CSCI 241
Scott Wehrwein
Runtime Analysis:
Counting Operations - I
Goals

Know how to count constant time operations in simple algorithms.

Know how to find the asymptotic runtime class (big-O runtime) of an algorithm given a count of its constant-time operations.
How can we compare algorithms?

- Which one finishes faster?
- Which one uses less memory?
- Which one has more lines of code?
- Which one executes more lines of code? (constant-time)
- How many operations does each perform as a function of the input data size?
How many constant-time operations are executed by the following algorithm?

```java
/** Return the sum of 0..N; Pre: N > 0 */
public static int alg1(int N) {
    int i = 0;
    int sum = 0;
    while (i < N) {
        sum += i;
        i += 1;
    }
    return sum;
}
```

Strategy:
1. Identify constant-time operations.
2. Determine how many times each happens.
Counting Operations

How many constant-time operations are executed by the following algorithm?

```java
/** Return the sum of 0..N; Pre: N > 0 */
public static int alg1(int N) {
    int i = 0;
    int sum = 0;
    while (i < N) {
        sum += i;
        i += 1;
    }
    return sum;
}
```

Strategy:
1. Identify constant-time operations.
2. Determine how many times each happens.

Convention: If a line has multiple primitive operations, count the line as 1.
Counting Operations

How many constant-time operations are executed by the following algorithm?

```java
/** Return the sum of 0..N; Pre: N > 0 */
public static int alg1(int N) {
    int i = 0;
    int sum = 0;
    while (i < N) {
        sum += i;
        i += 1;
    }
    return sum;
}
```

Strategy:
1. Identify constant-time operations.
2. Determine how many times each happens.

Convention: If a line has multiple primitive operations, count the line as 1.

Total: \(3N + 4\)
Properties of a good measurement system

- Explicitly depends on input size

- Doesn't sweat the details:
  - Doesn't depend on hardware specifics
  - Assigns same number to algorithms that are 'close enough'
Counting Operations

How many constant-time operations are executed by the following algorithm?

```java
/** Return the sum of 0..N; Pre: N > 0 */
public static int alg1(int N) {
    int i = 0;  // 1
    int sum = 0;  // 1
    while (i < N) {  // N+1
        sum += i;  // N
        i += 1;  // N
    }
    return sum;  // 1
}
```

Total: $3N + 4$

Runtime class: $O(N)$

Strategy:
1. Identify constant-time operations.
2. Determine how many times each happens.
3. Drop constants and lower-order terms. ?!?!
Properties of a good measurement system

✓ Explicitly depends on input size

• Doesn't sweat the details:

✓ Doesn't depend on hardware specifics

• Assigns same number to algorithms that are 'close enough'
A CS Definition of Close Enough

(aka "big-O" runtime)
The asymptotic runtime class of an algorithm is the number of constant-time operations it performs, with all constants and lower-order terms dropped.

Examples:

<table>
<thead>
<tr>
<th>Operations</th>
<th>Big-O Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>N + 2</td>
<td>O(N)</td>
</tr>
<tr>
<td>4N + 7</td>
<td>O(N)</td>
</tr>
<tr>
<td>3N^2 + 4N</td>
<td>O(N^2)</td>
</tr>
<tr>
<td>2^N + 3N^4 - N</td>
<td>O(2^N)</td>
</tr>
<tr>
<td>7</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

Strategy:
1. Identify constant-time operations.
2. Determine how many times each happens.
3. Drop constants and lower-order terms.
Properties of asymptotic runtime analysis

- Explicitly depends on input size
- Doesn't sweat the details:
  - Doesn't depend on hardware specifics
  - Assigns same number to algorithms that are 'close enough'