Lecture 6: More Code Analysis, Testing
public static boolean isPrime(int n) {
    for (int i = 2; i < n; i++){
        if (n % i == 0) { +2
            return false; +1
        }
    }
    return true; +1
}

Approach
--> start with basic operations, work inside out for control structures
- Each basic operation = +1
- Conditionals = worst case test operations + branch
- Loop = iterations (loop body)

Answer:
code model: $3n - 5$ or $C_1n + C_2$
simplified tight-O bound: $O(n)$
Homework 2 is out!
- You should have a repo
- If you have issues submitting post on piazza
Modeling Complex Loops

Write a mathematical model of the following code

```java
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        System.out.println("Hello!");
    }
}
```

Keep an eye on loop bounds!
Modeling Complex Loops

for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        System.out.print("Hello! ");
    }
    System.out.println();
}

T(n) = (0 + 1 + 2 + 3 +...+ i-1)

How do we model this part?
How do we model this part?

Summations!
Summations!

1 + 2 + 3 + 4 +... + n = \sum_{i=1}^{n} i

\sum_{i=a}^{b} f(i) = f(a) + f(a + 1) + f(a + 2) + ... + f(b-2) + f(b-1) + f(b)

T(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{i-1} 1

What is the Big O?
What is the Big O?

<table>
<thead>
<tr>
<th>On the ith iteration of the outer loop</th>
<th>Output of ith iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>1</td>
<td>&quot;Hello! &quot;</td>
</tr>
<tr>
<td>2</td>
<td>&quot;Hello! Hello! &quot;</td>
</tr>
</tbody>
</table>
Simplifying Summations

for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        System.out.println("Hello!");
    }
}

\[ T(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{i-1} 1 \]

• Summation of a constant
  \[ \sum_{i=0}^{n-1} c = cn \]

• Factoring out a constant
  \[ \sum_{i=a}^{b} cf(i) = c \sum_{i=a}^{b} f(i) \]

• Gauss’s Identity
  \[ \sum_{i=0}^{n-1} i = \frac{n(n-1)}{2} \]

Find closed form using summation identities (given on exams)

\[ T(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{i-1} 1 \]

• Closed form

\[ = \sum_{i=0}^{n-1} 1 \cdot i \]

• Simplified
  \[ = \sum_{i=0}^{n-1} i \]

• Tight big O
  \[ = \frac{n(n-1)}{2} = \frac{1}{2} n^2 - \frac{1}{2} n \]

\[ = O(n^2) \]
public static void primesUpToN(int n) {
    System.out.print("1 2 ");
    for (int i = 3; i <= n; i++) {
        for (int j = 2; j < i; j++) {
            if (j != i && j % i == 0) {
                System.out.print(i + " ");
                break;
            }
        }
    }
    System.out.println();
}

\[
T(n) = 1 + \sum_{i=3}^{n} \sum_{j=2}^{i-1} 5 = 1 + \sum_{i=0}^{n-3} \sum_{j=0}^{i-3} 5 = 1 + \sum_{i=0}^{n-3} 5(i - 2) = 1 + 5(\sum_{i=0}^{n-3} i - \sum_{i=0}^{n-3} 2) - = 1 + 5(\frac{(n-2)(n-3)}{2} - (n-2)(2))
\]

Adjusting summation bounds
Summation of a constant
Factoring out a constant
Gauss's identity
**Definition: Big-O**

We wanted to find an upper bound on our algorithm’s running time, but
- We don’t want to care about constant factors.
- We only care about what happens as $n$ gets large.

