BITS Pilani
Pilani Campus

Data Structures & Algorithms Design- SS ZG519
Lecture - 4

Dr. Padma Murali

Lecture 4 Topics

- Analysis of Algorithms -- space and time complexity
- Recurrences
- Binary Search
- Solving recurrences

- Slides source: 2008 Pearson Education, Inc. Publishing as Pearson Addison-Wesley
- Lecture notes

Orders of growth of some important functions

- All logarithmic functions $\log_a n$ belong to the same class
- $\Theta(\log n)$ no matter what the logarithm's base $a > 1$ is
- All polynomials of the same degree k belong to the same class:
- $a_k n^k + a_{k-1} n^{k-1} + \ldots + a_0 \in \Theta(n^k)$
- Exponential functions a^n have different orders of growth for different a's
- order $\log n$ < order n^α (α>0) < order a^n < order $n!$ < order n^n

Basic asymptotic efficiency classes

1	constant
$\log n$	logarithmic
n	linear
$n \log n$	n-log-n
n^2	quadratic
n^3	cubic
2^n	exponential
$n!$	factorial

Time efficiency of nonrecursive algorithms

General Plan for Analysis

- Decide on parameter *n* indicating ***input size***
- Identify algorithm's ***basic operation***
- Determine ***worst***, ***average***, and ***best*** cases for input of size *n*
- Set up a sum for the number of times the basic operation is executed
- Simplify the sum using standard formulas and rules (see Appendix A)

Useful summation formulas and rules

$\Sigma_{l \le i \le u} 1 = 1+1+\ldots+1 = u - l + 1$

In particular, $\Sigma_{l \le i \le u} 1 = n - 1 + 1 = n \in \Theta(n)$

$\Sigma_{1 \le i \le n}\, i = 1+2+\ldots+n = n(n+1)/2 \approx n^2/2 \in \Theta(n^2)$

$\Sigma_{1 \le i \le n}\, i^2 = 1^2+2^2+\ldots+n^2 = n(n+1)(2n+1)/6 \approx n^3/3 \in \Theta(n^3)$

$\Sigma_{0 \le i \le n}\, a^i = 1 + a + \ldots + a^n = (a^{n+1} - 1)/(a - 1)$ for any $a \neq 1$

In particular, $\Sigma_{0 \le i \le n}\, 2^i = 2^0 + 2^1 + \ldots + 2^n = 2^{n+1} - 1 \in \Theta(2^n)$

$\Sigma(a_i \pm b_i) = \Sigma a_i \pm \Sigma b_i \quad \Sigma c a_i = c \Sigma a_i \quad \Sigma_{l \le i \le u} a_i = \Sigma_{l \le i \le m} a_i + \Sigma_{m+1 \le i \le u} a_i$

Plan for Analysis of Recursive Algorithms

- Decide on a parameter indicating an input's size.
- Identify the algorithm's basic operation.
- Check whether the number of times the basic op. is executed may vary on different inputs of the same size. (If it may, the worst, average, and best cases must be investigated separately.)
- Set up a recurrence relation with an appropriate initial condition expressing the number of times the basic op. is executed.
- Solve the recurrence (or, at the very least, establish its solution's order of growth) by backward substitutions or another method.

Example 1: Recursive evaluation of n!

Definition: $n ! = 1 * 2 * \ldots *(n-1) * n$ for $n \geq 1$ and $0! = 1$

Recursive definition of n!: $F(n) = F(n-1) * n$ for $n \geq 1$ and
$F(0) = 1$

ALGORITHM $F(n)$

//Computes n! recursively
//Input: A nonnegative integer n
//Output: The value of n!
if $n = 0$ **return** 1
else return $F(n-1) * n$

Size:
Basic operation:
Recurrence relation:

Solving the recurrence for M(*n*)

$M(n) = M(n-1) + 1, \quad M(0) = 0$

Fibonacci numbers

The Fibonacci numbers:

0, 1, 1, 2, 3, 5, 8, 13, 21, …

The Fibonacci recurrence:

