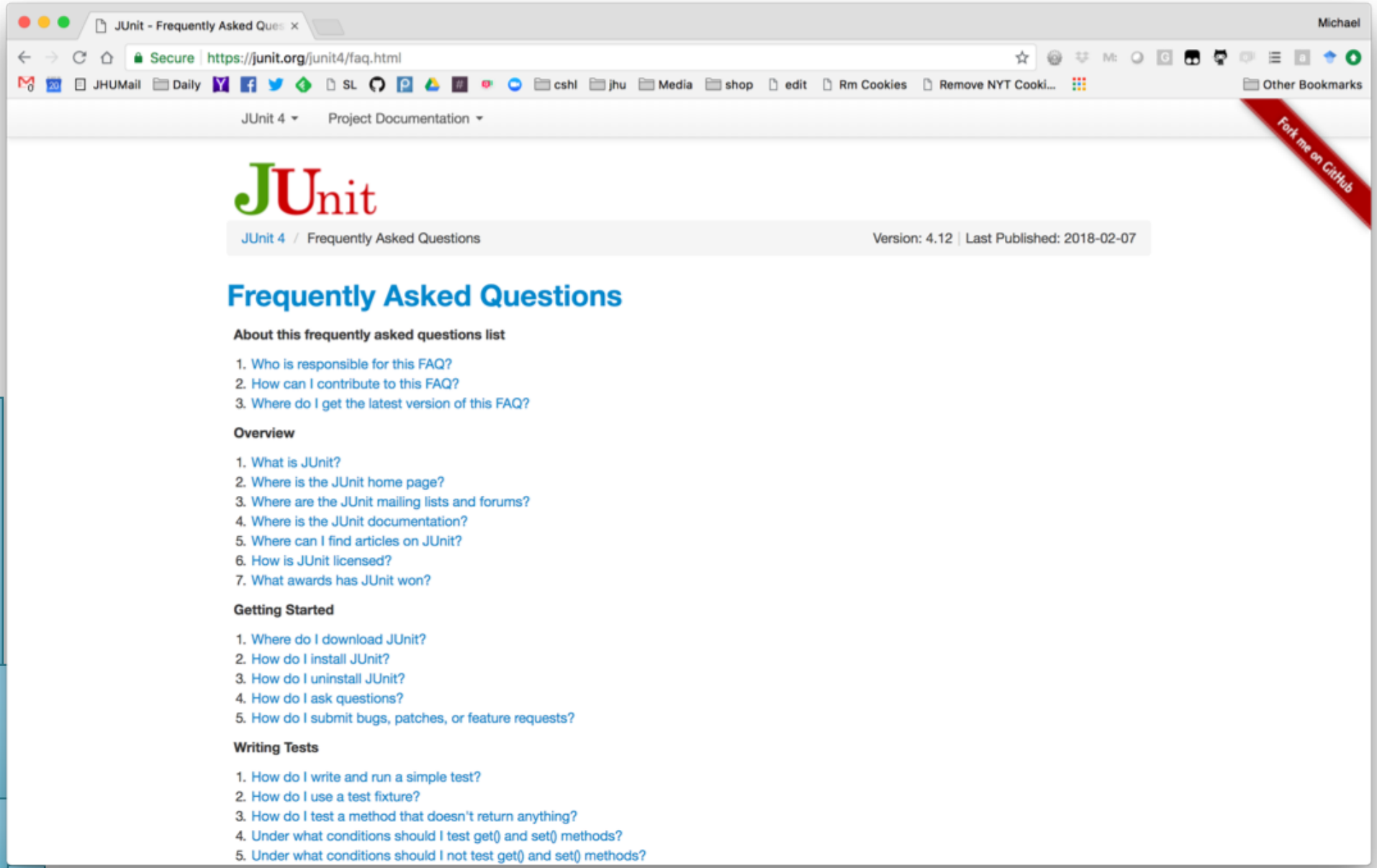
Guidelines

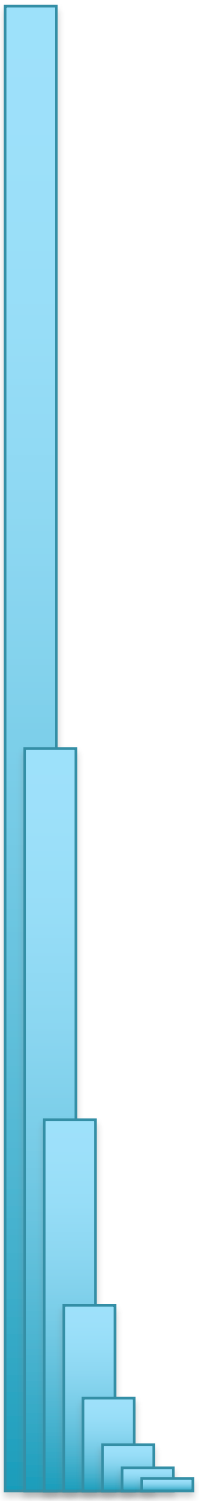
1. ***Every Method should be tested for correct outputs***
   - Try simple and complex examples (different lengths of arrays, etc)
   - Private methods can be tested implicitly, but the entire public interface should be evaluated

2. ***Every exception and error condition should also be tested***
   - This is how the ADT contract will be enforced

3. ***Write the test cases first, that way you will know when you are done***

More Help


JUnit - Frequently Asked Ques
Michael
Secure https://junit.org/junit4/faq.html
JHUMail Daily SL cshl jhu Media shop edit Rm Cookies Remove NYT Cooki... Other Bookmarks
JUnit 4 Project Documentation
Fork me on GitHub
JUnit
JUnit 4 / Frequently Asked Questions
Version: 4.12 | Last Published: 2018-02-07
Frequently Asked Questions
About this frequently asked questions list
1. Who is responsible for this FAQ?
2. How can I contribute to this FAQ?
3. Where do I get the latest version of this FAQ?
Overview
1. What is JUnit?
2. Where is the JUnit home page?
3. Where are the JUnit mailing lists and forums?
4. Where is the JUnit documentation?
5. Where can I find articles on JUnit?
6. How is JUnit licensed?
7. What awards has JUnit won?
Getting Started
1. Where do I download JUnit?
2. How do I install JUnit?
3. How do I uninstall JUnit?
4. How do I ask questions?
5. How do I submit bugs, patches, or feature requests?
Writing Tests
1. How do I write and run a simple test?
2. How do I use a test fixture?
3. How do I test a method that doesn't return anything?
4. Under what conditions should I test get() and set() methods?
5. Under what conditions should I not test get() and set() methods?



