5. Solving recurrences

Time Complexity Analysis of Merge Sort

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Q. How to prove that the run-time of merge sort is O(n log n)?

A.

Time Complexity Analysis of Merge Sort

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

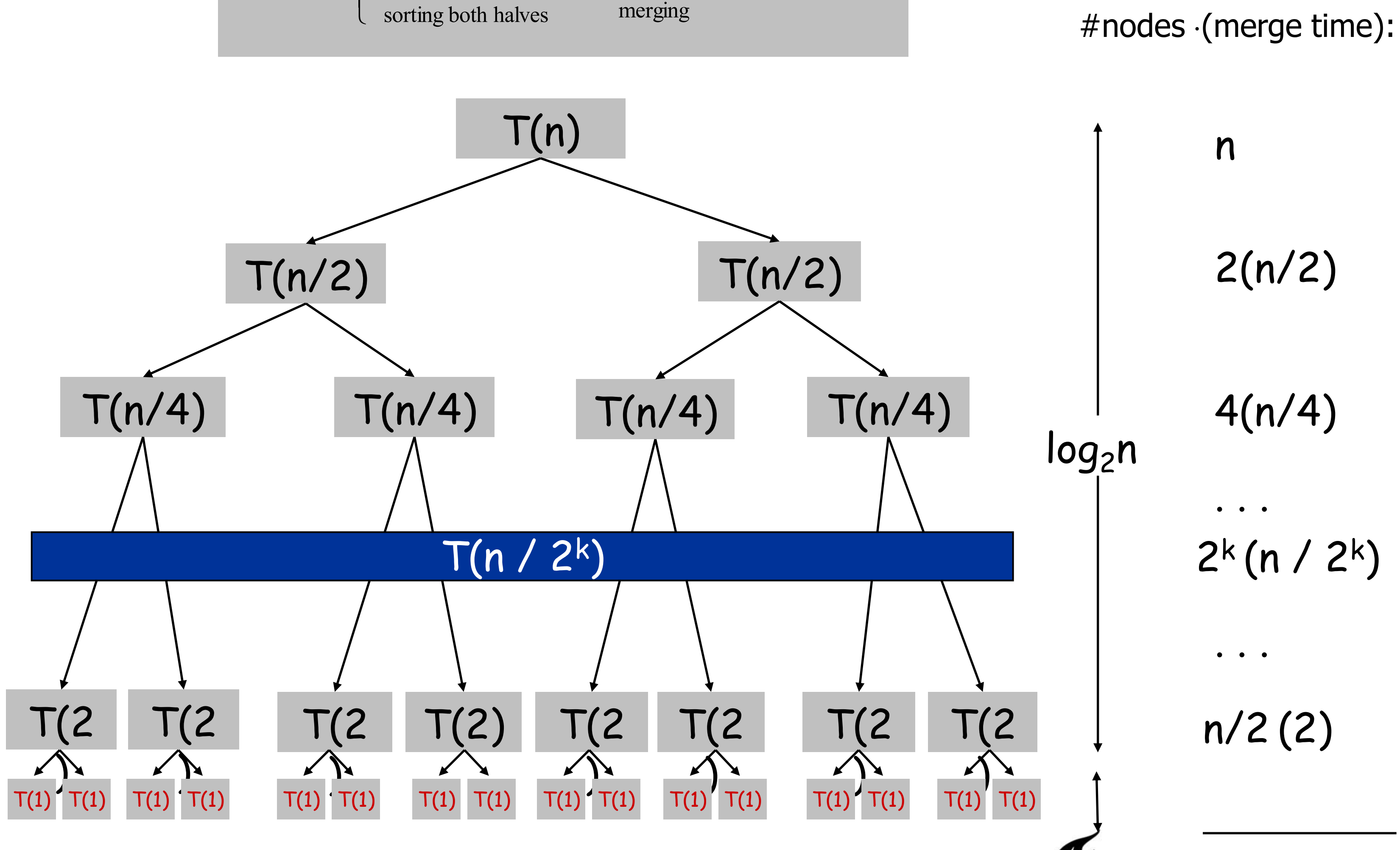
Q. How to prove that the run-time of merge sort is O(n log n)?

