Lecture 4: Introduction to Code Analysis
Warm Up

Read through the code on the worksheet given
Come up with a test case for each of the described test categories

Expected Behavior  add(1)
Forbidden Input    add(null)
Empty/Null         Add into empty list
Boundary/Edge      Add enough values to trigger internal array double and copy
Scale              Add 1000 times in a row

Socrative:
www.socrative.com
Room Name: CSE373
Please enter your name as: Last, First
Administrivia

- Fill out HW 2 Partner form
  Posted on class webpage at top
  Due TONIGHT Monday 4/8 by 11:59pm

- Fill out Student Background Survey, on website announcements

- Read Pair Programming Doc (on readings for Wednesday on calendar)
Algorithm Analysis
Code Analysis

How do we compare two pieces of code?
- Time needed to run
- Memory used
- Number of network calls
- Amount of data saved to disk
- Specialized vs generalized
- Code reusability
- Security
Comparing Algorithms with Mathematical Models

Consider overall trends as inputs increase
- Computers are fast, small inputs don’t differentiate
- Must consider what happens with large inputs

Identify trends without investing in testing

Model performance across multiple possible scenarios
- Worst case - what is the most expensive or least performant an operation can be
- Average case – what functions are most likely to come up?
- Best case – if we understand the ideal conditions can increase the likelihood of those conditions?
**Review: Sequential Search**

**Sequential search:** Locates a target value in a collection by examining each element sequentially.

- Example: Searching the array below for the value **42**:

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>-4</td>
<td>2</td>
<td>7</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>22</td>
<td>25</td>
<td>30</td>
<td>36</td>
<td><strong>42</strong></td>
<td>50</td>
<td>56</td>
<td>68</td>
<td>85</td>
<td>92</td>
<td>103</td>
</tr>
</tbody>
</table>

```
public int search(int[] a, int val) {
    for (int i = 0; i < a.length; i++) {
        if (a[i] == val) {
            return i;
        }
    }
    return -1;
}
```

- How many elements will be examined?
  - What is the best case? element found at index 0, 1 item examined, O(1)
  - What is the worst case? element found at index 16 or not found, all elements examined, O(n)
  - What is the average case? most elements examined, O(n)

\[ f(n) = n \]
**Review: Binary Search**

**binary search:** Locates a target value in a *sorted* array or list by successively eliminating half of the array from consideration.

- Example: Searching the array below for the value **42**:

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|    |
| value | -4| 2 | 7 | 10| 15| 20| 22| 25| 30| 36| **42**| 50| 56| 68| 85| 92| 103|

How many elements will be examined?
- What is the best case?
  - element found at index 8, 1 item examined, $O(1)$
- What is the worst case?
  - element found at index 9 or not found, $\frac{1}{2}$ elements examined, $O(?)$
- What is the average case?

```java
public static void binarySearch(int[] a, int val){
    ...
    while (first <= last){
        if (arr[mid] < key ){
            first = mid + 1;
        } else if ( arr[mid] == key ){
            return mid;
        } else{
            last = mid - 1;
        }
        mid = (first + last)/2;
    }
    return -1;
}
```
Analyzing Binary Search

What is the pattern?
- At each iteration, we eliminate half of the remaining elements

How long does it take to finish?
- 1\text{st} iteration – \( N/2 \) elements remain
- 2\text{nd} iteration – \( N/4 \) elements remain
- K\text{th} iteration - \( N/2^k \) elements remain

Finishes when \( \frac{n}{2^k} = 1 \)
\[
\frac{n}{2^k} = 1
\]
-> multiply right side by \( 2^k \)
\[
N = 2^k
\]
-> isolate K exponent with logarithm
\[
\log_2 N = k
\]
Asymptotic Analysis

asymptotic analysis – the process of mathematically representing runtime of a algorithm in relation to the number of inputs and how that relationship changes as the number of inputs grow

Two step process

1. Model – reduce code run time to a mathematical relationship with number of inputs
2. Analyze – compare runtime/input relationship across multiple algorithms
Code Modeling
Code Modeling

code modeling – the process of mathematically representing how many operations a piece of code will run in relation to the number of inputs $n$

Examples:
- Sequential search  $f(n) = n$
- Binary search  $f(n) = \log_2 n$

What counts as an “operation”?

**Basic operations**
- Adding ints or doubles
- Variable assignment
- Variable update
- Return statement
- Accessing array index or object field

**Consecutive statements**
- Sum time of each statement

**Function calls**
- Count runtime of function body

**Conditionals**
- Time of test + worst case scenario branch

**Loops**
- Number of iterations of loop body x runtime of loop body

Assume all operations run in equivalent time
Modeling Case Study

**Goal:** return ‘true’ if a sorted array of ints contains duplicates

**Solution 1: compare each pair of elements**

```java
public boolean hasDuplicate1(int[] array) {
    for (int i = 0; i < array.length; i++) {
        for (int j = 0; j < array.length; j++) {
            if (i != j && array[i] == array[j]) {
                return true;
            }
        }
    }
    return false;
}
```

**Solution 2: compare each consecutive pair of elements**

```java
public boolean hasDuplicate2(int[] array) {
    for (int i = 0; i < array.length - 1; i++) {
        if (array[i] == array[i + 1]) {
            return true;
        }
    }
    return false;
}
```
Modeling Case Study: Solution 2

Goal: produce mathematical function representing runtime \( f(n) \) where \( n = \text{array.length} \)

Solution 2: compare each consecutive pair of elements

```java
public boolean hasDuplicate2(int[] array) {
    for (int i = 0; i < array.length - 1; i++) {
        if (array[i] == array[i + 1]) {
            return true; +1
        }
    }
    return false;
}
```

\[ f(n) = 5(n - 1) + 1 \]

linear \( \rightarrow \) \( O(n) \)

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)
Modeling Case Study: Solution 1

Solution 1: compare each consecutive pair of elements

```java
public boolean hasDuplicate1(int[] array) {
    for (int i = 0; i < array.length; i++) {
        for (int j = 0; j < array.length; j++) {
            if (i != j && array[i] == array[j]) {
                return true; +1
            }
        }
    }
    return false; +1
}
```

\( f(n) = 5(n - 1) + 1 \)  
quadratic \( \rightarrow O(n^2) \)

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)
Your turn!

Write the specific mathematical code model for the following code and indicate the big O runtime.

```java
public void foobar (int k) {
    int j = 0; +1
    while (j < k) { +k/5 (body)
        for (int i = 0; i < k; i++) { +k(body)
            System.out.println("Hello world"); +1
        }
        j = j + 5; +2
    }
}
```

$f(k) = \frac{k(k + 2)}{5}$

quadratic -> $O(k^2)$

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)