16.070
Introduction to Computers & Programming

Algorithms: Recurrence

Prof. Kristina Lundqvist
Dept. of Aero/Astro, MIT
• If an algorithm contains a **recursive** call to itself, its running time can often be described by a **recurrence**

• A **recurrence** is an equation or inequality that describes a function in terms of its value on smaller inputs.

• Many natural functions are easily expressed as recurrences

  • $a_n = a_{n-1} + 1$; $a_1 = 1 \Rightarrow a_n = n$ (linear)
  • $a_n = a_{n-1} + 2n - 1$; $a_1 = 1 \Rightarrow a_n = n^2$ (polynomial)
  • $a_n = 2a_{n-1}$; $a_1 = 1 \Rightarrow a_n = 2^n$ (exponential)
  • $a_n = n \cdot a_{n-1}$; $a_1 = 1 \Rightarrow a_n = n!$ (others…)
Recurrence

- Recursion is **Mathematical Induction**

- In both, we have general and boundary conditions, with the *general* condition breaking the problem into smaller and smaller pieces.

- The *initial* or *boundary* condition terminate the recursion.
A recurrence equation defines a function, say $T(n)$. The function is defined recursively, that is, the function $T(.)$ appear in its definition. (recall recursive function call). The recurrence equation should have a base case.

For example:

$$T(n) = \begin{cases} 
T(n-1)+T(n-2), & \text{if } n>1 \\
1, & \text{if } n=1 \text{ or } n=0 
\end{cases}$$

for convenience, we sometime write the recurrence equation as:

$$T(n) = T(n-1)+T(n-2)$$

$$T(0) = T(1) = 1$$
Recurrences

- The expression:

\[ T(n) = \begin{cases} 
  c & n = 1 \\
  2T \left( \frac{n}{2} \right) + cn & n > 1 
\end{cases} \]

is a recurrence.

- Recurrence: an equation that describes a function in terms of its value on smaller functions
Recurrence Examples

\[ s(n) = \begin{cases} 0 & n = 0 \\ c + s(n-1) & n > 0 \end{cases} \]

\[ T(n) = \begin{cases} c & n = 1 \\ 2T\left(\frac{n}{2}\right) + c & n > 1 \end{cases} \]
Calculating Running Time
Through Recurrence Equation (1/2)

Algorithm A \( \text{min1}(a[1], a[2], \ldots, a[n]) \):

1. If \( n == 1 \), return \( a[1] \)
2. \( m := \text{min1}(a[1], a[2], \ldots, a[n-1]) \)
3. If \( m > a[n] \), return \( a[n] \), else return \( m \)

- Now, let’s count the number of comparisons
- Let \( T(n) \) be the total number of comparisons (in step 1 and 3).
  \[
  T(n) = 1 + T(n - 1) + 1;
  \]
  \[
  T(1) = 1;
  \]

\[ T(n) = n + 1, \quad \text{if } n > 1 \]
Calculating Running Time
Through Recurrence Equation (2/2)

Algorithm B \texttt{min2}(a[1],a[2],…,a[n]):

1. If \( n == 1 \) return the minimum of \( a[1] \);
2. Let \( m_1 := \texttt{min2}(a[1], a[2], …, a[n/2]) \);
   Let \( m_2 := \texttt{min2}(a[n/2+1], a[n/2+2], …, a[n]) \);
3. If \( m_1 > m_2 \) return \( m_1 \) else return \( m_2 \)

- For \( n>2 \), \( T(n) = T(n/2) + T(n/2) + 1 \),
  \( T(1)=1 \)

- To be precise, \( T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1 \),
  but for convenient, we ignore the “ceiling” and “floor”
  and assume \( n \) is a power of 2.
More Recurrence equations

\[ T(n) = 2 \cdot T(n/2) + 1, \quad T(1) = 1. \] Base case; initial condition.

