Lecture 3
Why study algorithms and performance?

• Algorithms help us to understand *scalability*.
• Performance often draws the line between what is feasible and what is impossible.
• Algorithmic mathematics provides a *language* for talking about program behavior.
• Performance is the *currency* of computing.
• The lessons of program performance generalize to other computing resources.
• Speed is fun!
The problem of sorting

**Input:** sequence $\langle a_1, a_2, \ldots, a_n \rangle$ of numbers.

**Output:** permutation $\langle a'_1, a'_2, \ldots, a'_n \rangle$ such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

**Example:**

**Input:** 8 2 4 9 3 6

**Output:** 2 3 4 6 8 9
Insertion sort

```
INSERTION-SORT (A, n)  \(\triangleright\) A[1..n]

for j \leftarrow 2 \text{ to } n
    do  key \leftarrow A[j]
        i \leftarrow j - 1
        while i > 0 and A[i] > key
            do  A[i+1] \leftarrow A[i]
                i \leftarrow i - 1
        A[i+1] = key
```

“pseudocode”
Insertion sort

**INSERTION-SORT** \((A, n) \Rightarrow A[1 \ldots n]\)

for \(j \leftarrow 2\) to \(n\)

\[
\begin{align*}
\text{do } & \text{key} \leftarrow A[j] \\
& i \leftarrow j - 1 \\
\text{while } & i > 0 \text{ and } A[i] > \text{key} \\
\text{do } & A[i+1] \leftarrow A[i] \\
& i \leftarrow i - 1 \\
& A[i+1] = \text{key}
\end{align*}
\]
Example of insertion sort

8 2 4 9 3 6
Example of insertion sort

8 2 4 9 3 6
Example of insertion sort

8 2 4 9 3 6
2 8 4 9 3 6
Example of insertion sort

8 2 4 9 3 6
2 8 4 9 3 6
Example of insertion sort

8  2  4  9  3  6
2  8  4  9  3  6
2  4  8  9  3  6
Example of insertion sort

8  2  4  9  3  6
2 8  4  9  3  6
2 4  8  9  3  6
Example of insertion sort

<table>
<thead>
<tr>
<th>8</th>
<th>2</th>
<th>4</th>
<th>9</th>
<th>3</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>8</td>
<td>4</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>9</td>
<td>3</td>
</tr>
</tbody>
</table>
Example of insertion sort

8 2 4 9 3 6
2 8 4 9 3 6
2 4 8 9 3 6
2 4 8 9 3 6
Example of insertion sort

8 2 4 9 3 6
2 8 4 9 3 6
2 4 8 9 3 6
2 4 8 9 3 6
2 3 4 8 9 6
Example of insertion sort

8  2  4  9  3  6
2  8  4  9  3  6
2  4  8  9  3  6
2  4  8  9  3  6
2  3  4  8  9  6
Example of insertion sort

8 2 4 9 3 6
2 8 4 9 3 6
2 4 8 9 3 6
2 4 8 9 3 6
2 3 4 8 9 6
2 3 4 6 8 9 done
Running time

• The running time depends on the input: an already sorted sequence is easier to sort.
• Parameterize the running time by the size of the input, since short sequences are easier to sort than long ones.
• Generally, we seek upper bounds on the running time, because everybody likes a guarantee.
Kinds of analyses

**Worst-case:** (usually)
- $T(n) =$ maximum time of algorithm on any input of size $n$.

**Average-case:** (sometimes)
- $T(n) =$ expected time of algorithm over all inputs of size $n$.
- Need assumption of statistical distribution of inputs.

**Best-case:** (bogus)
- Cheat with a slow algorithm that works fast on *some* input.
Machine-independent time

What is insertion sort’s worst-case time?
• It depends on the speed of our computer:
  • relative speed (on the same machine),
  • absolute speed (on different machines).

**Big Idea:**
• Ignore machine-dependent constants.
• Look at growth of $T(n)$ as $n \to \infty$.

“*Asymptotic Analysis*”
Θ-notation

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

**Engineering:**
- Drop low-order terms; ignore leading constants.
- Example: \( 3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3) \)
Asymptotic performance

When \( n \) gets large enough, a \( \Theta(n^2) \) algorithm *always* beats a \( \Theta(n^3) \) algorithm.

