Algorithms

Recursion II

Emanuele Rodolà
rodola@di.uniroma1.it

SAPIENZA
UNIVERSITÀ DI ROMA

2nd semester a.y. 2020/2021 · March 18, 2021

“Solving” recursion

For merge sort, we have encountered the expression:

$$T(n) = \begin{cases} \Theta(1) & n = 1 \\ 2T(\frac{n}{2}) + \Theta(n) & n > 1 \end{cases}$$

How to obtain asymptotic bounds on the solution?

"Solving" recursion

For merge sort, we have encountered the expression:

$$T(n) = \begin{cases} \Theta(1) & n = 1 \\ 2T(\frac{n}{2}) + \Theta(n) & n > 1 \end{cases}$$

How to obtain asymptotic bounds on the solution?

- Substitution method
- Recursion-tree method
- Master method

Substitution method

General idea:

1. Guess an expression for the solution.
2. Prove your guess by induction.

Good method when it is easy to guess.

Substitution method

General idea:

1. Guess an expression for the solution.
2. Prove your guess by induction.

Good method when it is easy to guess.

Example:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{and} \quad T(1) = 1$$

Substitution method

General idea:

1. Guess an expression for the solution.
2. Prove your guess by induction.

Good method when it is easy to guess.

Example:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{and} \quad T(1) = 1$$

Similar to something we have seen, so we guess $T(n) = O(n \lg n)$.

Substitution method

General idea:

1. Guess an expression for the solution.
2. Prove your guess by induction.

Good method when it is easy to guess.

Example:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{and} \quad T(1) = 1$$

Similar to something we have seen, so we guess $T(n) = O(n \lg n)$.

Since O measures upper bounds, we want to prove:

$$T(n) \leq cn \lg n \qquad \text{for } n \geq n_0$$

for some $c > 0$.

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{and} \quad T(1) = 1$$

Guess:

$$T(n) \le cn \lg n$$

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{and} \quad T(1) = 1$$

Guess:

$$T(n) \leq cn \lg n$$

Assume the guess holds for $T(\lfloor n/2 \rfloor)$:

$$T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor$$

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{and} \quad T(1) = 1$$

Guess:

$$T(n) \leq cn \lg n$$

Assume the guess holds for $T(\lfloor n/2 \rfloor)$:

$$T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor \lg\lfloor n/2 \rfloor$$

Then we substitute our guess into the recursion:

$$T(n) \leq 2(c\lfloor n/2 \rfloor \lg\lfloor n/2 \rfloor) + n$$

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{and} \quad T(1) = 1$$

Guess:

$$T(n) \leq cn \lg n$$

Assume the guess holds for $T(\lfloor n/2 \rfloor)$:

$$T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor \lg\lfloor n/2 \rfloor$$

Then we substitute our guess into the recursion:

$$\begin{aligned} T(n) &\leq 2(c\lfloor n/2 \rfloor \lg\lfloor n/2 \rfloor) + n \\ &\leq cn \lg(n/2) + n \end{aligned}$$

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{and} \quad T(1) = 1$$

Guess:

$$T(n) \leq cn \lg n$$

Assume the guess holds for $T(\lfloor n/2 \rfloor)$:

$$T(\lfloor n/2 \rfloor) \leq c \lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor$$

Then we substitute our guess into the recursion:

$$\begin{aligned} T(n) &\leq 2(c \lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor) + n \\ &\leq cn \lg(n/2) + n \\ &= cn \lg n - cn \lg 2 + n \end{aligned}$$

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{and} \quad T(1) = 1$$

Guess:

$$T(n) \leq cn \lg n$$

Assume the guess holds for $T(\lfloor n/2 \rfloor)$:

$$T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor \lg\lfloor n/2 \rfloor$$

Then we substitute our guess into the recursion:

$$\begin{aligned} T(n) &\leq 2(c\lfloor n/2 \rfloor \lg\lfloor n/2 \rfloor) + n \\ &\leq cn \lg(n/2) + n \\ &= cn \lg n - cn \lg 2 + n \\ &= cn \lg n - cn + n \end{aligned}$$