\[ T(n) = T(n-1) + n, \quad T(1) = 1. \] Selection Sort

\[ T(n) = 2 \cdot T(n/2) + n, \quad T(1) = 1. \] Merge Sort

\[ T(n) = T(n/2) + 1, \quad T(1) = 0. \] Binary search
We can use mathematical induction to prove that a general function solves for a recursive one. Guess a solution and prove it by induction.

\[ T_n = 2T_{n-1} + 1 \ ; \ T_0 = 0 \]

<table>
<thead>
<tr>
<th>n</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_n )</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>7</td>
<td>15</td>
<td>31</td>
<td>63</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Prove: $T_n = 2^n - 1$ by induction:

1. Show the base case is true: $T_0 = 2^0 - 1 = 0$

2. Now assume true for $T_{n-1}$

3. Substitute in $T_{n-1}$ in recurrence for $T_n$

$$T_n = 2T_{n-1} + 1$$

$$= 2 (2^{n-1} - 1) + 1$$

$$= 2^n - 1$$
Solving Recurrences

There are 3 general methods for solving recurrences

1. **Substitution**: “Guess & Verify”: **guess** a solution and **verify** it is correct with an **inductive proof**

2. **Iteration**: “Convert to Summation”: convert the recurrence into a **summation** (by expanding some terms) and then bound the summation

3. Apply **“Master Theorem”**: if the recurrence has the form

   \[ T(n) = aT\left(\frac{n}{b}\right) + f(n) \]

then there is a formula that can (often) be applied.

Recurrence formulas are notoriously **difficult to derive**, but **easy to prove valid** once you have them
Simplications

- There are two simplifications we apply that won't affect asymptotic analysis
  - ignore floors and ceilings
  - assume base cases are constant, i.e., $T(n) = \Theta(1)$ for $n$ small enough
Solving Recurrences: **Substitution**

- This method involves **guessing** form of solution
- use **mathematical induction** to find the constants and verify solution
- use to find an upper or a lower bound (do both to obtain a tight bound)
The Substitution method

Solve: \( T(n) = 2T(\lfloor n/2 \rfloor) + n \)

- **Guess:** \( T(n) = O(n \lg n) \), that is: \( T(n) \leq cn \lg n \)
- **Prove:**
  - **Base case:** assume constant size inputs take const time
  - \( T(n) \leq cn \lg n \) for a choice of constant \( c > 0 \)
  - Assume that the bound holds for \( \lfloor n/2 \rfloor \), that is, \( T(\lfloor n/2 \rfloor) \leq c \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor) \)

Substituting into the recurrence yields:
\[
T(n) \leq 2(c \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + n \\
\leq cn \lg(n/2) + n \\
= cn \lg n - cn \lg 2 + n \\
= cn \lg n - cn + n \\
\leq cn \lg n
\]

Where last step holds as long as \( c \geq 1 \)

Example: $T(n) = 4T(n/2) + n$ (upper bound)

guess $T(n) = \mathcal{O}(n^3)$ and try to show $T(n) \leq cn^3$ for some $c > 0$ (we'll have to find $c$)

basis?

assume $T(k) \leq ck^3$ for $k < n$, and prove $T(n) \leq cn^3$

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

$\leq 4(c(n/2)^3 + n$

$= c/2n^3 + n$

$= cn^3 - (c/2n^3 - n)$

$\leq cn^3$

where the last step holds if $c > 2$ and $n > 1$

We find values of $c$ and $n_0$ by determining when $c/2n^3 - n \geq 0$
Solving Recurrences by Guessing (1/3)

- Guess the form of the answer, then use induction to find the constants and show that solution works

- Examples:
  - $T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n \ lg \ n)$
  - $T(n) = 2T(\lfloor n/2 \rfloor) + n \Rightarrow ???$
Guess the form of the answer, then use induction to find the constants and show that solution works.

