CS 253: Algorithms

Chapter 3

Growth of Functions
Analysis of Algorithms

Goal:

- To analyze and compare algorithms in terms of *running time* and *memory requirements* (i.e. *time* and *space complexity*)

- In other words, how does the running time and space requirements change as we increase the input size n?

(sometimes we are also interested in the coding complexity)

- **Input size** (number of elements in the input)
  - size of an *array* or a *matrix*
  - # of bits in the binary representation of the input
  - vertices and/or edges in a graph, etc.
Types of Analysis

- **Worst case**
  - Provides an **upper bound** on running time
  - An absolute **guarantee** that the algorithm would not run longer, no matter what the inputs are

- **Best case**
  - Provides a **lower bound** on running time
  - Input is the one for which the algorithm runs the fastest

- **Average case**
  - Provides a **prediction** about the running time
  - Assumes that the input is random

\[
\text{Lower Bound} \leq \text{Running Time} \leq \text{Upper Bound}
\]
Computing the Running Time

- Measure the execution time?
  
  *Not a good idea!* It varies for different microprocessors!

- Count the number of statements executed?
  
  *Yes, but you need to be very careful!*

  High-level programming languages have statements which require a large number of low-level machine language instructions to execute (a function of the input size $n$).

  For example, a subroutine call can not be counted as one statement; it needs to be analyzed separately.

- Associate a "cost" with each statement.
  
  Find the "total cost" by multiplying the cost with the total number of times each statement is executed.

  *(we have seen examples before)*
Example

**Algorithm X**

```
sum = 0;
for(i=0; i<N; i++)
   for(j=0; j<N; j++)
      sum += arrY[i][j];
```

**Cost**

\[ \text{Total Cost} = c_1 + c_2 \times (N+1) + c_3 \times N \times (N+1) + c_4 \times N^2 \]
Asymptotic Analysis

- To compare two algorithms with running times $f(n)$ and $g(n)$, we need a **rough measure** that characterizes **how fast each function grows** with respect to $n$

- In other words, we are interested in how they behave **asymptotically** (i.e. for large $n$) (called **rate of growth**)

- **Big O notation**: asymptotic “less than” or “at most”:
  
  $$f(n)=O(g(n)) \text{ implies: } f(n) \leq g(n)$$

- **Ω notation**: asymptotic “greater than” or “at least”:
  
  $$f(n)= \Omega (g(n)) \text{ implies: } f(n) \geq g(n)$$

- **Θ notation**: asymptotic “equality” or “exactly”:
  
  $$f(n)= \Theta (g(n)) \text{ implies: } f(n) = g(n)$$
Big-O Notation

- We say $f_A(n) = 7n + 18$ is order $n$, or $O(n)$. It is, at most, roughly proportional to $n$.

- $f_B(n) = 3n^2 + 5n + 4$ is order $n^2$, or $O(n^2)$. It is, at most, roughly proportional to $n^2$.

- In general, any $O(n^2)$ function is faster-growing than any $O(n)$ function.
More Examples …

- \( n^4 + 100n^2 + 10n + 50 \Rightarrow O(n^4) \)
  \( 10n^3 + 2n^2 \Rightarrow O(n^3) \)
  \( n^3 - n^2 \Rightarrow O(n^3) \)

- constants
  
  10 is \( O(1) \)
  
  1273 is \( O(1) \)

- what is the rate of growth for Algorithm X studied earlier (in Big O notation)?

\[ \text{Total Time} = c_1 + c_2*(N+1) + c_2 * N*(N+1) + c_3*N^2 \]

If \( c_1, c_2, c_3 \), and \( c_4 \) are constants then \( \text{Total Time} = O(N^2) \)
Definition of Big O

- **O-notation**

\[ O(g(n)) = \{ f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0 \} . \]

\( g(n) \) is an *asymptotic upper bound* for \( f(n) \).
Note that $30n+8$ is $O(n)$. Can you find a $c$ and $n_0$ which can be used in the formal definition of Big O?

You can easily see that $30n+8$ isn’t less than $n$ anywhere ($n>0$).

But it is less than $31n$ everywhere to the right of $n=8$.

So, one possible $(c, n_0)$ pair that can be used in the formal definition:

\[ c = 31, \quad n_0 = 8 \]

$30n+8 \in O(n)$
**here Big-O Visualization**

$O(g(n))$ is the set of functions with smaller or same order of growth as $g(n)$
No Uniqueness

- There is no unique set of values for $n_0$ and $c$ in proving the asymptotic bounds

- Prove that $100n + 5 = O(n^2)$

  (i) $100n + 5 \leq 100n + n = 101n \leq 101n^2$ for all $n \geq 5$

  You may pick $n_0 = 5$ and $c = 101$ to complete the proof.