A. We have seen several methods:

- Recursion tree
- Substitution (by induction)

Proof by Recursion Tree

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$



merge time: $n\log_2 n$

Proof by Induction/Substitution (when n is power of 2)

Claim. If T(n) satisfies this recurrence, then $T(n) = n \log_2 n$.

assume n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Pf. (by induction on n)

Proof by Induction/Substitution (when n is power of 2)

Claim. If T(n) satisfies this recurrence, then $T(n) = n \log_2 n$.

assume n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Pf. (by induction on n)

- Base case: $n = 1$.
- Induction hypothesis: $T(n) = n \log_2 n$.
- Step: show that $T(2n) = 2n \log_2 (2n)$. Now for 2n (not n+1 as we are used to) !

Q. How do we proof this step?

Proof by Induction/Substitution (when n is power of 2)

Claim. If T(n) satisfies this recurrence, then $T(n) = n \log_2 n$.

assume n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Pf. (by induction on n)

- Base case: $n = 1$.
- Induction hypothesis: $T(n) = n \log_2 n$.
- Step: show that $T(2n) = 2n \log_2 (2n)$. Now for 2n (not n+1 as we are used to) !

$$\begin{aligned} T(2n) &= 2T(n) + 2n \\ &= 2n \log_2 n + 2n \\ &= \\ &= 2n \log_2 (2n) \end{aligned}$$

Induction hypothesis

TU Delft

Proof by Induction/Substitution (when n is power of 2)

Claim. If T(n) satisfies this recurrence, then $T(n) = n \log_2 n$.

assume n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Pf. (by induction on n)

- Base case: n = 1.
- Induction hypothesis: $T(n) = n \log_2 n$.
- Step: show that $T(2n) = 2n \log_2 (2n)$.

Now for 2n (not n+1 as we are used to)!

$$\begin{aligned} T(2n) &= 2T(n) + 2n \\ &= 2n \log_2 n + 2n \\ &= 2n(\log_2(2n) - 1) + 2n \\ &= 2n \log_2(2n) \end{aligned}$$

$$\boxed{\log_2(2n) = 1 + \log n}$$

Proof by Induction/Substitution (with rounding)

Claim. If T(n) satisfies the following recurrence, then $T(n) \leq n \lceil \log n \rceil$.

$\uparrow$ $\log_2 n$

$$T(n) \leq \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Pf. (by induction on n)

- Base case: $n = 1$.
- Define $n_1 = \lceil n / 2 \rceil$, $n_2 = \lfloor n / 2 \rfloor$.
- Hypothesis: assume true for 1, 2, ... , n–1.
- Step:

Proof by Induction/Substitution (with rounding)

Claim. If T(n) satisfies the following recurrence, then T(n) $\leq n \lceil \log n \rceil$.

$\uparrow$ $\log_2 n$

$$T(n) \leq \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Pf. (continued)

- Step:

$$\begin{aligned} T(n) &\leq T(n_1) + T(n_2) + n \\ &\leq n_1 \lceil \log n_1 \rceil + n_2 \lceil \log n_2 \rceil + n \\ \\ &= n \lceil \log n \rceil \end{aligned}$$

$\leftarrow$ Induction hypothesis

Proof by Induction/Substitution (with rounding)

Claim. If T(n) satisfies the following recurrence, then $T(n) \leq n \lceil \log n \rceil$.

$\uparrow$ $\log_2 n$

$$T(n) \leq \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Pf. (continued)

- Step:

$$\begin{aligned} T(n) &\leq T(n_1) + T(n_2) + n \\ &\leq n_1 \lceil \log n_1 \rceil + n_2 \lceil \log n_2 \rceil + n \\ &\leq n_1 \lceil \log n_1 \rceil + n_2 \lceil \log n_1 \rceil + n \\ &= n \lceil \log n_1 \rceil + n \\ &\leq \\ &= n \lceil \log n \rceil \end{aligned}$$

Proof by Induction/Substitution (with rounding)

Claim. If T(n) satisfies the following recurrence, then T(n) $\leq n \lceil \log n \rceil$.

