Design and Analysis of Algorithms

Supplemental Slides

Mingyu XIAO

School of Computer Science and Engineering
University of Electronic Science and Technology of China

Recurrences

Introduction

The branch-and-search approach (the divide-and-conquer approach) is a very popular technique in the development of exact algorithms and heuristic algorithms for difficult computational problems.

This technique always leads to recurrence relations.

Example

Mergesort:
Divide the problem of n elements into two subproblems of n/2 elements.
Get the recurrence relation:

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

Recurrences

Recurrence: an equation or inequality that describe a function in terms of its value on smaller inputs.

Some forms of recurrence:

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

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

$$T(n) = T(n-a) + T(n-b) + f(n)$$

The equals signs in these equations can be replaced by less than or equals signs (≤).

(≥ meaningless)

How to solve recurrences?

Some methods:

- Substitution method
- Recursion-tree method
- Master method

Others (Refer to books of special topical on
Recursion Theory)

Technicalities

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

Assume that integer arguments to functions:
  n may not be an integer.
Omit floor, ceiling functions:
  n/2 may not be an integer.
Omit boundary conditions:

$$T(1) = \Theta(1)$$

$$T(n) = \Theta(1) \quad \textit{for sufficient small n.}$$

I. Substitution Method

Substitution Method:

1. Guess the form of the solution.
2. Verify by mathematical induction and solve for constants.

Example of substitution method

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

Guess that $T(n)=O(n \lg n)$.
To prove that $T(n) \le cn\lg n$ for some constant $c>0$.

Assume $T(k) \le ck \lg k$ for $k<n$ and
Prove $T(n) \le cn \lg n$ by induction

(important: same constant c!)

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

as long as $c \ge 1$

Making a good guess

From a recurrence which is similar to one you have seen before.

like

$T(n)=2T(n/2+10)+n$

Prove loose lower and upper bounds. Then reduce the gap between them, like

$T(n)=\Omega(n)$ and $T(n)=O(n^2)$

guess $T(n)=\Theta(n\lg n)$

Fallacious Argument

Let's look at another example :
$T(n)=4T(n/2)+n$

Guess $T(n)=O(n^2)$
Prove it by mathematical induction.

Assume $T(k) \leq ck^2$ for all $k<n$.
By induction,

$$\begin{aligned} T(n) &= 4T(n/2)+n \\ &\leq 4c(n/2)^2+n \\ &\leq cn^2+n \\ &= O(n^2) \end{aligned}$$

What goes wrong?

WRONG!

Fallacious Argument

We must prove the exact form of the inductive hypothesis, that is $T(n) \le cn^2$

$T(n) \le cn^2+n$

$\quad !\le cn^2$ for any choice of $c>0$

$!= O(n^2)$

Correct proof

Idea: strengthen our inductive hypothesis by subtracting some lower-order terms.

Assume $T(k) \le c_1 k^2 - c_2 k$ for $k<n$

To prove $T(n) \le c_1 n^2 - c_2 n$

By induction

$$
\begin{aligned}
T(n) &= 4T(n/2)+n \\
&\le 4[c_1(n/2)^2 - c_2(n/2)]+n \\
&= c_1 n^2 - c_2 n - (c_2 n - n) \\
&\le c_1 n^2 - c_2 n \qquad \text{if } c_2 \ge 1
\end{aligned}
$$

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

Changing variables

$T(n)=2T(n^{1/2})+\lg n$

Simplify the recurrence by letting $m=\lg n$

$n=2^m$

we get $T(2^m)=2T(2^{m/2})+m$

Rename $S(m)= T(2^m)$

we get $S(m)=2S(m/2)+m$

$S(m)=O(m\lg m)$

$=O(\lg n\lg\lg n)$

II. Iteration Method

Two ways (algebraic and geometrical)
Direct expand (algebraic)
Recursion trees (geometrical)

Idea: Model the execution of the algorithm to compute the running time

Direct Expand

Idea: expand the recurrence and express it as a summation of terms dependent only on n and the initial conditions.