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

$F(0) = 0$

$F(1) = 1$

General 2nd order linear homogeneous recurrence with constant coefficients:

$$aX(n) + bX(n-1) + cX(n-2) = 0$$

Solving $aX(n) + bX(n-1) + cX(n-2) = 0$

- Set up the characteristic equation (quadratic)

$$ar^2 + br + c = 0$$

- Solve to obtain roots r_1 and r_2

- General solution to the recurrence

  if r_1 and r_2 are two distinct real roots: $X(n) = \alpha r_1^{\,n} + \beta r_2^{\,n}$

  if $r_1 = r_2 = r$ are two equal real roots: $X(n) = \alpha r^n + \beta n r^n$

- Particular solution can be found by using initial conditions

Application to the Fibonacci numbers

$F(n) = F(n-1) + F(n-2)$ or $F(n) - F(n-1) - F(n-2) = 0$

Characteristic equation:

Roots of the characteristic equation:

General solution to the recurrence:

Particular solution for $F(0) = 0$, $F(1)=1$:

Computing Fibonacci numbers

1. Definition-based recursive algorithm

2. Nonrecursive definition-based algorithm
3. Explicit formula algorithm

$$\begin{pmatrix} F(n-1) & F(n) \\ F(n) & F(n+1) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n$$

4. Logarithmic algorithm based on formula:

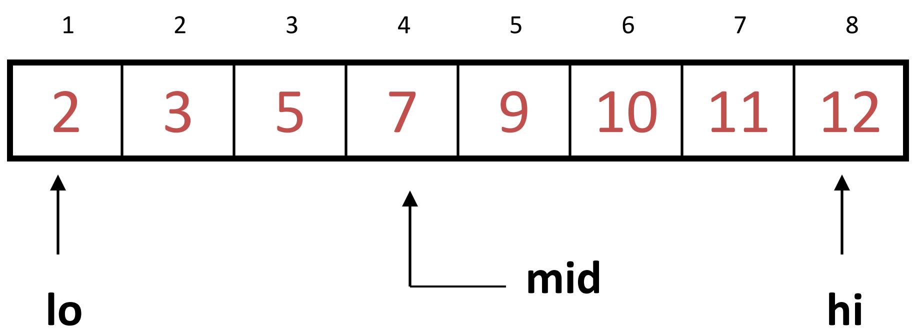
for $n \geq 1$, assuming an efficient way of computing matrix powers.

Recurrent Algorithms

BINARY – SEARCH

- for an ordered array A, finds if **x** is in the array A[lo…hi]

Alg.: BINARY-SEARCH (A, lo, hi, x)



if (lo > hi)
    return FALSE
mid ← $\lfloor$(lo+hi)/2$\rfloor$
if x = A[mid]
    return TRUE
if (x < A[mid])
    BINARY-SEARCH (A, lo, mid-1, x)
if (x > A[mid])
    BINARY-SEARCH (A, mid+1, hi, x)

Example

A[8] = {1, 2, 3, 4, 5, 7, 9, 11}

– lo = 1 hi = 8 x = 7

1	2	3	4	5	6	7	8
1	2	3	4	5	7	9	11

mid = 4, lo = 5, hi = 8

1	2	3	4	5	7	9	11

mid = 6, A[mid] = x Found!

Example

A[8] = {1, 2, 3, 4, 5, 7, 9, 11}

– lo = 1 hi = 8 x = 6

1	2	3	4	5	6	7	8	
1	2	3	4	5	7	9	11	mid = 4, lo = 5, hi = 8

1	2	3	4	5	7	9	11	mid = 6, A[6] = 7, lo = 5, hi = 5

1	2	3	4	5	7	9	11	mid = 5, A[5] = 5, lo = 6, hi = 5 NOT FOUND!

Analysis of BINARY-SEARCH

Alg.: BINARY-SEARCH (A, lo, hi, x)

```
if (lo > hi)
    return FALSE
mid ← ⌊(lo+hi)/2⌋
if x = A[mid]
    return TRUE
if ( x < A[mid] )
    BINARY-SEARCH (A, lo, mid-1, x)
if ( x > A[mid] )
    BINARY-SEARCH (A, mid+1, hi, x)
```

constant time: c_1

constant time: c_2

constant time: c_3

same problem of size n/2

same problem of size n/2

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

– $T(n)$ – running time for an array of size n

Recurrences and Running Time

- Recurrences arise when an algorithm contains recursive calls to itself
- What is the actual running time of the algorithm?
- Need to solve the recurrence
  - Find an explicit formula of the expression (the generic term of the sequence)

