SCC0602 - Algoritmos e Estruturas de Dados I

Divide and Conquer

Professor: André C. P. L. F. de Carvalho, ICMC-USP
PAE: Rafael Martins D'Addio
Monitor: Joao Pedro Rodrigues Mattos

Today

- Analysis the running time of recursive algorithms
  - Like divide-and-conquer
  - Writing and solving *recurrences*
- Tree recurrence method
- Master method

Revision: Insertion sort x Merge sort

- n^2 *versus nlgn*
  - *log(n)* : how many times do you need to divide n by 2 in order to reach 1 (assume $n = 2^m$)?

n^2	$nlgn$
$2^2 = 5$	$2lg(2) = 2$
$4^2 = 16$	$4lg(4) = 8$
…	
$32^2 = 1024$	$32lg(32) = 160$
…	
$512^2 = 262.144$	$512lg(512) = 4.606$

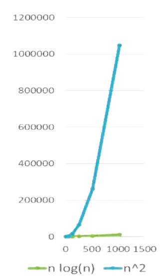
log(n) grows very slowly with n and much slower than n^2

lg(**number of particles in the universe**) < 280

Revision: Insertion sort x Merge sort

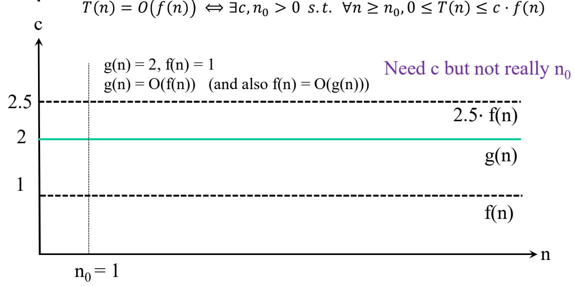
n^2 *versus* $nlgn$

n	n log(n)	n^2
8	24	64
16	64	256
32	160	1024
64	384	4096
128	896	16384
256	2048	65536
512	4608	262144
1024	10240	1048576



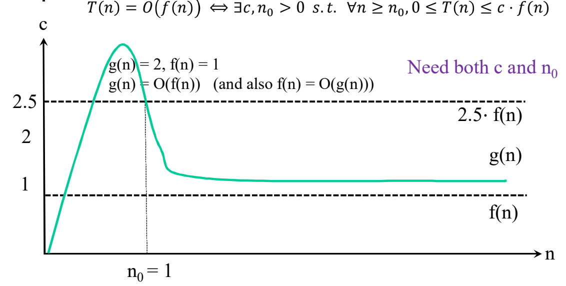
Why do we need n_0 and c

$$T(n) = O(f(n)) \Leftrightarrow \exists c, n_0 > 0 \ s.t. \ \forall n \geq n_0, 0 \leq T(n) \leq c \cdot f(n)$$



Why do we need n_0 and c

$$T(n) = O(f(n)) \Leftrightarrow \exists c, n_0 > 0 \ s.t. \ \forall n \geq n_0, 0 \leq T(n) \leq c \cdot f(n)$$



Divide and Conquer

- *Divide et impera* [Divide and rule]
  - Ancient political maxim cited by Machiavelli
    - Julius Caesar (102-44BC)
- The divide-and-conquer paradigm:
  - DIVIDE problem up into smaller problems
  - CONQUER by solving each subproblem
  - COMBINE results to solve the original problem

Recurrences

- **Recursive calls** in algorithms can be described using recurrences
- It is an equation or inequality that describes a function in terms of its value on smaller inputs

$$T(n) = \begin{cases} \text{solving\_trivial\_problem} & \text{if } n = 1 \\ \text{num\_pieces } T(n/\text{subproblem\_size\_factor}) + \text{dividing} + \text{combining} & \text{if } n > 1 \end{cases}$$

- Example: Merge Sort

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

Solving Recurrences

- Substitution method
  - Guess the solutions
  - Verify the solution by the mathematical induction
  - Repeated (backward) substitution method
    - Expand the recurrence by substitution and look for a pattern
- Recursion-trees
- Master method
  - templates for different classes of recurrences

Substitution method (ex. 2)

- Find the running time (upper bound) of merge sort
  - Assume that $n=2^b$, for some b

$$T(n) = \begin{cases} 2 & if\ n = 1 \\ 2T\left(\frac{n}{2}\right) + 2n + 3 & if\ n > 1 \end{cases}$$

$n_0 = 1 \rightarrow$ T(1) = 2 and c1lg1 = 0
2 ≤ 0 (impossible)