Examples:

- $T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n \lg n)$
- $T(n) = 2T(\lfloor n/2 \rfloor) + n \Rightarrow T(n) = \Theta(n \lg n)$
- $T(n) = 2T(\lfloor n/2 \rfloor + 17) + n \Rightarrow ???$
- Guess the form of the answer, then use induction to find the constants and show that solution works

- Examples:
  - $T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n \lg n)$
  - $T(n) = 2T(\lfloor n/2 \rfloor) + n \Rightarrow T(n) = \Theta(n \lg n)$
  - $T(n) = 2T(\lfloor n/2 \rfloor + 17) + n \Rightarrow \Theta(n \lg n)$
Recursion-Trees

- Although the substitution method can provide a succinct proof that a solution to a recurrence is correct, it is sometimes difficult to come up with a good guess.

- Drawing out a recursion-tree is a good way to devise a good guess.
Recursion Trees

\[ T(n) = 2 \ T(n/2) + n^2, \quad T(1) = 1 \]
Solving Recurrences: Iteration

- Expand the recurrence
- Work some algebra to express as a summation
- Evaluate the summation
- We will show several examples
\[ s(n) = \begin{cases} 
0 & n = 0 \\
\ c + s(n-1) & n > 0 
\end{cases} \]
So far for \( n \geq k \) we have

- \( s(n) = ck + s(n-k) \)

What if \( k = n \)?

- \( s(n) = cn + s(0) = cn \)
- So far for \( n \geq k \) we have
  - \( s(n) = ck + s(n-k) \)
- What if \( k = n \)?
  - \( s(n) = cn + s(0) = cn \)
- So
  \[
  s(n) = \begin{cases} 
  0 & n = 0 \\
  c + s(n-1) & n > 0 
  \end{cases}
  \]
- Thus in general
  - \( s(n) = cn \)
\[ s(n) = \begin{cases} 
0 & n = 0 \\
 n + s(n-1) & n > 0 
\end{cases} \]

- \( s(n) \)
  
  \[
  = n + s(n-1) \\
  = n + n-1 + s(n-2) \\
  = n + n-1 + n-2 + s(n-3) \\
  = n + n-1 + n-2 + n-3 + s(n-4) \\
  = \ldots \\
  = n + n-1 + n-2 + n-3 + \ldots + n-(k-1) + s(n-k)
  \]
\[ s(n) = \begin{cases} 
0 & n = 0 \\
n + s(n-1) & n > 0 
\end{cases} \]

- \( s(n) \)
  
  \[ = n + s(n-1) \]
  
  \[ = n + n-1 + s(n-2) \]
  
  \[ = n + n-1 + n-2 + s(n-3) \]
  
  \[ = n + n-1 + n-2 + n-3 + s(n-4) \]
  
  \[ = \ldots \]
  
  \[ = n + n-1 + n-2 + n-3 + \ldots + n-(k-1) + s(n-k) \]
  
  \[ = \sum_{i=n-k+1}^{n} i + s(n-k) \]
So far for $n \geq k$ we have

\[
\sum_{i=n-k+1}^{n} i + s(n-k) = \begin{cases} 
0 & n = 0 \\
n + s(n-1) & n > 0 
\end{cases}
\]
\( s(n) = \begin{cases} 
0 & n = 0 \\
 n + s(n-1) & n > 0 
\end{cases} \)

- So far for \( n \geq k \) we have
  \[
  \sum_{i=n-k+1}^{n} i + s(n-k)
  \]
- What if \( k = n \)?
\[ s(n) = \begin{cases} 
0 & n = 0 \\
n + s(n-1) & n > 0 
\end{cases} \]

- So far for \( n \geq k \) we have

\[
\sum_{i=n-k+1}^{n} i + s(n-k)
\]

- What if \( k = n \)?

\[
\sum_{i=1}^{n} i + s(0) = \sum_{i=1}^{n} i + 0 = n \frac{n+1}{2}
\]
So far for \( n \geq k \) we have

\[
\sum_{i=n-k+1}^{n} i + s(n-k)
\]

What if \( k = n \)?

\[
\sum_{i=1}^{n} i + s(0) = \sum_{i=1}^{n} i + 0 = n \frac{n+1}{2}
\]

Thus in general

\[
s(n) = n \frac{n+1}{2}
\]
\[ T(n) = \begin{cases} 
  c & n = 1 \\
  2T\left(\frac{n}{2}\right) + c & n > 1 
\end{cases} \]

