CSE 417: Algorithms and Computational Complexity

Winter 2001

Lecture 5

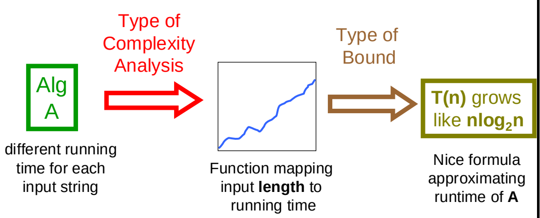
Instructor: Paul Beame

TA: Gidon Shavit

Fixing a Misunderstanding

- We have looked at
  - type of complexity analysis
    - worst-case, best-case, average-case
  - types of function bounds
    - O, Ω, Θ
- These two considerations are orthogonal to each other
  - one can do any type of function bound with any type of complexity analysis

Complexity analysis overview



Usually we represent the function in the middle using a recurrence relation rather than explicitly

Quicksort Analysis

- Partition does **n-1** comparisons on a list of length **n**
  - pivot is compared to each other element
- If **pivot** is i^{th} largest then two subproblems are of size **i-1** and **n-i**
- Pivot is equally likely to be any one of 1^{st} through n^{th} largest

$$T(n) = n-1+\frac{1}{n}\sum_{i=1}^{n}\left(T(i-1)+T(n-i)\right)$$

Quicksort analysis

$$T(n) = n-1+\frac{1}{n}\sum_{i=1}^{n}\left(T(i-1)+T(n-i)\right)$$

$$= n-1+\frac{2\,T(1)+2\,T(2)+\ldots+2\,T(n-1)}{n}$$

$$\therefore nT(n) = n(n-1)+2\,T(1)+2\,T(2)+\ldots+2\,T(n-1)$$

$$(n+1)T(n+1) = (n+1)n+2\,T(1)+2\,T(2)+\ldots+2\,T(n)$$

$$\therefore (n+1)T(n+1)-nT(n) = 2\,T(n)+2n$$

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

$$\therefore \frac{T(n+1)}{n+2} = \frac{T(n)}{n+1}+\frac{2n}{(n+1)(n+2)}$$

Quicksort analysis

Let $Q(n) = \frac{T(n)}{n+1}$

$$\therefore Q(n+1) \le Q(n)+\frac{2}{n+1}$$

$$\therefore Q(n) \le 2(1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n}) = 2H_n \approx 2\ln n = 1.38\log_2 n$$

(Recall that $\ln n = \int_1^n 1/x\ dx$)

$$\therefore T(n) \approx 1.38\, n\log_2 n$$

Algorithm Design Techniques

- General overall idea
  - Reduce solving a problem to a smaller problem or problems of the same type
- Greedy algorithms
  - Used when one needs to build something a piece at a time
  - Repeatedly make the **greedy** choice - the one that looks the best right away
    - e.g. closest pair in TSP search
  - Usually fast if they work

7

Algorithm Design Techniques

- Divide & Conquer
  - Reduce problem to one or more sub-problems of the same type
  - 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)

8

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 $a^n = a^m \cdot a^m$

9

Divide & Conquer Algorithm

- Power(a,n)

```
if n=0 then
    return(1)
else
    x ←Power(a,⌊n/2⌋)
    if n is even then
        return(x•x)
    else
        return(a•x•x)
```

10

Analysis

- Worst-case recurrence
  - $T(n)=T(\lfloor n/2 \rfloor)+2$
- By master theorem
  - $T(n)=O(\log n)$
- More precise analysis:
  - $T(n)= \lceil \log_2 n \rceil$ + # of 1's in n's binary representation

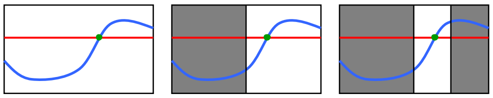
11

A Practical Application- RSA

- Instead of a^n want **a^n mod N**
  - a^{i+j} mod **N** = ((a^i mod **N**)•(a^j mod **N**)) mod **N**
  - same algorithm applies with each **x•y** replaced by
    - ((**x** mod **N**)•(**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

12

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**

13