- We shouldn’t ignore asymptotically slower algorithms, however.
- Real-world design situations often call for a careful balancing of engineering objectives.
- Asymptotic analysis is a useful tool to help to structure our thinking.
Insertion sort analysis

**Worst case:** Input reverse sorted.

\[
T(n) = \sum_{j=2}^{n} \Theta(j) = \Theta(n^2) \quad \text{[arithmetic series]}
\]

**Average case:** All permutations equally likely.

\[
T(n) = \sum_{j=2}^{n} \Theta(j / 2) = \Theta(n^2)
\]

*Is insertion sort a fast sorting algorithm?*

- Moderately so, for small \( n \).
- Not at all, for large \( n \).
Merge sort

**MERGE-SORT** $A[1 \ldots n]$

1. If $n = 1$, done.
2. Recursively sort $A[1 \ldots \lceil n/2 \rceil]$ and $A[\lfloor n/2 \rfloor + 1 \ldots n]$.
3. "Merge" the 2 sorted lists.

*Key subroutine: MERGE*
Merging two sorted arrays

20 12
13 11
 7  9
 2  1
Merging two sorted arrays

20 12
13 11
7 9
2 1
1
Merging two sorted arrays
Merging two sorted arrays

20 12 || 20 12
13 11 || 13 11
 7  9 ||   7  9
 2  1 ||   2
 1 ||   2
Merging two sorted arrays
Merging two sorted arrays
Merging two sorted arrays
Merging two sorted arrays

\[
\begin{array}{cccc}
20 & 12 & 20 & 12 \\
13 & 11 & 13 & 11 \\
7 & 9 & 7 & 9 \\
1 & 2 & 2 & 7 \\
1 & 2 & 7 & 9 \\
1 & 2 & 7 & 9 \\
\end{array}
\]
Merging two sorted arrays

20 12
13 11
7 9
2 1

20 12
13 11
7 9
2 2

20 12
13 11
7 9
7 7

20 12
13 11
7 9
9 9

20 12
13 11
7 9
9 9
Merging two sorted arrays
Merging two sorted arrays

20 12
13 11
7 9
2 1
1

20 12
13 11
7 9
2 2
2

20 12
13 11
7 9
7 9
7

20 12
13 11
9 9
9

20 12
13 11
11 11
11

20 12
13 13
12 12
12
Merging two sorted arrays

20 12
13 11
7 9
2 1

1 2

7 9

7

20 12
13 11
13 11
13 11
13
13
13

9

9

9

11

11

11

11

12
Merging two sorted arrays

Time = $\Theta(n)$ to merge a total of $n$ elements (linear time).
Analyzing merge sort

\[ T(n) \quad \text{MERGE-SORT } A[1 \ldots n] \]

- \( \Theta(1) \) \quad 1. If \( n = 1 \), done.
- \( 2T(n/2) \) \quad 2. Recursively sort \( A[1 \ldots \lfloor n/2 \rfloor] \) and \( A[\lceil n/2 \rceil +1 \ldots n] \).
- \( \Theta(n) \) \quad 3. "Merge" the 2 sorted lists

**Abuse**: Should be \( T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) \), but it turns out not to matter asymptotically.

**Sloppiness**: Should be \( T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) \), but it turns out not to matter asymptotically.
Recurrence for merge sort

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

• We shall usually omit stating the base case when \( T(n) = \Theta(1) \) for sufficiently small \( n \), but only when it has no effect on the asymptotic solution to the recurrence.
• CLRS and Lecture 2 provide several ways to find a good upper bound on \( T(n) \).
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

$h = \log n$

$\Theta(1)$
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

$h = \log n$

$\Theta(1)$
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant. 

$h = \lg n$

$\Theta(1)$

$\#\text{leaves} = n$

$\Theta(n)$
Recursion tree

Solve \( T(n) = 2T(n/2) + cn \), where \( c > 0 \) is constant.

\[
\begin{align*}
    h &= \lg n \\
    \text{#leaves} &= n \\
    \Theta(1) &
\end{align*}
\]

Total = \( \Theta(n \lg n) \)
Asymptotic notation

\textit{O-notation (upper bounds):}

We write $f(n) = O(g(n))$ if there exist constants $c > 0$, $n_0 > 0$ such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$. 

Asymptotic notation

$O$-notation (upper bounds):

We write $f(n) = O(g(n))$ if there exist constants $c > 0$, $n_0 > 0$ such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$.