  (ii) $100n + 5 \leq 100n + 5n = 105n \leq 105n^2$ for all $n \geq 1$

  You may pick $n_0 = 1$ and $c = 105$ to complete the proof.
Definition of $\Omega$

$\Omega(g(n)) = \{ f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0 \}$.

$\Omega(g(n))$ is the set of functions with larger or same order of growth as $g(n)$.

$g(n)$ is an asymptotic lower bound for $f(n)$. 
Examples

- $5n^2 = \Omega(n)$

  \[ \exists \ c, \ n_0 \text{ such that: } 0 \leq cn \leq 5n^2 \Rightarrow cn \leq 5n^2 \Rightarrow c = 1 \text{ and } n > n_0 = 1 \]

- $100n + 5 \neq \Omega(n^2)$

  \[ \exists \ c, \ n_0 \text{ such that: } 0 \leq cn^2 \leq 100n + 5 \]

  since \[ 100n + 5 \leq 100n + 5n \quad \forall \ n \geq 1 \]

  \[ cn^2 \leq 105n \Rightarrow n(cn - 105) \leq 0 \]

  Since $n$ is positive \[ (cn - 105) \leq 0 \Rightarrow n \leq 105/c \]

  \[ \Rightarrow \text{contradiction: } n \text{ cannot be smaller than a constant} \]

- $n = \Omega(2n), \quad n^3 = \Omega(n^2), \quad n = \Omega(\log n)$
Definition of $\Theta$

$$\Theta(g(n)) = \{ f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that}$$
$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0 \}.$$

$\Theta(g(n))$ is the set of functions with the same order of growth as $g(n)$.

$g(n)$ is an \textit{asymptotically tight bound} for $f(n)$.
Examples

\[ n^2/2 - n/2 = \Theta(n^2) \]

- \[ \frac{1}{2} n^2 - \frac{1}{2} n \leq \frac{1}{2} n^2 \quad \forall n \geq 0 \quad \Rightarrow \quad c_2 = \frac{1}{2} \]

- \[ \frac{1}{2} n^2 - \frac{1}{2} n \geq \frac{1}{2} n^2 - \frac{1}{2} n \cdot \frac{1}{2} n \quad (\forall n \geq 2) = \frac{1}{4} n^2 \quad \Rightarrow \quad c_1 = \frac{1}{4} \]

n \neq \Theta(n^2): \quad c_1 n^2 \leq n \leq c_2 n^2 \quad \Rightarrow \text{only holds for: } n \leq 1/c_1

6n^3 \neq \Theta(n^2): \quad c_1 n^2 \leq 6n^3 \leq c_2 n^2

\Rightarrow \text{only holds for: } n \leq c_2 /6

n \neq \Theta(\log n): \quad c_1 \log n \leq n \leq c_2 \log n

\Rightarrow c_2 \geq n/\log n, \forall n \geq n_0 - \text{impossible}
Relations Between Different Sets

- Subset relations between order-of-growth sets.
Common orders of magnitude
Table 1.4  Execution times for algorithms with the given time complexities

