Asymptotic Notation

**Ref:** Chapter 3 of text.

**Purposes:**
- Express resource requirements of an algorithm in a simple manner.
- Study behavior as input size becomes large.

**Assumptions:**
- Functions are defined only for non-negative integer inputs.
- The functions take on non-negative integer values.

**Big-O Notation:** (Upper bound)

**Definition:** $g(n)$ is $O(f(n))$ if there are non-negative constants $c$ and $n_0$ such that $g(n) \leq c f(n)$ for all $n \geq n_0$.

When $g(n) = O(f(n))$:
- For “large enough” $n$, $g(n)$ is bounded by $c f(n)$.
- $g(n)$ grows no faster than $f(n)$.

**Example 1:** Let $g(n) = 7n + 4$.
\[ g(n) \leq 11n \text{ for all } n \geq 1. \] So, $g(n) = O(n)$ with $c = 11$ and $n_0 = 1$.

**Example 2:** Let $g(n) = 2n \log_2 n + 7n$.
\[ g(n) \leq 9n \log_2 n \text{ when } \log_2 n \geq 1, \text{ that is, for all } n \geq 2. \] So, $g(n) = O(n \log_2 n)$ with $c = 9$ and $n_0 = 2$.

**Example 3:** Let $g(n) = \alpha$, where $\alpha$ is a positive constant.
\[ g(n) = \alpha \times 1 \text{ for all } n \geq 0. \] So, $g(n) = O(1)$ with $c = \alpha$ and $n_0 = 0$.

**Example 4:** Let $g(n)$ be the polynomial function defined by $g(n) = \sum_{i=0}^{k} a_i n^i$.
\[ g(n) \leq \sum_{i=0}^{k} |a_i| n^k \text{ for all } n \geq 0. \] So, $g(n) = O(n^k)$ with $c = \sum_{i=0}^{k} |a_i|$ and $n_0 = 0$. 

---

2-1

---

2-2
Lemma: Let \( g_1(n) = O(f_1(n)) \) and \( g_2(n) = O(f_2(n)) \). Then
1. \( g_1(n) + g_2(n) = O(f_1(n) + f_2(n)) \).
2. \( g_1(n) + g_2(n) = O(\max\{f_1(n), f_2(n)\}) \).
3. \( g_1(n) * g_2(n) = O(f_1(n) * f_2(n)) \).

Proof: From the given conditions, we know that there are constants \( c_1, c_2, n_1 \) and \( n_2 \) such that
\[
\begin{align*}
g_1(n) &\leq c_1 f_1(n) & \text{for all } n \geq n_1 \\
g_2(n) &\leq c_2 f_2(n) & \text{for all } n \geq n_2.
\end{align*}
\]

Part 1: Let \( c = \max\{c_1, c_2\} \) and \( n_0 = \max\{n_1, n_2\} \). Then,
\[
\begin{align*}
g_1(n) + g_2(n) &\leq c_1 f_1(n) + c_2 f_2(n) & \text{for all } n \geq n_0 \\
&\leq c [f_1(n) + f_2(n)] & \text{for all } n \geq n_0.
\end{align*}
\]
This proves Part 1.

Part 2: From the proof for Part 1
\[
g_1(n) + g_2(n) \leq 2c [\max\{f_1(n), f_2(n)\}]
\]
for all \( n \geq n_0 \).

Thus, Part 2 is proven.

Part 3: Exercise.

Comments:
1. Try to get the simplest possible function inside big-O (i.e., drop lower order terms).
2. Try to get as tight a big-O estimate as possible.
3. Suppose \( g(n) = n^2 \). Note that \( g(n) \neq O(n) \).
   (Prove this by contradiction.)
**Ω Notation:** (Lower bound)

**Definition:** $g(n)$ is $\Omega(f(n))$ if there are positive constants $c$ and $n_0$ such that $g(n) \geq cf(n)$ for all $n \geq n_0$.

When $g(n) = \Omega(f(n))$:  
- For “large enough” $n$, $g(n)$ is at least $cf(n)$.  
- $g(n)$ grows at least as fast as $f(n)$.

**Example 1:** Let $g(n) = 7n - 4$.  
$g(n) = 6n + (n - 4)$. So, for all $n \geq 4$, $g(n) \geq 6n$. So, $g(n) = \Omega(n)$ with $c = 6$ and $n_0 = 4$.

**Example 2:** Let $g(n) = n \log_2 n - 7n$.  
$g(n) = 0.5n \log_2 n + (0.5n \log_2 n - 7n)$. The quantity within parentheses is non-negative when $0.5 \log_2 n \geq 7$, that is, when $n \geq 2^{14}$. Thus, $g(n) \geq 0.5n \log_2 n$ for all $n \geq 2^{14}$. So, $g(n) = \Omega(n \log_2 n)$ with $c = 0.5$ and $n_0 = 2^{14}$.

**Exercise:** Let $g(n)$ be the polynomial function defined by $g(n) = \sum_{i=0}^{k} a_i n^i$ where $a_k > 0$. Show that $g(n) = \Omega(n^k)$.

**Θ Notation:** (Tight bound)

**Definition:** $g(n)$ is $\Theta(f(n))$ if $g(n) = O(f(n))$ and $g(n) = \Omega(f(n))$.

When $g(n) = \Theta(f(n))$:  
- There are positive constants $c_1$, $c_2$ and $n_0$ such that $c_1 f(n) \leq g(n) \leq c_2 f(n)$ for all $n \geq n_0$.  
- The growth rates of $f(n)$ and $g(n)$ are equal.