$\uparrow$ $\log_2 n$

$$T(n) \leq \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Pf. (continued)

- Step:

$$\begin{aligned} T(n) &\leq T(n_1) + T(n_2) + n \\ &\leq n_1 \lceil \log n_1 \rceil + n_2 \lceil \log n_2 \rceil + n \\ &\leq n_1 \lceil \log n_1 \rceil + n_2 \lceil \log n_1 \rceil + n \\ &= n \lceil \log n_1 \rceil + n \\ &\leq n (\lceil \log n \rceil - 1) + n \\ &= n \lceil \log n \rceil \end{aligned}$$

$$\begin{aligned} n_1 &= \lceil n/2 \rceil \\ &\leq \left\lceil 2^{\lceil \log n \rceil} / 2 \right\rceil \\ &= 2^{\lceil \log n \rceil} / 2 \\ \Rightarrow \; \log n_1 &\leq \lceil \log n \rceil - 1 \\ \Rightarrow \; \lceil \log n_1 \rceil &\leq \lceil \log n \rceil - 1 \end{aligned}$$

Because right side is an integer, rounding to nearest integer is OK.

Time Complexity Analysis of Merge Sort

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Q. How to prove that the run-time of merge sort is O(n log n)?

A. We have seen several methods:

- Recursion tree
- Substitution (by induction)

Time Complexity Analysis of Merge Sort

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Q. How to prove that the run-time of merge sort is O(n log n)?

A. We have seen several methods:

- Recursion tree
- Substitution (by induction)

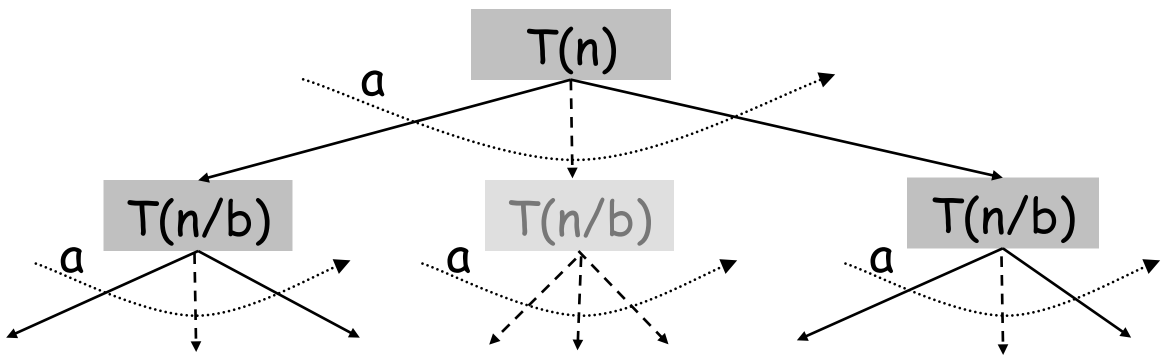
We don't like proofs. Can't you give us a *general rule* for the complexity of recursive functions?

General Recursion Tree

$$\text{for } a \geq 1,\ b > 1, \quad T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ \underbrace{aT(n/b)}_{\text{recursive calls}} + \underbrace{f(n)}_{\text{combining}} & \text{otherwise} \end{cases}$$

General Recursion Tree

$$\text{for } a \geq 1,\ b > 1, \quad \mathrm{T}(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ \underbrace{aT(n/b)}_{\text{recursive calls}} + \underbrace{f(n)}_{\text{combining}} & \text{otherwise} \end{cases}$$