<table>
<thead>
<tr>
<th>$n$</th>
<th>$f(n) = \log n$</th>
<th>$f(n) = n$</th>
<th>$f(n) = n \log n$</th>
<th>$f(n) = n^2$</th>
<th>$f(n) = n^3$</th>
<th>$f(n) = 2^n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.003 $\mu$s*</td>
<td>0.01 $\mu$s</td>
<td>0.033 $\mu$s</td>
<td>0.1 $\mu$s</td>
<td>1 $\mu$s</td>
<td>$\mu$s</td>
</tr>
<tr>
<td>20</td>
<td>0.004 $\mu$s</td>
<td>0.02 $\mu$s</td>
<td>0.086 $\mu$s</td>
<td>0.4 $\mu$s</td>
<td>8 $\mu$s</td>
<td>ms</td>
</tr>
<tr>
<td>30</td>
<td>0.005 $\mu$s</td>
<td>0.03 $\mu$s</td>
<td>0.147 $\mu$s</td>
<td>0.9 $\mu$s</td>
<td>27 $\mu$s</td>
<td>s</td>
</tr>
<tr>
<td>40</td>
<td>0.005 $\mu$s</td>
<td>0.04 $\mu$s</td>
<td>0.213 $\mu$s</td>
<td>1.6 $\mu$s</td>
<td>64 $\mu$s</td>
<td>18.3 ms</td>
</tr>
<tr>
<td>50</td>
<td>0.005 $\mu$s</td>
<td>0.05 $\mu$s</td>
<td>0.282 $\mu$s</td>
<td>2.5 $\mu$s</td>
<td>125 $\mu$s</td>
<td>13 days</td>
</tr>
<tr>
<td>$10^2$</td>
<td>0.007 $\mu$s</td>
<td>0.10 $\mu$s</td>
<td>0.664 $\mu$s</td>
<td>10 $\mu$s</td>
<td>1 ms</td>
<td>$4 \times 10^{12}$ years</td>
</tr>
<tr>
<td>$10^3$</td>
<td>0.010 $\mu$s</td>
<td>1.00 $\mu$s</td>
<td>9,966 $\mu$s</td>
<td>1 ms</td>
<td>1 s</td>
<td></td>
</tr>
<tr>
<td>$10^4$</td>
<td>0.013 $\mu$s</td>
<td>0 $\mu$s</td>
<td>130 $\mu$s</td>
<td>100 ms</td>
<td>16.7 min</td>
<td></td>
</tr>
<tr>
<td>$10^5$</td>
<td>0.017 $\mu$s</td>
<td>0.10 ms</td>
<td>1.67 ms</td>
<td>10 s</td>
<td>11.6 days</td>
<td></td>
</tr>
<tr>
<td>$10^6$</td>
<td>0.020 $\mu$s</td>
<td>1 ms</td>
<td>19.93 ms</td>
<td>16.7 min</td>
<td>31.7 years</td>
<td></td>
</tr>
<tr>
<td>$10^7$</td>
<td>0.023 $\mu$s</td>
<td>0.01 s</td>
<td>0.23 s</td>
<td>1.16 days</td>
<td>31,709 years</td>
<td></td>
</tr>
<tr>
<td>$10^8$</td>
<td>0.027 $\mu$s</td>
<td>0.10 s</td>
<td>2.66 s</td>
<td>115.7 days</td>
<td>$3.17 \times 10^7$ years</td>
<td></td>
</tr>
<tr>
<td>$10^9$</td>
<td>0.030 $\mu$s</td>
<td>1 s</td>
<td>29.90 s</td>
<td>31.7 years</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*1 $\mu$s = $10^{-6}$ second.

†1 ms = $10^{-3}$ second.
Logarithms and properties

- In algorithm analysis we often use the notation “$\log n$” without specifying the base.

**Binary logarithm**

- $\lg n = \log_2 n$

**Natural logarithm**

- $\ln n = \log_e n$

- $\log x^y = y \log x$

- $\log xy = \log x + \log y$

- $\log \frac{x}{y} = \log x - \log y$

- $a^{\log_b x} = x^{\log_b a}$

- $\log_b x = \frac{\log_a x}{\log_a b}$

- $\lg^k n = (\lg n)^k$

- $\lg \lg n = \lg(\lg n)$
For each of the following pairs of functions, either \( f(n) \) is \( O(g(n)) \), \( f(n) \) is \( \Omega(g(n)) \), or \( f(n) = \Theta(g(n)) \). Determine which relationship is correct.

- \( f(n) = \log n^2; \ g(n) = \log n + 5 \)  
  \( f(n) = \Theta(g(n)) \)
- \( f(n) = n; \ g(n) = \log n^2 \)  
  \( f(n) = \Omega(g(n)) \)
- \( f(n) = \log \log n; \ g(n) = \log n \)  
  \( f(n) = O(g(n)) \)
- \( f(n) = n; \ g(n) = \log^2 n \)  
  \( f(n) = \Omega(g(n)) \)
- \( f(n) = n \log n + n; \ g(n) = \log n \)  
  \( f(n) = \Omega(g(n)) \)
- \( f(n) = 10; \ g(n) = \log 10 \)  
  \( f(n) = \Theta(g(n)) \)
- \( f(n) = 2^n; \ g(n) = 10n^2 \)  
  \( f(n) = \Omega(g(n)) \)
- \( f(n) = 2^n; \ g(n) = 3^n \)  
  \( f(n) = O(g(n)) \)
Properties

**Theorem:**

\[ f(n) = \Theta(g(n)) \iff f = O(g(n)) \text{ and } f = \Omega(g(n)) \]

- **Transitivity:**
  - \( f(n) = \Theta(g(n)) \) and \( g(n) = \Theta(h(n)) \) \( \Rightarrow \) \( f(n) = \Theta(h(n)) \)
  - Same for \( O \) and \( \Omega \)

- **Reflexivity:**
  - \( f(n) = \Theta(f(n)) \)
  - Same for \( O \) and \( \Omega \)

- **Symmetry:**
  - \( f(n) = \Theta(g(n)) \) if and only if \( g(n) = \Theta(f(n)) \)

- **Transpose symmetry:**
  - \( f(n) = O(g(n)) \) if and only if \( g(n) = \Omega(f(n)) \)