CMPSC 465: LECTURE IV

Solving Recurrences

Ke Chen

September 05, 2025

Solving recurrences

Recall that the time complexity $T(n)$ of MergeSort satisfies:
$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n)$.

Simplification: assume n is a power of 2 so we can ignore floors and ceilings.

Solving recurrences

Recall that the time complexity $T(n)$ of MergeSort satisfies:
$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n)$.

Simplification: assume n is a power of 2 so we can ignore floors and ceilings.

$T(n) = 2T(n/2) + \Theta(n)$.

Solving recurrences

Recall that the time complexity $T(n)$ of MergeSort satisfies:
$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n)$.

Simplification: assume n is a power of 2 so we can ignore floors and ceilings.

$T(n) = 2T(n/2) + \Theta(n)$.

Method 1: Solve by substitution

- Make a guess, e.g., $T(n) = O(n \log n)$.
- Try to prove the guess by induction.

Solving recurrences by substitution

$T(n) = 2T(n/2) + \Theta(n).$

Let's guess $T(n) = O(n \log n)$.

Solving recurrences by substitution

$T(n) = 2T(n/2) + \Theta(n).$

Let's guess $T(n) = O(n \log n)$.

In order to prove our guess, we need to show that $T(n) \leq c \cdot n \log n$ for some constant c.

Solving recurrences by substitution

$T(n) = 2T(n/2) + \Theta(n).$

Let's guess $T(n) = O(n \log n)$.

In order to prove our guess, we need to show that $T(n) \leq c \cdot n \log n$ for some constant c.

Assume this is true for all $m < n$, in particular, for $m = n/2$. Substituting into the recurrence gives

Solving recurrences by substitution

$T(n) = 2T(n/2) + \Theta(n).$

Let's guess $T(n) = O(n \log n)$.

In order to prove our guess, we need to show that
$T(n) \leq c \cdot n \log n$ for some constant c.

Assume this is true for all $m < n$, in particular, for $m = n/2$.
Substituting into the recurrence gives

$$\begin{aligned}
T(n) &= 2T(n/2) + \Theta(n) \\
&\leq 2c \cdot (n/2) \log(n/2) + O(n) \\
&= 2c \cdot (n/2) \log n - 2c \cdot n/2 + O(n) \\
&\leq c \cdot n \log n - c \cdot n + c' \cdot n \\
&= c \cdot n \log n - (c - c') \cdot n \\
&\leq c \cdot n \log n.
\end{aligned}$$

The last step holds as long as we choose $c \geq c'$.

Solving recurrences by substitution

$\boxed{T(n) = 2T(n/2) + \Theta(n).}$

Let's guess $T(n) = O(n)$.

Solving recurrences by substitution

$T(n) = 2T(n/2) + \Theta(n).$

Let's guess $T(n) = O(n)$.

Need to show that $T(n) \leq c \cdot n$ for some constant c.

Solving recurrences by substitution

$T(n) = 2T(n/2) + \Theta(n).$

Let's guess $T(n) = O(n)$.

Need to show that $T(n) \le c \cdot n$ for some constant c.

Assume this is true for all $m < n$, in particular, for $m = n/2$.
Substituting into the recurrence gives

Solving recurrences by substitution

$T(n) = 2T(n/2) + \Theta(n).$

Let's guess $T(n) = O(n)$.

Need to show that $T(n) \leq c \cdot n$ for some constant c.

Assume this is true for all $m < n$, in particular, for $m = n/2$.
Substituting into the recurrence gives

$$\begin{aligned} T(n) &= 2T(n/2) + \Theta(n) \\ &\leq 2c \cdot n/2 + O(n) \\ &\leq 2c \cdot n/2 + c' \cdot n \\ &= (c + c')n \end{aligned}$$

Solving recurrences by substitution

$T(n) = 2T(n/2) + \Theta(n).$

Let's guess $T(n) = O(n)$.

Need to show that $T(n) \leq c \cdot n$ for some constant c.

