Divide and Conquer

Kleinberg and Tardos 5.1, 5.2

Divide-and-Conquer Strategy:
- Break up problem into parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

MergeSort

- Divide array into two halves.
- Recursively sort each half.
- Merge two halves to make sorted whole.

A Recurrence Relation for MergeSort

T(n) = number of comparisons required to mergesort an input of size n.

Mergesort recurrence.

\[ T(n) = \begin{cases} \tau \left( \left\lfloor \frac{n}{2} \right\rfloor \right) + T\left( \left\lceil \frac{n}{2} \right\rceil \right) + c0 & \text{if } n = 1 \\ \tau \left( \frac{n}{2} \right) + T\left( \frac{n}{2} \right) + cn & \text{otherwise} \end{cases} \]

Solution. T(n) = O(n log_2 n).

Assorted proofs. We describe several ways to prove this. We assume n is a power of 2 and replace \( n \) with \( 2^k \).

Unrolling the recursion
Proof by repeated substitution

**Claim.** If $T(n)$ satisfies this recurrence, then $T(n) = cn \log_2 n$.

\[
T(n) = \begin{cases} 
    \frac{cn}{2}, & \text{if } n = 1; \\
    2T(n/2) + cn, & \text{otherwise}.
\end{cases}
\]

**Proof.** For $n > 1$:

1. $T(n) = 2T(n/2) + cn$
2. $T(n/2) = 4T(n/4) + c(n/2)$
3. $T(n/4) = 8T(n/8) + c(n/4)$

\[
T(n) = 2^{\log_2 n} - 1 \cdot c(n) + cn - 2^{\log_2 n} - 1 
= O(n \log_2 n).
\]

Another example

$T(1)$ = 1
$T(n) = 3T(n/2) + n$

Proof by Induction

**Claim.** If $T(n)$ satisfies this recurrence, then $T(n) = cn \log_2 n$.

\[
T(n) = \begin{cases} 
    \frac{cn}{2}, & \text{if } n = 1; \\
    2T(n/2) + cn, & \text{otherwise}.
\end{cases}
\]

**Proof.** (by induction on $n$)

- **Base case:** $n = 1$.
- **Induction hypothesis:** $T(m) = cm \log_2 m$ for $m < n$.

\[
T(n) = 2T(n/2) + cn
= \left( \frac{cn}{2} + cn \log_2 \left( \frac{n}{2} \right) \right)
= c \log_2 n \cdot \frac{n}{2} + cn
= O(n \log_2 n).
\]

The Master Theorem

Let $f$ be an increasing function that satisfies

\[
f(n) = a \cdot f(n/b) + c \cdot n^d
\]

whenever $n = b^k$, where $k$ is a positive integer, $a \geq 1$, $b > 1$ is an integer $> 1$, and $c$ and $d$ are real numbers with $c$ positive and $d$ nonnegative. Then

\[
f(n) = \begin{cases} 
    \mathcal{O}(n^d), & \text{if } a < b^d; \\
    \mathcal{O}(n^{d+l}), & \text{if } a = b^d; \\
    \mathcal{O}(n^{d+1}), & \text{if } a > b^d.
\end{cases}
\]

From section 7.3 in Rosen

Binary Search using the Master Theorem

\[
f(n) = a \cdot f(n/b) + cn^d
\]

\[
a = \frac{1}{b - 1},
\]

\[
b = \frac{4}{3},
\]

\[
d = 0
\]

\[
f(n) = \begin{cases} 
    \mathcal{O}(n^d), & \text{if } a < b^d; \\
    \mathcal{O}(n^{d+l}), & \text{if } a = b^d; \\
    \mathcal{O}(n^{d+1}), & \text{if } a > b^d.
\end{cases}
\]
mergesort: Recurrence Analysis

\[ f(n) = a \cdot f(n/b) + cn^d \]

- \( a = \log_b n \)
- \( b = 2 \)
- \( d = 1 \)
- \( \Omega(\log \log n) \) if \( a > b \)
- \( \Omega(n^{\log_b a}) \) if \( a = b \)
- \( \Omega(n^a) \) if \( a < b \)

Finding maximum in unsorted array

Algorithm:
- If \( n=1 \), then element is the max.
- If \( n>1 \), divide array in half, find max of each and choose max of the two

Recurrence relation?

\[ f(n) = a \cdot f(n/b) + cn^d \]

Towers of Hanoi

Towers of Hanoi: move all disks to third peg without ever placing a larger disk on a smaller one.

What's the recurrence relation?

Can we solve it using the Master Theorem?

\[ f(n) = a \cdot f(n/b) + cn^d \]