Analyzing an Algorithm

Predicting the resources the algorithm requires

Resources: Memory
Communication Bandwidth
Logic Gates
Computation Time
Evaluating an algorithm

Mike: My algorithm can sort $10^6$ numbers in 3 seconds.
Bill: My algorithm can sort $10^6$ numbers in 5 seconds.

Mike: I’ve just tested it on my new Pentium IV processor.
Bill: I remember my result from my undergraduate studies (1985).

Mike: My input was a random permutation of 1..$10^6$.
Bill: My input was the sorted output, so I only needed to verify that it is sorted.
• **Processing time is surely a bad measure!!!**

• **We need a ‘stable’ measure, independent of the implementation.**
Time Complexity

• A function $T: \mathbb{N} \rightarrow \mathbb{N}$ of the input size.

• $T(n)$ is the number of operations the algorithm performs on an input of size $n$.

• Input Size:
  
  – Number of items in the input (sequence length)

  – Total number of bits representing the input

  – Possibly more than one size parameter  
    (In graph algorithms – # of nodes, # of edges)
Insertion Sort

Insertion-Sort(A)

1  for  i ← 2 to length[A]
2    key ← A[i]
3    j ← i-1
4    while j>0 and A[j]>key
6      j ← j-1
7      A[j+1] ← key
Insertion Sort

Insertion-Sort(A)

1. for i ← 2 to length[A]
2. key ← A[i]
3. j ← i-1
4. while j>0 and A[j]>key
5. \( A[j+1] \leftarrow A[j] \)
6. j ← j-1
7. \( A[j+1] \leftarrow key \)

\[
T(n) = 2n - 1 \quad + \quad 3 \sum_{i=2}^{n} t_i
\]
\[ T(n) = 2n - 1 + 3 \cdot \sum_{i=2}^{n} t_i \]
Insertion Sort - Best Case

Input: \(<1, 2, 3, 4, 5, 6>\) is already sorted

\[ t_i = 1 \quad \text{and} \quad \sum_{i=2}^{n} t_i = n - 1 \]

\[ T(n) = 2n - 1 + 3(n-1) = 5n - 4 \]

This is a linear function of \( n \)
Insertion Sort - Worst Case

Input: \( <6, 5, 4, 3, 2, 1> \)  
(sorted in reverse order)

\[ \Rightarrow t_i = i \quad \text{and} \quad \sum_{i=2}^{n} t_i = \frac{n(n + 1)}{2} - 1 \]

\[ \Rightarrow T(n) = 2n - 1 + 3 \cdot \left( \frac{n(n + 1)}{2} - 1 \right) = \frac{3n^2}{2} + \frac{7n}{2} - 4 \]

is a quadratic function of \( n \)
**Insertion Sort - Average Case**

On average:

1/2 the elements are smaller than $A[i]$ and 1/2 are bigger than $A[i]$

\[
\Rightarrow t_i = \frac{i}{2} \quad \text{and} \quad \sum_{i=2}^{n} t_i = \frac{1}{2} \cdot \left( \frac{n(n + 1)}{2} - 1 \right)
\]

\[
\Rightarrow T(n) = \frac{3n^2}{4} + \frac{11n}{4} - \frac{5}{2}
\]

is a **quadratic** function of $n$
Types of Analysis

- **Best case** \( T_{bc}(n) = \min_{I: |I|=n} \{ \text{time}(I) \} \)

- **Worst Case** \( T_{wc}(n) = \max_{I: |I|=n} \{ \text{time}(I) \} \)

- **Average Case** \( T_{avg} = \sum_{I: |I|=n} \text{time}(I) \cdot \Pr(I) \)

- **Other types** – amortized, common case, …
Worst Case Running Time

- *Gives upper bound on running time for any input.*
  A *guarantee* that the algorithm *never* takes longer.

- For some applications, worst case happens often
  
  **Example:** Searching a data base for missing information

- Average case is often roughly as bad as w.c.
  
  **Example:** Insertion Sort
Simplifying Assumptions

• The time to execute a simple statement, such as $x \leftarrow x+1$, depends only on the statement.

• The time to execute two statements, is the sum of the times to execute each.

• Any “simple” statement takes one time “unit” to execute.

• Any constant length sequence of instructions takes (the same) constant “unit”s of time.
The RAM Model of Computation

RAM: Random Access Machine
- Each simple operation takes 1 time step.
- Loops and subroutines are not simple operations.
- Each memory access takes one time step, and there is no shortage of memory.

