An analogy between the asymptotic comparison of two functions $f$ and $g$ and the comparison of two real numbers $a$ and $b$:

\[
\begin{align*}
    f(n) &= O(g(n)) & \approx a & \leq & b \\
    f(n) &= \Omega(g(n)) & \approx a & \geq & b \\
    f(n) &= \Theta(g(n)) & \approx a & = & b
\end{align*}
\]
Question:

What's the order of the following widely used functions:
lgn, n, n^2, 1, n^3, 2^n, n2^n, (n + 1)!, 2^{2^n}, (lgn)!, e^n, n!

Answer:

1 \leq lgn \leq n \leq n^2 \leq n^3 \leq (lgn)! \leq 2^n

\leq n2^n \leq e^n \leq n! \leq (n + 1)! \leq 2^{2^n}, where a \leq b means a = O(b)
Suppose one basic operation needs CPU time 0.000001 second.

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>0.00001 s</td>
<td>0.00002 s</td>
<td>0.00003 s</td>
<td>0.00004 s</td>
<td>0.00005 s</td>
<td>0.00006 s</td>
</tr>
<tr>
<td>n^2</td>
<td>0.0001 s</td>
<td>0.0004 s</td>
<td>0.0009 s</td>
<td>0.016 s</td>
<td>0.025 s</td>
<td>0.036 s</td>
</tr>
<tr>
<td>n^3</td>
<td>0.001 s</td>
<td>0.008 s</td>
<td>0.027 s</td>
<td>0.064 s</td>
<td>0.125 s</td>
<td>0.216 s</td>
</tr>
<tr>
<td>n^5</td>
<td>0.1 s</td>
<td>3.2 s</td>
<td>24.3 s</td>
<td>1.7 min</td>
<td>5.2 min</td>
<td>13.0 min</td>
</tr>
<tr>
<td>2^n</td>
<td>0.001 s</td>
<td>1.0 s</td>
<td>17.9 min</td>
<td>12.7 days</td>
<td>35.7 years</td>
<td>366 cent</td>
</tr>
<tr>
<td>3^n</td>
<td>0.59 s</td>
<td>58 min</td>
<td>6.5 years</td>
<td>3855 cent</td>
<td>2 × 10^8 cent</td>
<td>1.3 × 10^{13} cent</td>
</tr>
</tbody>
</table>
A Recurrence Example: Merge Sort

Merge sort is a good example to show how divide and conquer works. The idea is: Given an array \( A[1..n] \), divide it into two sub-array \( A[1..n/2] \) and \( A[n/2+1..n] \). Each sub-array is individually sorted, and the resulting sub-arrays are merged to produce a single sorted array of \( n \) elements. The algorithm:

\[
\text{MERGE-SORT}(A, p, r)
\]

1. if \((p == r)\) return;
2. \( q = (p + r)/2; \)
3. \( \text{Merge-Sort}(A, p, q); \)
4. \( \text{Merge-Sort}(A, q+1, r); \)
5. \( \text{Merge}(A, p, q, r); \)

To sort the whole array, \( \text{Merge-Sort}(A, 1, n) \) is called.
The operation of Merge Sort

Input: 5, 2, 4, 7, 1, 3, 2, 6

Figure 2.4  The operation of merge sort on the array $A = (5, 2, 4, 7, 1, 3, 2, 6)$. The lengths of the sorted sequences being merged increase as the algorithm progresses from bottom to top.
Complexity of Merge Sort

**Divide:** The divide step only compute the middle, takes constant time. \( D(n) = \Theta(1) \).

**Conquer:** Recursively sort 2 subarrays. \( C(n) = 2T(n/2) \).

**Combine:** Merge two \( n/2 \)-element subarrays, takes linear time \( \Theta(n) \).

**Overall:**

\[
T(n) = \begin{cases} 
\Theta(1) & \text{if } n = 1 \text{ (or smallsize) } \\
2T(n/2) + \Theta(1) + \Theta(n) & \text{if } n > 1 \text{ (or smallsize)} 
\end{cases}
\]

\[
= \begin{cases} 
C_1 & \text{if } n = 1 \text{ (or smallsize) } \\
2T(n/2) + C_2n & \text{if } n > 1 \text{ (or smallsize)} 
\end{cases}
\]
How to solve this recurrence?

Solution 1: Substitution method

1. Guess the form of the solution.

2. Use mathematical induction to find the constants and show the solution works.
Example: Merge Sort

\[ T(n) = \begin{cases} 
    C_1 & \text{if } n = 1 \\
    2T(n/2) + C_2n & \text{if } n > 1 
\end{cases} \]

**Step 1:** give a guess: \( T(n) = O(n \lg n) \)

**Step 2:** to show \( \exists \) const \( c \) and \( n_0 \), such that \( T(n) \leq c \cdot n \lg n \)
for all \( n \geq n_0 \)

**Base case:**
\[ T(1) = C_1 \leq c \cdot 1 \lg 1 = 0 \ldots \text{Impossible} \]

Take \( T(2) \) as the base case.
\[ T(2) = 2C_1 + 2C_2 \leq c \cdot 2 \lg 2 = 2c \]
as long as \( c \geq (C_1 + C_2) \).
Induction Step:

Suppose there exist a constant $c$ such that

$$T(n) \leq c \cdot n \log n$$

for all $n = 2, 3, \ldots, k-1$

We want to show $T(n) \leq c \cdot n \log n$

holds for $n = k$.

$$T(k) = 2T(k/2) + C_2k$$

Note: $k/2$ is in $\{2, 3, \ldots, k-1\}$

$$\leq 2(c \cdot (k/2) \log (k/2)) + C_2k$$

$$= ck \log k - ck \log 2 + C_2k$$

$$= ck \log k - ck + C_2k$$

$$\leq ck \log k$$

as long as $c \geq C_2$.

So we pick $n_0 = 2$, $c = C_1 + C_2$,

$$T(n) \leq c \cdot n \log n$$

for all $n \geq n_0 \implies T(n) = O(n \log n)$. 
Where to get the good guess?

**Solution 2: Iteration/Recursion tree method**: used to generate a good guess; can also be used as a direct proof.

**Example:**

\[ T(n) = \begin{cases} 
1 & \text{if } n = 1 \\
2T(n/2) + n & \text{if } n > 1 
\end{cases} \]

\[
T(n) = 2T\left(\frac{n}{2}\right) + n \\
= 2(2T\left(\frac{n}{4}\right) + \frac{n}{2}) + n \\
= 2^2T\left(\frac{n}{2^2}\right) + n + n \\
= 2^2(2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}) + 2n \\
= 2^3T\left(\frac{n}{2^3}\right) + 3n \\
\ldots \\
= 2^iT\left(\frac{n}{2^i}\right) + i\cdot n
\]
Question: When will the iteration procedure reach the boundary condition (hit the ground)?

Answer: \((n/2^i) = 1 \iff i = \lg n\)

Then \(T(n) = 2^{\lg n} T(1) + \lg n \times n\)
\[= n + n\lg n\]
\[= \Theta(n\lg n).\]