Stacks

Stacks

Stacks are very simple but surprisingly useful data structures for storing a collection of information

• Any number of items can be stored, but you can only manipulate the top of the stack:

- ***Push***: adds a new element to the top
- ***Pop***: takes off the top element
- ***Top***: Lets you peek at top element's value without removing it from stack

Many Applications

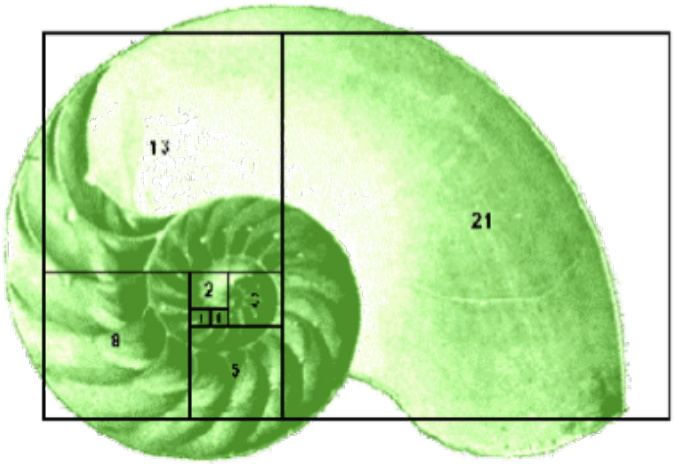
• In hardware call stack
• Memory management systems
• Parsing arithmetic instructions:

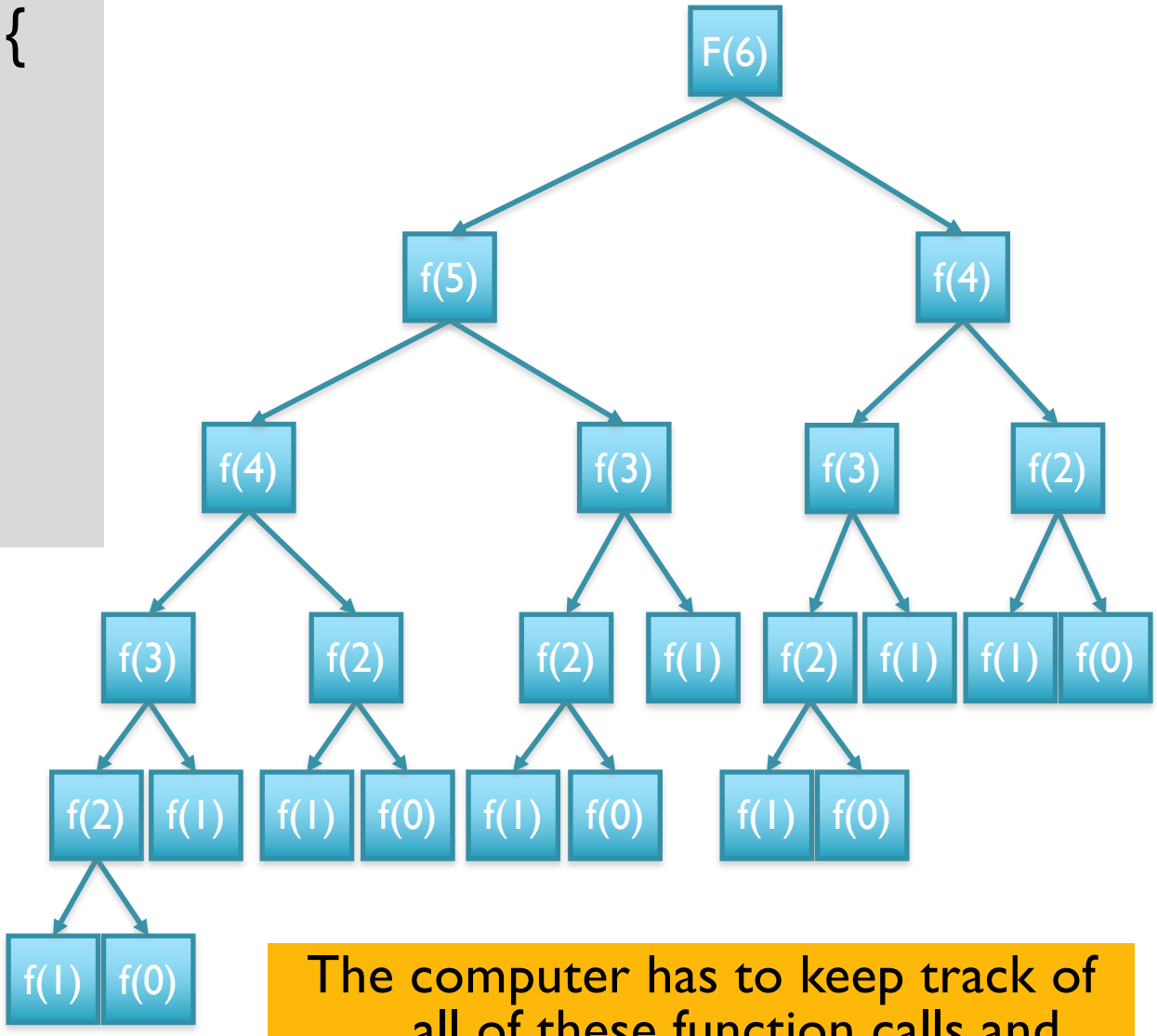
((x+3) / (x+9)) * (42 * sin(x))

• Back-tracing, such as searching within a maze

Fibonacci Sequence

```
public static int fib(int n) {
   if (n <= 1) {
     return 1;
   }
   return fib(n-1) +
            fib(n-2);
 }
```