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{and} \quad T(1) = 1$$

Guess:

$$T(n) \leq cn \lg n$$

Assume the guess holds for $T(\lfloor n/2 \rfloor)$:

$$T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor$$

Then we substitute our guess into the recursion:

$$\begin{aligned} T(n) &\leq 2(c\lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor) + n \\ &\leq cn \lg(n/2) + n \\ &= cn \lg n - cn \lg 2 + n \\ &= cn \lg n - cn + n \\ &\leq cn \lg n \qquad\qquad \text{for } c \geq 1. \end{aligned}$$

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{and} \quad T(1) = 1$$

Guess:

$$T(n) \leq cn \lg n$$

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{and} \quad T(1) = 1$$

Guess:

$$T(n) \leq cn \lg n$$

We are not done yet! We only showed:

The guess holds for $\lfloor n/2 \rfloor$ $\Rightarrow$ The guess holds for n

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{and} \quad T(1) = 1$$

Guess:

$$T(n) \le cn \lg n$$

We are not done yet! We only showed:

The guess holds for $\lfloor n/2 \rfloor \Rightarrow$ The guess holds for n

We should also show:

The guess holds for $\lfloor n/4 \rfloor \Rightarrow$ The guess holds for $\lfloor n/2 \rfloor$

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{and} \quad T(1) = 1$$

Guess:

$$T(n) \leq cn \lg n$$

We are not done yet! We only showed:

The guess holds for $\lfloor n/2 \rfloor$ $\Rightarrow$ The guess holds for n

We should also show:

The guess holds for $\lfloor n/8 \rfloor$ $\Rightarrow$ The guess holds for $\lfloor n/4 \rfloor$

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{and} \quad T(1) = 1$$

Guess:

$$T(n) \leq cn \lg n$$

We are not done yet! We only showed:

The guess holds for $\lfloor n/2 \rfloor \Rightarrow$ The guess holds for n

We should also show:

The guess holds for $\lfloor n/16 \rfloor \Rightarrow$ The guess holds for $\lfloor n/8 \rfloor$

...and so on.

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{and} \quad T(1) = 1$$

Guess:

$$T(n) \leq cn \lg n$$

We are not done yet! We only showed:

The guess holds for $\lfloor n/2 \rfloor$ $\Rightarrow$ The guess holds for n

We should also show:

The guess holds for $\lfloor n/16 \rfloor$ $\Rightarrow$ The guess holds for $\lfloor n/8 \rfloor$

...and so on.

We must show that the base case is satisfied.

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{and} \quad T(1) = 1$$

We must show that the base case is satisfied.

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{and} \quad T(1) = 1$$

We must show that the base case is satisfied.

Let us check:

$$T(n) \le cn \lg n$$

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{and} \quad T(1) = 1$$

We must show that the base case is satisfied.

Let us check:

$$T(1) \leq c1 \lg 1$$

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{and} \quad T(1) = 1$$

We must show that the base case is satisfied.

Let us check:

$$1 \leq 0 \qquad \text{fail}$$

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{and} \quad T(1) = 1$$

We must show that the base case is satisfied.

If the base case was different, say $T(2) = 3$, we would get:

$$T(2) \leq c2 \lg 2$$

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{and} \quad T(1) = 1$$

We must show that the base case is satisfied.

If the base case was different, say $T(2) = 3$, we would get:

$$3 \leq c2$$

And we could easily find a $c > 0$ satisfying the inequality.

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{and} \quad T(1) = 1$$

We must show that the base case is satisfied.

If the base case was different, say $T(2) = 3$, we would get:

$$3 \leq c2$$

And we could easily find a $c > 0$ satisfying the inequality.

Can we then "replace" the base case with something else?

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{and} \quad T(1) = 1$$

We must show that the base case is satisfied.

