Asymptotic Analysis-Ch. 3

- Names for order of growth for classes of algorithms:
  - **constant** \( \Theta(n^0) = \Theta(1) \)
  - **logarithmic** \( \Theta(\log n) \)
  - **linear** \( \Theta(n) \)
  - \(<\text{en log en}>>\) \( \Theta(n \log n) \)
  - **quadratic** \( \Theta(n^2) \)
  - **cubic** \( \Theta(n^3) \)
  - **polynomial** \( \Theta(n^k), k \geq 1 \)
  - **exponential** \( \Theta(a^n), a > 1 \)

Asymptotic analysis valid only in the limit

Example: an algorithm with running time of order \( n^2 \) will "eventually" (i.e., for sufficiently large \( n \)) run slower than one with running time of order \( n \), which in turn will eventually run slower than one with running time of order \( \log n \).

"Big Oh", "Theta", and "Big Omega" are the tools we will use to make these notions precise.

Note: By saying valid "in the limit" or "asymptotically", we mean that the comparison may not hold true for small values of \( n \).

"Big Oh" - Upper Bounding Running Time

**Definition:** \( f(n) \in O(g(n)) \) if there exist constants \( c > 0 \) and \( n_0 > 0 \) such that

\[
0 \leq f(n) \leq cg(n) \quad \text{for all } n \geq n_0.
\]

Intuition:
- \( f(n) \in O(g(n)) \) means \( f(n) \) is "of order at most", or "less than or equal to" \( g(n) \) when we ignore small values of \( n \)
- \( f(n) \) is eventually less than or equal to some constant multiple of \( g(n) \) for large enough \( n \)
- For large enough \( n \), some constant multiple of \( g(n) \) is an upper bound for \( f(n) \)

Example: \((\log n)^2 \) is \( O(n) \)

\[
f(n) = (\log n)^2 \quad g(n) = n
\]

\((\log n)^2 \leq n \) for \( c=1 \) and all \( n \geq 16 \), so \((\log n)^2 \) is \( O(n) \)

Basic idea: ignore constant factor differences and lower-order terms

\[
617n^3 + 277n^2 + 720n + 7n \in O(?)
\]

Proving Running Time:

finding values of \( c \) and \( n \)

Consider

\[
f(n) = 5n^3 + 2n^2 + 22n + 6
\]

We claim that

\[
f(n) \in O(n^3)
\]

Let \( c = 6 \) and \( n_0 = 6 \). Then

\[
5n^3 + 2n^2 + 22n + 6 \leq 6n^3
\]

for every \( n \geq 6 \)
**Proving Running Time:**
finding values of c and n

If
\[ f(n) = 5n^3 + 2n^2 + 22n + 6 \]
we have seen that
\[ f(n) \in O(n^3) \]
but \( f(n) \) is not in \( O(n^2) \), because no positive value for \( c \) or \( n_0 \) works for large enough \( n \).

**Logarithms**

- Asymptotics allow us to ignore log base
- Different base changes only constant factor
- When we say \( f(n) \in O(\log n) \), the base is unimportant. Usually, we use \( \log_2 \)

\[ \log_b n = \frac{\log_2 n}{\log_2 b} \]

**Important Notation**

Sometimes you will see notation like this:
\[ f(n) \in O(n^2) + O(n) \]

- Each occurrence of big-O symbol has a distinct constant multiple.
- But \( O(n^2) \) term dominates \( O(n) \) term, so the above is equivalent to \( f(n) \in O(n^2) \)

**Example: InsertionSort**

**INPUT:**
An array \( A \) of \( n \) numbers
\[ a_1, a_2, ..., a_n \]
**OUTPUT:**
A permutation of input array \( \{a_1', a_2', ..., a_n'\} \) such that \( a_1' \leq a_2' \leq ... \leq a_n' \).

```plaintext
1. for j = 2 to length(A)
2. key = A[j]
3. i = j - 1
4. while i > 0 and A[i] > key
6. i = i - 1
7. A[i+1] = key
```

Time for execution on input array of length \( n \) (if exact count is made of the number of times each line is executed):
- best-case: \( b(n) = 5n - 4 \)
- worst-case: \( w(n) = 3n^2/2 + 11n/2 - 4 \)