- \( T(n) = \)
  - \( 2T(n/2) + c \)
  - \( 2(2T(n/2/2) + c) + c \)
  - \( 2^2T(n/2^2) + 2c + c \)
  - \( 2^2(2T(n/2^2/2) + c) + 3c \)
  - \( 2^3T(n/2^3) + 4c + 3c \)
  - \( 2^3T(n/2^3) + 7c \)
  - \( 2^3(2T(n/2^3/2) + c) + 7c \)
  - \( 2^4T(n/2^4) + 15c \)
  - \( \ldots \)
  - \( 2^kT(n/2^k) + (2^k - 1)c \)
So far for $n > 2k$ we have

- $T(n) = 2^k T(n/2^k) + (2^k - 1)c$

What if $k = \lg n$?

- $T(n) = 2^{\lg n} T(n/2^{\lg n}) + (2^{\lg n} - 1)c$
  
  
  = $n T(n/n) + (n - 1)c$
  
  = $n T(1) + (n-1)c$
  
  = $nc + (n-1)c = (2n - 1)c$
Solving Recurrences: Iteration

\[ T(n) = \begin{cases} 
  c & n = 1 \\
  aT\left(\frac{n}{b}\right) + cn & n > 1
\end{cases} \]
$T(n) = \begin{cases} 
    c & n = 1 \\
    aT\left(\frac{n}{b}\right) + cn & n > 1 
\end{cases}$

- $T(n) =$
  
  $aT(n/b) + cn$
  
  $a(aT(n/b/b) + cn/b) + cn$
  
  $a^2T(n/b^2) + cna/b + cn$
  
  $a^2T(n/b^2) + cn(a/b + 1)$
  
  $a^2(aT(n/b^2/b) + cn/b^2) + cn(a/b + 1)$
  
  $a^3T(n/b^3) + cna^2/b^2 + cn(a/b + 1)$
  
  $a^3T(n/b^3) + cn(a^2/b^2 + a/b + 1)$
  
  $\ldots$
  
  $a^kT(n/b^k) + cn(a^k-1/b^{k-1} + a^{k-2}/b^{k-2} + \ldots + a^2/b^2 + a/b + 1)$
So we have

- \( T(n) = a^k T(n/b^k) + cn(a^{k-1}/b^{k-1} + ... + a^2/b^2 + a/b + 1) \)

For \( k = \log_b n \)

- \( n = b^k \)
- \( T(n) = a^k T(1) + cn(a^{k-1}/b^{k-1} + ... + a^2/b^2 + a/b + 1) \)
  - \( = a^k c + cn(a^{k-1}/b^{k-1} + ... + a^2/b^2 + a/b + 1) \)
  - \( = ca^k + cn(a^{k-1}/b^{k-1} + ... + a^2/b^2 + a/b + 1) \)
  - \( = cna^k/b^k + cn(a^{k-1}/b^{k-1} + ... + a^2/b^2 + a/b + 1) \)
  - \( = cn(a^k/b^k + ... + a^2/b^2 + a/b + 1) \)
\[ T(n) = \begin{cases} 
\frac{c}{n} & n = 1 \\
\alpha T\left(\frac{n}{b}\right) + cn & n > 1 
\end{cases} \]

- So with \( k = \log_b n \)
  - \( T(n) = cn(a^k/b^k + \ldots + a^2/b^2 + a/b + 1) \)
- What if \( a = b? \)
  - \( T(n) = cn(k + 1) \)
    - \( = cn(\log_b n + 1) \)
    - \( = \Theta(n \log n) \)
\[ T(n) = \begin{cases} 
 c & n = 1 \\
 aT\left(\frac{n}{b}\right) + cn & n > 1 
\end{cases} \]

- So with \( k = \log_b n \)
  - \( T(n) = cn(a^k/b^k + \ldots + a^2/b^2 + a/b + 1) \)
- What if \( a < b \)?
$T(n) = \begin{cases} 
\frac{c}{b} & n = 1 \\
 cT\left(\frac{n}{b}\right) + cn & n > 1 
\end{cases}$

