Topics

• How long does it run?
• Why measuring time won’t work
• Algorithm analysis
• Big-O analysis
• Asymptotic run-time estimate
Running Time

• We want to compare two algorithms (implementations) to see which is more efficient

• Why clock time won’t work-
  – Computers are multi-tasking
  – Different computer architectures
  – Measure different sizes of input, i.e. 1 to N
  – Programming language paradigms

• We estimate performance of the algorithm
Algorithms

• Definition of an algorithm: A set of ordered steps for solving a problem
• What do we typically analyze?
  How long an algorithm takes
  How much memory it uses
Algorithm analysis

• Suppose that algorithm $A$ processes $n$ data elements in time $T$ and uses up space $S$.
• Algorithm analysis attempts to estimate how $T$ and/or $S$ is affected by changes in $n$.
• In other words, $T$ and/or $S$ is a function of $n$ when we use $A$. 
Example (Find the sum of integers from 1 to n )

```c
int n = 1000;
long sum = 0;
for (int k = 1; k <= n; k++)
    sum += k;
```

Suppose that this code executes in 2 ms.
What is the expected execution time if \( n \) is changed to
- 500? 2 ms
- 2000? 5 ms
- 10,000? 21 ms

The relationship appears to be \( \frac{n}{500} + 1 \) ms.
Note that the change in \( T \) is linear, i.e., \( T \) approximately doubles when \( n \) doubles, etc.
Example, continued

• Using Big-O notation, we call this an O(n) algorithm. (O(n) is pronounced "order n").

• Big-O notation describes the asymptotic execution time of an algorithm.
Big-O Notation

• The formula was \((n/500)+1\) ms. What about the \(1/500\) and the \(+1\) ?

• With Big-O notation, it's generally safe to drop constant terms and factors.

  This means that \(O(3n)\), \(O(n+1)\), \(O(5n+2)\) all becomes \(O(n)\)

• Remember it's about the rate of change.
Big-O Example 1

• Print multiples of n from 1 to 500:

```java
int n = 127;
for(int k = 1; k <= 500; k++)
    System.out.print(k * n);
```

• Suppose that this code executes in 2 ms. What is the expected execution time if n is changed to

<table>
<thead>
<tr>
<th>n</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>254</td>
<td>2 ms</td>
</tr>
<tr>
<td>1270</td>
<td>2 ms</td>
</tr>
<tr>
<td>63</td>
<td>2 ms</td>
</tr>
</tbody>
</table>

The "change" in T is constant, i.e., T does not depend on the size of n.
Big-O Example 1

• We call this an O(1) algorithm.
  Note that it doesn't matter if it's 1 ... 5 or 1 ... 500.
  Yes, 1 ... 500 takes longer than 1 ... 5, but changing n doesn't change the time for either.

• Even if you think of it as O(500), 500 = 500 x 1, so drop the constant to get O(1).
Create a "multiplication table" with n rows and n columns:

```java
int n = 100;
int table[][] = new int[n][n];
for(int row = 0; row < n; row++)
    for(int col = 0; col < n; col++)
        table[row][col] = (row+1) * (col+1);
```

Suppose that this code executes in 10 ms. What is the expected execution time if n is changed to

<table>
<thead>
<tr>
<th>n</th>
<th>Expected Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>40 ms</td>
</tr>
<tr>
<td>1000</td>
<td>1000 ms</td>
</tr>
<tr>
<td>50</td>
<td>2.5 ms</td>
</tr>
</tbody>
</table>
Big-O Example 2

The change in $T$ is not linear.

$T$ quadruples when $n$ doubles.

$T$ increases by a factor of 100 when $n$ increases by a factor of 10 ... etc.

Hence, the algorithm is $O(n^2)$. 
Asymptotic Execution Time

• The order of an algorithm is an approximate measure of the change in $T$ as $n$ changes.
• The order of an algorithm is called the asymptotic execution time (it only matters when $n$ is a big value)
• Big-O notation is not very useful for small values of $n$.
• Big-O is asymptotic because we are trying to estimate the function that the execution time approaches as $n$ gets very large.
Big-O Example 3

Draw a triangle:

```java
int n = 40;
for (int row = 1; row <= n; row++) {
    for (int col = 1; col <= row; col++)
        System.out.print('*');
    System.out.println('');
}
```

How many times is `System.out.print('*');` executed?

1 + 2 + 3 + ... + 40 = 820 = n(n+1)/2 = (½)(n² + n)

How does T change when n is doubled to 80?

1 + 2 + 3 + ... + 80 = 3240 = approximately 4 x 820 (3280)

What is the order of this algorithm?

Drop the constant, and use the dominating term

This algorithm is O(n²) ... not O(n² + n)
Find out how many times an integer can be divided by 2:

```java
int n = 1000;
while (n > 0){
    System.out.println(n);
    n /= 2;
}
```

• How many times is `System.out.println(n);` executed?
• This algorithm is $O(\log_2 n) = O(\log n)$
Big-O Example 4

Note:
When we say $O(\log n)$ the base of the logarithm isn’t important. Why?

Because, e.g., $(\log_2 n) = \log_x n / (\log_x 2)$. $(\log_x 2)$ is a constant which can be dropped.
Comparison of Scaling

<table>
<thead>
<tr>
<th>n</th>
<th>log n</th>
<th>n²</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1.58</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>2.32</td>
<td>25</td>
</tr>
<tr>
<td>10</td>
<td>3.32</td>
<td>100</td>
</tr>
<tr>
<td>100</td>
<td>6.64</td>
<td>10,000</td>
</tr>
<tr>
<td>1000</td>
<td>9.97</td>
<td>1,000,000</td>
</tr>
<tr>
<td>1,000,000</td>
<td>19.93</td>
<td>10^{12}</td>
</tr>
</tbody>
</table>

If a single step takes 1 ms, the log n algorithm will take less than .02 seconds for n=1,000,000, whereas the n² algorithm will take over 31 years!
Comparison of Scaling

- Yellow – \( \log n \)
- Red – \( n \)
- Blue – \( n^2 \)
- Green – \( 2^n \)
How Big is \( N \)?

All students in CS162? \( 60 \)
All students at OSU? \( \sim 25,000 \)
All college students in the US? \( \sim 41 \text{ million} \)
The population of the Earth? \( \sim 7 \text{ billion} \)

The number of Earth-like planets in the universe? \( 1 \) to a gazillion
Summary

• Analyze the algorithm
• We ignore machine details
• This lesson is just a taste of the subject
• Analysis details for the future-
  – Best, worst, and average case analysis
  – Recurrence relations
  – Time versus space complexity