Analysis of Algorithms
Criteria of Analysis

Which criteria should be taken into account?

- Running time
- Memory footprint
- Disk IO
- Network bandwidth
- Power consumption
- Lines of codes
- …
Average Case Vs Worst Case

Running Time

- Best case
- Average case
- Worst case

Different inputs of the same size
Case Study: Insertion Sort

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

\[
\text{for } j = 2 \text{ to } n \\
\text{key} = A[j] \\
\text{// Insert } A[j] \text{ into the sorted sequence } A[1 \ldots j - 1]. \\
i = j - 1 \\
\text{while } i > 0 \text{ and } A[i] > key \\
\quad A[i + 1] = A[i] \\
\quad i = i - 1 \\
A[i + 1] = key
\]

<table>
<thead>
<tr>
<th>cost</th>
<th>times</th>
</tr>
</thead>
<tbody>
<tr>
<td>(c_1)</td>
<td>(n)</td>
</tr>
<tr>
<td>(c_2)</td>
<td>(n - 1)</td>
</tr>
<tr>
<td>(c_3)</td>
<td>(0)</td>
</tr>
<tr>
<td>(c_4)</td>
<td>(n - 1)</td>
</tr>
<tr>
<td>(c_5)</td>
<td>(\sum_{j=2}^{n} t_j)</td>
</tr>
<tr>
<td>(c_6)</td>
<td>(\sum_{j=2}^{n} (t_j - 1))</td>
</tr>
<tr>
<td>(c_7)</td>
<td>(\sum_{j=2}^{n} (t_j - 1))</td>
</tr>
<tr>
<td>(c_8)</td>
<td>(n - 1)</td>
</tr>
</tbody>
</table>
Insertion Sort

Best case

Worst case
Growth of Functions

- It is hard to compute the actual running time
- The cost of the worst-case is a good measure
- The *growth* of the function is what interests us (Big Data)
- We are more concerned with comparing two functions, i.e., two algorithms.
Growth of Functions

g(n)
f(n)
$O$-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)
**Ω-notation**

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

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

\( g(n) \) is an asymptotic lower bound for \( f(n) \)
The notation \( \exists c_1, c_2 > 0, n_0 > 0 \),
\[ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \]
\[ n \geq n_0 \]

This means that \( g(n) \) is an asymptotic tight bound for \( f(n) \).
o-notation

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

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

g(n) is a non-tight asymptotic upper-bound for f(n)
ω-notation

∀c > 0
∃n₀ > 0
0 ≤ cg(n) ≤ f(n)
n ≥ n₀

g(n) is a non-tight asymptotic lower-bound for f(n)

f(n) = ω(g(n))
Compare two functions

\[ \lim_{n \to \infty} \frac{f(n)}{g(n)} \]

- 0: \( f(n) = o(g(n)) \)
- \( c > 0 \): \( f(n) = \Theta(g(n)) \)
- \( \infty \): \( f(n) = \omega(g(n)) \)
### 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>
Simple Rules

- We can omit constants
- We can omit lower order terms
- $\Theta(an^2 + bn + c)$ becomes $\Theta(n^2)$
- $\Theta(c_1)$ and $\Theta(c_2)$ become $\Theta(1)$
- $\Theta(\log_{k_1} n)$ and $\Theta(\log_{k_2} n)$ become $\Theta(\lg n)$
- $\Theta(\lg(n^k))$ becomes $\Theta(\lg n)$
- $\lg^{k_1}(n) = o(n^{k_2})$ for any positive constants $k_1$ and $k_2$
Popular Classes of Functions

- **Constant:** \( f(n) = \Theta(1) \)
- **Logarithmic:** \( f(n) = \Theta(\log(n)) \)
- **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**($A, n$)

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

$\Theta(n^2)$

*n-times*

*j-times*
Analysis of Recursive Algorithms

Section 4.4
Divide-and-Conquer

Big Dataset

Smaller Dataset

Conquer

Partial Answer

Smaller Dataset

Conquer

Partial Answer

Combine

Final Answer
Merge Sort

\textsc{merge-sort}(A, p, r)

\hspace{1cm} \textbf{if} \ p < r \hspace{1cm} // check for base case

\hspace{1.5cm} q = \lfloor (p + r)/2 \rfloor \hspace{1cm} // divide

\hspace{1.5cm} \textsc{merge-sort}(A, p, q) \hspace{1cm} // conquer

\hspace{1.5cm} \textsc{merge-sort}(A, q + 1, r) \hspace{1cm} // conquer

\hspace{1.5cm} \textsc{merge}(A, p, q, r) \hspace{1cm} // combine

\[ T(n) = \begin{cases} 
\hspace{1cm} c & \text{if } n \leq 1, \\
\hspace{1cm} 2T(n/2) + c \cdot n & \text{otherwise}
\end{cases} \]
Recursion Tree

\[ T(n) \]
Recursion Tree

```
    cn
   /   \
T(n/2) T(n/2)
```
Recursion Tree

\[ \begin{align*}
 cn \\
 \frac{cn}{2} & \quad \frac{cn}{2} \\
 T(n/4) & \quad T(n/4) & \quad T(n/4) & \quad T(n/4)
\end{align*} \]
Recursion Tree

The diagram represents a recursion tree with nodes labeled by $cn$, $cn/2$, and $c$ at various levels. The height of the tree is $\lg n$, and the total cost at the leaves is $cn$. The tree structure shows how the cost accumulates as it breaks down the problem into smaller subproblems.
Recursion Tree

\[
\begin{array}{c|c}
  i & \# \text{ nodes} \\
 0 & 1 \\
1 & 2 \\
2 & 4 \\
\vdots & \vdots \\
d & n \\
\end{array}
\]
Recursion Tree

\[ n = 2^d \]
\[ d = \log_2 n \]
Total Cost

- \( T(n) = \sum_{i=0}^{\lg n} c \cdot n \)
- \( T(n) = c \cdot n(\lg n + 1) \)
- \( T(n) = O(n \lg n) \)