**Example:** $2n^2 = O(n^3)$  \hspace{1cm} (c = 1, n_0 = 2)
Asymptotic notation

**O-notation (upper bounds):**

We write \( f(n) = O(g(n)) \) if there exist constants \( c > 0, \ n_0 > 0 \) such that \( 0 \leq f(n) \leq cg(n) \) for all \( n \geq n_0 \).

**Example:** \( 2n^2 = O(n^3) \) \( (c = 1, \ n_0 = 2) \)

functions, not values
Asymptotic notation

\( O \)-notation (upper bounds):

We write \( f(n) = O(g(n)) \) if there exist constants \( c > 0, \ n_0 > 0 \) such that \( 0 \leq f(n) \leq cg(n) \) for all \( n \geq n_0 \).

**Example:** \( 2n^2 = O(n^3) \) \( (c = 1, \ n_0 = 2) \)

functions, not values

funny, “one-way” equality
Set definition of O-notation

\[ O(g(n)) = \{ f(n) : \text{there exist constants } c > 0, n_0 > 0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0 \} \]
Set definition of O-notation

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

\textbf{Example:} \hspace{1cm} 2n^2 \in O(n^3)
**Set definition of O-notation**

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

**Example:** \( 2n^2 \in O(n^3) \)

*(Logicians: \( \lambda n.2n^2 \in O(\lambda n.n^3) \), but it’s convenient to be sloppy, as long as we understand what’s really going on.)*
Macro substitution

Convention: A set in a formula represents an anonymous function in the set.
Macro substitution

Convention: A set in a formula represents an anonymous function in the set.

Example: \( f(n) = n^3 + O(n^2) \) means \( f(n) = n^3 + h(n) \) for some \( h(n) \in O(n^2) \).
Macro substitution

**Convention:** A set in a formula represents an anonymous function in the set.

**Example:** \( n^2 + O(n) = O(n^2) \)

means

for any \( f(n) \in O(n) \):

\[ n^2 + f(n) = h(n) \]

for some \( h(n) \in O(n^2) \).
Ω-notation (lower bounds)

$O$-notation is an *upper-bound* notation. It makes no sense to say $f(n)$ is at least $O(n^2)$. 
\(\Omega\)-notation (lower bounds)

\(O\)-notation is an upper-bound notation. It makes no sense to say \(f(n)\) is at least \(O(n^2)\).

\[\Omega(g(n)) = \{ f(n) : \text{there exist constants } c > 0, n_0 > 0 \text{ such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0 \}\]
**Ω-notation** (lower bounds)

*O*-notation is an *upper-bound* notation. It makes no sense to say $f(n)$ is at least $O(n^2)$.

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

**Example:** $\sqrt{n} = \Omega(\lg n)$ (c = 1, $n_0 = 16$)
Θ-notation (tight bounds)

\[ \Theta(g(n)) = O(g(n)) \cap \Omega(g(n)) \]
**Θ-notation (tight bounds)**

$$Θ(g(n)) = O(g(n)) \cap Ω(g(n))$$

**Example:** \( \frac{1}{2} n^2 - 2n = Θ(n^2) \)
Solving recurrences

• The analysis of merge sort required us to solve a recurrence.

• Recurrences are like solving integrals, differential equations, etc.
  ◦ Learn a few tricks.

• Applications of recurrences to divide-and-conquer algorithms.
Substitution method

*The most general method:*

1. **Guess** the form of the solution.
2. **Verify** by induction.
3. **Solve** for constants.
Substitution method

The most general method:

1. **Guess** the form of the solution.
2. **Verify** by induction.
3. **Solve** for constants.

**Example:**  \( T(n) = 4T(n/2) + n \)

- [Assume that \( T(1) = \Theta(1) \).]
- Guess \( O(n^3) \). (Prove \( O \) and \( \Omega \) separately.)
- Assume that \( T(k) \leq ck^3 \) for \( k < n \).
- Prove \( T(n) \leq cn^3 \) by induction.
Example of substitution

\[ T(n) = 4T(n/2) + n \]
\[ \leq 4cn(n/2)^3 + n \]
\[ = (c/2)n^3 + n \]
\[ = cn^3 - ((c/2)n^3 - n) \]  \(\leftarrow\) desired – residual
\[ \leq cn^3 \]  \(\leftarrow\) desired

whenever \((c/2)n^3 - n \geq 0\), for example,
if \(c \geq 2\) and \(n \geq 1\).
Example (continued)

• We must also handle the initial conditions, that is, ground the induction with base cases.