Direct Expand

$$
\begin{aligned}
T(n) &= n+3T(n/4) \\
&= n+3(n/4+ 3T(n/16)) \\
&= n+3(n/4+n+3(n/16+ 3T(n/64)) \\
&= n+3n/4+9n/16+27T(n/64)
\end{aligned}
$$

$$
\begin{aligned}
T(n) &\le n+3n/4+9n/16+\cdots+3^{\log_4 n}\Theta(1) \\
&\le n\sum_{i=0}^{\infty}\left(\frac{3}{4}\right)^i + \Theta(n^{\log_4 3}) \\
&= 4n+o(n) = O(n)
\end{aligned}
$$

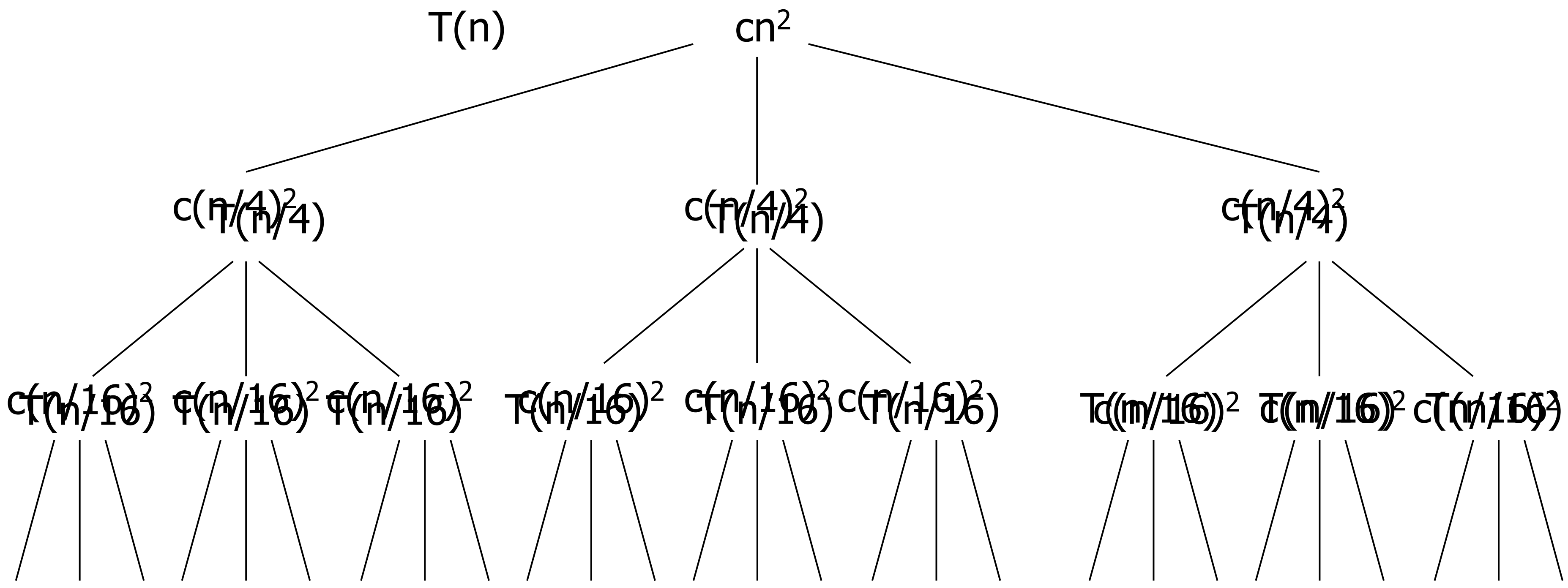
We can add the floor signs in the functions.

