Divide and Conquer

Divide and Conquer

1. Break Problem Up
2. Solve Problems when they are small
3. Aggregate Solutions

Equation

$(a + bi)(c + di) = ac + adi + bci - bd = ac - bd + i(ad + bc)$
where
$bc + ad = ((a + b)(c + d)) - ac - bd$
Runtimes:
Multiplication = O(n)

In Class Problem

$x = xlxr = 2^{\frac{n}{2}}xl + xr$

Depth of the Tree
Number of Nodes at level k (branching factor)
Size of subproblem
Number of subproblems = 4^k

Runtime Analysis

$n/2^k$ = $k = log2n$

$O(n) = 2^k$

$T(n) = 4T(\frac{n}{2}) + O(n)$=$O(\frac{n}{2^k})4^k$=$O(n)(\frac{4}{2})^k$=$O(n)2^k$

Mergesort

- Top heavy has a O(n) complexity at the top
- Bottom Heavy have a O(1) complexity at the top (majority of work is being done in the bottom)
- Mid weight algorithms speed depend on both the span of the tree and the problems themselves

Apporaches to Divide and Conquor Algorithms

Dive Into

i.e. Binary Search

- Enters the Center of the problem
- Cuts out portions of the problem that cannot contain the solution
- $T(n) = T(\frac{n}{s}) + c$

Dive + Aggregate

i.e. Merge Sort

- You break the problem into simple to solve subproblems
- You then combine these solved problems into the problem again
- $T(n) = k * T(\frac{n}{s}) + c$
  - k = number of new problems we break things into
  - s = factor problems get smaller by
  - c = the amount of time it takes to solve a problem of size n
- Formally: $T(n) = a * T(\frac{n}{b}) + O(n^d)$
- Mergesort: $T(n) = 2 * T(\frac{n}{2}) + O(n)$

Geometric Series

$1 + s + s^2 + ... + s^n = c$

- multiply by s
  $s + s^2 + ... + s^{n+1} = cs$
  $s^{n+1} - 1 = cs - c$
  $s^{n+1} - 1 = c(s-1)$
  $\frac{s^{n+1}-1}{(s-1)} = c$
  if s<1, the limit as n->inf of s^{n+1} = 0
  if s=1, $1 + 1 + 1 ... + 1$ = O(n)
  if s>1, $\frac{s^{n+1}-1}{s-1} \leq a * s^n, ***$

Master Theorem

if $T(n) = aT(\lceil \frac{n}{b} \rceil) + O(n^d)$ for constants $a > 0$, $b > 1$, $d \geq 0$
Therefore, T(n) =
if $d > \log_b(a)$ **then** T(n) = $O(n^d)$
if $d = \log_b(a)$ **then** T(n) = $O(n^d)\log_b(n)$
if $d < \log_b(a)$ **then** T(n) = $O(n^{\log_b(a)})$
next week pick back up....

Master Theorum

$T(n) = aT(\lceil \frac{\pi}{b} \rceil) + O(n^d)$

- a > 0
- b > 1
- d >= 0
  where
- a = #(multiplicitive factor) (double means 2) of new Problems at next level from previous level
- b = size (dividing factor) of new problems at next level from previous level (prev prob size/ new prob size)
- d = Exponent of n indicating how much work we do for an arbatrary problem of size n

T(n)=

- if $d > log_b a$ **then** T(n)=$O(n^d)$
- if $d = log_b a$ **then** T(n)= $O(n^d * log_b n)$
- if $d < log_b a$ **then** T(n)= $O(n^{log_b a})$

Levels of K

- k'th level has a^k subproblems, each of size $\frac{n}{b^k}$
- work at level k
  - $a^k * O(\frac{n}{b^k})^d$ = $O(n^d)(\frac{a}{b^d})^k$
- **Potential Cases:**
  - $\frac{a}{b^d} < 1$ so T(n)=O(n^d)
  - $\frac{a}{b^d} = 1$ so T(n)=O($n^d * log_b n$)
  - $\frac{a}{b^d} > 1$ so T(n)=O(n^d)$(\frac{a}{b^d})^{log_b(n)}$