**Insertion Sort - Time Complexity**

Time complexities for insertion sort are:
- best-case: \( b(n) = 5n - 4 \)
- worst-case: \( w(n) = 3n^2/2 + 11n/2 - 4 \)

Questions:
1. is \( b(n) \in O(n) \) ? Yes (\( 5n - 4 < 6n \) for all \( n \geq 0 \))
2. is \( w(n) \in O(n) \) ? No (\( 3n^2/2 + 11n/2 - 4 \geq 3n \) for all \( n \geq 1 \))
3. is \( w(n) \in O(n^2) \)? Yes (\( 3n^2/2 + 11n/2 - 4 < 4n^2 \) for all \( n \geq 0 \))
4. is \( w(n) \in O(n^3) \)? Yes (\( 3n^2/2 + 11n/2 - 4 \leq 2n^3 \) for all \( n \geq 2 \))

**Plotting run-time graphically**

\[ f(n) = 2n+6 \]
\[ g(n) = 4n \]

\( 2n+6 \) is \( O(n) \) since \( 2n_0+6 \leq 4n_0 \) for \( n_0 \geq 3 \)

\( n_0 = 3 \) input size
On the other hand... \( n^2 \) is not \( O(n) \) because there are no \( c \) and \( n_0 \) such that:
\[ n^2 \leq cn \text{ for all } n \geq n_0 \]

"Big Omega" - Lower Bounding Running Time

**Definition:** \( f(n) \in \Omega(g(n)) \) if there exist constants \( c > 0 \) and \( n_0 > 0 \) such that
\[ f(n) \geq cg(n) \text{ for all } n \geq n_0. \]

**Intuition:**
- \( f(n) \in \Omega(g(n)) \) means \( f(n) \) is "of order at least" or "greater than or equal to" \( g(n) \) when we ignore small values of \( n \).
- \( f(n) \) is eventually greater than or equal to some constant multiple of \( g(n) \) for large enough \( n \).
- For large enough \( n \), some constant multiple of \( g(n) \) is a lower bound for \( f(n) \).

"Theta" - Tightly Bounding Running Time

**Definition:** \( f(n) \in \Theta(g(n)) \) if there exist constants \( c_1, c_2 > 0 \) and \( n_0 > 0 \) such that
\[ c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0. \]

**Intuition:**
- \( f(n) \in \Theta(g(n)) \) means \( f(n) \) is "of the same order as", or "equal to" \( g(n) \) when we ignore small values of \( n \).
- \( f(n) \) is eventually trapped between two constant multiples of \( g(n) \) for large enough \( n \).

**Showing "Theta" relationships:**
- Show both a "Big Oh" and "Big Omega" relationship.

Insertion Sort - Time Complexity

Time complexities for insertion sort are:

- best-case: \( b(n) = 5n - 4 \)
- worst-case: \( w(n) = 3n^2/2 + 11n/2 - 4 \)

**Questions:**
1. is \( b(n) \in \Theta(n) \)? Yes because \( b(n) = O(n) \) and \( \Omega(n) \)
2. is \( w(n) \in \Theta(n) \)? No because \( w(n) \neq O(n) \)
3. is \( w(n) \in \Theta(n^2) \)? Yes because \( w(n) = O(n^2) \) and \( \Omega(n^2) \)
4. is \( w(n) \in \Theta(n^3) \)? No because \( w(n) \neq \Omega(n^3) \)

Asymptotic Analysis

- Classifying algorithms is generally done in terms of worst-case running time:
  - \( O(f(n)) \): Big Oh--asymptotic upper bound.
  - \( \Omega(f(n)) \): Big Omega--asymptotic lower bound
  - \( \Theta(f(n)) \): Theta--asymptotic tight bound
Useful Properties for Asymptotic Analysis

- If \( f(n) \in O(g(n)) \) and \( g(n) \in O(h(n)) \), then \( f(n) \in O(h(n)) \) (transitivity)
  
  **Intuition:** if \( f(n) \leq c \cdot g(n) \) and \( g(n) \leq c' \cdot h(n) \) then \( f(n) \leq c'' \cdot h(n) \)

- \( f(n) \in O(g(n)) \) iff \( g(n) \in \Omega(f(n)) \) (transpose symmetry)
  
  **Intuition:** \( f(n) \leq c \cdot g(n) \) iff \( g(n) \geq c' \cdot f(n) \)

- \( f(n) \in o(g(n)) \) iff \( g(n) \in \Theta(f(n)) \) (symmetry)
  
  **Intuition:** \( f(n) \leq c \cdot g(n) \) iff \( g(n) = c' \cdot f(n) \)

**Little Oh**

"Little Oh" notation is used to denote strict upper bounds. (Big-Oh bounds are not necessarily strict inequalities).