Assume this is true for all $m < n$, in particular, for $m = n/2$.
Substituting into the recurrence gives

$$\begin{aligned} T(n) &= 2T(n/2) + \Theta(n) \\ &\leq 2c \cdot n/2 + O(n) \\ &\leq 2c \cdot n/2 + c' \cdot n \\ &= (c + c')n > c \cdot n \end{aligned}$$

Solving recurrences by substitution

$T(n) = 2T(n/2) + \Theta(n).$

Let's guess $T(n) = O(n)$.

Need to show that $T(n) \leq c \cdot n$ for some constant c.

Assume this is true for all $m < n$, in particular, for $m = n/2$.
Substituting into the recurrence gives

$$\begin{aligned} T(n) &= 2T(n/2) + \Theta(n) \\ &\leq 2c \cdot n/2 + O(n) \\ &\leq 2c \cdot n/2 + c' \cdot n \\ &= (c + c')n > c \cdot n \end{aligned}$$

Note: When the induction proof fails, it does not necessarily mean the initial guess was wrong.

Solving recurrences by substitution

$T(n) = 2T(n/2) + \Theta(n).$

We can also guess a lower bound $T(n) = \Omega(n \log n)$.

Need to show that $T(n) \geq c \cdot n \log n$ for some constant c.

Assume this is true for all $m < n$, in particular, for $m = n/2$.
Substituting into the recurrence gives

$$\begin{aligned}
T(n) &= 2T(n/2) + \Theta(n) \\
&\geq 2c \cdot (n/2) \log(n/2) + \Omega(n) \\
&= 2c \cdot (n/2) \log n - 2c \cdot n/2 + \Omega(n) \\
&\geq c \cdot n \log n - c \cdot n + c' \cdot n \\
&= c \cdot n \log n - (c - c') \cdot n \\
&\geq c \cdot n \log n.
\end{aligned}$$

The last step holds as long as we choose $c \leq c'$.

Solving recurrences by unrolling

$T(n) = 2T(n/2) + \Theta(n).$

To get an upper bound, we can try unrolling the recurrence:

$$T(n) \leq 2T(n/2) + c \cdot n$$

Solving recurrences by unrolling

$T(n) = 2T(n/2) + \Theta(n).$

To get an upper bound, we can try unrolling the recurrence:

$$\begin{aligned} T(n) &\leq 2T(n/2) + c \cdot n \\ &\leq 2\left(2T(n/4) + c \cdot (n/2)\right) + c \cdot n \end{aligned}$$

Solving recurrences by unrolling

$T(n) = 2T(n/2) + \Theta(n).$

To get an upper bound, we can try unrolling the recurrence:

$$\begin{aligned} T(n) &\leq 2T(n/2) + c \cdot n \\ &\leq 2\left(2T(n/4) + c \cdot (n/2)\right) + c \cdot n \\ &= 4T(n/4) + 2c \cdot n \end{aligned}$$

Solving recurrences by unrolling

$T(n) = 2T(n/2) + \Theta(n).$

To get an upper bound, we can try unrolling the recurrence:

$$
\begin{aligned}
T(n) &\leq 2T(n/2) + c \cdot n \\
&\leq 2\left(2T(n/4) + c \cdot (n/2)\right) + c \cdot n \\
&= 4T(n/4) + 2c \cdot n \\
&\leq 4\left(2T(n/8) + c \cdot (n/4)\right) + 2c \cdot n
\end{aligned}
$$

Solving recurrences by unrolling

$T(n) = 2T(n/2) + \Theta(n).$

To get an upper bound, we can try unrolling the recurrence:

$$\begin{aligned}
T(n) &\leq 2T(n/2) + c \cdot n \\
&\leq 2\left(2T(n/4) + c \cdot (n/2)\right) + c \cdot n \\
&= 4T(n/4) + 2c \cdot n \\
&\leq 4\left(2T(n/8) + c \cdot (n/4)\right) + 2c \cdot n \\
&= 8T(n/8) + 3c \cdot n
\end{aligned}$$