Example Recurrences

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

Recursive algorithm that loops through the input to eliminate one item

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

Recursive algorithm that halves the input in one step

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

Recursive algorithm that halves the input but must examine every item in the input

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

Recursive algorithm that splits the input into 2 halves and does a constant amount of other work

Methods for Solving Recurrences

Iteration method

Substitution method

Recursion tree method

Master method

Iteration Method – Example

innovate achieve lead

Assume: $n = 2^k$

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

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

$$
\begin{aligned}
T(n) &= n + 2T(n/2) \\
&= n + 2(n/2 + 2T(n/4)) \\
&= n + n + 4T(n/4) \\
&= n + n + 4(n/4 + 2T(n/8)) \\
&= n + n + n + 8T(n/8) \\
\ldots\ &= in + 2^i T(n/2^i) \\
&= kn + 2^k T(1) \\
&= n\lg n + nT(1) = \Theta(n\lg n)
\end{aligned}
$$

Iteration Method

SOLVE :

1. $T(n) = T(n-1) + n$ $\Theta(n^2)$

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

3. $T(n) = 2T(n/2) + 1$ $\Theta(n)$

The substitution method

1. Guess a solution

2. Use induction to prove that the solution works

Substitution method

Guess a solution

- $T(n) = O(g(n))$
- Induction goal: apply the definition of the asymptotic notation
    - $T(n) \le d\, g(n)$, for some $d > 0$ and $n \ge n_0$
- Induction hypothesis: $T(k) \le d\, g(k)$ for all $k < n$

Prove the induction goal

- Use the **induction hypothesis** to find some values of the constants d and n_0 for which the **induction goal** holds

Recurrences

When an algorithm contains a recursive call to itself, its' running time can be described by a recurrence equation or recurrence.

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

Three methods for solving recurrences. -(i.e) for obtaining asymptotic "Θ" or "O" bounds on the solution.

(1) Substitution method

→ Guess a solution

→ use mathematical induction to prove our guess is correct.

(2) Recursion tree method

→ converts recurrences into a tree whose nodes represent the costs incurred at various levels of the recursion.

→ use techniques for bounding summations to solve the recurrences.

(3) Master method

→ provide bounds for recurrences of the form
$T(n) = a\,T(n/b) + f(n)$
where $a \geq 1$, $b > 1$ & $f(n)$ is a given function.

Assumptions

→ we assume that the input n is always an integer.
→ we ignore boundary condition.
→ while stating & solving recurrences, floors, ceilings are omitted.
... case running

floor, ...

For example: The worst case running time of Merge sort is given by

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases} \quad \text{— (1)}$$

The solution of the above is claimed to be $\Theta(n \log n)$.

When $n > 1$, we have

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) \quad \text{— (2)}$$

But, from our assumption, (1) & (2) is usually written as

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

since although changing the value of $T(1)$ changes the solution to the recurrence, the solution typically doesn't change by more than a constant factor, so the order of growth is unchanged.

Substitution method

(1) Guess the form of the solution.

(2) use mathematical induction to find the constants and show that the solution works.

→ The method is powerful, but can be applied only in cases when we can guess the solution.

→ The method can be used to establish either upper or lower bounds on a recurrence.

For example:

Find an upper bound on the recurrence

$T(n) = 2T(\lfloor n/2 \rfloor) + n$. — (3)

Sol: This is similar to the recurrence which we saw in (1) & (2).

we guess that the solution is

$T(n) = O(n \log n)$.

we have to prove that

$T(n) \leq cn \log n$ for an appropriate choice of the constant $c > 0$.

we will use mathematical induction to prove this: ... holds for $\lfloor n/2$

to prove this.

Assume that this bound holds for $\lfloor n/2 \rfloor$.

(ie) $T(\lfloor n/2 \rfloor) \leq c(\lfloor n/2 \rfloor) \log \lfloor n/2 \rfloor$.

Substituting into the recurrence (3),

we get

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

$$\leq 2c\frac{n}{2} \log n/2 + n$$

$$= cn \log n/2 + n$$

$$= cn \log n - cn \log 2 + n$$

$$= cn \log n - cn + n$$

$$= cn \log n + n(1-c)$$

$$\leq cn \log n \quad , \text{ for } c \geq 1$$

(2) Show that the solution of the recurrence
$T(n) = 2T(\lfloor n/2 \rfloor + 15) + n$ is $T(n) = O(n \log n)$.