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

Guess that $T(n) = O(nlogn)$
Prove that T(n) ≤ cn lg n for a proper choice of c

Substitution method (ex. 2)

- Find the running time (upper bound) of merge sort
  - Assume that $n=2^b$, for some b

$$T(n) = \begin{cases} 2 & if\ n = 1 \\ 2T\left(\frac{n}{2}\right) + 2n + 3 & if\ n > 1 \end{cases}$$

~~$n_0 = 1 \rightarrow$ T(1) = 2 and c1lg1 = 0~~
~~2 ≤ 0 (impossible)~~
Use $n_0 = 2$
T(2) = 11 and c2lg2 = 2c
T(3) = 13 and c3lg3 = 3c1.6
For n> 3, recurrence does not depend on T(1)

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

Guess that $T(n) = \Theta(nlogn)$
Prove that T(n) ≤ cn lg n for a proper choice of c

Substitution method (ex. 2)

T(n) = 2T(n/2) + 2n + 3
Prove that T(n) ≤ cn lg n

Assuming that the bound holds for n/2, T(n/2) ≤ cn/2 lg (n/2)

T(n) ≤ 2[cn/2 lg (n/2)] + 2n + 3
≤ cn lg (n/2) + 2n + 3
≤ cn lg n – cnlg 2 + 2n + 3
≤ cn lg n – cn + 2n + 3 *(ignore terms < nlgn)*
≤ cn lg n

Choose positive value of c that holds for T(2) and T(3)

Substitution method (ex. 2)

$T(n) = 2T(n/2) + 2n + 3$
Prove that $T(n) \leq cn \lg n$

Assuming that the bound holds for $n/2$, $T(n/2) \leq c n/2 \lg (n/2)$

$$
\begin{aligned}
T(n) &\leq 2[c n/2 \lg (n/2)] + 2n + 3 \\
&\leq cn \lg (n/2) + 2n + 3 \\
&\leq cn \lg n - cn \lg 2 + 2n + 3 \\
&\leq cn \lg n - cn + 2n + 3 \\
&\leq cn \lg n
\end{aligned}
$$

Choose positive value of c that holds for T(2) and T(3)
If c = 1
  T(2) ≤ 1x2lg2
  11 ≤ 2 → does not hold
If c = 2
  T(2) ≤ 2x2lg2
  11 ≤ 4 → does not hold

Substitution method (ex. 2)

$T(n) = 2T(n/2) + 2n + 3$
Prove that $T(n) \leq cn \lg n$

Assuming that the bound holds for $n/2$, $T(n/2) \leq c n/2 \lg (n/2)$

$$
\begin{aligned}
T(n) &\leq 2[c n/2 \lg (n/2)] + 2n + 3 \\
&\leq cn \lg (n/2) + 2n + 3 \\
&\leq cn \lg n - cn \lg 2 + 2n + 3 \\
&\leq cn \lg n - cn + 2n + 3 \\
&\leq cn \lg n
\end{aligned}
$$

Choose positive value of c that holds for T(2) and T(3)
If c = 3
  T(2) ≤ 3x2lg2
  11 ≤ 6 → does not hold
If c = 6
  T(2) ≤ 6x2lg2
  11 ≤ 12 → holds

Substitution method (ex. 2)

$T(n) = 2T(n/2) + 2n + 3$
Prove that $T(n) \leq cn \lg n$

Assuming that the bound holds for $n/2$, $T(n/2) \leq c n/2 \lg (n/2)$

$$
\begin{aligned}
T(n) &\leq 2[c n/2 \lg (n/2)] + 2n + 3 \\
&\leq cn \lg (n/2) + 2n + 3 \\
&\leq cn \lg n - cn \lg 2 + 2n + 3 \\
&\leq cn \lg n - cn + 2n + 3 \\
&\leq cn \lg n \text{ (holds if } c > 5)
\end{aligned}
$$

Choose positive value of c that holds for T(2) and T(3)
If c = 6
  T(3) ≤ 6x3lg3
  13 ≤ 18x1,6 → holds

Thus, c = 6 and
T(n) = O(nlg n)

Repeated Substitution Method

- Variation of substitution method
  - Close to the tree
- Simple procedure:
  - Substitute
  - Expand
  - Substitute
  - Expand
  - ...