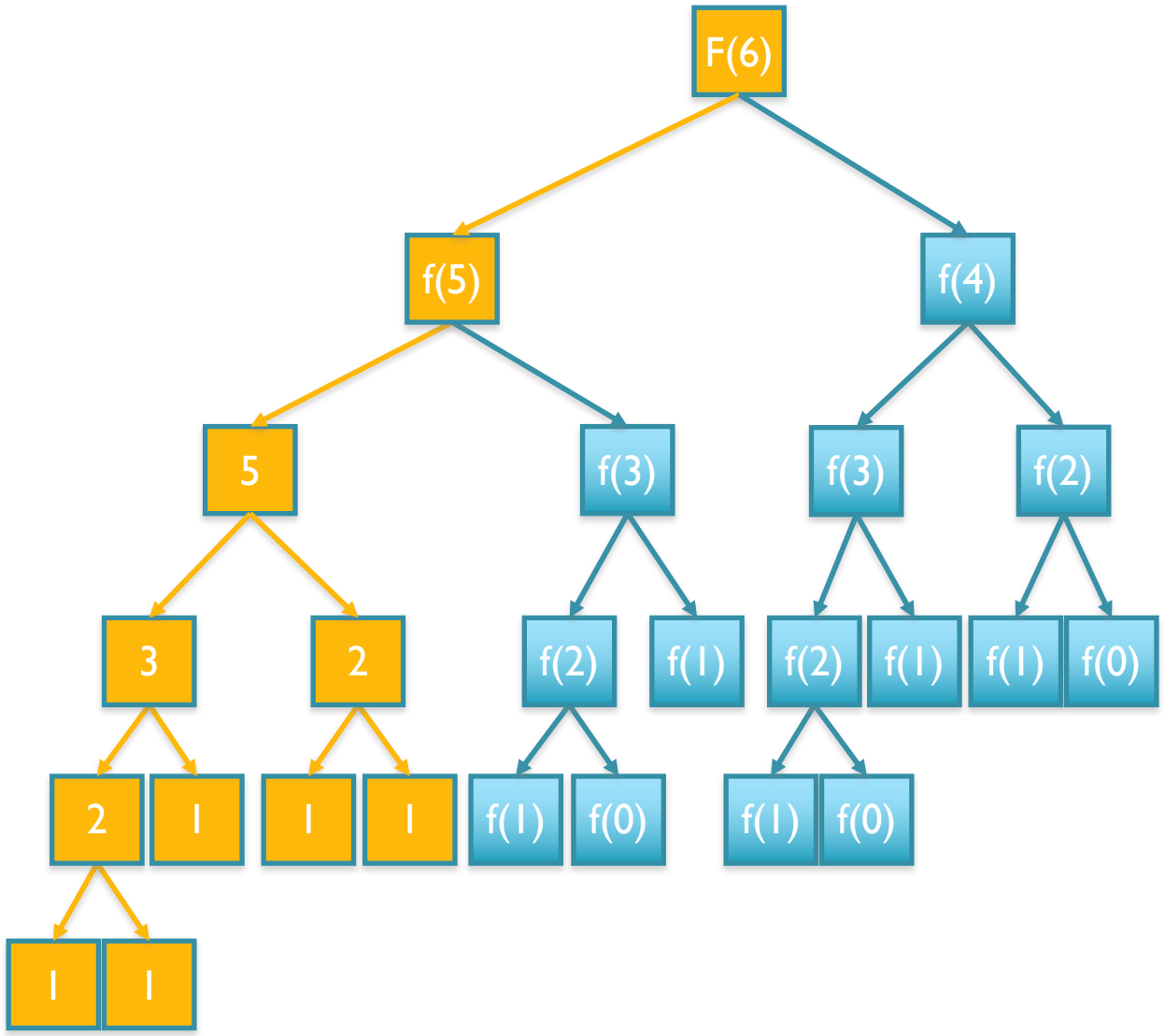


The computer has to keep track of all of these function calls and keep everything in order!

Fibonacci Sequence

Equivalent to running through a maze and always keeping your right hand on the wall

Notice we only look at the top of the stack to keep track of where we have been and were we should go next!



F(5)
a = F(4)

F(6)
a = F(5)

RPN Notation

We usually write arithmetic with "infix" notation:

1 + 2 * 3 => 7

But this can be tricky to parse because of complex order of operations

To simplify this, many systems use "Reverse Polish Notation" (RPN) invented by logician Jan Łukasiewicz in the 1920s:

- Operator follows the operands:

3 4 + => 7

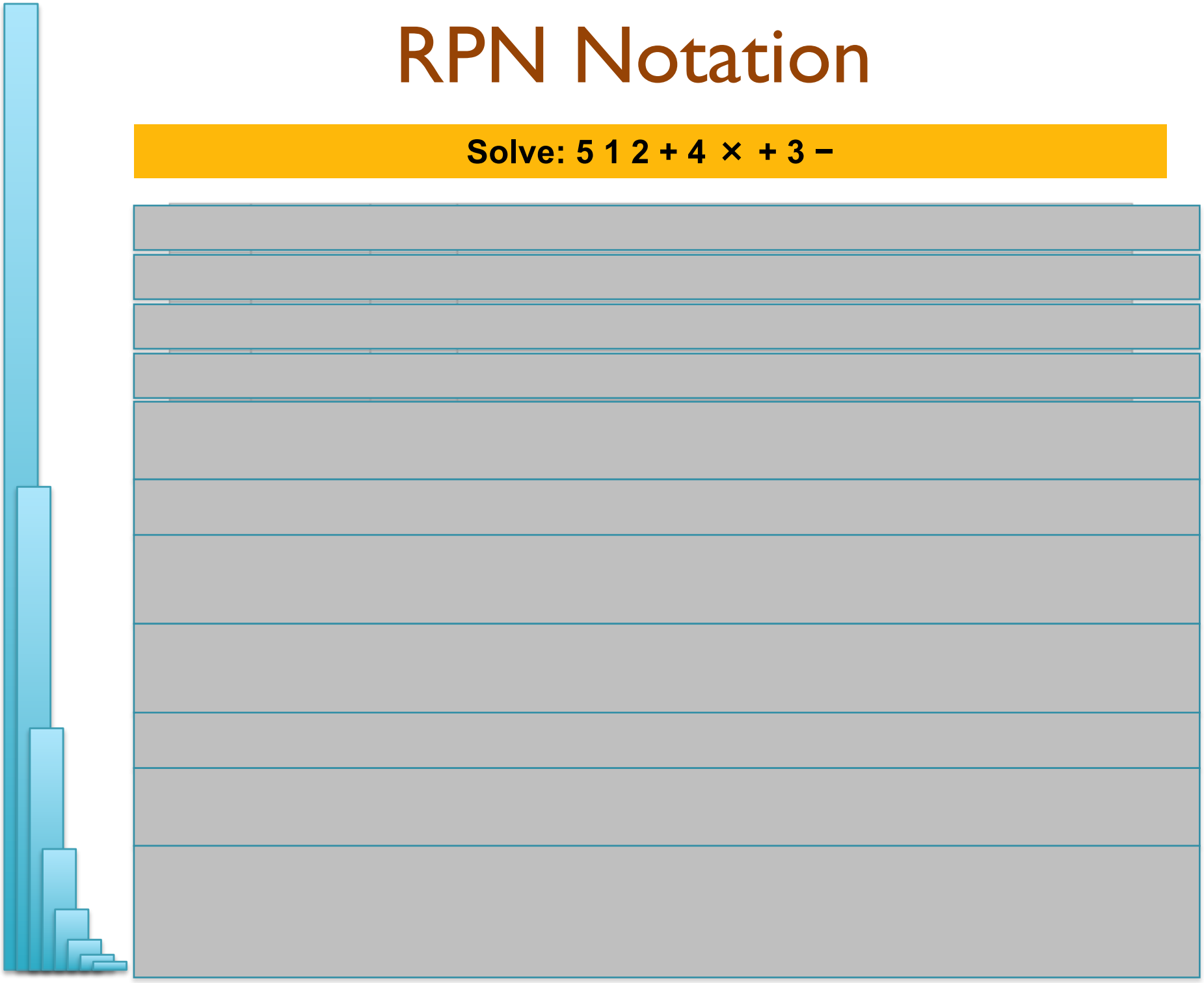
Such systems often use a stack to manage the calculation:

- Numbers pushed onto stack
- Operator means pop off the last two values from the stack, calculate the results, and then push result back onto stack

1 2 3 * + => 1 (2 3 *) + => 1 6 + => 7

RPN Notation

Solve: 5 1 2 + 4 × + 3 −



RPN Notation

Solve: 5 1 2 + 4 × + 3 −

Input	Action	Stack	Notes
5	Operand	5	Push onto stack.
1	Operand	1 5	Push onto stack.
2	Operand	2 1 5	Push onto stack.
+	Operator	3 5	Pop the two operands (1, 2), calculate (1 + 2 = 3) and push onto stack.
4	Operand	4 3 5	Push onto stack.
×	Operator	12 5	Pop the two operands (3, 4), calculate (3 * 4 = 12) and push onto stack.
+	Operator	17	Pop the two operands (5, 12), calculate (5 + 12 = 17) and push onto stack.
3	Operand	3 17	Push onto stack.
−	Operator	14	Pop the two operands (17, 3), calculate (17 - 3 = 14) and push onto stack.
	Result	14	

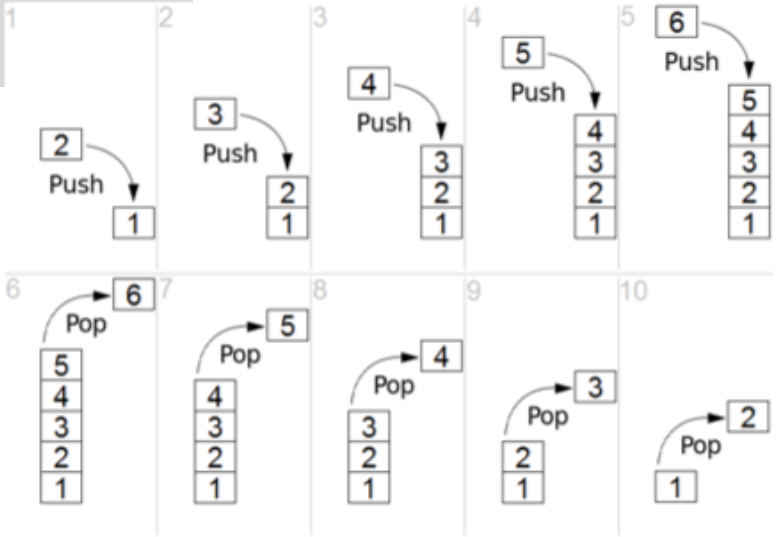
https://en.wikipedia.org/wiki/Reverse_Polish_notation

Stack Interface

```
public interface Stack<T> {
    // checks if empty
    boolean empty();

    // peeks at top value without removing
    T top() throws EmptyException;

    // removes top element
    void pop() throws EmptyException;

    // adds new element to top of stack
    void push(T t);
}
```

How would you implement this interface?

Why?



```java
/**
    Stack implemented using a linked list.

    All operations take O(1) time in the worst case; however
    each push() results in a new object being allocated which
    may be inappropriate for some applications.

    @param <T> Element type.
*/
public class ListStack<T> implements Stack<T> {
    private static class Node<T> {
        Node<T> next;
        T data;
    }

    private Node<T> first;

    /**
        Create an empty stack.
    */
    public ListStack() {
    }

    @Override
    public T top() throws EmptyException {
        try {
            return this.first.data;
        } catch (NullPointerException e) {
            throw new EmptyException();
        }
    }

    @Override
    public void pop() throws EmptyException {
        try {
            this.first = this.first.next;
        } catch (NullPointerException e) {
            throw new EmptyException();
        }
    }

    @Override
    public void push(T t) {
        Node<T> n = new Node<T>();
        n.data = t;
        n.next = this.first;
        this.first = n;
    }

    @Override
    public boolean empty() {
        return this.first == null;
    }

    @Override
    public String toString() {
        String s = "[";
        for (Node<T> n = this.first; n != null; n = n.next) {
            s += n.data.toString();
            if (n.next != null) {
                s += ", ";
            }
        }
        s += "]";
        return s;
    }
}
```

