Background

- If you do not have necessary background in analysis of algorithms
  - See the book
    - Introduction to Algorithms by
      - Cormen, Leiserson, Rivest and Stein
    - Or go online
  - Topics to study
    - Introduction
    - Growth of functions
    - Summations
    - Recurrences

Background: Orders of Magnitude

- O, Ω, Θ
  - $f(x) = O(g(x))$ iff $\exists\ c, n_0 : f(x) < c.g(x)\ \ \forall\ n > n_0$
    - used for **upper bound** of algorithm complexity:
      this particular algorithm takes **at most** c.g(n) time
  - $f(x) = \Omega(g(x))$ iff $\exists\ c, n_0 : f(x) > c.g(x)\ \ \forall\ n > n_0$
    - used for **lower bound** of problem complexity:
      any algorithm for solving this problem takes **at least** c.g(n) time
  - $f(x) = \Theta(g(x))$ iff $f(x)=O(g(x))$ and $f(x)=\Omega(g(x))$
    - "Tight" bound

Background: Closed problems

- Closed problem P:
  - ∃ algorithm X with O(X) = Ω(P)
    eg. Sort has tight bound: Θ(nlog(n))
- Problem P has algorithmic gap:
  - P is not closed, eg., all NP Complete problems (problems with polynomial lower bound but currently exponential upper bound, such as TSP)

Recurrence Relations

- Algorithmic complexity often described using recurrence relations: f(n) = R(f(1) .. f(n-1))
- Two important types of recurrence relations
  - Linear
  - Divide and Conquer
- cs420(dl) covers these

Repeated substitution

- Simple recurrence relations (one recurrent term in the rhs) can sometimes be solved using repeated substitution
- Two types: Linear and DivCo
  - Linear F(n) = aF(n-d)+g(n), base: $F(1)=v_1$
  - Divco F(n)= aF(n/d)+g(n), base: $F(1)=v_1$
- Two questions:
  - what is the pattern
  - how often is it applied until we hit the base case

Linear Example

M(n)=2M(n-1)+1, M(1)=1

recognize this recurrence?

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

$4M(n-2)+2+1 = 4(2M(n-3)+1)+2+1=$

$8M(n-3)+4+2+1=$ … inductive step …

$2^kM(n-k)+2^{k-1}+2^{k-2}+...+2+1=$

hit base for k = n-1:

$= 2^{n-1}M(1)+2^{n-1}+2^{n-2}+...+2+1 = 2^n-1$

for more on Linear recurrence relations, see 420dl

DivCo example

Merge sort:

$T(n) = 2T(n/2) + n, \quad T(1)=1$

$n = 2^k$

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

$8T(n/8) + 3n$ ... inductive step … =

$2^kT(n/2^k)+kn$

hit base for k = logn

$= 2^kT(n/2^k)+kn = \quad n+kn = O(nlogn)$

Another one: binary search

$f(n) = f(n/2)+c \quad f(1)=1$

let $n = 2^k$

$f(n)=f(n/2)+c = f(n/4)+2c =$

$f(n/8)+3c = f(n/2^k)+kc =$

hit base for k=log n:

$f(1)+ c \log n = O(\log n)$

Master Method

Cookbook approach to solution, based on repeated substitution (Cormen et.al. or Rosen)

$A_n = C\,A_{n/d} + kn^p$

- $A_n = O(n^p)$ if $C < d^p$ eg $A_n = 3\,A_{n/2} + n^2$
- $A_n = O(n^p \log(n))$ if $C = d^p$ eg $A_n = 2A_{n/2} + n$
- $A_n = O(n^{\log_d c})$ if $C > d^p$ eg $A_n = 3\,A_{n/2} + n$

Do binary search and merge sort with this method

Examples

- Merge Sort

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

  C=? d=? p=? d^p=?

  T(n) = O(???)
- Binary Search

  $f(n) = f(n/2)+c \quad f(1)=1$

  C=? d=? p=? d^p=?

  f(n) = O(???)