Repeated Substitution Method

- Simple procedure:
  - Make some substitutions
  - Observe a pattern and write how the expression looks after the i^{th} substitution
  - Find out which value i should have (e.g., lg$n$) to get the base case of the recurrence ($T(1)$)
  - Insert the value of $T(1)$ and the expression of i into the expression

Repeated Substitution (Ex. 2)

- Sequence of expand-substitute operations

$$
T(n) = \begin{cases} 2 & \text{if } n = 1 \\ 2T(n/2) + 2n + 3 & \text{if } n > 1 \end{cases}
$$

$$
\begin{aligned}
T(n) &\leq 2T(n/2) + 2n + 3 && \textit{Substitute } T(n/2) \\
&\leq 2[2T(n/4) + 2n/2 + 3] + 2n + 3 && \textit{Expand outside []} \\
&\leq 2^2T(n/4) + 4n + 2 \times 3 + 3 && \textit{Substitute } T(n/4) \\
&\leq 2^2 [2T(n/8) + 2n/4 + 3] + 4n + 2 \times 3 + 3 && \textit{Expand outside []} \\
&\leq 2^3 T(n/2^3) + 2 \times 3n + (2^2 + 2^1 + 2^0) \times 3 && \textit{There is a pattern} \\
&\leq 2^i T(n/2^i) + 2 \times in + 3 \times \textstyle\sum_{j=0}^{i-1} 2^j && \textit{Look for value of i to} \\
&\leq 2^{\lg n} T(n/n) + 2n\lg n + 3(2^{\lg n} - 1) && \textit{reach the base case: lg n} \\
&\leq 2n + 2n\lg n + 3n - 3 = 5n + 2n\lg n - 3 \\
&\leq n \lg n
\end{aligned}
$$

Exercise

- Running time of tromino tiling algorithm for a 2^nx2^n board

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 4T(n-1)+1 & \text{if } n > 1 \end{cases}$$

- Find its upper bound

Repeated Substitution (Ex. 3)

- Running time of tromino tiling algorithm for a 2^nx2^n board

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 4T(n-1)+1 & \text{if } n > 1 \end{cases}$$

$T(n) \le 4T(n-1) + 1$ *Substitute T(n-1)*
$\le 4[4T(n-2) + 1] + 1$ *Expand outside []*
$\le 4^2T(n-2) + 4 + 1$ *Substitute T(n-2)*
$\le 4^2[4T(n-3) + 1] + 4 + 1$ *Expand outside []*
$\le 4^3T(n-3) + 4^2 + 4^1 + 4^0$ *Look for value of i to reach* the base case: *n-1*
$\le 4^i\, T(n-i) + \sum_{j=0}^{i-1} 4^j$
$\le 4^{n-1}\, T(1) + \frac{4^{n-1}-1}{4-1} = \frac{4^n - 1}{3}$

Substitution method

Solve $T(n) = 4T(n/2) + n$
1) Guess that $T(n) = O(n^3)$
2) Assume that $T(n) \le ck^3$ *for* $k \le n/2$
3) Prove by induction that $T(n) \le cn^3$

$T(n) = 4T(n/2) + n$ *(Recurrence)*
$\le 4c(n/2)^3 + n$ *(Induction)*
$\le c/2n^3 + n$ *(Simplification)*
$\le cn^3$

$n_0 = 1 \rightarrow$ T(1) = 1 and $cn^3 = c\text{x}1 = c$

Substitution method

Solve $T(n) = 4T(n/2) + n$
1) Guess that $T(n) = O(n^3)$
2) Assume that $T(n) \le ck^3$ *for* $k \le n/2$
3) Prove by induction that $T(n) \le cn^3$

$T(n) = 4T(n/2) + n$ *(Recurrence)*
$\le 4c(n/2)^3 + n$ *(Induction)*
$\le c/2n^3 + n$ *(Simplification)*
$\le cn^3$

$n_0 = 1 \rightarrow$ T(1) = 1 and $cn^3 = c\text{x}1 = c$
Choose positive value of c that holds for T(1) and T(2)
If c = 1
$T(1) \le 1\text{x}1^3$
$1 \le 1 \rightarrow$ holds
$T(2) \le 1\text{x}2^3$
$6 \le 8 \rightarrow$ holds