Solving recurrences by unrolling

$T(n) = 2T(n/2) + \Theta(n).$

To get an upper bound, we can try unrolling the recurrence:

$$\begin{aligned}
T(n) &\le 2T(n/2) + c \cdot n \\
&\le 2\left(2T(n/4) + c \cdot (n/2)\right) + c \cdot n \\
&= 4T(n/4) + 2c \cdot n \\
&\le 4\left(2T(n/8) + c \cdot (n/4)\right) + 2c \cdot n \\
&= 8T(n/8) + 3c \cdot n \\
&\ \ \vdots \\
&\le 2^k T(n/2^k) + k \cdot c \cdot n
\end{aligned}$$

Solving recurrences by unrolling

$T(n) = 2T(n/2) + \Theta(n).$

To get an upper bound, we can try unrolling the recurrence:

$$
\begin{aligned}
T(n) &\leq 2T(n/2) + c \cdot n \\
&\leq 2\left(2T(n/4) + c \cdot (n/2)\right) + c \cdot n \\
&= 4T(n/4) + 2c \cdot n \\
&\leq 4\left(2T(n/8) + c \cdot (n/4)\right) + 2c \cdot n \\
&= 8T(n/8) + 3c \cdot n \\
&\ \vdots \\
&\leq 2^k T(n/2^k) + k \cdot c \cdot n
\end{aligned}
$$

When $n/2^k = 1$, or $k = \log n$, we reach the base case $T(1) = O(1)$. So the final result is $T(n) \leq 2^k O(1) + k \cdot c \cdot n$.

Solving recurrences by unrolling

$T(n) = 2T(n/2) + \Theta(n).$

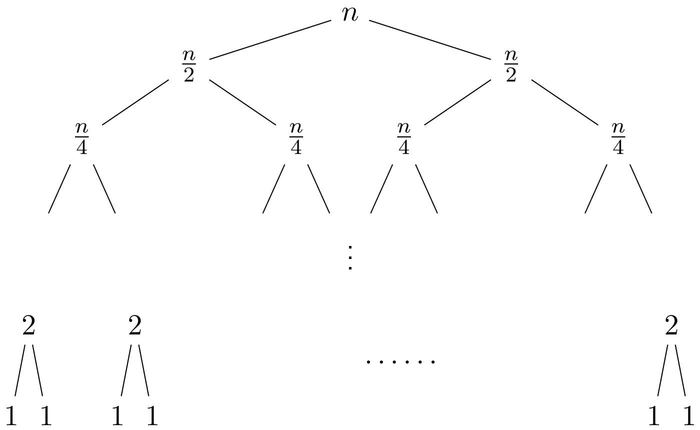
To get an upper bound, we can try unrolling the recurrence:

$$\begin{aligned}
T(n) &\le 2T(n/2) + c \cdot n \\
&\le 2\left(2T(n/4) + c \cdot (n/2)\right) + c \cdot n \\
&= 4T(n/4) + 2c \cdot n \\
&\le 4\left(2T(n/8) + c \cdot (n/4)\right) + 2c \cdot n \\
&= 8T(n/8) + 3c \cdot n \\
&\;\;\vdots \\
&\le 2^k T(n/2^k) + k \cdot c \cdot n
\end{aligned}$$

When $n/2^k = 1$, or $k = \log n$, we reach the base case $T(1) = O(1)$. So the final result is $T(n) \le 2^k O(1) + k \cdot c \cdot n = O(n) + c \cdot n \log n = O(n \log n)$.