<u>solution</u>:

Assume that
$T(m) \leq cm \log m$ for all values $m < n$.

$$\therefore T(n) = 2(T\lfloor n/2 \rfloor + 15) + n$$
$$\leq 2c(\lfloor n/2 \rfloor + 15) \log(\lfloor \tfrac{n}{2} \rfloor + 15) + n$$
$$\leq 2c(\tfrac{n}{2} + 15) \log(\tfrac{n}{2} + 15) + n$$

Now, $\log(\frac{n}{2} + 15) = \log\left[\frac{n}{2}\left(1 + \frac{30}{n}\right)\right]$

$$= \log \frac{n}{2} + \log\left(1 + \frac{30}{n}\right) \qquad \left[\because \log(ab) = \log a + \log b\right]$$

$$= \log \frac{n}{2} + \frac{30}{n} - \frac{1}{2}\left(\frac{30}{n}\right)^2 + \frac{1}{3}\left(\frac{30}{n}\right)^3 - \cdots$$

$$\left[\because \log(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots\right]$$

$$\therefore T(n) \le 2c\left(\frac{n}{2} + 15\right) \log\left(\frac{n}{2} + 15\right) + n$$

$$= 2c\left(\frac{n}{2} + 15\right)\left[\log \frac{n}{2} + \frac{30}{n} - \frac{1}{2}\left(\frac{30}{n}\right)^2 + \frac{1}{3}\left(\frac{30}{n}\right)^3 \cdots\right] + n$$

$$+ n$$

$$\leq 2c\left(\frac{n}{2} + 15\right)\left[\log\left(\frac{n}{2}\right) + \frac{30}{n}\right] + n$$

$$\leq 2c\left(\frac{n}{2}\log\frac{n}{2} + 15\log\frac{n}{2} + 15 + \frac{450}{n}\right) + n$$

Choose $n \geq n_0 \geq 30$.

$$\therefore \leq 2c\left[\frac{n}{2}\log\frac{n}{2} + \frac{n}{2}\log\frac{n}{2} + \frac{n}{2}\log\frac{n}{2} + \frac{n}{2}\log\frac{n}{2}\right] + n$$

$$= 2c \cdot \frac{4n}{2}\log n/2 + n$$

$$= 4cn\log n/2 + n$$

$$= 2c \cdot \frac{4n}{2} \log n/2 + n$$

$$= 4cn \log n/2 + n$$

$$= 4cn(\log n - \log 2) + n$$

$$= 4cn \log n - 4cn + n$$

$$= 4cn \log n + n(1 - 4c)$$

$$\leq cn \log n \qquad \text{for } c \leq 1/4$$

$$\Rightarrow T(n) = O(n \log n)$$

Example 1

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

Guess: $T(n) = O(\lg n)$

- Induction goal: $T(n) \le d \lg n$, for some d and $n \ge n_0$
- Induction hypothesis: $T(n/2) \le d \lg(n/2)$

Proof of induction goal:

$$T(n) = T(n/2) + c \le d \lg(n/2) + c$$

$$= d \lg n - d + c \le d \lg n$$

if: $-d + c \le 0,\ d \ge c$

Example 2

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

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

- Induction goal: $T(n) \le c\, n^2$, for some c and $n \ge n_0$
- Induction hypothesis: $T(n-1) \le c(n-1)^2$ for all $k < n$

Proof of induction goal:

$$T(n) = T(n-1) + n \le c\,(n-1)^2 + n$$

$$= cn^2 - (2cn - c - n) \le cn^2$$

$$\text{if: } 2cn - c - n \ge 0 \Leftrightarrow c \ge n/(2n-1) \Leftrightarrow c \ge 1/(2 - 1/n)$$

- For $n \ge 1 \Rightarrow 2 - 1/n \ge 1 \Rightarrow$ any $c \ge 1$ will work

Example 3

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

Guess: $T(n) = O(n\lg n)$

- Induction goal: $T(n) \le cn \lg n$, for some c and $n \ge n_0$
- Induction hypothesis: $T(n/2) \le cn/2 \lg(n/2)$

Proof of induction goal:

$$T(n) = 2T(n/2) + n \le 2c\,(n/2)\lg(n/2) + n$$

$$= cn \lg n - cn + n \le cn \lg n$$

$$\text{if: } -cn + n \le 0 \Rightarrow c \ge 1$$

(3) Solve the recurrence

$T(n) = 2T(\lfloor\sqrt{n}\rfloor) + \log n$

By changing variable.