Thus $c \ge 1$ and $T(n) = O(n^3)$

Substitution Method

- Achieving tighter bounds

Try to show that $T(n) = O(n^2)$
Assume that $T(k) \le ck^2$

$T(n) \le 4T(n/2) + n$
$\le 4c(n/2)^2 + n$
$\le cn^2 + n$
$\le cn^2$

$n_0 = 1 \rightarrow$ T(1) = 1 and $cn^2 = c\text{x}1 = c$

Substitution Method

- Achieving tighter bounds

Try to show that $T(n) = O(n^2)$
Assume that $T(k) \le ck^2$

$T(n) \le 4T(n/2) + n$
$\le 4c(n/2)^2 + n$
$\le cn^2 + n$
$\le cn^2$

$n_0 = 1 \rightarrow$ T(1) = 1 and $cn^2 = c\text{x}1 = c$
Choose positive value of c that holds for T(1) and T(2)
If c = 1
$T(1) \le 1\text{x}1^2$
$1 \le 1 \rightarrow$ holds
$T(2) \le 1\text{x}2^2$
$5 \le 4 \rightarrow$ Does not hold
If c = 2
$T(1) \le 2\text{x}1^2$
$1 \le 2 \rightarrow$ holds
$T(2) \le 2\text{x}2^2$
$9 \le 8 \rightarrow$ Does not hold

Substitution Method

- Achieving tighter bounds

Try to show that $T(n) = O(n^2)$
Assume that $T(k) \le ck^2$

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

$n_0 = 1 \rightarrow$ $T(1) = 1$ and $cn^2 = c\text{x}1 = c$
Choose positive value of c that holds for T(1) and T(2)
If c = 3
$T(1) \le 3\text{x}1^2$
$1 \le 3 \rightarrow$ holds
$T(2) \le 3\text{x}2^2$
$13 \le 12 \rightarrow$ Does not hold
If c = 4
$T(1) \le 4\text{x}1^2$
$1 \le 4 \rightarrow$ holds
$T(2) \le 4\text{x}2^2$
$17 \le 16 \rightarrow$ Does not hold

False for any value of c
Never holds

Substitution Method (2)

- What is the problem?
  - The inequality (cn^2 + positive value) $\le$ cn^2 is not possible
- To prove inductive step, try to make the hypothesis stronger
  - T(n) $\le$ (answer you want) - (something > 0)

Substitution Method (3)

- Corrected proof:
  - Strength the inductive hypothesis by subtracting lower-order terms!

Assume $T(k) \le c_1k^2 - c_2k$ for $k < n$

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\le 4(c_1(n/2)^2 - c_2(n/2)) + n \\
&= c_1n^2 - 2c_2n + n \\
&= c_1n^2 - c_2n - (c_2n - n) \\
&\le c_1n^2 - c_2n \text{ if } c_2 \ge 1
\end{aligned}
$$

Substitution Method

- Powerful, but we need to guess the form of the solution
  - Making a good guess for the substitution method can be difficult
  - Recursion trees can give good guesses of asymtotic solutions to recurrences
    - Which can be confirmed by the substitution method
    - They can even be the direct proof of the solution to a recurrence

Recursion Trees

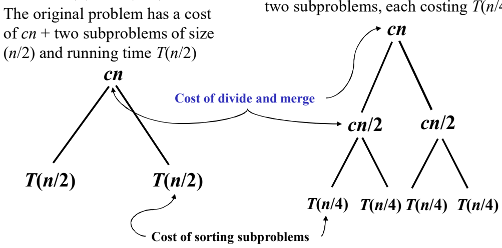
- Show successive expansions of recurrences using trees
  - Convenient way to visualize what happens when a recurrence is iterated
    - Keep track of the time spent on the subproblems of a divide and conquer algorithm
  - Help to sum the processing times necessary to solve a recurrence

Recursion Tree for Merge Sort

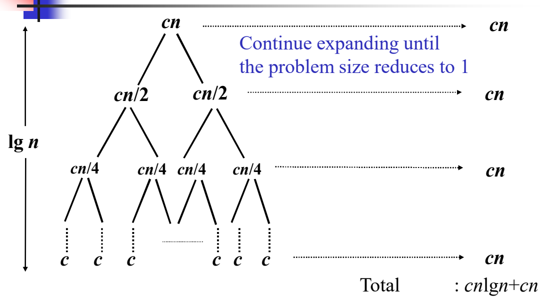
$T(n)=2T(n/2) + \Theta(n)$
$T(n)=2T(n/2) + cn$

The original problem has a cost of cn + two subproblems of size (n/2) and running time $T(n/2)$

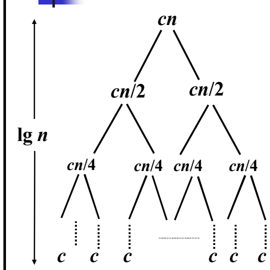
Each n/2 size problem has a cost of cn/2 + two subproblems, each costing $T(n/4)$