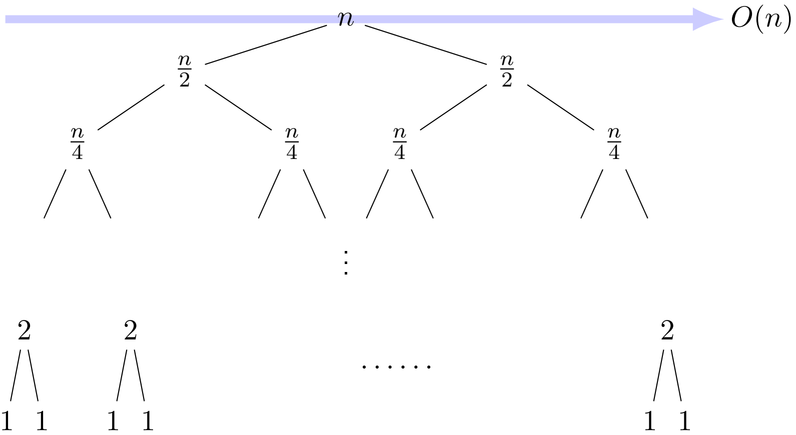
Solving recurrences by recursion tree

Recall that the recursive calls of MergeSort form a tree structure:



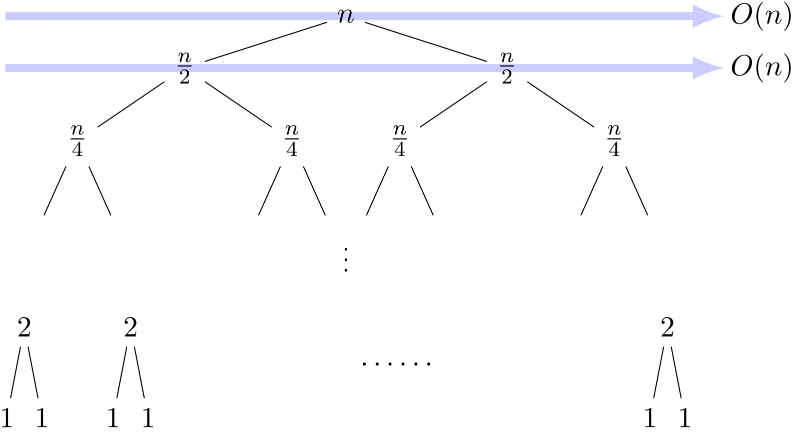
Solving recurrences by recursion tree

Recall that the recursive calls of MergeSort form a tree structure:



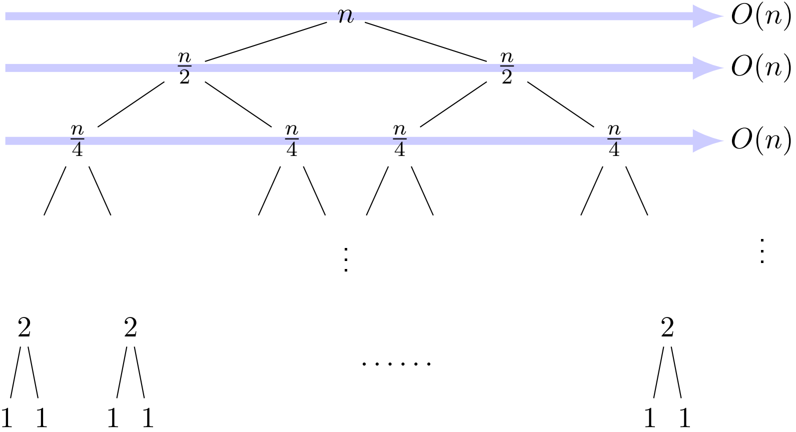
Solving recurrences by recursion tree

Recall that the recursive calls of MergeSort form a tree structure:



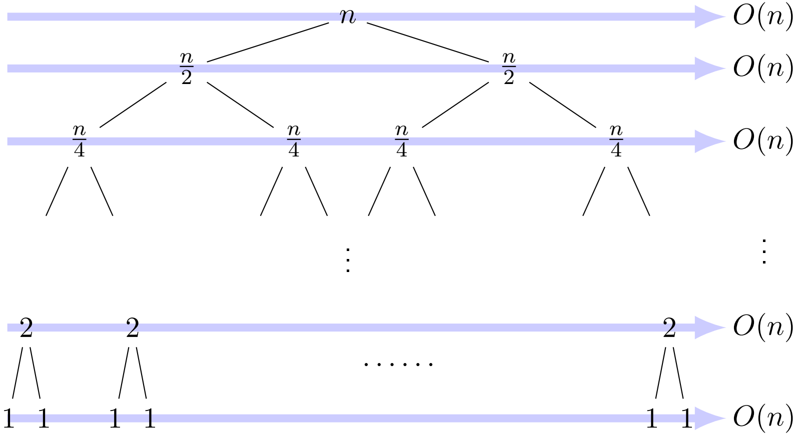
Solving recurrences by recursion tree

Recall that the recursive calls of MergeSort form a tree structure:



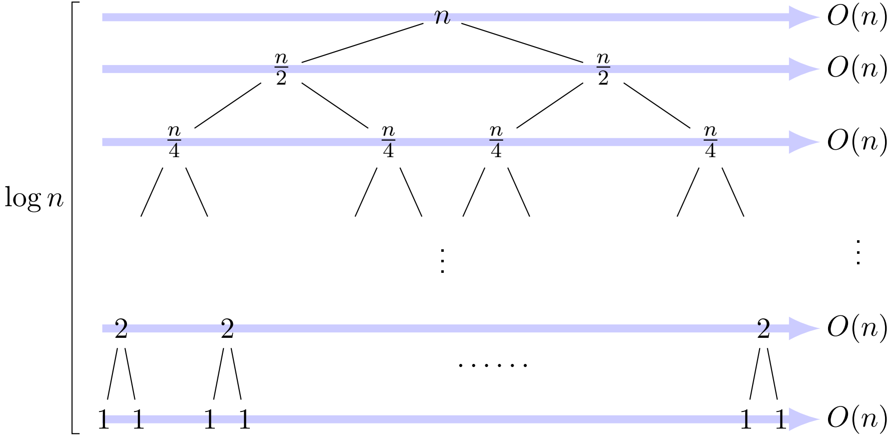
Solving recurrences by recursion tree

Recall that the recursive calls of MergeSort form a tree structure:



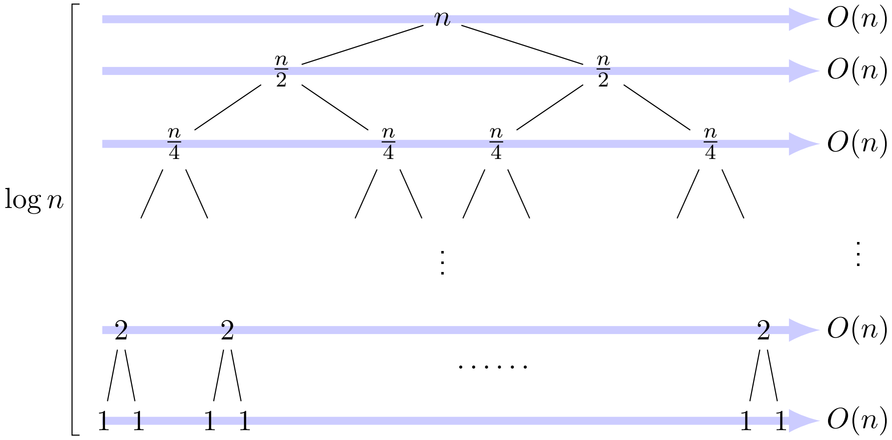
Solving recurrences by recursion tree

Recall that the recursive calls of MergeSort form a tree structure:



Solving recurrences by recursion tree

Recall that the recursive calls of MergeSort form a tree structure:



