Comprehending. Understanding the implementation details of a program.

Modeling. Counting the number of steps in terms of N, the size of the input.

Case Analysis. How certain conditions affect the program execution.

Asymptotic Analysis. Describing what happens for very large N, as N→∞.

Formalizing. Summarizing the final result in precise English or math notation.

The reading described the implementation details for dup1 and dup2 (Comprehension) and introduced the idea of counting steps (Modeling). In this lecture, we will go in-depth on modeling and formalizing.

?: Where did case analysis come up in the reading?

Consider every pair

boolean dup1(int[] A)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Best case</th>
<th>Worst case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array contains a duplicate at front</td>
<td>Constant time</td>
<td>Quadratic time</td>
</tr>
<tr>
<td>Array contains no duplicate items</td>
<td>Θ(1)</td>
<td>Θ(N²)</td>
</tr>
<tr>
<td>Overall Asymptotic Runtime Bound</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q: Asymptotic Analysis vs. Case Analysis

For a very large array with billions of elements (asymptotic analysis), how is it possible for dup1 to execute only 2 less-than (<) operations?

<table>
<thead>
<tr>
<th>Operation</th>
<th>dup1: Quadratic/Parabolic</th>
<th>dup2: Linear</th>
</tr>
</thead>
<tbody>
<tr>
<td>i = 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>less-than (&lt;)</td>
<td>2 to (N² + 3N + 2) / 2</td>
<td>1 to N</td>
</tr>
<tr>
<td>increment (+= 1)</td>
<td>0 to (N² + N) / 2</td>
<td>0 to N - 1</td>
</tr>
<tr>
<td>equality (==)</td>
<td>1 to (N² - N) / 2</td>
<td>1 to N - 1</td>
</tr>
<tr>
<td>array accesses</td>
<td>2 to N² - N</td>
<td>2 to 2N - 2</td>
</tr>
</tbody>
</table>

public static boolean dup1(int[] A) {
    for (int i = 0; i < A.length; i += 1) {
        for (int j = i + 1; j < A.length; j += 1) {
            if (A[i] == A[j]) {
                return true;
            }
        }
    }
    return false;
}

Q1: For a very large array with billions of elements (asymptotic analysis), how is it possible for dup1 to execute only 2 less-than (<) operations?

?: What does the runtime for dup1 vs. dup2 look like if we only consider the best case asymptotic analysis? How does that result compare to the worst case asymptotic analysis?
What is the order of growth of each function? (Informally, what is the shape of each function for very large N?)

<table>
<thead>
<tr>
<th>Function</th>
<th>Order of Growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>N^3 + 3N^4</td>
<td>4</td>
</tr>
<tr>
<td>(1 / N) + N^2</td>
<td>3</td>
</tr>
<tr>
<td>(1 / N) + 5</td>
<td>3</td>
</tr>
<tr>
<td>Ne^N + N</td>
<td>Ne</td>
</tr>
<tr>
<td>40 sin(N) + 4N^2</td>
<td>2</td>
</tr>
</tbody>
</table>

Q1: What is the order of growth of each function? (Informally, what is the shape of each function for very large N?)

Big-Theta Definition

\[ R(N) \in \Theta(f(N)) \]

means there exist positive constants \( k_1 \) and \( k_2 \) such that

\[ k_1 \cdot f(N) \leq R(N) \leq k_2 \cdot f(N) \]

for all values of \( N \) greater than some \( N_0 \).

? : What is a value that we can choose for \( N_0 \) according to the plot on the right?
Find a simple \( f(N) \) and corresponding \( k_1 \) and \( k_2 \).

\[
R(N) = \frac{4N^2 + 3N \ln N}{2}
\]

\( R(N) \in \Theta(f(N)) \)

means there exist positive constants \( k_1 \) and \( k_2 \) such that

\[
k_1 \cdot f(N) \leq R(N) \leq k_2 \cdot f(N)
\]

for all values of \( N \) greater than some \( N_0 \).

**Q1:** Find a simple \( f(N) \) and corresponding \( k_1 \) and \( k_2 \).

\[
R(N) \in O(f(N))
\]

means there exists a positive constant \( k_2 \) such that

\[
R(N) \leq k_2 \cdot f(N)
\]

for all values of \( N \) greater than some \( N_0 \).

Why can we say that \( 40 \sin(N) + 4N^2 \) is in \( O(N^4) \)? Explain in terms of the formal definition of Big-O.

Why is it incorrect to say that \( 40 \sin(N) + 4N^2 \) is in \( \Theta(N^4) \)? Explain in terms of the formal definition of Big-Theta.
Give an overall asymptotic runtime bound for $R$ as a combination of $\Theta$, $O$, and/or $\Omega$ notation. Take into account both the best and the worst case runtimes ($R_{\text{best}}$ and $R_{\text{worst}}$).

Then, give a few other valid runtime bounds for $R_{\text{best}}$, $R_{\text{worst}}$, and $R$ using asymptotic notation.

**Q1:** Give an overall asymptotic runtime bound for $R$ as a combination of $\Theta$, $O$, and/or $\Omega$ notation. Take into account both the best and the worst case runtimes ($R_{\text{best}}$ and $R_{\text{worst}}$).

**Q2:** Then, give a few other valid runtime bounds for $R_{\text{best}}$, $R_{\text{worst}}$, and $R$ using asymptotic notation.

**Mystery**

Give a tight asymptotic runtime bound for mystery as a function of $N$, the length of the array, in the best case, worst case, and overall.

```java
boolean mystery(int[] a, int target) {
    int N = a.length;
    for (int i = 0; i < N; i += 1)
        if (a[i] == target)
            return true;
    return false;
}
```