• Measuring the running time of algorithms
  
  o The **complexity** of an algorithm is a measure of its running time, or perhaps the memory space it takes

• **Θ-complexity**

  • **Notation**
    
    ∃: there exists
    ∀: for all
    s.t.: such that
    iff: if and only if

  • **Big-Oh: O**
    
    f(n) = O(g(n)) iff ∃ constants C₁ > 0, x₁ ≥ 0 s.t. ∀ n ≥ x₁, f(n) ≤ C₁ * g(n)

    Example: f(n) = n² + n

    f(n) = O(n²) because 2n² ≥ (n² + n) ∀ n ≥ 1

    Big-Oh specifies an *upper bound* on f(n)

  • **Big-Omega: Ω**
    
    f(n) = Ω(g(n)) iff ∃ constants C₂ > 0, x₂ ≥ 0 s.t. ∀ n ≥ x₂, f(n) ≥ C * g(n)

    Example: f(n) = Ω(n²) because n² ≤ (n² + n) ∀ n ≥ 0

    Ω specifies a *lower bound* on f(n)

  • **Big-Theta: Θ**
    
    f(n) = Θ(g(n)) iff f(n) = O(g(n)) and f(n) = Ω(g(n))

    Example: f(n) = Θ(n²) because 2n² ≥ (n² + n) ≥ n² ∀ n ≥ 1

    Big-Theta specifies both a lower and an upper bound on f(n). Note that g(n) must be the same function in both Big-Oh and Ω
Example

```java
public int howManyPairsSumToZero(int[] x) {
    int n = x.length;
    int count = 0;
    for (int i = 0; i < n - 1; i++)
        for (int j = i+1; j < n; j++)
            if (x[i] + x[j] == 0)
                count++;
    return count;
}
```

- Characterization of running time: how many times is the if statement executed?

<table>
<thead>
<tr>
<th>i</th>
<th># times</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>n-1</td>
</tr>
<tr>
<td>1</td>
<td>n-2</td>
</tr>
<tr>
<td>2</td>
<td>n-3</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>n-4</td>
<td>3</td>
</tr>
<tr>
<td>n-3</td>
<td>2</td>
</tr>
<tr>
<td>n-2</td>
<td>1</td>
</tr>
</tbody>
</table>

- \( f(n) = (n-1) + (n-2) + (n-3) + \ldots + 3 + 2 + 1 = \frac{n(n-1)}{2} \)
- \( f(n) = .5n^2 - .5n \)
- Hypothesis: \( f(n) = \Theta(n^2) \)
- To prove, show that \( f(n) = O(n^2) \) and \( f(n) = \Omega(n^2) \)
  - Big-Oh
    \[ .75n^2 \geq (.5n^2 - .5n) \forall n \geq 0 \]
    Therefore, \(.5n^2 - .5n\) is \(O(n^2)\)
  - \( \Omega \)
    \[ .25n^2 \leq (.5n^2 - .5n) \forall n \geq 2 \]
    Therefore, \(.5n^2 - .5n\) is \(\Omega(n^2)\)
- Given that \(.5n^2-.5n\) is \(O(n^2)\) and \(\Omega(n^2)\), it is \(\Theta(n^2)\).
- Graphically:
Another example

```java
public int howManyTriplesSumToZero(int[] x) {
    int n = x.length;
    int count = 0;
    for (int i = 0; i < n - 2; i++)
        for (int j = i + 1; j < n - 1; j++)
            for (int k = j + 1; k < n; k++)
                if (x[i] + x[j] + x[k] == 0)
                    count++;
    return count;
}
```

- How many times is the `if` statement executed?

\[ f(n) = \binom{n}{3} = \frac{n!}{(3! \cdot (n-3)!)} = \frac{n(n-1)(n-2)}{6} = \frac{n^3 - 3n^2 + 2n}{6} = \frac{n^3}{6} + \frac{n^2}{2} + \frac{n}{3}. \]