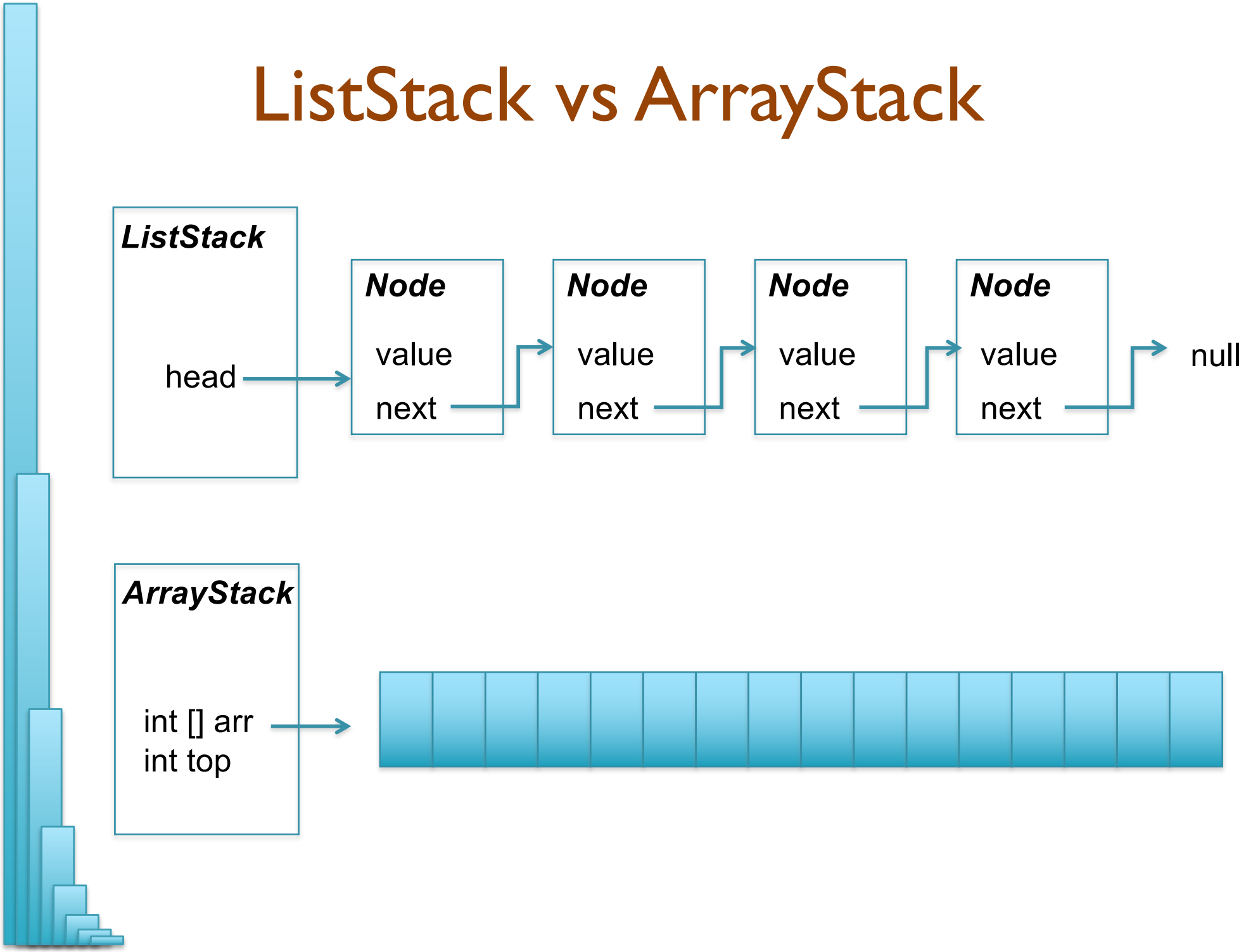
ListStack

vs.

ArrayStack

```java
/**
    Stack implemented using a growing array.

    All operations except push() take O(1) time in the worst
    case; push() takes O(1) amortized time because the array
    may need to be resized; however, compared to ListStack,
    fewer push() operations result in objects being allocated.

    @param <T> Element type.
*/
public class ArrayStack<T> implements Stack<T> {
    private T[] data;
    private int used;

    /**
        Create an empty stack.
    */
    public ArrayStack() {
        this.data = (T[]) new Object[1];
    }

    @Override
    public T top() throws EmptyException {
        if (this.empty()) {
            throw new EmptyException();
        }
        return this.data[this.used - 1];
    }

    @Override
    public void pop() throws EmptyException {
        if (this.empty()) {
            throw new EmptyException();
        }
        this.used -= 1;
    }

    private boolean full() {
        return this.data.length == this.used;
    }

    private void grow() {
        T[] bigger = (T[]) new Object[this.data.length * 2];
        for (int i = 0; i < this.used; i++) {
            bigger[i] = this.data[i];
        }
        this.data = bigger;
    }

    @Override
    public void push(T t) {
        if (this.full()) {
            this.grow();
        }
        this.data[this.used] = t;
        this.used += 1;
    }

    @Override
    public boolean empty() {
        return this.used == 0;
    }

    @Override
    public String toString() {
        String s = "[";
        for (int i = this.used - 1; i >= 0; i--) {
            s += this.data[i].toString();
            if (i > 0) {
                s += ", ";
            }
        }
        s += "]";
        return s;
    }
}
```

ListStack vs ArrayStack



ArrayStack versus ListStack

ListStack has some nice properties:

- Unbounded: keep adding items to the stack until you run out of memory
- Push, pop, and top are all constant time
- (Pretty) Simple implementation

But also has significant overhead:

- You may end up dedicating more memory to node references (8 bytes each) than to the values that you are storing (4 bytes per int, 1 byte for char, etc)
- The values may be distributed all over RAM which can incur a penalty on some hardware
    - See computer architecture class for details

ArrayStack has some nice properties:

- Minimal overhead to storing items:
    - 8 bytes for a reference to the entire array with N elements
- (Pretty) simple implementation:
    - pop just decrements one number, top is a quick array lookup

But also has a significant challenge/limitation

- Sometimes push() may be impossible (throw a StackOverflow exception), or expensive (copy entire array to a new larger array)

Stack Interfaces

```
public interface BoundedStack<T> {
    void push(T t) throws FullStackException;
    void pop() throws EmptyStackException;
    T top() throws EmptyStackException;
    boolean empty();
    boolean full();
}

public interface UnboundedStack<T> {
    void push(T t);
    void pop() throws EmptyStackException;
    T top() throws EmptyStackException;
    boolean empty();
}
```

Could we have UnboundedStack extend BoundedStack or vice-versa?