For a given problem instance:
- Running time of an algorithm = # RAM steps.

Useful abstraction ⇒ allows us to analyze algorithms in a machine-independent fashion.
Which Function Grows Faster?

\[ n^3 + 2n^2 \quad \text{vs.} \quad 100n^2 + 1000 \]
Which Function Grows Faster?

\[ n^{0.1} \text{ vs. } \log n \]
Which Function Grows Faster?

$5n^5$ vs. $n!$
Order of Growth

• We are interested in the type of function the running time was, not the specific function (linear, quadratic,...)

• Really interested only in the leading terms

• Mostly interested only in the Rate of Growth of the leading terms
  ⇒ ignore constant coefficients
Big Oh notation
Upper Bound on Running Time

**Definition:** \( f(n) \in O(g(n)) \)

if there are \( c > 0 \) and \( n_0 > 0 \) such that

\[
f(n) \leq c \cdot g(n) \quad \text{for all } n > n_0
\]

**Intuition:** \( f(n) \) is “less than” \( g(n) \)

when we ignore small values of \( n \)

and constant multiples
Big-Oh - Example

The function \( T(n) = 3n^3 + 2n^2 \) is in \( O(n^3) \)

Proof: Let \( n_0 = 0 \) and \( c = 5 \)

for all \( n > n_0 \): \( 3n^2 + 2n^2 \leq 5n^3 \)

Note:

It is also true that \( T(n) \) is in \( O(n^4) \)
**Omega \( \Omega \) notation**

**Lower Bound on Running Time**

**Definition:** \( f(n) \in \Omega(g(n)) \)

if there are \( c > 0 \) and \( n_0 > 0 \) such that

\[
f(n) \geq c \cdot g(n) \quad \text{for all } n > n_0
\]

**Intuition:** \( f(n) \) is “greater than” \( g(n) \)
when we ignore small values of \( n \)
and constant multiples
Notice that:

$g(n) \in \Omega(f(n))$ 

$\iff$

$f(n) \in O(g(n))$
Big-Oh and Omega $\Omega$

A useful way to show Big-Oh relationships:

$$f(n) \in O(g(n)) \iff \lim_{n \to \infty} \frac{f(n)}{g(n)} \leq c$$

for some constant $c \geq 0$

A useful way to show Omega relationships:

$$f(n) \in \Omega(g(n)) \iff \lim_{n \to \infty} \frac{g(n)}{f(n)} \leq c$$

for some constant $c \geq 0$
Theta \( \Theta \) notation:
Tightly Bounding Running Time

**Definition:** \( f(n) \in \Theta(g(n)) \)

if there are \( c_1, c_2 > 0 \) and \( n_0 > 0 \) such that
\[
 c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)
\]
for all \( n > n_0 \)

**Intuition:** \( f(n) \) is “equal” to \( g(n) \)
when we ignore small values of \( n \)
and constant multiples
Theta $\Theta$

Note: $f(n) \in \Theta(g(n))$

\[\iff \quad f(n) \in O(g(n)) \amp f(n) \in \Omega(g(n))\]

A useful way to show Theta relationships:

$g(n) \in \Theta(f(n)) \iff \lim_{n \to \infty} \frac{g(n)}{f(n)} = c$

for some constant $c > 0$
Useful Properties

\[ f(n) = \mathcal{O}(g(n)) \land g(n) = \mathcal{O}(h(n)) \]
\[ \Rightarrow f(n) = \mathcal{O}(h(n)) \]

\[ f(n) = \Theta(g(n)) \iff g(n) = \Theta(f(n)) \]

\[ f(n) = \Omega(g(n)) \iff g(n) = \mathcal{O}(f(n)) \]
More Useful Properties

- \( O(f(n) + g(n)) = O(\max\{f(n), g(n)\}) \)
  
  \( \text{e.g.} \)
  
  \( O(n^3 + n) = O(n^3) \)

- \( \Omega(f(n) + g(n)) = \Omega(\max\{f(n), g(n)\}) \)

- \( \Theta(f(n) + g(n)) = \Theta(\max\{f(n), g(n)\}) \)
Example:

Show that for any real constants, \( a, b \in \mathbb{R} \) where \( b > 0 \), \((n + a)^b = \Theta(n^b)\)