Main idea: Running time is measured in the limit as the input size $n$ grows to infinity.

• Calculate algorithm running time in terms of its rate of growth with increasing problem size. To make this task easier, we can
  - identify terms of highest order and ignore lower order terms
  - disregard multiplicative constants

Saying an algorithm has running time $\theta(n^2)$ says that the order of growth of the running time is in the set of functions whose running time is $n^2$, a quadratic function of $n$
Asymptotic Analysis

- Names for classes of algorithms:

  - **constant**: \( \Theta(n^0) = \Theta(1) \)
  - **logarithmic**: \( \Theta(lgn) \)
  - **polylogarithmic**: \( \Theta(lg^k n), \ k \geq 1 \)
  - **linear**: \( \Theta(n) \)
  - **linearithmic**: \( \Theta(n \lg n) \)
  - **quadratic**: \( \Theta(n^2) \)
  - **cubic**: \( \Theta(n^3) \)
  - **polynomial**: \( \Theta(n^k), \ k \geq 1 \)
  - **exponential**: \( \Theta(a^n), \ a > 1 \)

Growth Rate Increasing
Asymptotic Analysis

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 \( \log n \).

Asymptotic analysis in terms of "Big Oh", "Theta", and "Big Omega" are the tools we will use to make these notions precise.

Note: Our conclusions will only be valid "in the limit" or "asymptotically". That is, they may not hold true for small values of \( n \). And we really don't care about small values of \( n \).
"Big Oh" - Upper Bounding Running Time

**Definition:** \( f(n) \in O(g(n)) \) if there exist constants \( c > 0 \) and \( n_0 \geq 1 \) such that

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

**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 \( n \) and constants

- \( f(n) \) is eventually trapped below (or = to) some constant multiple of \( g(n) \)

- some constant multiple of \( g(n) \) is an upper bound for \( f(n) \) (for large enough \( n \))
Alternate Definition:

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

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 \( n \) and constants
- \( f(n) \) is eventually trapped below (or = to) some constant multiple of \( g(n) \)
- some constant multiple of \( g(n) \) is an upper bound for \( f(n) \)
  (for large enough \( n \))

(note: \( n_0 \) must be at least 1)
Example: \((\lg n)^2\) is \(O(n)\)

\[(\lg n)^2 \leq n\] for all \(n \geq 16\), so \((\lg n)^2\) is \(O(n)\)
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'.

**InsertionSort(A)**

1. for j = 2 to A.length
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:
- best-case: \( b(n) \approx 5n - 4 \)
- worst-case: \( w(n) \approx 3n^2/2 + 11n/2 - 4 \)
Insertion Sort - Time Complexity

Time complexities for insertion sort are:
- best-case: \( b(n) = 5n - 4 \)
- worst-case: \( w(n) = 3n^2/2 + 7n/2 - 4 \)

Questions:
1. is \( b(n) = O(n) \) ? Yes (\( 5n - 4 < 6n \) for all \( n \geq 0 \))
2. is \( w(n) = O(n) \) ? No (\( 3n^2/2 + 7n/2 - 4 \geq 3n \) for all \( n \geq 1 \))
3. is \( w(n) = O(n^2) \) ? Yes (\( 3n^2/2 + 7n/2 - 4 \leq 4n^2 \) for all \( n \geq 0 \))
4. is \( w(n) = O(n^3) \) ? Yes (\( 3n^2/2 + 7n/2 - 4 \leq 2n^3 \) for all \( n \geq 2 \))
Confused?

Basic idea: ignore constant factors and lower-order terms

• $617n^3 + 277x^2 + 720x + 7 \in \Theta(?)$
• $200 \in \Theta(?)$
• $(n \ (n+1))/2 \in \Theta(?)$
Consider

\[ f_1(n) = 5n^3 + 24n + 6 \]

We claim that

\[ f_1(n) = O(n^3) \]

Let \( c = 6 \) and \( n_0 = 10 \). Then

\[ 5n^3 + 24n + 6 \leq 6n^3 \]

for every \( n \geq 10 \)
If

\[ f_1(n) = 5n^3 + 24n + 6 \]

we have seen that

\[ f_1(n) = O(n^3) \]

but \( f_1(n) \) is not in \( O(n^2) \), because no positive value for \( c \) or \( n_0 \) works.
"Big Omega" - Lower Bounding Running Time

**Definition:** $f(n) \in \Omega(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 1$ such that

$$f(n) \geq cg(n) \quad \text{for all} \quad n \geq n_0.$$  

**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 trapped above (or = to) some constant multiple of $g(n)$.

- Some constant multiple of $g(n)$ is a **lower bound** for $f(n)$ (for large enough $n$).
"Big Omega" - Lower Bounding Running Time

Alternate Definition:

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

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 trapped above (or = to) some constant multiple of \( g(n) \)

• some constant multiple of \( g(n) \) is a lower bound for \( f(n) \) (for large enough \( n \))

(note: \( n_0 \) must be at least 1)
Insertion Sort - Time Complexity

Time complexities for insertion sort are:
- best-case: \( b(n) = 5n - 4 \)
- worst-case: \( w(n) = \frac{3n^2}{2} + \frac{7n}{2} - 4 \)

