CSE 417: Algorithms and Computational Complexity

Divide & Conquer

Autumn 2002
Paul Beame

1

Algorithm Design Techniques

- Divide & Conquer
  - Reduce problem to one or more sub-problems of the same type
  - Typically, each sub-problem is **at most a constant fraction** of the size of the original problem
    - e.g. Mergesort, Binary Search, Strassen's Algorithm, Quicksort (kind of)

2

Fast exponentiation

- Power(**a**,**n**)
  - **Input:** integer **n** and number **a**
  - **Output:** a^n
- Obvious algorithm
  - **n-1** multiplications
- Observation:
  - if **n** is even, **n=2m**, then $a^n=a^m \bullet a^m$

3

Divide & Conquer Algorithm

- Power(**a**,**n**)
  - if **n=0** then return(**1**)
  - else if **n=1** then return(**a**)
  - else
    - $x \leftarrow$ Power(**a**,$\lfloor n/2 \rfloor$)
    - if **n** is even then
      - return($x \bullet x$)
    - else
      - return($a \bullet x \bullet x$)

4

Analysis

- Worst-case recurrence
  - $T(n)=T(\lfloor n/2 \rfloor)+2$ for $n=1$
  - $T(1)=0$
- Time
  - $T(n)=T(\lfloor n/2 \rfloor)+2 = T(\lfloor n/4 \rfloor)+2+2 = \ldots$
    $= T(1)+\underbrace{2+\ldots+2}_{\log_2 n \text{ copies}} = 2\log_2 n$
- More precise analysis:
  - $T(n)= \lceil \log_2 n \rceil$ + # of **1**'s in **n**'s binary representation

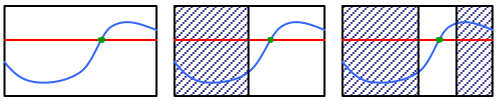
5

A Practical Application- RSA

- Instead of a^n want a^n **mod N**
  - a^{i+j} mod **N** = ((a^i mod **N**)$\bullet$(a^j mod **N**)) mod **N**
  - same algorithm applies with each $x \bullet y$ replaced by
    - ((**x** mod **N**)$\bullet$(**y** mod **N**)) mod **N**
- In RSA cryptosystem (widely used for security)
  - need a^n **mod N** where **a**, **n**, **N** each typically have **1024** bits
  - Power: at most **2048** multiplies of 1024 bit numbers
    - relatively easy for modern machines
  - Naive algorithm: 2^{1024} multiplies

6

Binary search for roots (bisection method)



- Given:
  - continuous function **f** and two points **a<b** with **f(a) = 0** and **f(b) > 0**
- Find:
  - approximation to **c** s.t. **f(c)=0** and **a<c<b**

Bisection method

Bisection(**a**,**b**, e)

  if (**a**-**b**) < e then

    return(**a**)

  else

    c ← (**a**+**b**)/**2**

    if **f**(**c**) = **0** then

      return(Bisection(**c**,**b**,e))

    else

      return(Bisection(**a**,**c**,e))

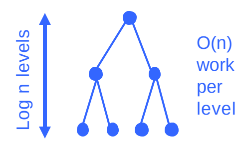
Time Analysis

- At each step we halved the size of the interval
- It started at size **b-a**
- It ended at size e
- # of calls to f is $\log_2($ (**b-a**)/e)

Mergesort (review)

Mergesort: (recursively) sort 2 half-lists, then merge results.

- T(n)=2T(n/2)+cn, n≥2
- T(1)=0
- Solution: Θ(n log n)



Why Balanced Subdivision?

- Alternative "divide & conquer" algorithm:
  - Sort first **n-1**
  - Sort last **1**
  - Merge them
- Recurrence
  - **T(n)=T(n-1)+T(1)+3n** for **n≥2**
  - **T(1)=0**
- Solution:
  - **3n** + **3(n-1)** + **3(n-2)** … = Θ(**n²**)

Another D&C Approach

- Suppose we've already invented DumbSort, taking time **n²**
- Try *Just One Level* of divide & conquer:
  - DumbSort(first **n/2** elements)
  - DumbSort(last **n/2** elements)
  - Merge results
- Time:
  - **(n/2)² + (n/2)² + n = n²/2 + n**
  - Almost twice as fast!