**Big-O**

$f(n)$ is $O(g(n))$ if there exist positive constants $c, n_0$ such that for all $n \geq n_0$,

$$f(n) \leq c \cdot g(n)$$

We also say that $g(n)$ “dominates” $f(n)$

$O(g(n))$ is the “family” or “set” of all functions that are dominated by $g(n)$
Prove the function \( f(n) = \frac{n^2}{2} - \frac{3n}{2} \in O(n^2) \) by finding a \( c \) and \( n_0 \). Show your work.

\[
\frac{n^2}{2} \leq c \cdot n^2 \quad \text{when } c = \frac{1}{2} \quad \text{and } n_0 = 1
\]

\[
-\frac{3n}{2} \leq c \cdot n^2 \quad \text{when } c = 1 \quad \text{and } n_0 = 1
\]

\[
\text{combing it all together ...}
\]

\[
\frac{n^2}{2} - \frac{3n}{2} \leq \frac{1}{2} n^2 + n^2 \leq \frac{3}{2} n^2 \quad \text{when } n_0 = 1
\]

\[
c = \frac{3}{2} \quad \text{and } n_0 = 1 \text{ show that } f(n) \leq g(n)
\]
O, Omega, Theta [oh my?]

Big-O is an **upper bound**
- My code takes at most this long to run

**Big-Omega** is a **lower bound**

Big-Omega

\[ f(n) \text{ is } \Omega(g(n)) \text{ if there exist positive constants } c, n_0 \text{ such that for all } n \geq n_0, \]
\[ f(n) \geq c \cdot g(n) \]

**Big Theta is “equal to”**

Big-Theta

\[ f(n) \text{ is } \Theta(g(n)) \text{ if } f(n) \text{ is } O(g(n)) \text{ and } f(n) \text{ is } \Omega(g(n)). \]
What is the tight big-O bound?  
O(n)

What is the tight big-Ω bound?  
Ω(1)

What is the big-Θ bound?  
Doesn’t exist :/
Viewing $O$ as a class

Big-O can also be defined as a family or set of functions.

**Big-O (alternative definition)**

$O(g(n))$ is the set of all functions $f(n)$ such that there exist positive constants $c, n_0$ such that for all $n \geq n_0$, $f(n) \leq c \cdot g(n)$

You can write $f(n) \in O(g(n))$

Equivalent to “$f(n)$ is $O(g(n))$” or “$f(n) = O(g(n))$”

The set of all functions that run in linear time (i.e. $O(n)$) is a “complexity class.”

- We never write $O(5n)$ instead of $O(n)$ – they’re the same thing!
- It’s like writing $\frac{6}{2}$ instead of 3. It just looks weird.
Practice

5n + 3 ∈ O(n)  True
n ∈ O(5n + 3)  True
5n + 3 = O(n)  True
O(5n + 3) = O(n)  True
O(n^2) = O(n)  False
n^2 ∈ O(1)  False
n^2 ∈ O(n)  False
n^2 ∈ O(n^2)  True
n^2 ∈ O(n^3)  True
n^2 ∈ O(n^{100})  True

Big-O

\( f(n) ∈ O(g(n)) \) if there exist positive constants \( c, n_0 \) such that for all \( n ≥ n_0 \),
\[ f(n) ≤ c \cdot g(n) \]

Big-Omega

\( f(n) ∈ Ω(g(n)) \) if there exist positive constants \( c, n_0 \) such that for all \( n ≥ n_0 \),
\[ f(n) ≥ c \cdot g(n) \]

Big-Theta

\( f(n) ∈ Θ(g(n)) \) if \( f(n) \) is \( O(g(n)) \) and \( f(n) \) is \( Ω(g(n)) \).
### Examples

<table>
<thead>
<tr>
<th>Expression</th>
<th>Big-O</th>
</tr>
</thead>
<tbody>
<tr>
<td>$4n^2 \in \Omega(1)$</td>
<td>False</td>
</tr>
<tr>
<td>$4n^2 \in \Omega(n)$</td>
<td>True</td>
</tr>
<tr>
<td>$4n^2 \in \Omega(n^2)$</td>
<td>True</td>
</tr>
<tr>
<td>$4n^2 \in \Omega(n^3)$</td>
<td>False</td>
</tr>
<tr>
<td>$4n^2 \in \Omega(n^4)$</td>
<td>False</td>
</tr>
</tbody>
</table>