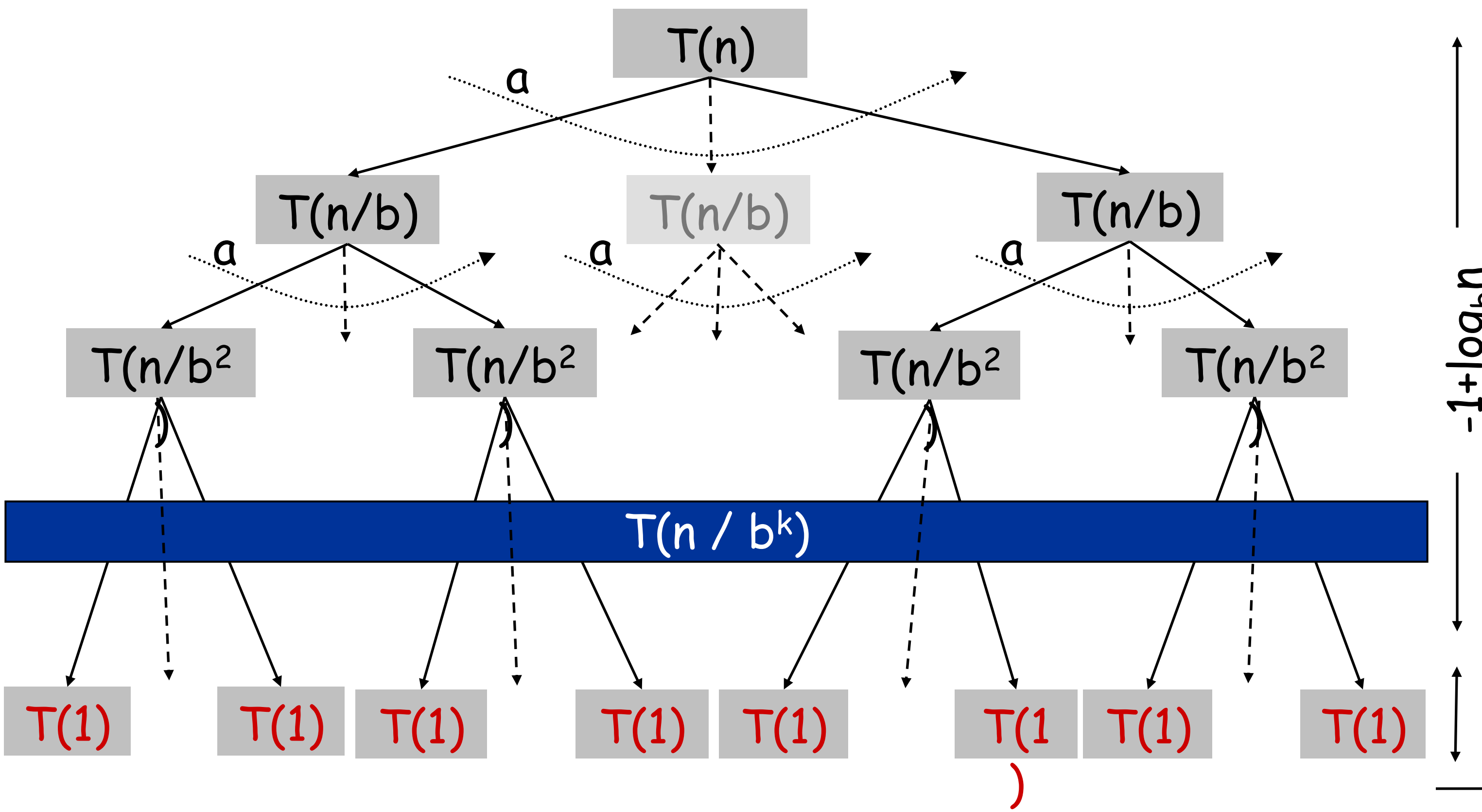
#nodes ·(time):

f(n)

a f(n/b)

General Recursion Tree

$$\text{for } a \ge 1,\ b > 1, \quad T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ \underbrace{aT(n/b)}_{\text{recursive calls}} + \underbrace{f(n)}_{\text{combining}} & \text{otherwise} \end{cases}$$



#nodes ·combining:

f(n)

a f(n/b)

a^2 f(n/b²)

. . .

a^k f(n / b^k)

. . .