**Examples:**

(a) $g(n) = 3n^2 + 8n - 3$. Then $g(n) = \Theta(n^2)$.
(b) $g(n) = n \log_2 n - 7n$. Then $g(n) = \Theta(n \log_2 n)$.
(c) $g(n) = \sum_{i=0}^{k} a_i n^i$ where $a_k > 0$. Then $g(n) = \Theta(n^k)$.  


Little-o Notation: (Weak upper bound)

Definition: \( g(n) \) is \( o(f(n)) \) if \( \lim_{n \to \infty} \frac{g(n)}{f(n)} = 0 \).

When \( g(n) = o(f(n)) \), the growth rate of \( g(n) \) is less than that of \( f(n) \).

Examples:
(a) \( g(n) = 2n \). Then \( g(n) = o(n^2) \).
(b) \( g(n) = n \log_2 n + 7n \). Then \( g(n) = o(n^2) \).

Notes:
1. For any positive constant \( k \), the function \( (\log_2 n)^k \) is \( o(n) \).
2. The function \( 2n \neq o(n) \). (Recall that \( 2n = \Theta(n) \).)

Using limits to decide relationships:
(a) Suppose \( \lim_{n \to \infty} \frac{g(n)}{f(n)} = 0 \). Then \( g(n) = o(f(n)) \). (Also, \( f(n) = \Omega(g(n)) \).)
(b) Suppose \( \lim_{n \to \infty} \frac{g(n)}{f(n)} = \infty \). Then \( g(n) = \Omega(f(n)) \). (Also, \( f(n) = o(g(n)) \).)
(c) Suppose \( \lim_{n \to \infty} \frac{g(n)}{f(n)} = c \), where \( c \) is a positive constant. Then \( g(n) = \Theta(f(n)) \).

Some common functions:
- \( k \) is a constant (independent of \( n \)).
- Functions are listed in increasing order of growth rate.

\[
\begin{array}{c|c|c}
\text{Function} & \text{Rate} & \text{Notes} \\
\hline
\log n & \text{Logarithmic} & \\
(\log n)^k & \text{Poly logarithmic} & \\
n^k & \text{Polynomial} & \\
2^n & \text{Exponential} & \\
2^{2^n} & \text{Double Exponential} & \\
c \text{ (or O(1))} & \text{Constant} & \\
\end{array}
\]
Note: In $O(\log n)$, the base of the logarithm is assumed to be a constant.

Reason: For any two constants $a$, $b$ such that $a > 1$, $b > 1$ and $a \leq b$, $\log_a n = (\log_b a \log_b n)$.

Simple observations:
- For an array of size $n$, the time to access any element is $O(1)$.
- For a linked list with $n$ nodes, the time to access the $i^{th}$ node is $O(i)$.

Example of running time analysis:

**Insertion Sort:** (Increasing order)

Given array $A[1 .. n]$ (with $n = \text{length}(A)$)
2. Assume that $A[1 .. j - 1]$ is sorted.
4. Repeat Steps 2 and 3 until the whole array is sorted.

**Pseudocode:** (from page 17 of text)

```plaintext
for $j = 2$ to length($A$)
    key = $A[j]$;  $i = j - 1$
    while ($i > 0$ and $A[i] > key$)
        $A[i + 1] = A[i]$;  $i = i - 1$
    $A[i + 1] = key$
```

A crude analysis:
- The no. of iterations of for loop = $n - 1$.
- Time spent in each iteration = $O(n)$ due to the while loop.
- So, running time = $O(n(n - 1)) = O(n^2)$.

A more careful analysis:
- In for loop, $j$ varies from 2 to $n$.
- For each value of $j$
  - Statements outside the while loop take $O(1)$ (say, $c_1$) time.
  - No. of iterations of the while loop is at most $j - 1$. 

2.9
Each iteration of while loop takes $O(1)$ (say, $c_2$) time.
- Time for while loop $\leq c_2(j - 1)$.
- Time taken by this iteration of for loop $\leq c_1 + c_2(j - 1)$.

So, total time $\leq \sum_{j=2}^{n-1} [c_1 + c_2(j - 1)] = O(n^2)$.

**Lower bound on the running time:**
- For this input, for each value of $j$, the while loop executes $j - 1$ times.

For this input, the running time of the algorithm is at least

$$\sum_{j=2}^{n} c_2(j - 1)$$

That is, running time of Insertion Sort $= \Omega(n^2)$.

**Conclusion:** Running time of Insertion Sort $= \Theta(n^2)$.

---

**Satisfiability problem:** (SAT)

**Input:** A Boolean formula $F$ with $n$ variables.

**Output:** Is there an assignment of Boolean values to the variables so that $F$ evaluates to true?

**Example instances:**
(a) $(x_1 \lor x_2 \lor \overline{x}_3) \land (\overline{x}_1 \lor \overline{x}_2) \land (x_2 \lor \overline{x}_3)$ : Satisfiable.
(b) $(x_1 \lor x_2) \land \overline{x}_1 \land \overline{x}_2$ : Not satisfiable.

**Exhaustive search algorithm for SAT:**
1. Compute the value of the expression for each combination of truth values to the $n$ variables.
2. If one of the combinations results in true, output “Yes”; otherwise, output “No”.

**Running time:** $O(2^n T(|F|))$, where $|F|$ is the size of $F$ and $T(|F|)$ is the time needed to evaluate $F$ for one combination of values.

**Note:** No significantly better algorithm is known for SAT.