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{and} \quad T(1) = 1$$

We must show that the base case is satisfied.

Recall that the guess $T(n) = O(n \lg n)$ means:

$$T(n) \le cn \lg n \qquad \text{for } n \ge n_0$$

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{and} \quad T(1) = 1$$

We must show that the base case is satisfied.

Recall that the guess $T(n) = O(n \lg n)$ means:

$$T(n) \leq cn \lg n \qquad \text{for } n \geq n_0$$

Which allows us to use a different base case for the inductive proof.

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \cancel{\text{and} \quad T(1) = 1}$$
$$\text{and} \quad T(n_0) = \cdots$$

We must show that the base case is satisfied.

Recall that the guess $O(n \lg n)$ means:

$$T(n) \leq cn \lg n \qquad \text{for } n \geq n_0$$

So, let us find a good value for n_0.

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \cancel{\text{and} \quad T(1) = 1}$$
$$\text{and} \quad T(n_0) = \cdots$$

We must show that the base case is satisfied.

Recall that the guess $O(n \lg n)$ means:

$$T(n) \leq cn \lg n \qquad \text{for } n \geq n_0$$

So, let us find a good value for n_0.

$$n_0 = 1 \quad \text{fail}$$

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \cancel{\text{and} \quad T(1) = 1}$$
$$\text{and} \quad T(2) = 4$$

We must show that the base case is satisfied.

Recall that the guess $O(n \lg n)$ means:

$$T(n) \leq cn \lg n \qquad \text{for } n \geq n_0$$

So, let us find a good value for n_0.

$$n_0 = 1 \quad \text{fail}$$
$$n_0 = 2$$

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \cancel{\text{and} \quad T(1) = 1}$$
$$\text{and} \quad T(2) = 4$$

We must show that the base case is satisfied.

Recall that the guess $O(n \lg n)$ means:

$$T(2) \leq c2 \lg 2 \qquad \text{for } n \geq 2$$

So, let us find a good value for n_0.

$$n_0 = 1 \quad \text{fail}$$
$$n_0 = 2$$

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \cancel{\text{and} \quad T(1) = 1}$$
$$\text{and} \quad T(2) = 4$$

We must show that the base case is satisfied.

Recall that the guess $O(n \lg n)$ means:

$$4 \leq c2 \qquad \text{for } n \geq 2$$

So, let us find a good value for n_0.

$$n_0 = 1 \quad \text{fail}$$
$$n_0 = 2 \quad \text{success}$$

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \cancel{\text{and} \quad T(1) = 1}$$
$$\text{and} \quad T(2) = 4$$

We must show that the base case is satisfied.

Recall that the guess $O(n \lg n)$ means:

$$4 \leq c2 \qquad \text{for } n \geq 2$$

So, let us find a good value for n_0.

$$n_0 = 1 \quad \text{fail}$$
$$n_0 = 2 \quad \text{success}$$

However, we can not compute $T(3)$ because we changed the base case.

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \cancel{\text{and} \quad T(1) = 1}$$
$$\text{and} \quad T(2) = 4 \quad \text{and} \quad T(3) = 5$$

We must show that the base case is satisfied.

Recall that the guess $O(n \lg n)$ means:

$$T(3) \leq c3 \lg 3 \qquad \text{for } n \geq 3$$

So, let us find a good value for n_0.

$$n_0 = 1 \quad \text{fail}$$
$$n_0 = 2 \quad \text{success}$$
$$n_0 = 3$$

Substitution method

To solve:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \cancel{\text{and} \quad T(1) = 1}$$
$$\text{and} \quad T(2) = 4 \quad \text{and} \quad T(3) = 5$$

We must show that the base case is satisfied.