Recursion tree

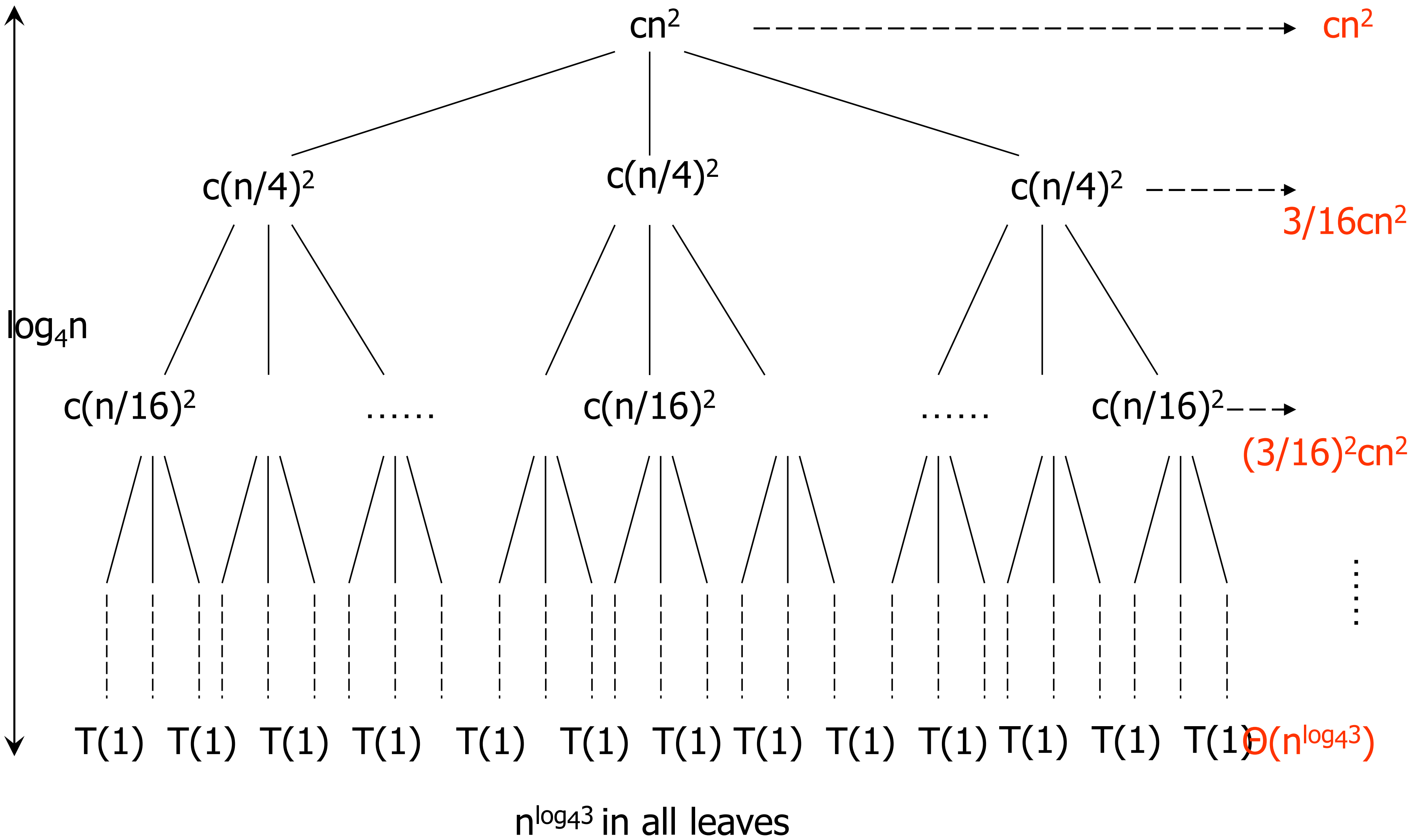
Draw the recursion tree
- T() denotes the problem, and Θ() denotes the time of separation of the problem.

Example

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



Construction of a recursion tree



Calculation by using the recursion tree

The kernel of subproblems at level k is $n/4^k$
each node at level k costs $c(n/4^k)^2$.
3^k nodes at level k
each level contributes $3^k * c(n/4^k)^2 = (3/16)^k cn^2$
The tree has $\log_4 n+1$ levels
There are $3^{\log_4 n} = n^{\log_4 3}$ leaves, each contributing $\Theta(1)$
We can simply verify it by substitution method.

$$T(n) = \sum_{i=0}^{\log_4 n - 1} (3/16)^i cn^2 + \theta\left(n^{\log_4 3}\right)$$