$O(n \log n)$

Solving recurrences by recursion tree

For a more general recurrence relation $T(n) = a \cdot T(n/b) + f(n)$

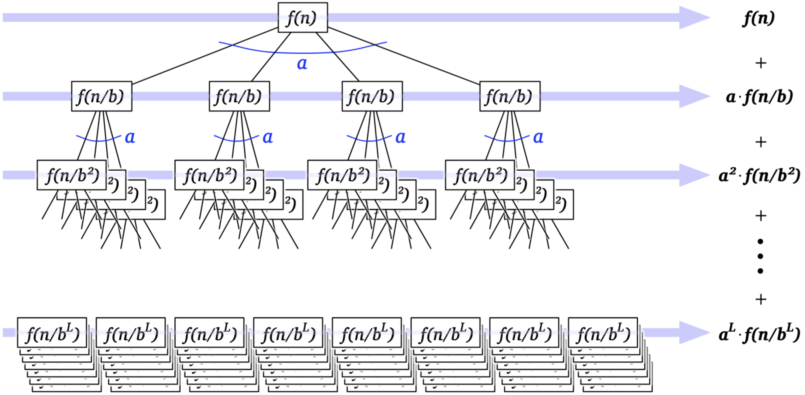


Image by: Jeff Erickson, 2018

Solving recurrences by recursion tree

For a more general recurrence relation $T(n) = a \cdot T(n/b) + f(n)$

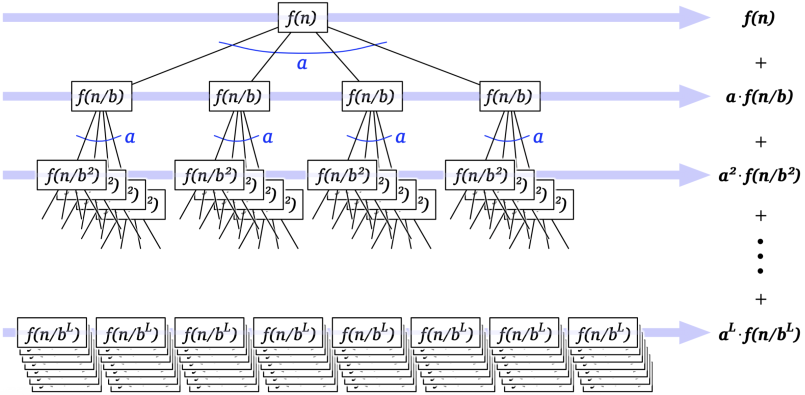


Image by: Jeff Erickson, 2018

So $T(n) = \sum_{i=0}^{L} a^i \cdot f(n/b^i)$ where $L = \log_b n$.

Solving recurrences by recursion tree

$T(n) = a \cdot T(n/b) + f(n)$ solves to $T(n) = \sum_{i=0}^{\log_b n} a^i \cdot f(n/b^i)$.

If $f(n)$ is bounded above by $f(n) = O(n^d)$, we have

$$T(n) = \sum_{i=0}^{\log_b n} a^i \cdot O\left(\left(\frac{n}{b^i}\right)^d\right)$$

Solving recurrences by recursion tree

$T(n) = a \cdot T(n/b) + f(n)$ solves to $T(n) = \sum_{i=0}^{\log_b n} a^i \cdot f(n/b^i)$.

If $f(n)$ is bounded above by $f(n) = O(n^d)$, we have

$$\begin{aligned} T(n) &= \sum_{i=0}^{\log_b n} a^i \cdot O\left(\left(\frac{n}{b^i}\right)^d\right) \\ &= \sum_{i=0}^{\log_b n} \left(\frac{a}{b^d}\right)^i \cdot O(n^d) \end{aligned}$$

Solving recurrences by recursion tree

$T(n) = a \cdot T(n/b) + f(n)$ solves to $T(n) = \sum_{i=0}^{\log_b n} a^i \cdot f(n/b^i)$.

If $f(n)$ is bounded above by $f(n) = O(n^d)$, we have

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\log_b n} a^i \cdot O\left(\left(\frac{n}{b^i}\right)^d\right) \\
&= \sum_{i=0}^{\log_b n} \left(\frac{a}{b^d}\right)^i \cdot O(n^d) \\
&= O(n^d) \cdot \sum_{i=0}^{\log_b n} \left(\frac{a}{b^d}\right)^i
\end{aligned}
$$

Solving recurrences by recursion tree

$T(n) = a \cdot T(n/b) + O(n^d)$ solves to $T(n) = O(n^d) \cdot \sum_{i=0}^{\log_b n} \left(\frac{a}{b^d}\right)^i$.