$a^{\log_b n}$ f(1)

$$\Theta\left(n^{\log_b a}\right) + \sum_{k=0}^{-1+\log_b n} a^k f\left(\frac{n}{b^k}\right)$$

$a^{\log_b n} = n^{\log_b a}$ operations in the leaves

Master Method

$$\text{for } a \ge 1,\ b > 1, \quad T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ \underbrace{aT(n/b)}_{\text{recursive calls}} + \underbrace{f(n)}_{\text{combining}} & \text{otherwise} \end{cases}$$

So, $T(n) = \Theta\left(n^{\log_b a}\right) + \sum_{k=0}^{-1+\log_b n} a^k f\left(\frac{n}{b^k}\right)$ where

- first term is cost of all $n^{\log_b a}$ subproblems of size 1, and
- second term cost for combining in each level.

Three common cases:

- Running time dominated by cost at leaves
- Running time evenly distributed throughout the tree
- Running time dominated by cost at root

Consequently, to solve the recurrence, we need only to characterize the dominant term, $n^{\log_b a}$ or $f(n)$

TU Delft

Master Method

Given a recurrence of the form $T(n) = aT(n/b) + f(n)$

We can distinguish three common cases:

1. Running time dominated by cost at leaves: for an ε > 0
   if $f(n) = O\left(n^{\log_b a - \varepsilon}\right)$ then $T(n) = \Theta\left(n^{\log_b a}\right)$

2. Running time evenly distributed throughout the tree:
   if $f(n) = \Theta\left(n^{\log_b a}\right)$ then $T(n) = \Theta\left(n^{\log_b a} \log n\right)$

3. Running time dominated by cost at root: for an ε > 0
   if $f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right)$ then $T(n) = \Theta\left(f(n)\right)$

If f(n) satisfies regularity condition: a f(n/b) ≤ c f(n) for some c < 1 (polynomials always do)

The master method cannot solve every recurrence of this form.

TU Delft

Summary Master Method

- Extract *a*, *b*, and *f*(*n*) from a given recurrence
- Determine $n^{\log_b a}$
- Compare *f*(*n*) and $n^{\log_b a}$ asymptotically
- Determine appropriate Master Method case and apply:

1. Running time dominated by cost at leaves:
   if $f(n) = O\left(n^{\log_b a-\varepsilon}\right)$ then $T(n) = \Theta\left(n^{\log_b a}\right)$ for an ε > 0
2. Running time evenly distributed throughout the tree:
   if $f(n) = \Theta\left(n^{\log_b a}\right)$ then $T(n) = \Theta\left(n^{\log_b a}\log n\right)$
3. Running time dominated by cost at root:
   if $f(n) = \Omega\left(n^{\log_b a+\varepsilon}\right)$ then $T(n) = \Theta\left(f(n)\right)$ for an ε > 0

   Case 3: Only If f(n) satisfies regularity condition:
   a f(n/b) ≤ c f(n) for some c < 1
   (polynomials always do)

Examples

- Extract *a*, *b*, and *f*(*n*) from a given recurrence
- Determine $n^{\log_b a}$
- Compare *f*(*n*) and $n^{\log_b a}$ asymptotically
- Determine appropriate Master Method case, and apply

Example. Analyze Merge Sort using the Master Method:

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

$$a = 2,\ b = 2; \qquad n^{\log_b a} = n^{\log_2 2} = n$$

Q. What is dominant? (leaves, equal, root node)

Examples

- Extract *a*, *b*, and *f*(*n*) from a given recurrence
- Determine $n^{\log_b a}$
- Compare *f*(*n*) and $n^{\log_b a}$ asymptotically
- Determine appropriate Master Method case, and apply

Example. Analyze Merge Sort using the Master Method:

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

$$a = 2,\ b = 2; \qquad n^{\log_b a} = n^{\log_2 2} = n$$

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

this is case 2, $f(n) = \Theta\left(n^{\log_b a}\right)$, so

$$T(n) = \Theta\left(n^{\log_b a} \log n\right)$$

TU Delft

Examples

```
Binary-search(A, p, r, s):
   q←(p+r)/2
   if A[q]=s then return q
   else if A[q]>s then
        Binary-search(A, p, q-1, s)
   else Binary-search(A, q+1, r, s)
```