Recursion Tree for Merge Sort



Recursion Tree for Merge Sort



- Each level has total cost ***cn***
- When we go down one level, number of subproblems doubles, but the cost per subproblem halves $\Rightarrow$ *cost per level remains the same*
- There are lg n+1 levels, height is lg n (Assuming $n = 2^m$)
  - Can be proved by induction
- Total cost=sum of costs at each level = (lg n+1)cn = cnlg$n$ + cn = $\Theta(n \lg n)$

Recursion Tree for Merge Sort

- Confirm with the Substitution Method

Try to show that $T(n) = O(n\lg n)$
Assume that $T(n) = 2T(n/2) + cn$

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

Exemplo 2

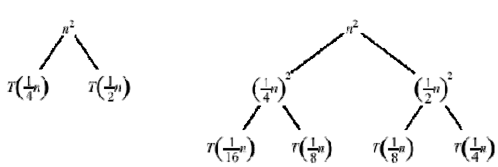
- Show the recurrence tree for the recurrence

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

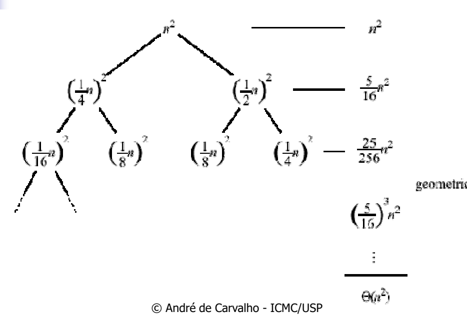
Exemplo 2

- Show the recurrence tree for the recurrence

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



Example 2



Exercícios

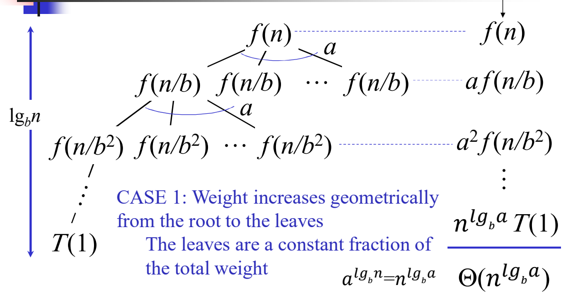
- Use the recursion-tree method to determine a guess for the recurrences
  - *Ex. 2:* $T(n) = T(n/3) + T(2n/3) + O(n)$
  - *Ex. 3:* $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$

Master Method

- Try to solve a class of recurrences of the form
  $$T(n) = aT(n/b) + f(n)$$
  - Where $a \geq 1$, $b > 1$ and f is asymptotically positive
- T(n) is the runtime for an algorithm and it is known that
  - a subproblems of size n/b are solved recursively, each in time $T(n/b)$
  - $f(n)$ is the cost of dividing the problem and combining the results
    - In merge-sort $T(n) = 2T(n/2) + \Theta(n)$

Master Method (2)

Functions increase from top to bottom geometrically
Thus, only the last bottom term is needed



CASE 1: Weight increases geometrically from the root to the leaves

The leaves are a constant fraction of the total weight

$a^{\lg_b n} = n^{\lg_b a}$

Master Method (3)

- Number of leaves: $a^{\lg_b n} = n^{\lg_b a}$
- Iterating the recurrence, expanding the tree yields

$$\begin{aligned} T(n) &= f(n) + aT(n/b) \\ &= f(n) + af(n/b) + a^2T(n/b^2) \\ &= f(n) + af(n/b) + a^2f(n/b^2) + \ldots \\ &\quad + a^{\log_b n-1}f(n/b^{\log_b n-1}) + a^{\log_b n}\, T(1) \\ &= \sum_{j=0}^{\log_b n-1} a^j f(n/b^j) + \Theta(n^{\lg_b a}) \end{aligned}$$

  - The first term is a division/recombination cost (totaled across all levels of the tree)
  - The second term is the cost of doing all $n^{\log_b a}$ subproblems of size 1 (total of all work pushed to leaves)

Master Method intuition

- Three common cases:
  - Running time dominated by cost at leaves
  - Running time evenly distributed throughout the tree
  - Running time dominated by cost at the root
- Thus, to solve the recurrence, we need only to characterize the dominant term
- In each case compare $f(n)$ with $O(n^{\lg_b a})$