- If $a/b^d = 1$, $T(n) = O(n^d \log n)$.

Fact: The sum of any geometric series is a constant times its largest term:

- If $a/b^d < 1$, largest term is the first term, $T(n) = O(n^d)$.
- If $a/b^d > 1$, largest term is the last term

$$(a/b^d)^{\log_b n} = a^{\log_b n}/(b^{\log_b n})^d = n^{\log_b a}/n^d,$$

so $T(n) = O(n^{\log_b a})$.

Solving recurrences by recursion tree

Master theorem If $T(n) = aT(\lceil n/b \rceil) + O(n^d)$ for some constants $a > 0$, $b > 1$, and $d \geq 0$, then

$$T(n) = \begin{cases} O(n^d) & \text{if } a/b^d < 1 \\ O(n^d \log n) & \text{if } a/b^d = 1 \\ O(n^{\log_b a}) & \text{if } a/b^d > 1 \end{cases}.$$

Solving recurrences by recursion tree

Master theorem If $T(n) = aT(\lceil n/b \rceil) + O(n^d)$ for some constants $a > 0$, $b > 1$, and $d \geq 0$, then

$$T(n) = \begin{cases} O(n^d) & \text{if } a/b^d < 1 \\ O(n^d \log n) & \text{if } a/b^d = 1 \\ O(n^{\log_b a}) & \text{if } a/b^d > 1 \end{cases}.$$

Examples.

- $T(n) = 2T(n/2) + O(n)$

Solving recurrences by recursion tree

Master theorem If $T(n) = aT(\lceil n/b \rceil) + O(n^d)$ for some constants $a > 0$, $b > 1$, and $d \geq 0$, then

$$T(n) = \begin{cases} O(n^d) & \text{if } a/b^d < 1 \\ O(n^d \log n) & \text{if } a/b^d = 1 \\ O(n^{\log_b a}) & \text{if } a/b^d > 1 \end{cases}.$$

Examples.

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

Solving recurrences by recursion tree

Master theorem If $T(n) = aT(\lceil n/b \rceil) + O(n^d)$ for some constants $a > 0$, $b > 1$, and $d \geq 0$, then

$$T(n) = \begin{cases} O(n^d) & \text{if } a/b^d < 1 \\ O(n^d \log n) & \text{if } a/b^d = 1 \\ O(n^{\log_b a}) & \text{if } a/b^d > 1 \end{cases} .$$

Examples.

- $T(n) = 2T(n/2) + O(n)$
- $T(n) = T(n/2) + O(1)$
- $T(n) = 19T(n/3) + O(n^3)$

Solving recurrences by recursion tree

Master theorem If $T(n) = aT(\lceil n/b \rceil) + O(n^d)$ for some constants $a > 0$, $b > 1$, and $d \geq 0$, then

$$T(n) = \begin{cases} O(n^d) & \text{if } a/b^d < 1 \\ O(n^d \log n) & \text{if } a/b^d = 1 \\ O(n^{\log_b a}) & \text{if } a/b^d > 1 \end{cases} .$$

Examples.

- $T(n) = 2T(n/2) + O(n)$
- $T(n) = T(n/2) + O(1)$
- $T(n) = 19T(n/3) + O(n^3)$
- $T(n) = 4T(n/2) + O(n \log n)$

Solving recurrences by recursion tree

Master theorem If $T(n) = aT(\lceil n/b \rceil) + O(n^d)$ for some constants $a > 0$, $b > 1$, and $d \geq 0$, then

$$T(n) = \begin{cases} O(n^d) & \text{if } a/b^d < 1 \\ O(n^d \log n) & \text{if } a/b^d = 1 \\ O(n^{\log_b a}) & \text{if } a/b^d > 1 \end{cases}.$$

Examples.

- $T(n) = 2T(n/2) + O(n)$
- $T(n) = T(n/2) + O(1)$
- $T(n) = 19T(n/3) + O(n^3)$
- $T(n) = 4T(n/2) + O(n \log n)$
- $T(n) = 4T(n/3) + O(n \log n)$