• **Base:** $T(n) = \Theta(1)$ for all $n < n_0$, where $n_0$ is a suitable constant.

• For $1 \leq n < n_0$, we have “$\Theta(1)$” $\leq cn^3$, if we pick $c$ big enough.
Example (continued)

• We must also handle the initial conditions, that is, ground the induction with base cases.

• **Base:** $T(n) = \Theta(1)$ for all $n < n_0$, where $n_0$ is a suitable constant.

• For $1 \leq n < n_0$, we have “$\Theta(1)$” $\leq cn^3$, if we pick $c$ big enough.

---

*This bound is not tight!*
A tighter upper bound?

We shall prove that $T(n) = O(n^2)$.
A tighter upper bound?

We shall prove that $T(n) = O(n^2)$.

Assume that $T(k) \leq ck^2$ for $k < n$:

\[
T(n) = 4T(n/2) + n \\
\leq 4c(n/2)^2 + n \\
= cn^2 + n \\
= O(n^2)
\]
A tighter upper bound?

We shall prove that $T(n) = O(n^2)$.

Assume that $T(k) \leq ck^2$ for $k < n$:

$T(n) = 4T(n/2) + n$

$\leq 4c(n/2)^2 + n$

$= cn^2 + n$

$= O(n^2)$ \textbf{Wrong!} We must prove the I.H.
A tighter upper bound?

We shall prove that $T(n) = O(n^2)$.

Assume that $T(k) \leq c k^2$ for $k < n$:

\[
T(n) = 4T(n/2) + n \\
\leq 4c(n/2)^2 + n \\
= c n^2 + n \\
= O(n^2) \quad \text{Wrong! We must prove the I.H.} \\
= c n^2 - (-n) \quad \text{[desired – residual]} \\
\leq c n^2 \quad \text{for no choice of } c > 0. \quad \text{Lose!}
\]
A tighter upper bound!

**Idea:** Strengthen the inductive hypothesis.

- *Subtract* a low-order term.

*Inductive hypothesis:* $T(k) \leq c_1 k^2 - c_2 k$ for $k < n$. 

A tighter upper bound!

**IDEA:** Strengthen the inductive hypothesis.
- *Subtract* a low-order term.

*Inductive hypothesis:* $T(k) \leq c_1 k^2 - c_2 k$ for $k < n$.

\[
T(n) = 4T(n/2) + n \\
= 4\left(c_1 \left(\frac{n}{2}\right)^2 - c_2 \left(\frac{n}{2}\right)\right) + n \\
= c_1 n^2 - 2c_2 n + n \\
= c_1 n^2 - c_2 n - (c_2 n - n) \\
\leq c_1 n^2 - c_2 n \quad \text{if } c_2 \geq 1.
\]
A tighter upper bound!

**Idea:** Strengthen the inductive hypothesis.  
• *Subtract* a low-order term.

**Inductive hypothesis:** $T(k) \leq c_1 k^2 - c_2 k$ for $k < n$.

\[
T(n) = 4T(n/2) + n \\
= 4(c_1(n/2)^2 - c_2(n/2)) + n \\
= c_1 n^2 - 2c_2 n + n \\
= c_1 n^2 - c_2 n - (c_2 n - n) \\
\leq c_1 n^2 - c_2 n \text{ if } c_2 \geq 1.
\]

Pick $c_1$ big enough to handle the initial conditions.
Recursion-tree method

- A recursion tree models the costs (time) of a recursive execution of an algorithm.
- The recursion-tree method can be unreliable, just like any method that uses ellipses (…).
- The recursion-tree method promotes intuition, however.
- The recursion tree method is good for generating guesses for the substitution method.
Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$: 
Example of recursion tree

Solve \( T(n) = T(n/4) + T(n/2) + n^2: \)

\( T(n) \)
Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:
Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:
Example of recursion tree

Solve \( T(n) = T(n/4) + T(n/2) + n^2: \)

\[
\begin{array}{c}
\Theta(1) \\
\vdots \\
(n/8)^2 \\
(n/8)^2 \\
(n/4)^2 \\
(n/2)^2 \\
n^2
\end{array}
\]
Example of recursion tree