**Big-O**

$f(n) \in O(g(n))$ if there exist positive constants $c, n_0$ such that for all $n \geq n_0$,

$$f(n) \leq c \cdot g(n)$$

**Big-Omega**

$f(n) \in \Omega(g(n))$ if there exist positive constants $c, n_0$ such that for all $n \geq n_0$,

$$f(n) \geq c \cdot g(n)$$

**Big-Theta**

$f(n) \in \Theta(g(n))$ if $f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$. 
Testing Your Code
Testing

Computers don’t make mistakes- people do!

“I’m almost done, I just need to make sure it works”
– Naive 14Xers

**Software Test:** a separate piece of code that exercises the code you are assessing by providing input to your code and finishes with an assertion of what the result should be.

1. **Isolate** - break your code into small modules
2. **Build in increments** - Make a plan from simplest to most complex cases
3. **Test as you go** - As your code grows, so should your tests
Types of Tests

Black Box
- Behavior only – ADT requirements
- From an outside point of view
- Does your code uphold its contracts with its users?
- Performance/efficiency

White Box
- Includes an understanding of the implementation
- Written by the author as they develop their code
- Break apart requirements into smaller steps
- “unit tests” break implementation into single assertions
What to test?

**Expected behavior**
- The main use case scenario
- Does your code do what it should given friendly conditions?

**Forbidden Input**
- What are all the ways the user can mess up?

**Empty/Null**
- Protect yourself!
- How do things get started?
- 0, -1, null, empty collections

**Boundary/Edge Cases**
- First items
- Last item
- Full collections

**Scale**
- Is there a difference between 10, 100, 1000, 10000 items?
Testing Strategies

You can’t test everything
 - Break inputs into categories
 - What are the most important pieces of code?

Test behavior in combination
 - Call multiple methods one after the other
 - Call the same method multiple times

Trust no one!
 - How can the user mess up?

If you messed up, someone else might
 - Test the complex logic
## Thought Experiment

**Discuss with your neighbors:** Imagine you are writing an implementation of the List interface that stores integers in an Array. What are some ways you can assess your program’s correctness in the following cases:

<table>
<thead>
<tr>
<th>Expected Behavior</th>
<th>Boundary/Edge Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Create a new list</td>
<td></td>
</tr>
<tr>
<td>- Add some amount of items to it</td>
<td></td>
</tr>
<tr>
<td>- Remove a couple of them</td>
<td></td>
</tr>
<tr>
<td>- Add 1 item to an empty list</td>
<td></td>
</tr>
<tr>
<td>- Set an item at the front of the list</td>
<td></td>
</tr>
<tr>
<td>- Set an item at the back of the list</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Forbidden Input</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Add a negative number</td>
<td></td>
</tr>
<tr>
<td>- Add duplicates</td>
<td></td>
</tr>
<tr>
<td>- Add extra large numbers</td>
<td></td>
</tr>
<tr>
<td>- Add 1000 items to the list</td>
<td></td>
</tr>
<tr>
<td>- Remove 100 items in a row</td>
<td></td>
</tr>
<tr>
<td>- Set the value of the same item 50 times</td>
<td></td>
</tr>
</tbody>
</table>
JUnit

JUnit: a testing framework that works with IDEs to give you a special GUI experience when testing your code

```java
@Test
public void myTest() {
    Map<String, Integer> basicMap = new LinkedListDict<String, Integer>();
    basicMap.put("Kasey", 42);
    assertEquals(42, basicMap.get("Kasey"));
}
```

Assertions:
- assertEquals(item1, item2)
- assertTrue(Boolean expression)
- assertFalse(bolean expression)
- assertNotNull(item)

More: https://junit.org/junit5/docs/5.0.1/api/org/junit/jupiter/api/Assertions.html