Recall that the guess $O(n \lg n)$ means:

$$5 \leq c3 \lg 3 \qquad \text{for } n \geq 3$$

So, let us find a good value for n_0.

$$\begin{aligned} n_0 &= 1 \quad \text{fail} \\ n_0 &= 2 \quad \text{success} \\ n_0 &= 3 \quad \text{success} \end{aligned}$$

Exercise

For the recursion:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \text{and} \quad T(1) = 1$$

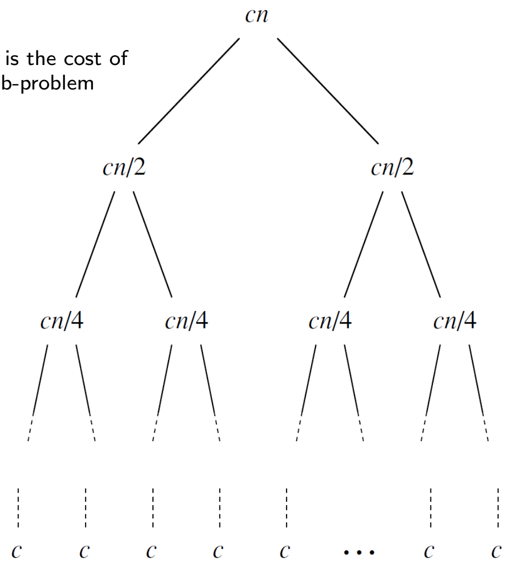
Prove the "loose" worst-case complexity:

$$T(n) = O(n^2)$$

Use the substitution method for your proof.

Recursion-tree method

each node is the cost of
a single sub-problem



Recursion-tree method

Can be used to generate good guesses for the substitution method.

Recursion-tree method

Can be used to generate good guesses for the substitution method.

Example:

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$$

We are interested in an upper bound for the cost.

Recursion-tree method

Can be used to generate good guesses for the substitution method.

Example:

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$$

We are interested in an upper bound for the cost.

Therefore we consider the recursion:

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

Recursion-tree method

Can be used to generate good guesses for the substitution method.

Example:

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$$

We are interested in an upper bound for the cost.

Therefore we consider the recursion:

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

We can also assume that $n = 4^m$ for some m.

For merge sort, we also assumed that $n = 2^m$ for some m.

Recursion-tree method

Can be used to generate good guesses for the substitution method.

Example:

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$$

We are interested in an upper bound for the cost.

Therefore we consider the recursion:

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

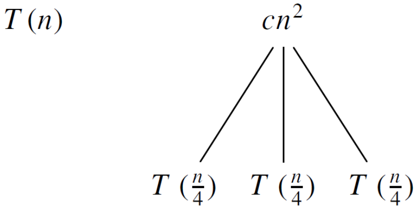
We can also assume that $n = 4^m$ for some m.

For merge sort, we also assumed that $n = 2^m$ for some m.

Further, we assume the base case is $T(1)$.

Recursion-tree method

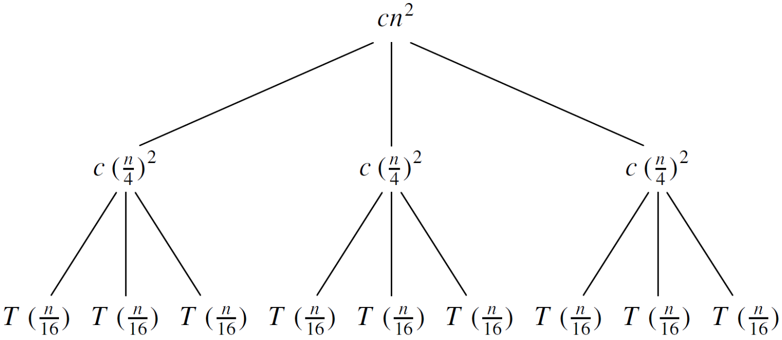
$$T(n) = 3T(n/4) + cn^2$$



(a) (b)