Nah: BoundedStack extends Unbounded will have different semantics!

Unbounded extends Bounded will have weird methods/exceptions:
full() method or FullException for Unbounded

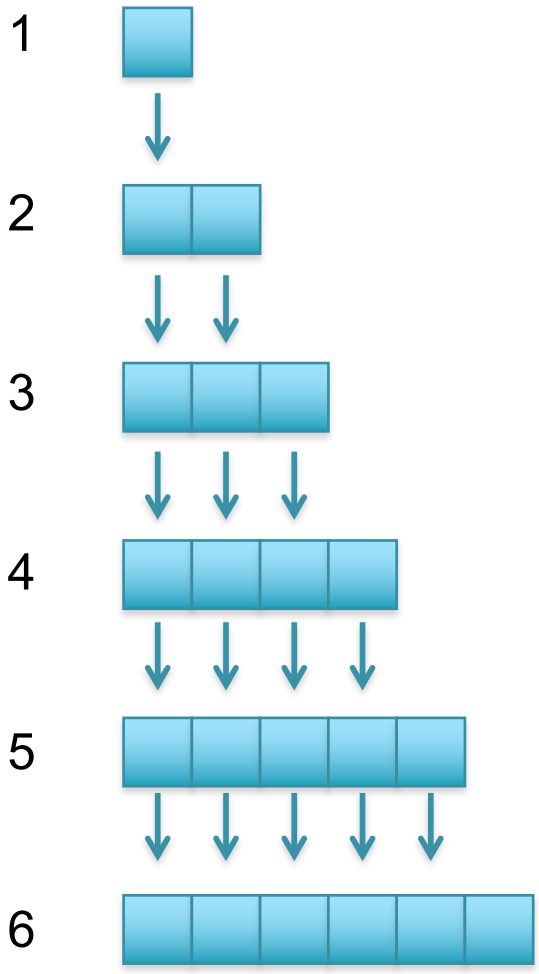
Stack Interfaces

```
public interface Stack<T> {
    void pop() throws EmptyStackException;
    T top() throws EmptyStackException;
    boolean empty();
}

public interface UnboundedStack<T> extends Stack<T> {
    void push(T t);
}

public interface BoundedStack<T> extends Stack<T> {
    void push(T t) throws FullStackException;
    boolean full();
}
```

This would work, although weird because push() has to be separately introduced

ArrayStack Growing

If the array size starts at 1, how expensive will it be to grow to 1M if we copy one element at a time?