Solve \( T(n) = T(n/4) + T(n/2) + n^2: \)

\[
\begin{align*}
&n^2 \\
&\quad \quad (n/4)^2 \\
&\quad \quad \quad \quad (n/16)^2 \\
&\quad \quad \quad \quad \quad \Theta(1) \\
&\quad \quad \quad \quad \quad \ddots \\
&\quad \quad (n/8)^2 \\
&\quad \quad (n/8)^2 \\
&\quad \quad (n/4)^2 \\
&\quad \quad \quad \quad (n/2)^2 \\
&\quad \quad \quad \quad n^2
\end{align*}
\]
Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:
Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:

$\Theta(1)$
Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:

\[
\begin{align*}
\Theta(1) & \quad \Theta(1) \\
(n/16)^2 & \quad (n/8)^2 \\
(n/4)^2 & \quad (n/2)^2 \\
& \quad \vdots \\
& \quad \text{Total} = n^2 \left( 1 + \frac{5}{16} + \left( \frac{5}{16} \right)^2 + \left( \frac{5}{16} \right)^3 + \cdots \right) \\
& = \Theta(n^2) \quad \text{geometric series}
\end{align*}
\]
The master method

The master method applies to recurrences of the form

\[ T(n) = a \cdot T(n/b) + f(n), \]

where \( a \geq 1 \), \( b > 1 \), and \( f \) is asymptotically positive.
Three common cases

Compare $f(n)$ with $n^{\log ba}$:

1. $f(n) = O(n^{\log ba} - \varepsilon)$ for some constant $\varepsilon > 0$.
   - $f(n)$ grows polynomially slower than $n^{\log ba}$ (by an $n^{\varepsilon}$ factor).

Solution: $T(n) = \Theta(n^{\log ba})$.
Three common cases

Compare $f(n)$ with $n^{\log_b a}$:

1. $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$.
   - $f(n)$ grows polynomially slower than $n^{\log_b a}$ (by an $n^\varepsilon$ factor).
   **Solution:** $T(n) = \Theta(n^{\log_b a})$.

2. $f(n) = \Theta(n^{\log_b a \lg^k n})$ for some constant $k \geq 0$.
   - $f(n)$ and $n^{\log_b a}$ grow at similar rates.
   **Solution:** $T(n) = \Theta(n^{\log_b a \lg^{k+1} n})$. 
Three common cases (cont.)

Compare $f(n)$ with $n^{\log_b a}$:

3. $f(n) = \Omega(n^{\log_b a} + \varepsilon)$ for some constant $\varepsilon > 0$.
   - $f(n)$ grows polynomially faster than $n^{\log_b a}$ (by an $n^\varepsilon$ factor),
   
   and $f(n)$ satisfies the *regularity condition* that $a f(n/b) \leq c f(n)$ for some constant $c < 1$.

**Solution:** $T(n) = \Theta(f(n))$. 
Examples

Ex. \( T(n) = 4T(n/2) + n \)

\( a = 4, \, b = 2 \Rightarrow n^{\log_b^a} = n^2 ; \, f(n) = n. \)

Case 1: \( f(n) = O(n^{2 - \varepsilon}) \) for \( \varepsilon = 1. \)

\[ \therefore \, T(n) = \Theta(n^2). \]
Examples

**Ex.** \( T(n) = 4T(n/2) + n \)
\( a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n. \)

**Case 1:** \( f(n) = O(n^{2-\varepsilon}) \) for \( \varepsilon = 1. \)
\[ \therefore T(n) = \Theta(n^2). \]

**Ex.** \( T(n) = 4T(n/2) + n^2 \)
\( a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n^2. \)

**Case 2:** \( f(n) = \Theta(n^2 \lg^0 n), \) that is, \( k = 0. \)
\[ \therefore T(n) = \Theta(n^2 \lg n). \]
Examples

**Ex.** $T(n) = 4T(n/2) + n^3$

$a = 4$, $b = 2 \Rightarrow n^\log_b a = n^2$; $f(n) = n^3$.

**Case 3:** $f(n) = \Omega(n^{2+\varepsilon})$ for $\varepsilon = 1$

and $4(n/2)^3 \leq cn^3$ (reg. cond.) for $c = 1/2$.