Recursion-tree method

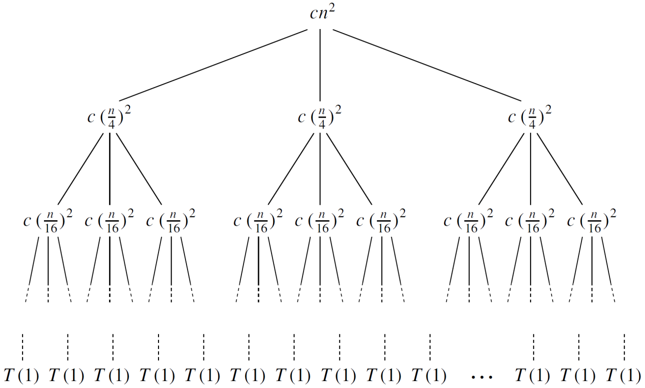
$$T(n) = 3T(n/4) + cn^2$$



(c)

Recursion-tree method

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

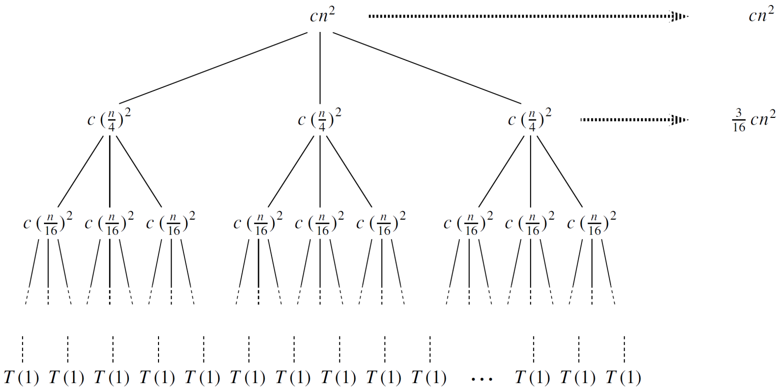


(d)

Recursion-tree method

each level has $3\times$ more nodes than the level above

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

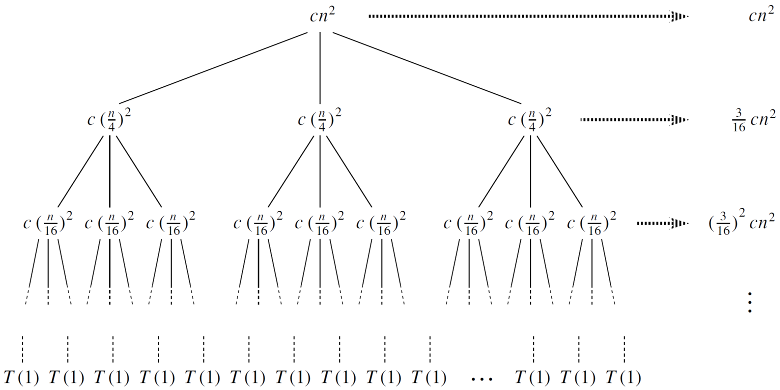


(d)

Recursion-tree method

each level has $3\times$ more nodes than the level above

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

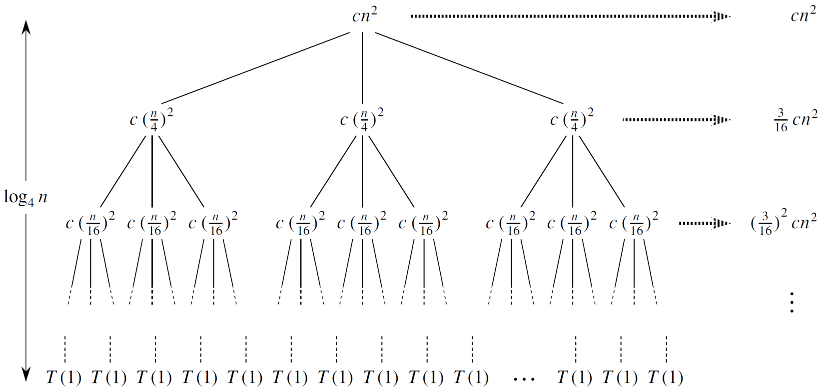


(d)