Let $m = \log n$

$\Rightarrow n = 2^m$

$\therefore T(2^m) + 2T(2^{m/2}) + m.$

Let $S(m) = T(2^m)$.

$\Rightarrow S(m) = 2S(m/2) + m$ — (1)

This recurrence is similar to the recurrence

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

whose solution is $T(n) = O(n\log n)$.

$\therefore$ solution of (1) is

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

$\Rightarrow T(n) = T(2^m) = S(m) = O(m\log m)$

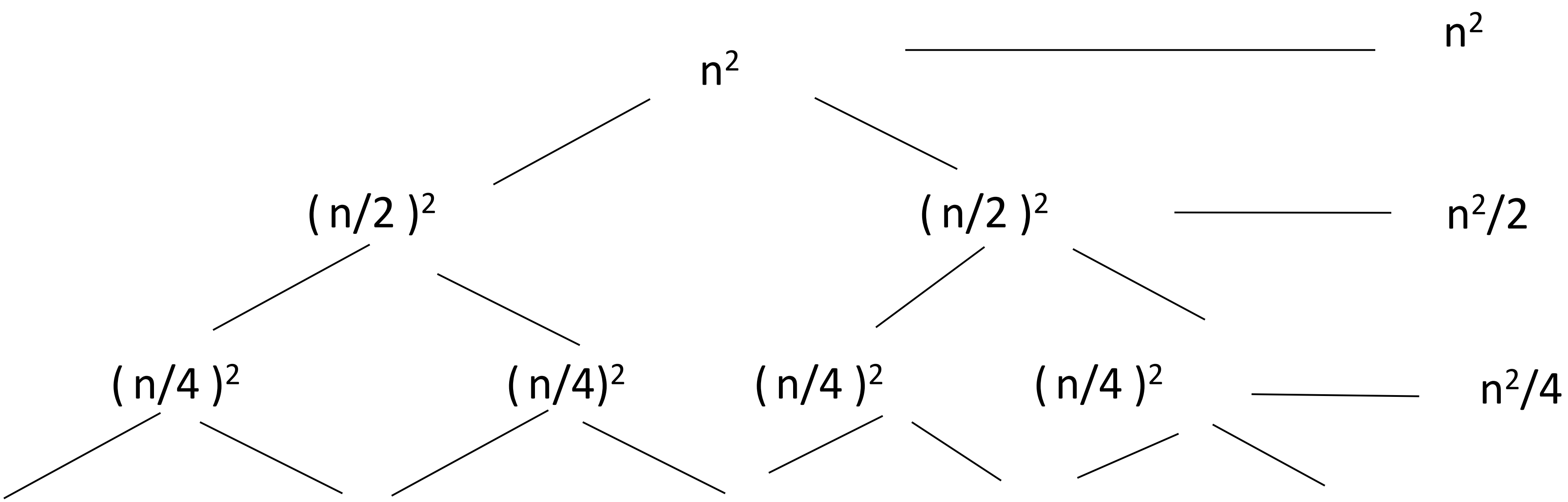
$= O(\log n \log(\log n))$

Recursion-tree method

- Convert the recurrence into a tree:
  - Each node represents the cost incurred at that level of recursion
  - Sum up the costs of all levels
- Used to “guess” a solution for the recurrence