1M push()s will require a total of

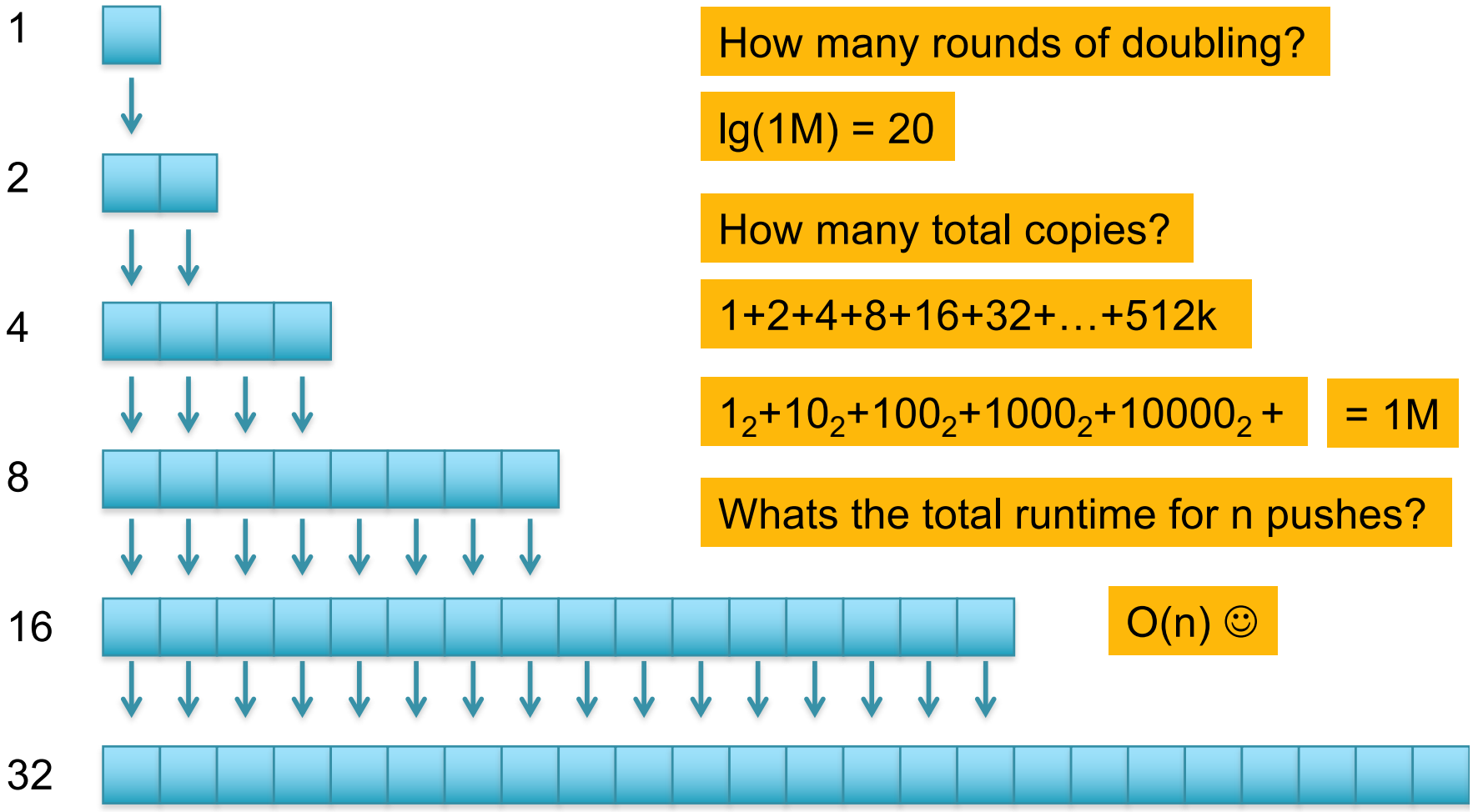
1+2+3+4+5+6+…+999,999 copies

= 0.5MM steps!

$O(n^2)$ performance ☹

ArrayStack Doubling

If the array size starts at 1, how expensive will it be to grow to 1M?
How many doublings will it take?
How many times will an item be copied?



How many rounds of doubling?

lg(1M) = 20

How many total copies?

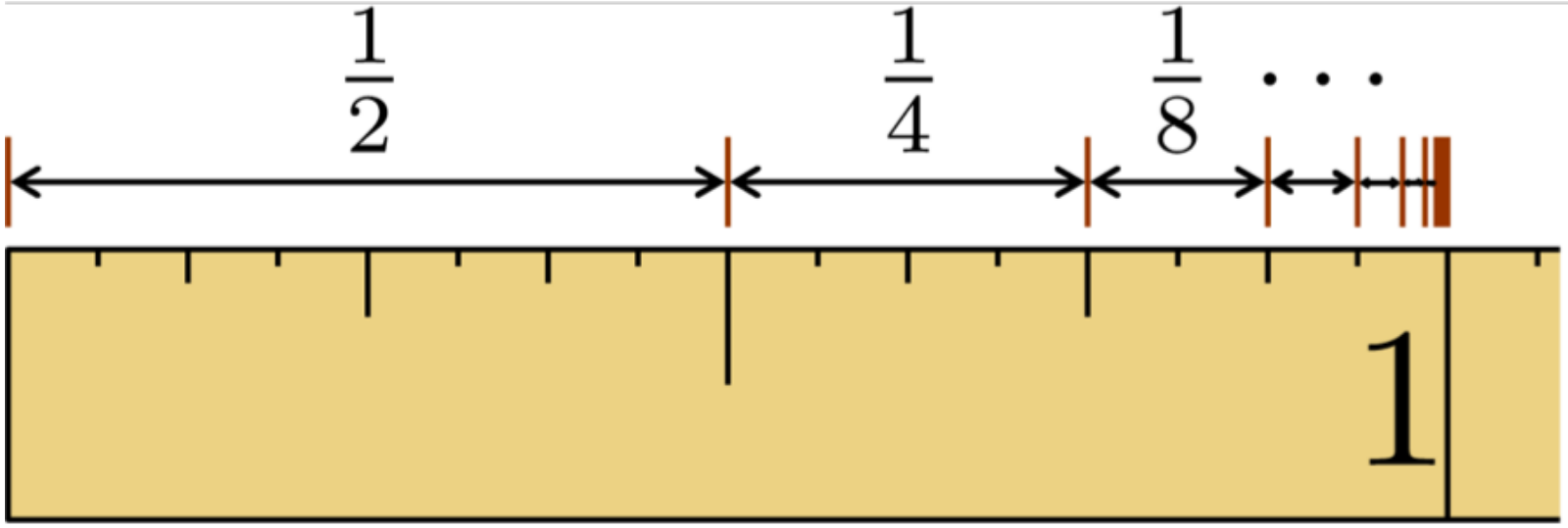
1+2+4+8+16+32+…+512k

$1_2+10_2+100_2+1000_2+10000_2+$ = 1M

Whats the total runtime for n pushes?

O(n) ☺

Geometric series



½ + ¼ + 1/8 + 1/16 + 1/32 + 1/64 + 1/128 … = 1

A geometric series is a series with a constant ratio between successive terms.

$$a + ar + ar^2 + ar^3 + \cdots + ar^{n-1} = \sum_{k=0}^{n-1} ar^k = a\left(\frac{1-r^n}{1-r}\right)$$

a = ½ r = ½ n = lg x

(½) [(1-1/2 $^{\lg n}$) / (1- ½)] = (½) [(1-0) / (½)] = (½) / (½) = 1

Amortized Analysis

The amortized cost per operation for a sequence of n operations is the total cost of the operations divided by n

Example: If we have 100 operations at cost 1, followed by one operation at cost 100, the amortized cost per operation is 200/101 < 2.
Note the worst case operation analysis yields 100

Amortized cost analysis is helpful because many important data structures occasionally incur a large cost as they perform some kind of rebalancing or improvement of their internal state, but those expensive operations cannot occur too frequently. In this case, amortized analysis can give a much tighter bound on the true cost of using the data structure than a standard worst-case-per-operation bound.

Note that even though the definition of amortized cost is simple, analyzing it will often require some thought.