Q. Analyze complexity of binary search using the master method (1 min)

A. Analysis:

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

TU Delft

Examples

```
Binary-search(A, p, r, s):
    q←(p+r)/2
    if A[q]=s then return q
    else if A[q]>s then
            Binary-search(A, p, q-1, s)
    else Binary-search(A, q+1, r, s)
```

Q. Analyze complexity of binary search using the master method (1 min)

A. Analysis:

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

$$a = 1,\ b = 2; \qquad n^{\log_b a} = n^{\log_2 1} = n^0 = \Theta(1)$$

$$f(n) = \Theta(1)$$

This is case 2, $f(n) = \Theta\left(n^{\log_b a}\right)$, so:

$$T(n) = \Theta\left(n^{\log_b a} \log n\right) = \Theta(\log n)$$

Examples

Q. Use the master method to solve the following recurrence relation:

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

A. Analysis:

$$a = 9,\ b = 3; \qquad n^{\log_3 9} = \ldots$$

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

Q. What is dominant? (leaves, equal, root node)

Examples

Q. Use the master method to solve the following recurrence relation:

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

A. Analysis:

$$a = 9,\ b = 3; \qquad n^{\log_3 9} = n^2$$

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

This is case 1, $f(n) = O\left(n^{\log_b a - \varepsilon}\right)$, so:

$$T(n) = \Theta\left(n^{\log_b a}\right) = \Theta\left(n^2\right)$$

TU Delft

Examples

Q. Use the master method to solve the following recurrence relation:

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

A. Analysis:

$$a = 3,\ b = 4; \qquad n^{\log_4 3} = n^{0.793}$$

$$f(n) = \Theta(n \log n)$$

Q. What is dominant? (leaves, equal, root node)

TU Delft

Examples

Q. Use the master method to solve the following recurrence relation:

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

WARNING: is not a polynomial
Check regularity condition.

A. Analysis:

$$a = 3,\ b = 4; \qquad n^{\log_4 3} = n^{0.793}$$

$$f(n) = \Theta(n \log n)$$

This is case 3, $f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right)$, so:

$$T(n) = \Theta\left(f(n)\right) = \Theta\left(n \log n\right)$$

Check regularity condition:

$$af(n/b) = 3(n/4)\log(n/4) \le (3/4)n \log n = cf(n)$$

OK, for example for c=3/4 (and this c < 1)

Examples

Q. Use the master method to solve the following recurrence relation:

$$T(n) = 2T(n/2) + n\log n$$

A. Analysis:

$$a = 2,\ b = 2; \qquad n^{\log_2 2} = n$$

$$f(n) = \Theta(n\log n)$$

Q. What is dominant? (leaves, equal, root node)

Examples

Q. Use the master method to solve the following recurrence relation:

$$T(n) = 2T(n/2) + n\log n$$

I'll be back!

A. Analysis:

$$a = 2,\ b = 2; \qquad n^{\log_2 2} = n$$

$$f(n) = \Theta(n \log n)$$

This is not case 3, $f(n) = \Omega\left(n^{\log_b a+\varepsilon}\right)$, for c>0, nor one of the others!

Back to substitution method and induction proof (try $n \log^2 n$).

Because n^c for c>0 is $\Omega(\log n)$,
so n^{1+c} is $\Omega(n \log n)$
so n log n is not $\Omega(n^{1+c})$

Examples

Q. Use the master method to solve the following recurrence relation:

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

A. Analysis:

$$a = 4,\ b = 2; \qquad n^{\log_2 4} = \ldots$$

$$f(n) = \Theta(n^3)$$

Q. What is dominant? (leaves, equal, root node)

TU Delft

Examples

Q. Use the master method to solve the following recurrence relation:

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

A. Analysis:

$$a = 4,\ b = 2; \qquad n^{\log_2 4} = n^2$$

$$f(n) = \Theta(n^3)$$

This is case 3, $f(n) = \Omega\left(n^{\log_b a+\varepsilon}\right)$, so:

$$T(n) = \Theta\left(f(n)\right) = \Theta\left(n^3\right)$$

Check regularity condition:

$$af(n/b) = 4\left(n/2\right)^3 = (4/8)n^3 \le cf(n)$$

OK, for example for c=3/4 (and this c < 1)