- So with $k = \log_b n$
  - $T(n) = cn(a^k/b^k + ... + a^2/b^2 + a/b + 1)$

- What if $a < b$?
  - Recall that $\sum(x^k + x^{k-1} + ... + x + 1) = (x^{k+1} - 1)/(x-1)$
\[ T(n) = \begin{cases} 
\frac{c}{n} & n = 1 \\
\alpha T\left(\frac{n}{b}\right) + cn & n > 1 
\end{cases} \]

- So with \( k = \log_b n \)
  - \( T(n) = cn(a^k/b^k + \ldots + a^2/b^2 + a/b + 1) \)
- What if \( a < b \)?
  - Recall that \( \sum (x^k + x^{k-1} + \ldots + x + 1) = (x^{k+1} - 1)/(x-1) \)
  - So:

\[
\frac{a^k}{b^k} + \frac{a^{k-1}}{b^{k-1}} + \ldots + \frac{a}{b} + 1 = \frac{(a/b)^{k+1} - 1}{(a/b) - 1} = \frac{1 - (a/b)^{k+1}}{1 - (a/b)} < \frac{1}{1 - a/b}
\]
\[ T(n) = \begin{cases} \frac{c}{n} & n = 1 \\ aT\left(\frac{n}{b}\right) + cn & n > 1 \end{cases} \]

- So with \( k = \log_b n \)
  - \( T(n) = cn(a^k/b^k + \cdots + a^2/b^2 + a/b + 1) \)
- What if \( a < b \)?
  - Recall that \( \sum(x^k + x^{k-1} + \cdots + x + 1) = (x^{k+1} - 1)/(x-1) \)
  - So:
    \[
    \frac{a^k}{b^k} + \frac{a^{k-1}}{b^{k-1}} + \cdots + \frac{a}{b} + 1 = \frac{(a/b)^{k+1} - 1}{(a/b) - 1} = \frac{1 - (a/b)^{k+1}}{1 - a/b} < \frac{1}{1 - a/b}
    \]
- \( T(n) = cn \cdot \Theta(1) = \Theta(n) \)
So with $k = \log_b n$

- $T(n) = cn(a^k/b^k + \ldots + a^2/b^2 + a/b + 1)$

What if $a > b$?
So with \( k = \log_b n \)

- \( T(n) = cn(a^k/b^k + \ldots + a^2/b^2 + a/b + 1) \)

What if \( a > b \)?

\[
\frac{a^k}{b^k} + \frac{a^{k-1}}{b^{k-1}} + \ldots + \frac{a}{b} + 1 = \frac{(a/b)^{k+1} - 1}{(a/b) - 1} = \Theta\left((a/b)^k\right)
\]
\[ T(n) = \begin{cases} \alpha T\left(\frac{n}{b}\right) + cn & n > 1 \\ c & n = 1 \end{cases} \]

- So with \( k = \log_b n \)
  - \( T(n) = cn(a^k/b^k + \cdots + a^2/b^2 + a/b + 1) \)
- What if \( a > b \)?
  \[
  \frac{a^k}{b^k} + \frac{a^{k-1}}{b^{k-1}} + \cdots + \frac{a}{b} + 1 = \frac{(a/b)^{k+1} - 1}{(a/b) - 1} = \Theta((a/b)^k)
  \]
- \( T(n) = cn \cdot \Theta(a^k / b^k) \)
\( T(n) = \begin{cases} 
  c n & \text{if } n = 1 \\
  aT\left(\frac{n}{b}\right) + cn & \text{if } n > 1 
\end{cases} \)

- So with \( k = \log_b n \)
  - \( T(n) = cn(a^k/b^k + \ldots + a^2/b^2 + a/b + 1) \)
- What if \( a > b \)?
  \[
  \frac{a^k}{b^k} + \frac{a^{k-1}}{b^{k-1}} + \ldots + \frac{a}{b} + 1 = \frac{(a/b)^{k+1} - 1}{(a/b) - 1} = \Theta((a/b)^k)
  \]