$$< \sum_{i=0}^{\infty} (3/16)^i cn^2 + \theta\left(n^{\log_4 3}\right)$$

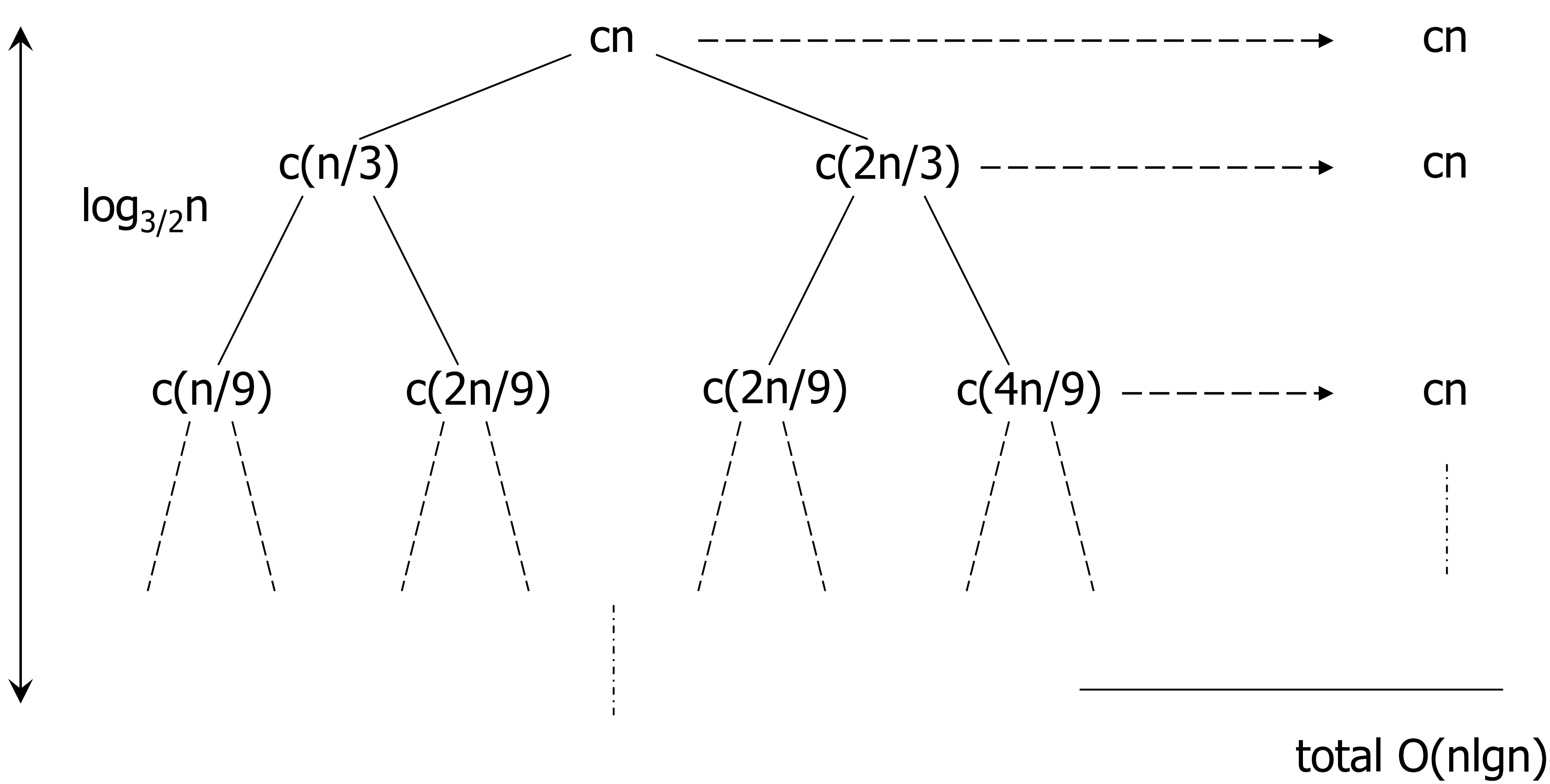
$$= \frac{1}{1\text{-}3/16} cn^2 + \theta\left(n^{\log_4 3}\right)$$

$$= \frac{16}{13} cn^2 + \theta\left(n^{\log_4 3}\right)$$

$$= O\left(n^2\right)$$

A more intricate example

Let's look at another example $T(n)=T(n/3)+T(2n/3)+O(n)$



Calculation by using the recursion tree

The height of tree h: $n(2/3)^h=1 \rightarrow h=\log_{3/2}n$

each level contributing cn

$T(n)=O(cn*\log_{3/2}n)=O(n\lg n)$

Is it right?

We haven't include the cost of leaves!

The number of leaves is at most $2^{\log_{3/2}n}$, each contributing$\Theta(1)$.

So the total cost of leaves is $\Theta(2^{\log_{3/2}n})=\Theta(n^{\log_{3/2}2})=\omega(n\lg n)$.

However, this is not a complete binary tree, so less than $2^{\log_{3/2}n}$ leaves!

III. Master Method

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

Case 1:

If $f(n) = O(n^{\log_b a-\varepsilon})$ for some constant $\varepsilon > 0$, then T(n)=$O(n^{\log_b a})$

Case 2:

If $f(n) = \Theta(n^{\log_b a})$, then T(n)=$\Theta(n^{\log_b a} \lg n)$

Case 3:

If $f(n) = \Omega(n^{\log_b a+\varepsilon})$ and $af(n/b) \le cf(n)$ for c<1 and sufficently large n,

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

Examples

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

For this recurrence, a=9, b=3, f(n)=n.

$$n^{\log_b a} = n^{\log_3 9} = \Theta(n^2),\ \ f(n) = O(n^{\log_3 9-\varepsilon}),\ \ \varepsilon = 1.$$

Apply case 1 of the master thm. T(n)=Θ(n^2)

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

a=1, b=3/2, f(n)=1.

$$n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1,\ \ f(n) = \Theta(n^{\log_b a}) = \Theta(1)$$

Case 2, T(n)= Θ(lgn).

T(n)=2T(n/2)+nlgn

cannot use this theorem!

Comments

The three cases do not cover all the possibilities for f(n).

f(n) is smaller than $n^{\log b(a)}$ but not polynomially smaller.

f(n) is larger than $n^{\log b(a)}$ but not polynomially larger.