∴ $T(n) = \Theta(n^3)$. 
Examples

Ex. \( T(n) = 4T(n/2) + n^3 \)
\( a = 4, \, b = 2 \Rightarrow n^{\log_b a} = n^2; \, f(n) = n^3. \)

Case 3: \( f(n) = \Omega(n^{2+\varepsilon}) \) for \( \varepsilon = 1 \)
and \( 4(n/2)^3 \leq cn^3 \) (reg. cond.) for \( c = 1/2. \)
\( \therefore T(n) = \Theta(n^3). \)

Ex. \( T(n) = 4T(n/2) + n^2/\lg n \)
\( a = 4, \, b = 2 \Rightarrow n^{\log_b a} = n^2; \, f(n) = n^2/\lg n. \)
Master method does not apply. In particular, for every constant \( \varepsilon > 0, \) we have \( n^\varepsilon = \omega(\lg n). \)
Idea of master theorem

Recursion tree:

\[ T(1) \]

\[ f(n) \]

\[ a \]

\[ f(n/b) \]

\[ a \]

\[ f(n/b^2) \]

\[ \cdots \]

\[ f(n/b^2) \]

\[ \cdots \]

\[ f(n/b) \]

\[ \cdots \]

\[ f(n/b) \]
Idea of master theorem

Recursion tree:

```
/\  \  \  \  \  \  \n|  |  |  |  |  |
f(n) a f(n)
\  \  /  /
f(n/b) f(n/b) af(n/b)
\  /  /  /
f(n/b^2) f(n/b^2) a^2 f(n/b^2)
/ /  /  /
⋅⋅⋅⋅⋅⋅⋅⋅⋅
T(1)
```
Idea of master theorem

Recursion tree:

\[
\begin{align*}
T(n) &= f(n) \\
&= af(n/b) + a^2 f(n/b^2) + \cdots + T(1)
\end{align*}
\]

where

\[
h = \log_b n
\]

and

\[
a, b, f(n)
\]
Idea of master theorem

Recursion tree:

\[ f(n) \]

\[ f(n/b) \]

\[ f(n/b^2) \]

\[ \ldots \]

\[ a \]

\[ a \]

\[ a^2 \]

\[ \vdots \]

\[ h = \log_b n \]

\[ T(1) \]

\[ a^h = a^{\log_b n} \]

\[ n^{\log_b a} T(1) \]

\[ \text{leaves} = a^h \]
Idea of master theorem

Recursion tree:

\[ h = \log_b n \]

CASE 1: The weight increases geometrically from the root to the leaves. The leaves hold a constant fraction of the total weight.

\[ \Theta(n^{\log_b a}) \]
**Idea of master theorem**

**Recursion tree:**

- **CASE 2**: \( k = 0 \) The weight is approximately the same on each of the \( \log_b n \) levels.

\[
T(1) = n^\log_b a \cdot T(1) \\
\Theta(n^\log_b a \cdot \log n)
\]
Idea of master theorem

Recursion tree:

CASE 3: The weight decreases geometrically from the root to the leaves. The root holds a constant fraction of the total weight.

\[ T(1) \]

\[ \Theta(f(n)) \]
• **Proof of the master theorem** \( n \) is exact power of \( b 

\[
T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)
\]
• **Proof of the master theorem** \( n \) is exact power of \( b \)

**Lemma 1.** Determine asymptotic bounds on this summation

\[
g(n) = \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)
\]

**Case 1.** If \( f(n) = O(n^{\log_b a - \varepsilon}) \), for some constant \( \varepsilon > 0 \), then \( g(n) = O(n^{\log_b a}) \)

**Case 2.** If \( f(n) = \Theta(n^{\log_b a}) \), then \( g(n) = \Theta(n^{\log_b a} \log n) \)

**Case 3.** If \( af(n/b) \leq cf(n) \) for some constant \( c < 1 \) and for all \( n \geq b \), then \( g(n) = \Theta(f(n)) \)
• **Proof of the master theorem** \( n \) is exact power of \( b \\

**Lemma 1.** Determine asymptotic bounds on this summation \( g(n) \)

\[
g(n) = \sum_{j=0}^{\log_b n - 1} a^j f(n / b^j)
\]

**Case 1.** If \( f(n) = O(n^{\log_b a - \epsilon}) \) , for some constant \( \epsilon > 0 \), then \( g(n) = O(n^{\log_b a}) \)

*Assumed fact:* \( f(n) = O(n^{\log_b a - \epsilon}) \rightarrow f(n / b^j) = O((n / b^j)^{\log_b a - \epsilon}) \)

*Substitution:* \( g(n) = O(\sum_{j=0}^{\log_b n - 1} a^j (n / b^j)^{\log_b a - \epsilon}) \)
• **Proof of the master theorem** $n$ is exact power of $b$