Master Method Case 1

- $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$
  - $f(n)$ grows polynomially (by factor n^{ε}) slower than $n^{\log_b a}$
- The work at the leaf level dominates
  - Summation of recursion-tree levels $O(n^{\log_b a})$
  - Cost for all the leaves $\Theta(n^{\log_b a})$
  - Thus, the overall cost $\Theta(n^{\log_b a})$

Master Method Case 2

- $f(n) = \Theta(n^{\log_b a})$
  - $f(n)$ and $n^{\log_b a}$ are asymptotically the same
- The work is equally distributed throughout the tree $T(n) = \Theta(n^{\log_b a} \lg n)$
  - (level cost) × (number of levels)

Master Method Case 3

- $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$
  - Inverse of Case 1
  - *f*(*n*) grows polynomially faster than $n^{\log_b a}$
  - Also need a regularity condition

    $\exists c < 1 \text{ and } n_0 > 0 \text{ such that } af(n/b) \le cf(n)\ \forall n > n_0$
- The work at the root dominates

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

Master Method Summarized

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

  1. $f(n) = O\left(n^{\log_b a - \varepsilon}\right)$

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

     $\Rightarrow T(n) = \Theta\left(n^{\log_b a} \lg n\right)$
  3. $f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right)$ and $af(n/b) \le cf(n)$, for some $c < 1, n > n_0$

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

  - The master method cannot solve every recurrence of this form; there is a gap between cases 1 and 2, as well as cases 2 and 3
    - *f(n)* is smaller (larger) than $n^{lg_b a}$ but not polynomially smaller (larger)

Strategy

- 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 MT case, and apply
- Example merge sort

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

  $a = 2, b = 2, n^{lg_b a} = n^{lg_2 2} = \mathrm{n} = \Theta(n), f(n) = \Theta(n),$

  $\rightarrow$ Case 2: $T(n) = \Theta(n^{lg_b a} lg\ n) = \Theta(n\ lg\ n)$

Example of Master Method 2

```
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)
```

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

$a = 1, b = 2, n^{lg_2 1} = 1 = \Theta(1), f(n) = 1 = \Theta(1),$

$f(n) = \Theta(n^{lg_b a}) \rightarrow$ Case 2: $T(n) = \Theta(n^{lg_b a} lg\ n) = \Theta(lg\ n)$

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

$a = 9, b = 3, n^{lg_3 9} = \Theta(n^2), f(n) = n = O(n^{lg_3 9 - \varepsilon})$ where $\varepsilon = 1$

$f(n) = O(n^{lg_3 9 - \varepsilon}) \rightarrow$ Case 1: $T(n) = \Theta(n^{lg_b a}) = \Theta(n^2)$

Multiplication Example (I)

- Multiplying two *n*-digit (or *n*-bit) numbers costs n^2 digit multiplications using a classical procedure
- Observation:
  - $23*14 = (2 \times 10^1 + 3)*(1 \times 10^1 + 4) =$

    $(2*1)10^2 + (3*1 + 2*4)10^1 + (3*4)$
- To save one multiplication use the trick:
  - $(3*1 + 2*4) = (2+3)*(1+4) - (2*1) - (3*4)$

Multiplication Example (II)

- To multiply a and b, which are n-digit numbers, use a divide and conquer algorithm
  - Split a and b in half:
    - $a = a_1 \times 10^{n/2} + a_0$ and $b = b_1 \times 10^{n/2} + b_0$
  - Then:
    - $a * b = (a_1 * b_1)10^n + (a_1 * b_0 + a_0 * b_1)10^{n/2} + (a_0 * b_0)$
  - Use a trick to save one multiplication:
    - $(a_1 * b_0 + a_0 * b_1) = (a_1 + a_0)*(b_1 + b_0) - (a_1 * b_1) - (a_0 * b_0)$

Multiplication Example(III)

- Number of single-digit multiplications performed by the algorithm can be described by a recurrence:

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 3T(n/2) & \text{if } n > 1 \end{cases}$$

- Solution: $T(n) = n^{\log_2 3} = n^{1.585}$

Next Week

- Sorting
  - QuickSort
  - HeapSort

Acknowledgement

- A large part of this material was adapted from
  - Simonas Šaltenis, Algorithms and Data Structures, Aalborg University, Denmark
  - Mary Wootters, Design and Analysis of Algorithms, Stanford University, USA
  - George Bebis, Analysis of Algorithms CS 477/677, University of Nevada, Reno
  - David A. Plaisted, Information Comp 550-001, University of North Carolina at Chapel Hill

Questions