SOLVE

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

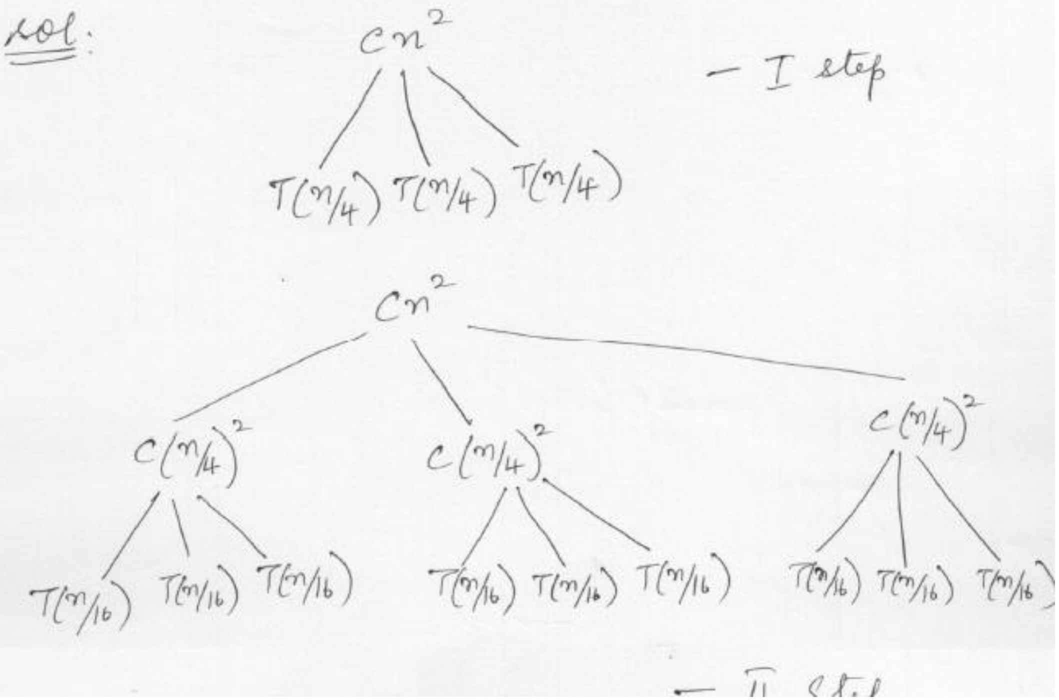


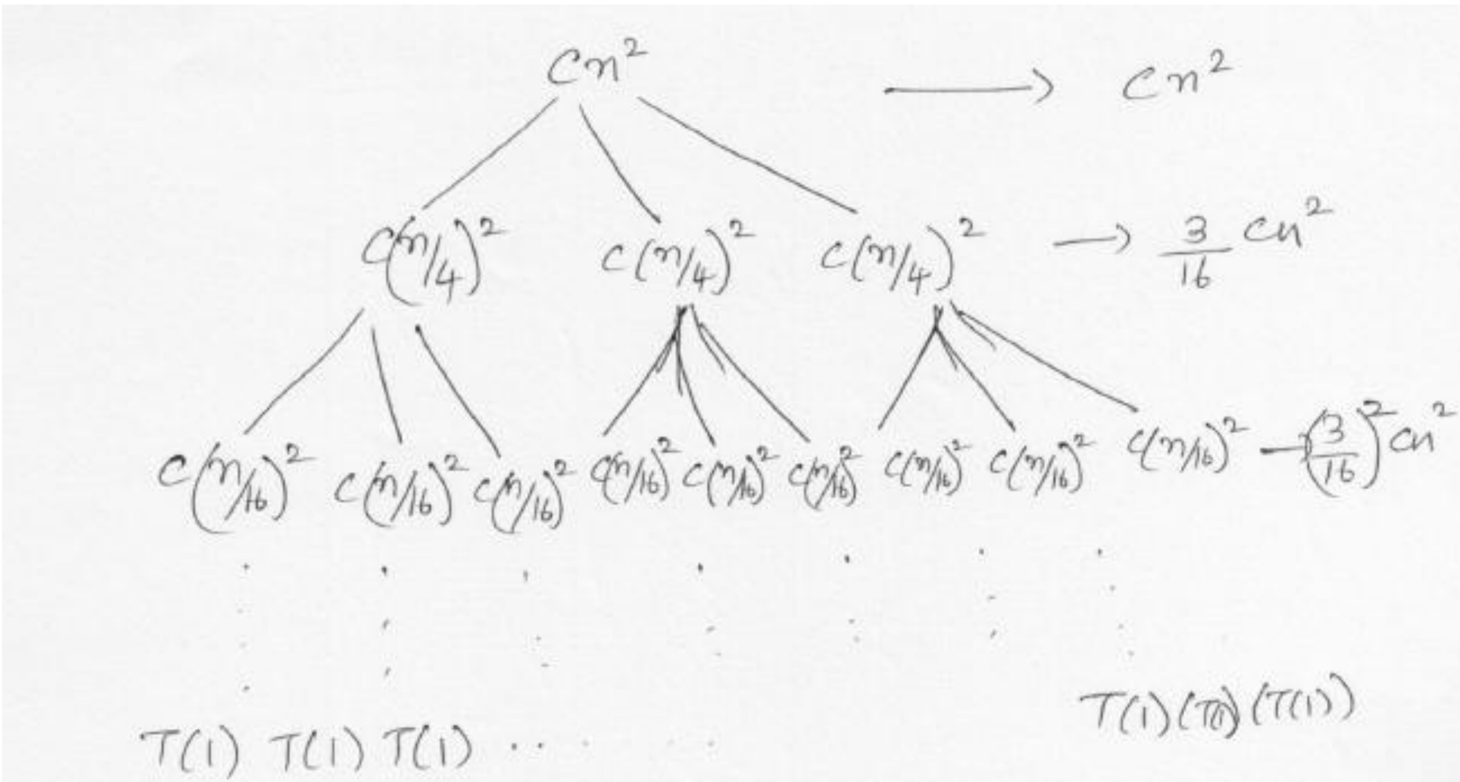
$$T(n) = \theta(n^2)$$

Recursion Tree Method

(1) Prove that the solution of the recurrence $T(n) = 3T(\lfloor \frac{n}{4} \rfloor) + cn^2$ is $O(n^2)$ using a recursion tree.

sol:





cn²
→ cn²
c(n/4)² c(n/4)² c(n/4)²
→ 3/16 cn²
c(n/16)² c(n/16)² c(n/16)² c(n/16)² c(n/16)² c(n/16)² c(n/16)² c(n/16)² c(n/16)²
→ (3/16)² cn²
T(1) T(1) T(1) . .
T(1) (T(1)) (T(1))

$T(1)\ T(1)\ T(1) \cdots$ $T(1)(T(1))(T(1))$

At the last level, we have

$\frac{n}{4^k} = 1$ (ie) at the k^{th} level

$\Rightarrow n = 4^k$ or $n = (2^2)^k$

or $\Rightarrow n = 2^{2k}$

$k = \log_4 n$

or $2k = \log_2 n$

or $k = \frac{1}{2} \log_2 n$

No. of nodes at each level.

$3^0 + 3^1 + 3^2 + 3^3 + \cdots + 3^k$.

(ie) 3^k nodes at the k^{th} level

or $k = \log_4 n$

or $3^k = 3^{\log_4 n}$

No. of nodes at each level	cost of node at each level
3^0	cn^2
3^1	$c(n/4)^2$
3^2	$c\left(\frac{n}{4^2}\right)^2$
$\vdots$	$\vdots$
3^{k-1}	$c\left(\frac{n}{4^{k-1}}\right)^2$
3^k	$c\left(\frac{n}{4^k}\right)^2$

Adding up we get

$$T(n) = cn^2 + 3c\left(\frac{n}{4}\right)^2 + 3^2 c\left(\frac{n}{4^2}\right)^2$$