**Definition:** \( f(n) \in o(g(n)) \) if for every \( c > 0 \), there exists some \( n_0 > 0 \) such that for all \( n \geq n_0 \), \( f(n) < cg(n) \).

**Intuition:**
- \( f(n) \in o(g(n)) \) means \( f(n) \) is "strictly less than" any constant multiple of \( g(n) \) when we ignore small values of \( n \)
- \( f(n) \) is trapped below any constant multiple of \( g(n) \) for large enough \( n \)

**Little Omega**

"Little Omega" notation is used to denote strict lower bounds (\( \Omega \) bounds are not necessarily strict inequalities).

**Definition:** \( f(n) \in \omega(g(n)) \) if for every \( c > 0 \), there exists some \( n_0 > 0 \) such that for all \( n \geq n_0 \), \( f(n) > cg(n) \).

**Intuition:**
- \( f(n) \in \omega(g(n)) \) means \( f(n) \) is "strictly greater than" any constant multiple of \( g(n) \) when we ignore small values of \( n \)
- \( f(n) \) is trapped above any constant multiple of \( g(n) \) for large enough \( n \)

**Handy Asymptotic Facts**

- If \( T(n) \) is a polynomial function of degree \( k \), then \( T(n) \in O(n^k) \)
- \( n^b \in O(a^n) \) for any constants \( a > 1, b > 0 \) (Exponentials dominate polynomials). In particular, any exponential function with a base strictly greater than 1 grows faster than any polynomial function.
- \( n! \in o(n^n) \)
- \( n! \in \omega(2^n) \)
- \( \lg(n!) \in \Theta(n \lg n) \) (by Stirling's approximation)
- The base of an exponential function and the degree of a polynomial matter asymptotically, but the base of a logarithm does not.
• Iterated logarithm function \((\log^* n)\):
  - the number of times the \(\log\) function can be iteratively applied before the result is less than or equal to 1
  - "log star of \(n\"
  - Very slow growing, e.g. \(\log^*(2^{65536}) = 5\)

eg:
\[
\begin{align*}
\log^2 &= 1 \\
\log^4 &= 2 \\
\log^6 &= 3 \\
\log^{65536} &= 4
\end{align*}
\]

---

### Basic asymptotic efficiency of code

<table>
<thead>
<tr>
<th>Class</th>
<th>Name</th>
<th>Sample algorithm type</th>
</tr>
</thead>
<tbody>
<tr>
<td>(O(1))</td>
<td>Constant</td>
<td>Algorithm ignores input (i.e., can't even scan input)</td>
</tr>
<tr>
<td>(O(\log n))</td>
<td>Logarithmic</td>
<td>Cuts problem size by constant fraction on each iteration</td>
</tr>
<tr>
<td>(O(n))</td>
<td>Linear</td>
<td>Algorithm scans its input (at least)</td>
</tr>
<tr>
<td>(O(n\log n))</td>
<td>&quot;n-log-n&quot;</td>
<td>Some divide and conquer</td>
</tr>
<tr>
<td>(O(n^n))</td>
<td>Quadratic</td>
<td>Loop inside loop = &quot;nested loop&quot;</td>
</tr>
<tr>
<td>(O(n^2))</td>
<td>Cubic</td>
<td>Loop inside nested loop</td>
</tr>
<tr>
<td>(O(2^n))</td>
<td>Exponential</td>
<td>Algorithm generates all subsets of (n)-element set of binary values</td>
</tr>
<tr>
<td>(O(n!))</td>
<td>Factorial</td>
<td>Algorithm generates all permutations of (n)-element set</td>
</tr>
</tbody>
</table>

---

End of Lecture 3