- Hypothesis: \( f(n) = \Theta(n^3) \).
- To prove, show that \( f(n) = O(n^3) \) and \( f(n) = \Omega(n^3) \)
  - \( O(n^3) \): Choose \( C_1 = 1/3 \)
    \[ n^3/3 \geq n^3/6 + n^2/2 + n/3. \]
  - \( \Omega(n^3) \): Choose \( C_2 = 1/9 \)
\[ n^3/9 \leq n^3/6 + n^2/2 + n/3. \]

- Therefore, \( n^3/6 + n^2/2 + n/3 = \Theta(n^3) \)

- Graphically

\[ \text{Exercise} \]

What is the \( \Theta \)-complexity of \( f(n) = 3n^2 - 9n + 4 \)? Demonstrate by showing Big-Oh and \( \Omega \).

\[ \text{Textbook Definitions for Algorithm Analysis} \]

Sedgewick and Wayne, p. 178-179

- \textbf{Tilde approximations}

We write \( \sim g(n) \) to represent any (more complicated) function \( f(n) \) that, when divided by \( g(n) \), approaches 1 as \( N \) approaches infinity, and we write \( f(n) \sim g(n) \) to indicate that \( f(n)/g(n) \) approaches 1 as \( n \) approaches infinity (\( f \) is a more complicated function than \( g \)).

In order words, we have determined that \( f(n) \) is a good characterization of the running time of an algorithm \( A \), but \( f \) is complicated (e.g., it contains many terms). We would like to characterize the running time of \( A \) in terms of a simpler function \( g \). Then we can do so if

\[ f(n) \sim g(n). \]
• **Order of growth**

  • Most often, we work with tilde approximations of the form

  \[ f(n) \sim a \cdot g(n) \]

  where \( g(n) = n^b \cdot (\log n)^c \)

  with \( a, b, \) and \( c \) as constants (and \( g(n) \) simpler than \( f(n) \)) Then, we refer to \( g(n) \) as the **order of growth** of \( f(n) \).

• **Example**

  • Assume that \( f(n) = .5n^2 - .5n \) characterizes the running time of an algorithm. Then \( f(n) \sim .5n^2 \)

  ![Graph](image)

  • The order of growth of \( f(n) \) is \( n^2 \).

• **Order of growth and \( \Theta \) notation are interchangable.**

  • Example: Assume that \( f(n) \) characterizes the running time of an algorithm.
  • Saying that \( f(n) \) has an order of growth of \( n^2 \) is the same as saying that \( f(n) = \Theta(n^3) \).

  We will not try to prove this.
• **Takeways**

  - For small datasets, complexity is often unimportant
  - For competing algorithms of similar complexity, the constant $a$ in $\sim a \cdot g(n)$ may be important. Example: Sorting

  Mergesort vs. Heapsort

  ![Mergesort vs. Heapsort](image1.png)

  - However, in comparing algorithms of different complexity, order of growth ($\Theta$) is all that matters

  Selection sort vs. mergesort and heapsort

  ![Selection sort vs. mergesort and heapsort](image2.png)
Some hints on complexity of algorithms

- a single loop

    for (int i=0; i

If there is one simple loop, the algorithm is \( \Theta(n) \). This is true even if there are some variations:

    for (int i=0; i<n/2; i++) { ... }
    for (int i=0; i<n; i+=5) { ... }

These are still \( \Theta(n) \)

- Here is a loop that is not \( \Theta(n) \)

    for (int i=1; i<n; i+=i)

Since \( i \) is doubled each time through the loop, this algorithm is \( \Theta(\log(n)) \)

- Nested loops

    for (int i=0; i<n; i++)
        for (int j=0; j<n; j++) { ... }
    for (int i=0; i<n; i++)
        for (int j=i; j<n; j++) { ... }

These are \( \Theta(n^2) \)

- Further nesting

    if \( f(n) \) is a polynomial, then \( f(n) = \Theta(n^k) \), where \( k \) is the degree of the polynomial

- Sequential looping

    If loop construct A has complexity \( f_1(n) \) and loop construct B has complexity \( f_2(n) \), then these constructs in sequence have complexity \( \max(f_1,f_2)(n) \)

Example: the code below is \( \Theta(n^2) \)

    int sum = 0;
    for (int i=0; i<n; i++)
        for (int j=0; j<n; j++)
            sum++;
    for (int k=0; k<n; k++)
        sum++;