Recursion-tree method

each level has $3\times$ more nodes than the level above

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

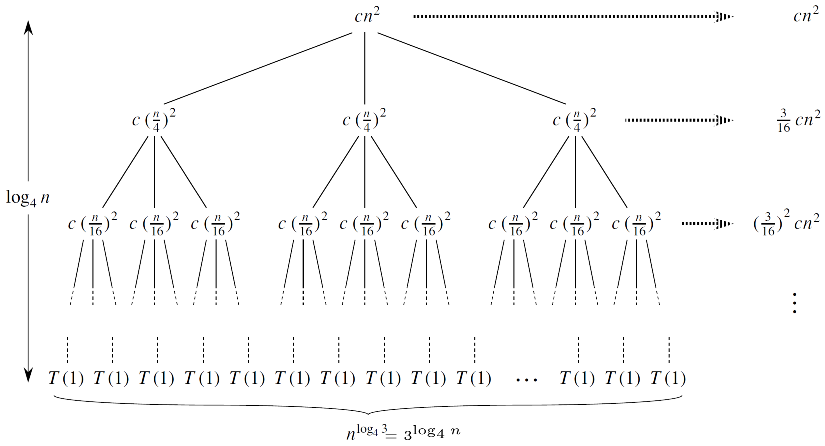


(d)

Recursion-tree method

each level has $3\times$ more nodes than the level above

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

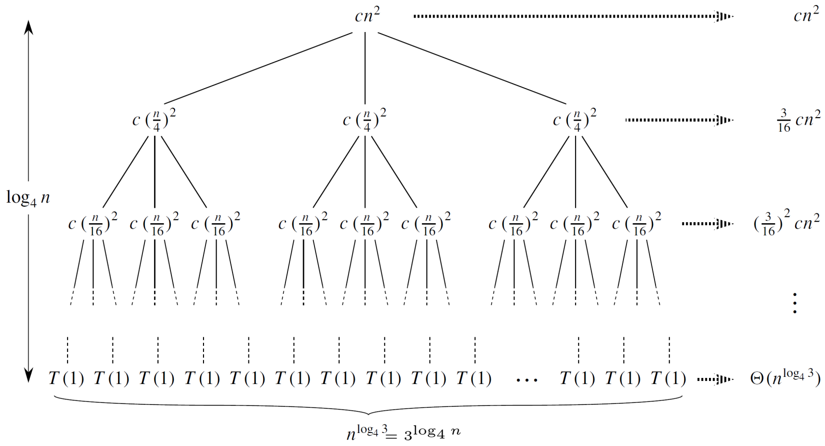


(d)

Recursion-tree method

each level has $3\times$ more nodes than the level above

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

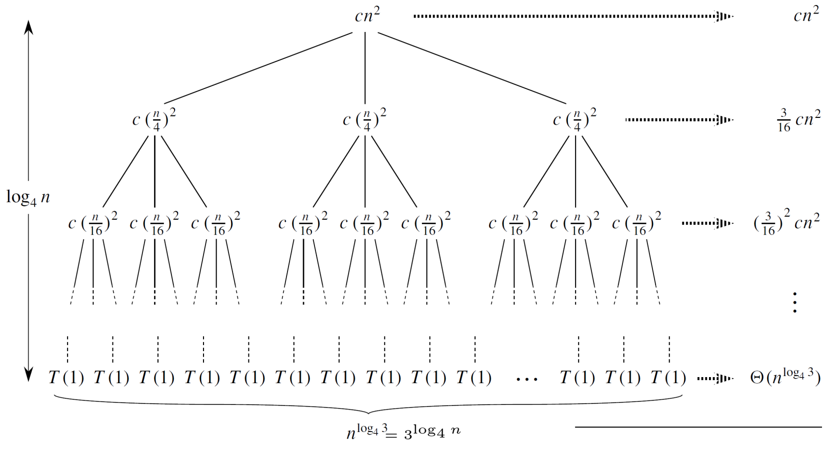


(d)