$$+ \cdots + 3^{\log_4 n - 1} c\left(\frac{n}{4^{\log_4 n - 1}}\right)^2$$

$$+ 3^{\log_4 n} \cdot c\left(\frac{n}{4^{\log_4 n}}\right)^2$$

$\Rightarrow$

$$T(n) = cn^2 + \frac{3}{16}cn^2 + \frac{3^2}{16^2}cn^2$$

$$+ \cdots + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3})$$

$$\left[\because a^{\log_c b} = b^{\log_c a}\right]$$

$\Rightarrow$

$$T(n) = cn^2 + \frac{3}{16}cn^2 + \frac{3^2}{16^2}cn^2 + \cdots + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + \Theta\left(n^{\log_4 3}\right)$$

$$\left[\because a^{\log_c b} = b^{\log_c a}\right]$$

$$\leq cn^2\left(\frac{1}{1-\frac{3}{16}}\right) + \Theta\left(n^{\log_4 3}\right)$$

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

& since $\log_4 3 < 1$, omitting the term, we can write

$$T(n) \leq \frac{16}{13}cn^2$$

$$\Rightarrow T(n) = O(n^2).$$

Master's Theorem

In Master's theorem, the function $f(n)$ is compared with the function $n^{\log_b a}$.

The solution to the recurrence is determined by the larger of the two functions.

innovate achieve lead

To solve recurrences of the form

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

where $a \geq 1$ and $b > 1$ are constants and $f(n)$ is an asymptotically positive function.

