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?

From this point forward, we’ll almost always be working in the mode of asymptotic analysis: considering the behavior of programs as N grows very large.

?: How can we characterize the range of step counts that we saw in `dup1` and `dup2`?
Orders of Growth

?: Why might we choose to focus on very large N rather than small N?

?: How do multiplicative constants, e.g. 100N or N^2 / 2, affect the order of growth of the runtime of different algorithms?

---

### Table 2.1: Running Times (Rounded Up) of Different Algorithms on Inputs of Increasing Size

<table>
<thead>
<tr>
<th>n</th>
<th>n log₂ n</th>
<th>n²</th>
<th>n³</th>
<th>1.5^n</th>
<th>2^n</th>
<th>n!</th>
</tr>
</thead>
<tbody>
<tr>
<td>n = 30</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>4 sec</td>
<td></td>
</tr>
<tr>
<td>n = 30</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>18 min</td>
<td>10^35 years</td>
</tr>
<tr>
<td>n = 50</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>11 min</td>
<td>36 years</td>
</tr>
<tr>
<td>n = 100</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>1 sec</td>
<td>12,892 years</td>
<td>10^17 years</td>
</tr>
<tr>
<td>n = 1,000</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>1 sec</td>
<td>18 min</td>
<td>very long</td>
<td>very long</td>
</tr>
<tr>
<td>n = 10,000</td>
<td>&lt; 1 sec</td>
<td>2 min</td>
<td>12 days</td>
<td>very long</td>
<td>very long</td>
<td>very long</td>
</tr>
<tr>
<td>n = 100,000</td>
<td>&lt; 1 sec</td>
<td>3 hours</td>
<td>32 years</td>
<td>very long</td>
<td>very long</td>
<td>very long</td>
</tr>
<tr>
<td>n = 1,000,000</td>
<td>1 sec</td>
<td>20 sec</td>
<td>12 days</td>
<td>31,710 years</td>
<td>very long</td>
<td>very long</td>
</tr>
</tbody>
</table>

---

Asymptotic Analysis and Case Analysis

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

```java
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 (i.e. asymptotic analysis), 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?
Identifying Orders of Growth

Consider the algorithm step counts below. What do you expect will be the order of growth of the runtime for the algorithm?

A. N [linear]
B. N² [quadratic]
C. N³ [cubic]
D. N⁶ [sextic]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>less-than (&lt;)</td>
<td>100N² + 3N</td>
</tr>
<tr>
<td>greater-than (&gt;)</td>
<td>2N³ + 1</td>
</tr>
<tr>
<td>and (&amp;&amp;)</td>
<td>5,000</td>
</tr>
</tbody>
</table>

Q1: What do you expect will be the order of growth of the runtime for the algorithm? In other words, if we plotted total runtime vs. N, which curve would we expect?

Simplification 3: Eliminate Lower-Order Terms

Ignore lower-order terms.

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;
}

?: Why can we ignore lower-order terms?
Your Turn: Worst Case Order of Growth for dup2

1. Only consider the **worst case**.
2. Pick a representative operation (cost model).
3. Ignore lower order terms.
4. Ignore multiplicative constants.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Worst Case Growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>i = 0</td>
<td>1</td>
</tr>
<tr>
<td>less-than (&lt;)</td>
<td>0 to N</td>
</tr>
<tr>
<td>increment (+= 1)</td>
<td>0 to N - 1</td>
</tr>
<tr>
<td>equality (==)</td>
<td>1 to N - 1</td>
</tr>
<tr>
<td>array accesses</td>
<td>2 to 2N - 2</td>
</tr>
</tbody>
</table>

"The worst case order of growth of the runtime for dup2 is …"
Worst Case Order of Growth: Exact Count of $==$ Operations

```
int N = A.length; // N == 6
for (int i = 0; i < N; i += 1)
  for (int j = i + 1; j < N; j += 1)
    if (A[i] == A[j])
      return true;
  return false;
```

"The worst case order of growth of the runtime for dup1 is $N^2$."

Worst Case Order of Growth: Geometric Argument

```
int N = A.length; // N == 6
for (int i = 0; i < N; i += 1)
  for (int j = i + 1; j < N; j += 1)
    if (A[i] == A[j])
      return true;
  return false;
```

Area of right triangle of side length $N - 1$.
Order of growth of area is $N^2$.

\[
C = 1 + 2 + 3 + \cdots + (N - 3) + (N - 2) + (N - 1)
C = (N - 1) + (N - 2) + (N - 3) + \cdots + 3 + 2 + 1
2C = N + N + N + \cdots + N + N + N = N(N - 1)
\therefore \quad C = \frac{N(N - 1)}{2}
\]
Q: Order of Growth Exercise

Informally, what is the shape of each function for very large $N$? In other words, what is the order of growth of each function?

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

Q1: Informally, what is the shape of each function for very large $N$? In other words, what is the order of growth of each function?

?: What is a value that we can choose for $N_0$ according to the plot on the right?

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 $N_0$.
Big-Theta Challenge

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

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

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

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 \( k_2 \).

Big-O Definition

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

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 \( N_0 \).

\[ 40 \sin(N) + 4N^2 \]

Plot of \( 40 \sin(N) + 4N^2 \)

\[ 5N^4 \]

\( k_2 = 5 \)

“Very large \( N \)”

?: 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.
Big-Omega Definition

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

means there exists a positive constant \( k_1 \) such that

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

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

Likewise, we have a Big-Omega definition for the other half of the inequality.

Q: Describe \( 40 \sin(N) + 4N^2 \in \Omega(N) \) in your own words using the plot on the right.

Q: Does \( \Theta(f(N)) \) imply \( O(f(N)) \) and \( \Omega(f(N)) \)? Does \( O(f(N)) \) and \( \Omega(f(N)) \) imply \( \Theta(f(N)) \)?

Overall Asymptotic Runtime Bound for dup1

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 \( R_{\text{worst}} \)).

\[ R_{\text{best}}(N) = 2 \]
\[ R_{\text{worst}}(N) = \frac{N^2 + 3N + 2}{2} \]

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 \( R_{\text{worst}} \)).