Some Divide &Conquer morals

- Moral 1:
  - Two problems of half size are *better* than one full-size problem, even given the **O(n)** overhead of recombining, since the base algorithm has *super-linear* complexity.
- Moral 2:
  - If a little's good, then more's better
    - **2** levels of D&C would be almost **4** times faster, **3** levels almost **8**, etc., even though overhead is growing.
    - Best is usually full recursion down to some small constant size (balancing "work" vs "overhead").

13

Divide & Conquer morals

- Moral 3: unbalanced division less good:
  - $(.1n)^2 + (.9n)^2 + n = .82n^2/2 + n$
    - The **18%** savings compounds significantly if you carry recursion to more levels, actually giving **O(n log n)**, but with a bigger constant.
    - worth doing if you can't get **50-50** split, but balanced is better if you can.
    - This is intuitively why Quicksort with random splitter is good – badly unbalanced splits are rare, and not instantly fatal.
  - $(1)^2 + (n-1)^2 + n = n^2 - 2n + 2 + n$
    - Little improvement here.

14

Sometimes two sub-problems aren't enough

- More general divide and conquer
  - You've broken the problem into **a** different sub-problems
  - Each has size at most **n/b**
  - The cost of the break-up and recombining the sub-problem solutions is $O(n^k)$
- Recurrence
  - $T(n)= a\cdot T(n/b)+c\cdot n^k$

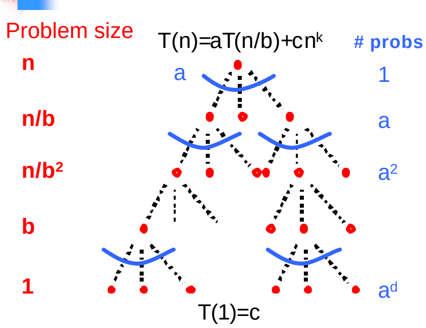
15

Master Divide and Conquer Recurrence

- If $T(n)= a\cdot T(n/b)+c\cdot n^k$ for $n>b$ then
  - if $a>b^k$ then $T(n)$ is $Q(n^{\log_b a})$
  - if $a<b^k$ then $T(n)$ is $Q(n^k)$
  - if $a=b^k$ then $T(n)$ is $Q(n^k \log n)$
- Works even if it is $\lceil n/b \rceil$ instead of n/b.

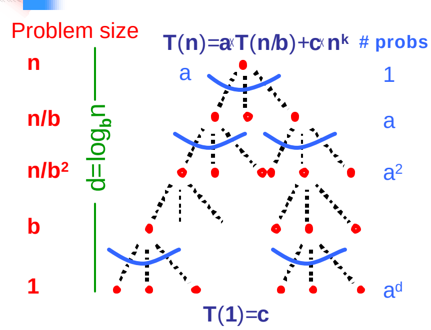
16

Proving Master recurrence



17

Proving Master recurrence



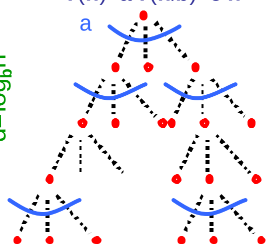
18

Proving Master recurrence

Problem size $\quad T(n)=a\cdot T(n/b)+c\cdot n^k \quad$ # probs $\quad$ cost

Problem size	# probs	cost
n	1	cn^k
n/b	a	$c\cdot a\cdot n^k/b^k$
n/b^2	a^2	$c\cdot a^2\cdot n^k/b^{2k} = c\cdot n^k(a/b^k)^2$
b		
1	a^d	$c\cdot n^k(a/b^k)^d = c\cdot a^d$

$d=\log_b n$



$T(1)=c$

19

Geometric Series

- $S \quad = t + tr + tr^2 + \ldots + tr^{n-1}$
- $r\cdot S \quad = \quad tr + tr^2 + \ldots + tr^{n-1} + tr^n$
- $(r-1)S = tr^n - t$
- so $S=t\,(r^n-1)/(r-1)$ if r?1.

- Simple rule
  - If r ? 1 then S is a constant times largest term in series

20

Total Cost

- Geometric series
  - ratio a/b^k
  - $d+1=\log_b n+1$ terms
  - first term cn^k, last term ca^d
- If $a/b^k=1$
  - all terms are equal $T(n)$ is Q($n^k \log n$)
- If $a/b^k<1$
  - first term is largest $T(n)$ is $\Theta(n^k)$
- If $a/b^k>1$
  - last term is largest $T(n)$ is $\Theta(a^d)=\Theta(a^{\log_b n}) = \Theta(n^{\log_b a})$
  (To see this take $\log_b$ of both sides)

21