Recursion-tree method

each level has $3\times$ more nodes than the level above

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



(d)

Recursion-tree method

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$$

Use the substitution method with the guess $T(n) = O(n^2)$, that is:

$$T(n) \leq dn^2 \quad \text{for some } d > 0$$

Recursion-tree method

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$$

Use the substitution method with the guess $T(n) = O(n^2)$, that is:

$$T(n) \leq dn^2 \quad \text{for some } d > 0$$

Substitute the guess in the recursion to get:

$$T(n) \leq 3d\lfloor n/4 \rfloor^2 + cn^2$$

Recursion-tree method

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$$

Use the substitution method with the guess $T(n) = O(n^2)$, that is:

$$T(n) \leq dn^2 \quad \text{for some } d > 0$$

Substitute the guess in the recursion to get:

$$\begin{aligned} T(n) &\leq 3d\lfloor n/4 \rfloor^2 + cn^2 \\ &\leq 3d(n/4)^2 + cn^2 \end{aligned}$$

Recursion-tree method

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$$

Use the substitution method with the guess $T(n) = O(n^2)$, that is:

$$T(n) \le dn^2 \quad \text{for some } d > 0$$

Substitute the guess in the recursion to get:

$$\begin{aligned} T(n) &\le 3d\lfloor n/4 \rfloor^2 + cn^2 \\ &\le 3d(n/4)^2 + cn^2 \\ &= \frac{3}{16}dn^2 + cn^2 \end{aligned}$$

Recursion-tree method

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$$

Use the substitution method with the guess $T(n) = O(n^2)$, that is:

$$T(n) \leq dn^2 \quad \text{for some } d > 0$$

Substitute the guess in the recursion to get:

$$\begin{aligned}
T(n) &\leq 3d\lfloor n/4 \rfloor^2 + cn^2 \\
&\leq 3d(n/4)^2 + cn^2 \\
&= \frac{3}{16}dn^2 + cn^2 \\
&\leq dn^2 \qquad\qquad \text{for } d \geq \frac{16}{13}c
\end{aligned}$$

Master method

Applies to recursion of the form:

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

with $a \geq 1, b > 1$.

We always assume $f(n)$ to be asymptotically positive.

Master method

Applies to recursion of the form:

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

with $a \geq 1, b > 1$.

We always assume $f(n)$ to be asymptotically positive.

- We have a sub-problems.

Master method

Applies to recursion of the form:

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

with $a \geq 1, b > 1$.

We always assume $f(n)$ to be asymptotically positive.

- We have a sub-problems.
- Each has size n/b.

Master method

Applies to recursion of the form:

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

with $a \geq 1, b > 1$.

We always assume $f(n)$ to be asymptotically positive.

- We have a sub-problems.
- Each has size n/b.
- The cost of dividing + combining is $f(n)$ (i.e. the root of the tree).

Master method

Applies to recursion of the form:

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

with $a \geq 1, b > 1$.

We always assume $f(n)$ to be asymptotically positive.

- We have a sub-problems.
- Each has size n/b.
- The cost of dividing + combining is $f(n)$ (i.e. the root of the tree).
- By n/b we mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$.

Master method

We use the master theorem, which can be applied in three cases:

Master method

We use the master theorem, which can be applied in three cases:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.

Master method

We use the master theorem, which can be applied in three cases:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.

Master method

We use the master theorem, which can be applied in three cases:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some $c < 1$ and large n, then $T(n) = \Theta(f(n))$.

Master method

We use the master theorem, which can be applied in three cases:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some $c < 1$ and large n, then $T(n) = \Theta(f(n))$.

Intuitively: compare $f(n)$ with $n^{\log_b a}$ and see who's bigger.

Master method

We use the master theorem, which can be applied in three cases:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some $c < 1$ and large n, then $T(n) = \Theta(f(n))$.