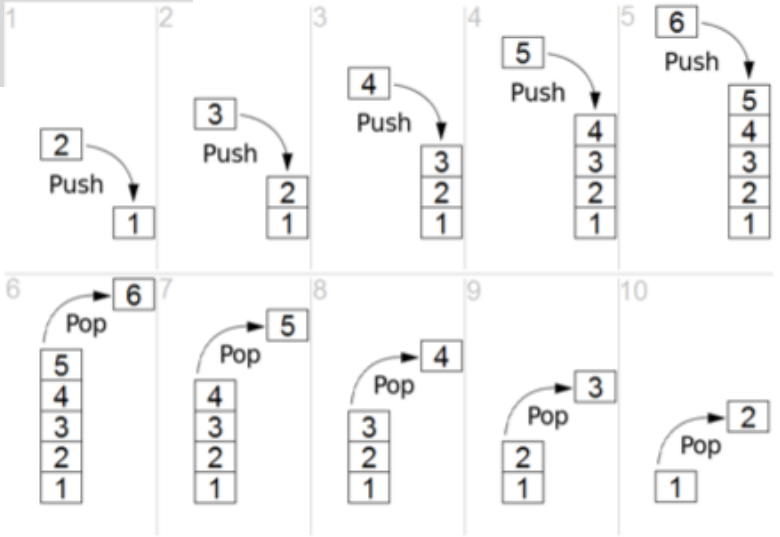
http://www.cs.cmu.edu/afs/cs/academic/class/15451-s10/www/lectures/lect0203.pdf

Stack Interface

```
public interface Stack<T> {
    // checks if empty
    boolean empty();

    // peeks at top value without removing
    T top() throws EmptyException;

    // removes top element
    void pop() throws EmptyException;

    // adds new element to top of stack
    void push(T t);
}
```

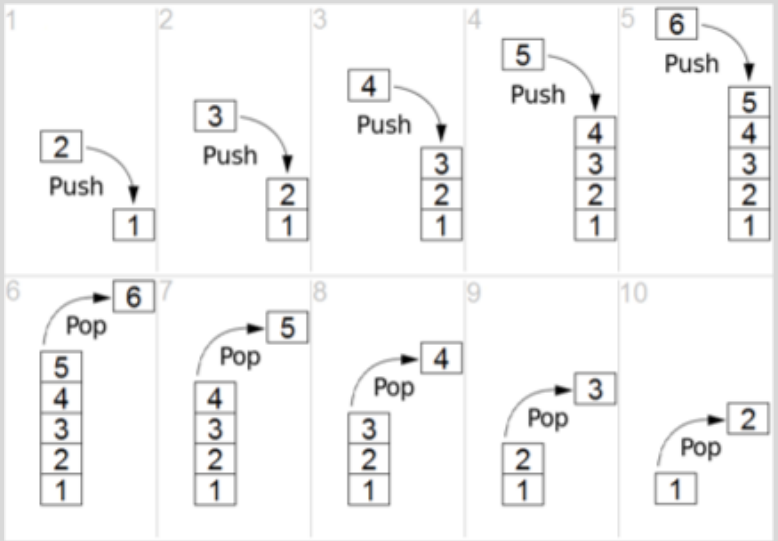
How would you *test* the implementation?

Why?



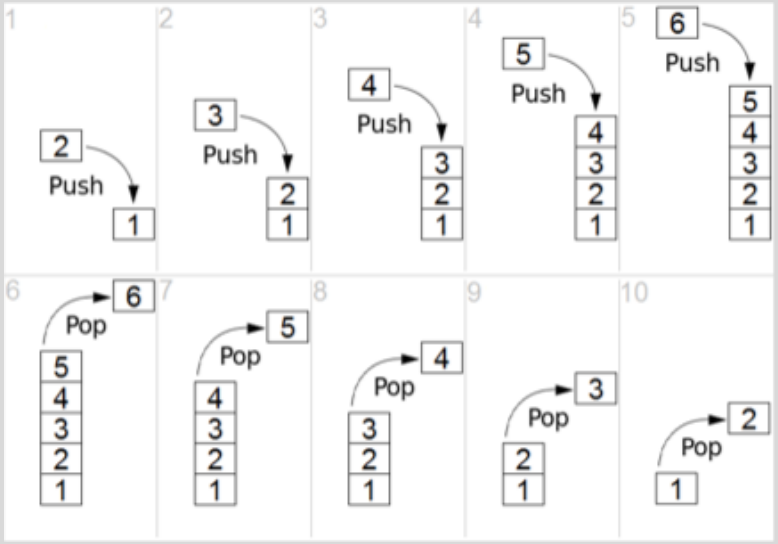
Stack JUnit Tests (1/2)

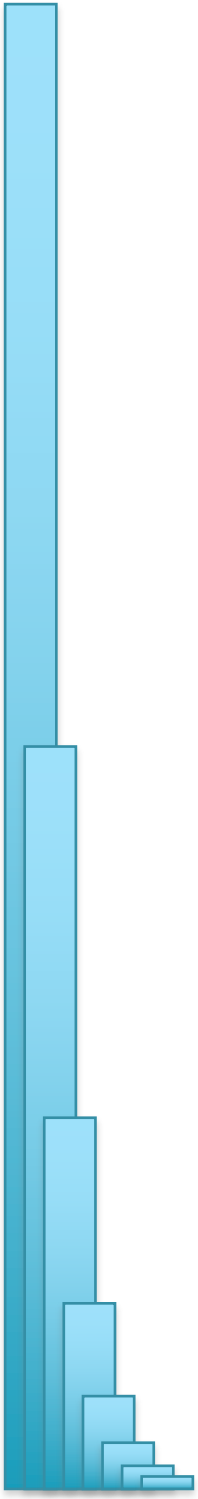
```
import org.junit.Test;
import org.junit.Before;
import static org.junit.Assert.assertEquals;

public class TestStack {
    private Stack<Integer> stack;

    @Before
    public void makeStack() {
        stack = new ListStack<Integer>();
    }

    @Test
    public void testNewEmpty() {
        assertEquals(true, stack.empty());
    }

    @Test
    public void testPushNotEmpty() {
        stack.push(12);
        assertEquals(false, stack.empty());
    }

    @Test
    public void testPushPopEmpty() {
        stack.push(12);
        stack.pop();
        assertEquals(true, stack.empty());
    }
```



Stack JUnit Tests (2/2)

```
    @Test
    public void testPushTopEqual() {
        stack.push(12);
        int hack = stack.top();
        assertEquals(12, hack);
    }


    @Test
    public void testLotsOfPush() {
        for (int i = 0; i < 100; i++) {
            stack.push(i);
        }
        int pops;
        for (pops = 0; !stack.empty(); pops++) {
            int hack = stack.top();
            assertEquals(99-pops, hack);
            stack.pop();
        }
        assertEquals(100, pops);
    }
}
```





Queues

Next Steps

1. Work on HW2
2. Get ready for HW3
3. Check on Piazza for tips & corrections!