- \( T(n) = cn \cdot \Theta(a^k / b^k) \)
  \[
  = cn \cdot \Theta(a^{\log n} / b^{\log n}) = cn \cdot \Theta(a^{\log n} / n)
  \]
\[ T(n) = \begin{cases} 
\frac{c}{n} & n = 1 \\
 aT\left(\frac{n}{b}\right) + cn & n > 1 
\end{cases} \]

- So with \( k = \log_b n \)
  - \( T(n) = cn\left(a^k/b^k + \ldots + a^2/b^2 + a/b + 1\right) \)
- What if \( a > b \)?

\[
\frac{a^k}{b^k} + \frac{a^{k-1}}{b^{k-1}} + \cdots + \frac{a}{b} + 1 = \frac{(a/b)^{k+1} - 1}{(a/b) - 1} = \Theta\left((a/b)^k\right)
\]

- \( T(n) = cn \cdot \Theta(a^k/b^k) \)
  \[
  = cn \cdot \Theta(a^{\log n}/b^{\log n}) = cn \cdot \Theta(a^{\log n}/n)
  \]
  
  *recall logarithm fact: \( a^{\log n} = n^{\log a} \)
\[ T(n) = \begin{cases} 
\frac{c}{n} & \text{if } n = 1 \\
\alpha T\left(\frac{n}{b}\right) + cn & \text{if } n > 1
\end{cases} \]

- So with \( k = \log_b n \)
  - \( T(n) = cn(a^k/b^k + \ldots + a^2/b^2 + a/b + 1) \)
- What if \( a > b \)?

\[
\frac{a^k}{b^k} + \frac{a^{k-1}}{b^{k-1}} + \cdots + \frac{a}{b} + 1 = \frac{(a/b)^{k+1} - 1}{(a/b) - 1} = \Theta((a/b)^k)
\]

- \( T(n) = cn \cdot \Theta(a^k / b^k) \)

\[
= cn \cdot \Theta(a^{\log n} / b^{\log n}) = cn \cdot \Theta(a^{\log n} / n)
\]

*recall logarithm fact: \( a^{\log n} = n^{\log a} \)*

\[
= cn \cdot \Theta(n^{\log a} / n) = \Theta(cn \cdot n^{\log a} / n)
\]
\[ T(n) = \begin{cases} 
  \frac{c}{n} & n = 1 \\
  aT\left(\frac{n}{b}\right) + cn & n > 1 
\end{cases} \]

- So with \( k = \log_b n \)
  - \( T(n) = cn(a^k/b^k + \ldots + a^2/b^2 + a/b + 1) \)
- What if \( a > b \)?
  \[
  \frac{a^k}{b^k} + \frac{a^{k-1}}{b^{k-1}} + \ldots + \frac{a}{b} + 1 = \frac{(a/b)^{k+1} - 1}{(a/b) - 1} = \Theta\left((a/b)^k\right)
  \]
- \( T(n) = cn \cdot \Theta(a^k / b^k) \)
  \[
  = cn \cdot \Theta(a^{\log n} / b^{\log n}) = cn \cdot \Theta(a^{\log n} / n)
  \]
  *recall logarithm fact: \( a^{\log n} = n^{\log a} \)*
  \[
  = cn \cdot \Theta(n^{\log a} / n) = \Theta(cn \cdot n^{\log a} / n)
  \]
  \[
  = \Theta(n^{\log a})
  \]
So...

\[ T(n) = \begin{cases} 
\Theta(n) & a < b \\
\Theta(n \log_b n) & a = b \\
\Theta(n^{\log_b a}) & a > b 
\end{cases} \]
The Master Method

- Provides a “cookbook” method for solving recurrences of the form

- \( T(n) = aT(n/b) + f(n) \), where \( a \geq 1 \) and \( b > 1 \) are constants and \( f(n) \) is an asymptotically positive function.