Intuitively: compare $f(n)$ with $n^{\log_b a}$ and see who's bigger.

We can read the theorem as follows:

1. The total cost is dominated by the base cases.
   $f(n)$ must be polynomially smaller than $n^{\log_b a}$ (i.e. by a factor n^{ϵ}).

Master method

We use the master theorem, which can be applied in three cases:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some $c < 1$ and large n, then $T(n) = \Theta(f(n))$.

Intuitively: compare $f(n)$ with $n^{\log_b a}$ and see who's bigger.

We can read the theorem as follows:

1. The total cost is dominated by the base cases.
   $f(n)$ must be polynomially smaller than $n^{\log_b a}$ (i.e. by a factor n^{ϵ}).
2. The total cost is distributed across the levels of the recursion tree.

Master method

We use the master theorem, which can be applied in three cases:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some $c < 1$ and large n, then $T(n) = \Theta(f(n))$.

Intuitively: compare $f(n)$ with $n^{\log_b a}$ and see who's bigger.

We can read the theorem as follows:

1. The total cost is dominated by the base cases.
   $f(n)$ must be polynomially smaller than $n^{\log_b a}$ (i.e. by a factor n^{ϵ}).
2. The total cost is distributed across the levels of the recursion tree.
3. The total cost is dominated by the root.
   $f(n)$ must be polynomially larger than $n^{\log_b a}$ and be regular.

Master method: Examples

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

We get $n^{\log_b a} = \Theta(n^2)$ and $f(n) = O(n^{\log_3 9 - \epsilon})$ with $\epsilon = 1$. Thus, we are in case (1) and the solution is $T(n) = \Theta(n^2)$.

Master method: Examples

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

We get $n^{\log_b a} = \Theta(n^2)$ and $f(n) = O(n^{\log_3 9 - \epsilon})$ with $\epsilon = 1$. Thus, we are in case (1) and the solution is $T(n) = \Theta(n^2)$.

$$T(n) = T(2n/3) + 1$$

Here $n^{\log_b a} = 1$ and case (2) applies, hence $T(n) = \Theta(\lg n)$.

Master method: Examples

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

We get $n^{\log_b a} = \Theta(n^2)$ and $f(n) = O(n^{\log_3 9-\epsilon})$ with $\epsilon = 1$. Thus, we are in case (1) and the solution is $T(n) = \Theta(n^2)$.

$$T(n) = T(2n/3) + 1$$

Here $n^{\log_b a} = 1$ and case (2) applies, hence $T(n) = \Theta(\lg n)$.

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

Here $n^{\log_b a} = O(n^{0.793})$ and $f(n) = \Omega(n^{\log_4 3+\epsilon})$. Further, we can show that the regularity condition holds for $f(n)$. Case (3) applies, hence $T(n) = \Theta(n \lg n)$.

Master method: Examples

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

We get $n^{\log_b a} = \Theta(n^2)$ and $f(n) = O(n^{\log_3 9 - \epsilon})$ with $\epsilon = 1$. Thus, we are in case (1) and the solution is $T(n) = \Theta(n^2)$.

$$T(n) = T(2n/3) + 1$$

Here $n^{\log_b a} = 1$ and case (2) applies, hence $T(n) = \Theta(\lg n)$.

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

Here $n^{\log_b a} = O(n^{0.793})$ and $f(n) = \Omega(n^{\log_4 3 + \epsilon})$. Further, we can show that the regularity condition holds for $f(n)$. Case (3) applies, hence $T(n) = \Theta(n \lg n)$.

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

The master theorem does not apply here, since $f(n) = n \lg n$ is not polynomially larger than $n^{\log_b a} = n$. In fact, $\frac{f(n)}{n^{\log_b a}} = \frac{n \lg n}{n} = \lg n < n^{\epsilon}$ asymptotically.

Suggested reading

Chapters 4.1, 4.2, 4.3 of:

"Introduction to Algorithms – 2nd Ed.", Cormen et al.