Growth of Functions
Learning Objectives

- Understand the meaning of growth of functions.
- Measure the growth of the running time of an algorithm.
- Use the Big-Oh notation to compare the growth of two functions.
Growth of Functions
$O$-notation

$\exists c > 0, n_0 > 0$

$0 \leq f(n) \leq cg(n)$

$n \geq n_0$

$g(n)$ is an asymptotic upper-bound for $f(n)$

$f(n) = O(g(n))$
**Ω-notation**

\[ f(n) = \Omega(g(n)) \]

\[ \exists c > 0, n_0 > 0 \]
\[ 0 \leq cg(n) \leq f(n) \]
\[ n \geq n_0 \]

\( g(n) \) is an asymptotic lower-bound for \( f(n) \)
$\Theta$-notation

$\exists c_1, c_2 > 0, n_0 > 0$\n
$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$\n
$n \geq n_0$

$g(n)$ is an asymptotic **tight** bound for $f(n)$
**o-notation**

\[ f(n) = o(g(n)) \]

\[ \forall c > 0 \]
\[ \exists n_0 > 0 \]
\[ 0 \leq f(n) \leq cg(n) \]
\[ n \geq n_0 \]

\[ g(n) \text{ is a non-tight asymptotic upper-bound for } f(n) \]
\( \omega \)-notation

\[ f(n) = \omega(g(n)) \]

\( \forall c > 0 \)
\( \exists n_0 > 0 \)
\( 0 \leq cg(n) \leq f(n) \)
\( n \geq n_0 \)

\( g(n) \) is a non-tight asymptotic lower-bound for \( f(n) \)
### Analogy to real numbers

<table>
<thead>
<tr>
<th>Functions</th>
<th>Real numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(n) = O(g(n)) )</td>
<td>( a \leq b )</td>
</tr>
<tr>
<td>( f(n) = \Omega(g(n)) )</td>
<td>( a \geq b )</td>
</tr>
<tr>
<td>( f(n) = \Theta(g(n)) )</td>
<td>( a = b )</td>
</tr>
<tr>
<td>( f(n) = o(g(n)) )</td>
<td>( a &lt; b )</td>
</tr>
<tr>
<td>( f(n) = \omega(g(n)) )</td>
<td>( a &gt; b )</td>
</tr>
</tbody>
</table>
Standard Classes of Functions

- Constant: \( f(n) = \Theta(1) \)
- Logarithmic: \( f(n) = \Theta(\lg(n)) \)
- Sublinear: \( f(n) = o(n) \)
- Linear: \( f(n) = \Theta(n) \)
- Super-linear: \( f(n) = \omega(n) \)
- Quadratic: \( f(n) = \Theta(n^2) \)
- Polynomial: \( f(n) = \Theta(n^k); \ k \) is a constant
- Exponential: \( f(n) = \Theta(k^n); \ k \) is a constant
Insertion Sort (Revisit)

**Insertion-Sort** *(A, n)*

```
for j = 2 to n
    key = A[j]
    // Insert A[j] into the sorted sequence A[1 ... j - 1].
    i = j - 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i - 1
    A[i + 1] = key
```

\[ \Theta(n^2) \]
Using L'Hopital's rule

- Determine the relative growth rates by using L'Hopital's rule

  - compute \( \lim_{n \to \infty} \frac{f(N)}{g(N)} \)

- if 0: \( f(N) = o(g(N)) \)
- if constant \( \neq 0 \): \( f(N) = \Theta(g(N)) \)
- if \( \infty \): \( g(N) = o(f(N)) \)
- limit oscillates: no relation
Recursion

- In math:
  - Factorial: \( n! = (n-1)! \cdot n, \quad 0! = 1 \)
  - Fibonacci: \( F(n) = F(n-1) + F(n-2), \quad F(0) = 0, \quad F(1) = 1 \)
- In programming:
  ```
  int fib(int number)
  {
    if (number == 0)
      return 0;
    if (number == 1)
      return 1;
    return fib(number - 1) + fib(number - 2);
  }
  ```

Question: Who is the recursion's worst enemy?
Function calls

- main() {
  F1(...);
}

- F1(...) {
  F2(...);
}

- F2(...) {
  F3(...);
}

Stack

Local variables

Other things you do not want to know