- The Master method requires memorization of three cases, but then the solution of many recurrences can be determined quite easily, often without pencil and paper.
The Master Method

- Given: a *divide and conquer* algorithm
  - An algorithm that divides the problem of size $n$ into $a$ subproblems, each of size $n/b$
  - Let the cost of each stage (i.e., the work to divide the problem + combine solved subproblems) be described by the function $f(n)$

- Then, the Master Method gives us a cookbook for the algorithm’s running time:
**Master Theorem:** Let $a > 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on nonnegative integers as:

$$T(n) = aT(n/b) + f(n),$$

Then, $T(n)$ can be bounded asymptotically as follows:

1. $T(n) = \Theta(n^{\log_b a})$ if $f(n) = \Theta(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$.

2. $T(n) = \Theta(n^{\log_b a \log n})$ if $f(n) = \Theta(n^{\log_b a})$.

3. $T(n) = \Theta(f(n))$ if $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$ and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large $n$. 
The Master Theorem

- if \( T(n) = aT(n/b) + f(n) \) then

\[
T(n) = \begin{cases} 
\Theta(n^{\log_b a}) & f(n) = O(n^{\log_b a - \varepsilon}) \\
\Theta(n^{\log_b a \log n}) & f(n) = \Theta(n^{\log_b a}) \\
\Theta(f(n)) & f(n) = \Omega(n^{\log_b a + \varepsilon}) \quad \text{AND} \quad af(n/b) < cf(n) \quad \text{for large } n 
\end{cases}
\]

\( \varepsilon > 0 \)

\( c < 1 \)
Intuition: compare $f(n)$ with $\Theta(n^{\log_b a})$

- case 1: $f(n)$ is `polynomially smaller than’ $\Theta(n^{\log_b a})$
- case 2: $f(n)$ is `asymptotically equal to’ $\Theta(n^{\log_b a})$
- case 3: $f(n)$ is `polynomially larger than' $\Theta(n^{\log_b a})$
In general (Master Theorem, CLR, p.62), $T(1) = d$, and for $n > 1$,

$$T(n) = aT(n/b) + cn$$

has solution

- if $a < b$, $T(n) = O(n)$;
- if $a = b$, $T(n) = O(n \log n)$;
- if $a > b$, $T(n) = O(n^{\log_b a})$
Case I

Example: $T(n) = 9T(n/3) + n$

- $a = 9, b = 3, f(n) = n, \quad n^{\log_b a} = n^{\log_3 9} = n^2$
- compare $f(n) = n$ with $n^{\log_b a} = n^2$
- $n = O(n^{2-\varepsilon})$ (f(n) is polynomially smaller than $n^{\log_b a}$)
- case 1 applies:

$$T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$$
Case II

Example: \( T(n) = T(2n/3) + 1 \)

- \( a = 1, \ b = 3/2, \ f(n) = 1, n^{\log_b a} = n^{\log_3 2} = n^0 = 1 \)
- compare \( f(n) = 1 \) with \( n^{\log_b a} = 1 \)
- \( 1 = \Theta(1) \) (\( f(n) \) is asymptotically equal to \( n^{\log_b a} \))
- case 2 applies:
  \( T(n) = \Theta(n^{\log_b a} \log n) = \Theta(\log n) \)
Case III

Example: $T(n) = 3T(n/4) + n \log n$

- $a = 3$, $b = 4$, $f(n) = n \log n$, $n^{\log_b a} = n^{\log_4 3} = n^{0.793}$
- compare $f(n) = n \log n$ with $n^{\log_b a} = n^{0.793}$
- $n \log n = \Omega(n^{0.793-\epsilon})$ $f(n)$ is polynomially larger than $n^{\log_b a}$

- case 3 might apply: need to check `regularity' of $f(n)$
  - find $c < 1$ s.t. $af(n/b) \leq cf(n)$ for large enough $n$
  - i.e. $\frac{3n}{4} \log\frac{n}{4} \leq cn \log n$ Which is true for $c = \frac{3}{4}$

- case 3 applies: $T(n) = \Theta(f(n)) = \Theta(n \log n)$