Questions:
1. is \( b(n) = \Omega(n) \) ? Yes... \( (5n - 4 \geq 2n) \) for all \( n_0 \geq 2 \)
2. is \( w(n) = \Omega(n) \) ? Yes... \( \left( \frac{3n^2}{2} + \frac{7n}{2} - 4 \geq 3n \right) \) for all \( n_0 \geq 1 \)
3. is \( w(n) = \Omega(n^2) \) ? Yes... \( \left( \frac{3n^2}{2} + \frac{7n}{2} - 4 \geq n^2 \right) \) for all \( n_0 \geq 1 \)
4. is \( w(n) = \Omega(n^3) \) ? No... \( \left( \frac{3n^2}{2} + \frac{7n}{2} - 4 < n^3 \right) \) for all \( n_0 \geq 3 \)
"Theta" - Tightly Bounding Running Time

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

$$c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \text{for all } n \geq n_0.$$

*Useful way to show "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: \( w(n) = \frac{3n^2}{2} + \frac{7n}{2} - 4 \)

Questions:
1. is \( b(n) = \theta(n) \)? Yes because \( b(n) = O(n) \) and \( \Omega(n) \)
2. is \( w(n) = \theta(n) \)? No because \( w(n) \neq O(n) \)
3. is \( w(n) = \theta(n^2) \)? Yes because \( w(n) = O(n^2) \) and \( \Omega(n^2) \)
4. is \( w(n) = \theta(n^3) \)? 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
"Little Oh" – Strict upper bound

**Definition:** \( f(n) \in o(g(n)) \) if for every \( c > 0 \), there exists some \( n_0 \geq 1 \) 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" – Strict Lower Bound

**Definition:** \( f(n) \in \omega(g(n)) \) if for every \( c > 0 \), there exists some \( n_0 \geq 1 \) 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 \( n \).
Using Limits to Determine Complexity

Showing "Little Oh" and "Little Omega" relationships:

\[ f(n) \in o(g(n)) \quad \text{iff} \quad \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \]

\[ f(n) \in \omega(g(n)) \quad \text{iff} \quad \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty \]

Showing Theta relationships:

\[ f(n) \in \Theta(g(n)) \quad \text{iff} \quad \lim_{n \to \infty} \frac{f(n)}{g(n)} = c > 0 \]
Analysis of PrefixAverages-v1

1. Create an array $A$ such that $\text{length}[A] = \text{length}[X] = n$
2. $s = 0$
3. for $(j = 1$ to $\text{length}[X])$
4. $s = s + X[j]$
5. $A[j] = s / j$
6. return $A$

$$\sum_{i=1}^{n} 1 = n - 1 + 1 = n$$

1. $T(n) = \Theta(n)$

2. Are there best- and worst-case inputs? No
Analysis of PrefixAverages-v2

1. Create an array A such that length[A] = n
2. \textbf{for} (j = 1 to n)
3. a = 0
4. \textbf{for} (i = 1 to j)
5. a = a + X[i]
7. return A

\[ T(n) = \Theta(n^2) \]

2. Are there best and worst case inputs? No
## Basic asymptotic efficiency classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Name</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Constant</td>
<td>Algorithm ignores input (i.e., can’t even scan or print input)</td>
</tr>
<tr>
<td>(lgn)</td>
<td>Logarithmic</td>
<td>Cuts problem size by constant fraction on each iteration</td>
</tr>
<tr>
<td>(n)</td>
<td>Linear</td>
<td>Algorithm scans its input (at least); one or more non-nested loops</td>
</tr>
<tr>
<td>(nlgn)</td>
<td>Linearithmic</td>
<td>Some divide and conquer algorithms; best sorting time.</td>
</tr>
<tr>
<td>(n^2)</td>
<td>Quadratic</td>
<td>Loop inside loop = “nested loop”</td>
</tr>
<tr>
<td>(n^3)</td>
<td>Cubic</td>
<td>Loop inside nested loop</td>
</tr>
<tr>
<td>(2^n)</td>
<td>Exponential</td>
<td>Algorithm generates all subsets of (n)-element set set</td>
</tr>
<tr>
<td>(n!)</td>
<td>Factorial</td>
<td>Algorithm generates all permutations of (n)-element set set</td>
</tr>
</tbody>
</table>
Handy Asymptotic Facts

a) If $T(n)$ is a polynomial function of degree $k$, then $T(n) = O(n^k)$.

b) $\log^k n = (\log n)^k = O(n)$

c) $n^b = o(a^n)$ for any constants $a > 1$, $b > 0$.

d) $n! = o(n^n)$

e) $n! = \omega(2^n)$

f) $\lg(n!) = \Theta(n\lg n)$

g) $n^a = O(n^{a+\epsilon})$, $a^n = O(a + \epsilon)^n$
Iterated logarithm function \((\lg^* n)\):

- the number of times the log function must be iteratively applied before the result is less than or equal to 1
- "log star of \(n\)"
- *Very slow growing*, e.g. \(\lg^* (2^{65536}) = 5\)
  (and \(2^{65536}\) is much larger than the number of atoms in the observable universe!!)

eg: \(\lg^* 2 = 1\)

\(\lg^* 4 = 2\)

\(\lg^* 16 = 3\)

\(\lg^* 65536 = 4\)