Master's Theorem

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function and let $T(n)$ be defined on the nonnegative integers by the recurrence

$T(n) = a\,T(n/b) + f(n)$ where

n/b can be $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then, $T(n)$ can be bounded asymptotically as follows.

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

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

(3) If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $a \cdot f(n/b) \leq c f(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$

Solve using master's method

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

Sol. Here, $a = 9$, $b = 3$, $f(n) = n$.

$\therefore n^{\log_b a} = n^{\log_3 9} = \Theta(n^2)$

Since, $f(n) = O(n^{\log_3 9 - \epsilon})$, where $\epsilon = 1$, applying case 1 of the Master theorem, we have

$T(n) = \Theta(n^2)$

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

Sol. $a = 3$, $b = 4$, $f(n) = n \log n$.

$n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$.

since $f(n) = \Omega(n^{\log_4 3+\epsilon})$, where $\epsilon \approx 0.2$, case 3 applies if we can show that the regularity condition holds for $f(n)$.

For, large n,

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

$\left(\frac{3}{4}\right) n \log n$

$= c f(n)$

for $c = 3/4$.

$\therefore$ By case 3, solution is

$T(n) = \Theta(n \log n)$.

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

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

$$n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$$

Case 2 applies,

since $f(n) = \Theta(n^{\log_b a}) = \Theta(1)$,

∴ the solution is

$$T(n) = \Theta(\log n).$$

The master method

The master method applies to recurrences of the form

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

where $a \geq 1$, $b > 1$, and f is asymptotically positive.

Three common cases

Compare $f(n)$ with $n^{\log_b a}$:

1. $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$.
   - $f(n)$ grows polynomially slower than $n^{\log_b a}$ (by an n^{ε} factor).

Solution: $T(n) = \Theta(n^{\log_b a})$.

Three common cases

Compare $f(n)$ with $n^{\log_b a}$:

1. $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$.
   - $f(n)$ grows polynomially slower than $n^{\log_b a}$ (by an n^{ε} factor).

   Solution: $T(n) = \Theta(n^{\log_b a})$.

2. $f(n) = \Theta(n^{\log_b a} \lg^k n)$ for some constant $k \geq 0$.
   - $f(n)$ and $n^{\log_b a}$ grow at similar rates.

   Solution: $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$.

Three common cases (cont.)

Compare $f(n)$ with $n^{\log_b a}$:

3. $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$.
   - $f(n)$ grows polynomially faster than $n^{\log_b a}$ (by an n^{ε} factor),

   and $f(n)$ satisfies the ***regularity condition*** that $a\,f(n/b) \le c\,f(n)$ for some constant $c < 1$.

   Solution: $T(n) = \Theta(f(n))$.

Examples

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

$a = 2$, $b = 2$, $\log_2 2 = 1$

Compare $n^{\log_2 2}$ with $f(n) = n$

$\Rightarrow f(n) = \Theta(n) \Rightarrow$ Case 2

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

Examples (cont.)

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

$a = 2$, $b = 2$, $\log_2 2 = 1$

Compare n with $f(n) = n^2$

$\Rightarrow f(n) = \Omega(n^{1+\varepsilon})$ Case 3 $\Rightarrow$ verify regularity cond.

$a\, f(n/b) \leq c\, f(n)$

$\Leftrightarrow 2\, n^2/4 \leq c\, n^2 \Rightarrow c = \frac{1}{2}$ is a solution $(c<1)$

$\Rightarrow T(n) = \Theta(n^2)$

Examples (cont.)

$$T(n) = 2T(n/2) + \sqrt{n}$$

$a = 2$, $b = 2$, $\log_2 2 = 1$

Compare n with $f(n) = n^{1/2}$

$\Rightarrow f(n) = O(n^{1-\varepsilon})$ Case 1

$\Rightarrow T(n) = \Theta(n)$

Examples (cont.)

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

$a = 3$, $b = 4$, $\log_4 3 = 0.793$

Compare $n^{0.793}$ with $f(n) = n\lg n$

$f(n) = \Omega(n^{\log_4 3+\varepsilon})$ Case 3

Check regularity condition:

$$3*(n/4)\lg(n/4) \le (3/4)n\lg n = c*f(n),\ c=3/4$$

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

The master method

Solve the following

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

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

The master method

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

   $T(n) = \theta(\lg n)$

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

   $T(n) = \theta(n^2)$