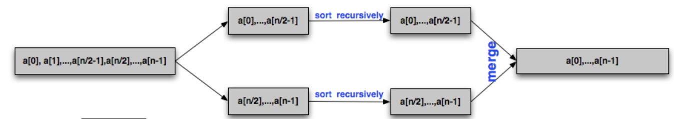
Divide-and-conquer

Prof. Tanya Berger-Wolf

Motivating Example: MergeSort

Sorting numbers a[0], **a[1], a[2], …, a[n-2], a[n-1]:**



T(n)

Running time on n numbers =

Split time + 2*(Running time on n/2 numbers) + Merge time of 2*(n/2)

Constant O(1)

T(n/2)

Exactly (we can prove) n

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

2

Divide and Conquer Recurrences

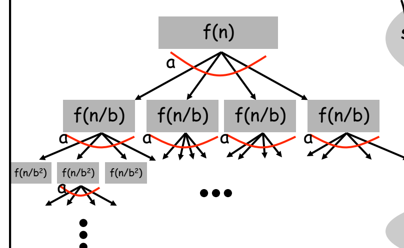
General form: $T(n) = aT(n/b) + f(n)$

What do the algorithms look like?
Divide the problem into a subproblems of size n/b.
Solve those subproblems (recursively).
Conquer the solution in time f(n).

Examples: mergesort and binary search.

Divide and Conquer Recurrences

General form: $T(n) = aT(n/b) + f(n)$
To solve a problem of size n, we require time f(n), plus the time it takes to solve a subproblems of size n/b.

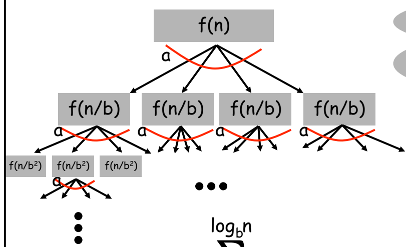


We don't have simple recipes for solving these in all cases, though sometimes we do…

Total running time is sum of the values in the grey rectangles.

Divide and Conquer Recurrences

General form: $T(n) = aT(n/b) + f(n)$



Sum over levels…

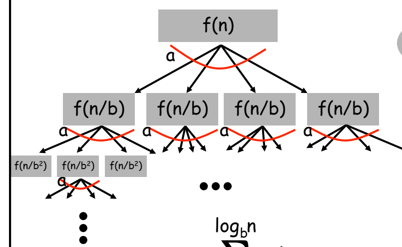
How many?

$\log_b n$

$$\sum_{i=0}^{\log_b n}$$

Divide and Conquer Recurrences

General form: $T(n) = aT(n/b) + f(n)$



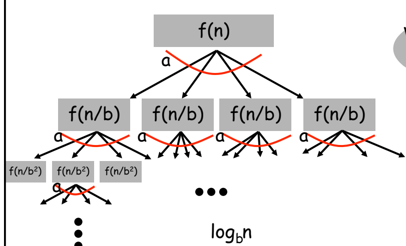
How many blocks at level i?

a^i

$$\sum_{i=0}^{\log_b n} a^i f(?)$$

Divide and Conquer Recurrences

General form: $T(n) = aT(n/b) + f(n)$



What is the work for the recursive call at level i?

n/b^i

$$\sum_{i=0}^{\log_b n} a^i f\left(\frac{n}{b^i}\right)$$

Divide and Conquer Recurrences

General form: $T(n) = aT(n/b) + f(n)$

$$\sum_{i=0}^{\log_b n} a^i f\left(\frac{n}{b^i}\right)$$

We no longer have recursive terms, but we do have a sum to deal with.

Consider binary search, and write a recurrence for the # of comparisons:

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

$a = 1$, $b = 2$, $f(n) = 1$.

$$\sum_{i=0}^{\log_2 n} 1^i \cdot 1 = \log_2 n + 1$$

Divide and Conquer Recurrences

General form: $T(n) = aT(n/b) + f(n)$

$$\sum_{i=0}^{\log_b n} a^i f\left(\frac{n}{b^i}\right)$$

We no longer have recursive terms, but we do have a sum to deal with.

How about our old favorite merge sort?
$T(n) = 2T(n/2) + n$

$a = 2$, $b = 2$, $f(n) = n$.

$$\sum_{i=0}^{\log_2 n} 2^i \cdot (n/2^i) = n \log_2 n$$

Example: Exponentiation

Iterative: $x^n = x \bullet x \bullet x \bullet x \bullet \ldots \bullet x$
n-1 operations

Recursive:

$$x^n = \begin{cases} x^{\frac{n}{2}} \cdot x^{\frac{n}{2}} & if \quad n \quad even \\ x^{\frac{n-1}{2}} \cdot x^{\frac{n-1}{2}} \cdot x & if \quad n \quad odd \end{cases}$$

return pow(x, n/2) * pow (x, n/2);

$T(n) = 2T(n/2) + 2$
$a = b = 2$, $f(n) = 2$

$$\sum_{i=0}^{\log_2 n} a^i f\left(\frac{n}{b^i}\right) = \sum_{i=0}^{\log_2 n} 2^i f\left(\frac{n}{2^i}\right) = 2\sum_{i=0}^{\log_2 n} 2^i = 2(2^{\log_2 n+1} - 1) = 4n - 2$$

a = pow(x, n/2);
return (a*a);

$T(n) = T(n/2) + 2$
$a = 1$, $b = 2$, $f(n) = 2$ then $T(n) = O(\log n)$

10

Master Theorem

The recurrence $T(n) = aT(n/b) + f(n)$ can be solved as follows:
If $a \cdot f(n/b) = kf(n)$ for some constant $k < 1$, then $T(n) = \Theta(f(n))$
If $a \cdot f(n/b) = Kf(n)$ for some constant $K > 1$, then $T(n) = \Theta(n^{\log_b a})$
If $a \cdot f(n/b) = f(n)$, then $T(n) = \Theta(f(n)\log_b n)$

You should check that this works for the recurrences we've seen here.