**Lemma 1.** Determine asymptotic bounds on this summation

Proof for Case 1

Substitute into $g(n) : g(n) = O\left( \sum_{j=0}^{\log_b^n - 1} a^j (n / b^j)^{\log_b a - \varepsilon} \right)$

$$\sum_{j=0}^{\log_b^n - 1} a^j (n / b^j)^{\log_b a - \varepsilon} = n^{\log_b a - \varepsilon} \sum_{j=0}^{\log_b^n - 1} \left( \frac{ab^\varepsilon}{b^{\log_b a}} \right)^j$$

$$= n^{\log_b a - \varepsilon} \sum_{j=0}^{\log_b^n - 1} (b^\varepsilon)^j$$

$$= n^{\log_b a - \varepsilon} \frac{b^{\varepsilon \log_b^n} - 1}{b^{\varepsilon} - 1}$$

$$= n^{\log_b a - \varepsilon} \frac{n^{\varepsilon} - 1}{b^{\varepsilon} - 1}$$
Proof of the master theorem $n$ is exact power of $b$

Lemma 1. Determine asymptotic bounds on this summation

Proof for Case 1

Substitute into $g(n): g(n) = O\left( \sum_{j=0}^{\log_b n - 1} a^j (n / b^j)^{\log_b n - \varepsilon} \right)$

$$g(n) = O\left( n^{\log_b n - \varepsilon} \left( \frac{n^\varepsilon - 1}{b^\varepsilon - 1} \right) \right)$$

$$= O\left( n^{\log_b n - \varepsilon} n^\varepsilon \right)$$

$$= O\left( n^{\log_b n} \right) \text{ (Now we proved case 1)}$$

Case 1. If $f(n) = O\left( n^{\log_b a - \varepsilon} \right)$, for some constant $\varepsilon > 0$, then $g(n) = O\left( n^{\log_b a} \right)$
• **Proof of the master theorem** $n$ is exact power of $b$

**Lemma 1. Determine asymptotic bounds on this summation**

**Proof for Case 2**

$$g(n) = \sum_{j=0}^{\log_b n - 1} a^j f(n / b^j)$$

**Case 2.** If $f(n) = \Theta(n^{\log_b a})$, then $g(n) = \Theta(n^{\log_b a} \log n)$

**Assumed fact:** $f(n) = \Theta(n^{\log_b a}) \rightarrow f(n / b^j) = \Theta((n / b^j)^{\log_b a})$

**Substitute into $g(n)$:** $g(n) = \Theta(\sum_{j=0}^{\log_b n - 1} a^j (n / b^j)^{\log_b a})$
• **Proof of the master theorem** \( n \) is exact power of \( b 

**Lemma 1.** Determine asymptotic bounds on this summation

Proof for Case 2

Substitute into \( g(n) \):

\[
\sum_{j=0}^{\log_b n - 1} a^j (n / b^j)^{\log_b a} = \Theta \left( \sum_{j=0}^{\log_b n - 1} a^j (n / b^j)^{\log_b a} \right)
\]

\[
\rightarrow \sum_{j=0}^{\log_b n - 1} a^j (n / b^j)^{\log_b a} = n^{\log_b a} \sum_{j=0}^{\log_b n - 1} \left( \frac{a}{b^{\log_b a}} \right)^j
\]

\[
= n^{\log_b a} \sum_{j=0}^{\log_b n - 1} (1)
\]

\[
= n^{\log_b a} \log_b n
\]
• **Proof of the master theorem** \( n \) is exact power of \( b 

**Lemma 1.** Determine asymptotic bounds on this summation Proof for Case 2

Substitute into \( g(n) \): \( g(n) = \Theta \left( \sum_{j=0}^{\log_b^n - 1} a^j (n/b^j)^{\log_b^n} \right) \)

\[
\rightarrow \sum_{j=0}^{\log_b^n - 1} a^j (n/b^j)^{\log_b^n} = n^{\log_b^n \log_b^n} = \Theta(n^{\log_b^n \log_b^n})
\]

\( = \Theta(n^{\log_b^n \log_b^n}) \)  \( (\text{Now we proved case 2})\)

---

**Case 2.** If \( f(n) = \Theta(n^{\log_b^n a}) \), then \( g(n) = \Theta(n^{\log_b^n \log n}) \)