Divide and Conquer

Recurrence Relations
Divide-and-Conquer

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

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

Jon von Neumann (1945)

```
divide    O(1)
sort      2T(n/2)
merge     O(n)
```
Complexity of merge

**time**

$O(n)$

**space**

$O(n)$

Often done having 2 $n$ sized arrays

Can you do it in less than $2n$ space?
A Recurrence Relation for MergeSort

\[ T(n) = \text{number of comparisons required to mergesort an input of size } n. \]

**MergeSort recurrence.**

\[
T(n) \leq \begin{cases} 
  c & \text{if } n = 1 \\
  T\left(\lfloor n/2 \rfloor\right) + T\left(\lceil n/2 \rceil\right) + cn & \text{otherwise}
\end{cases}
\]
A recurrence relation for the sequence \( \{a_n\} \) is an equation that expresses \( a_n \) in terms of one of more of the previous terms of the sequence, namely, \( a_0, a_1, \ldots, a_{n-1} \), for all integers \( n \) with \( n \geq n_0 \) where \( n_0 \) is a nonnegative integer.

A sequence is defined by a recurrence relation + initial conditions ("base cases")

Example: Towers of Hanoi:

\[
a_n = 2a_{n-1} + 1, \quad a_1 = 1
\]
A Recurrence Relation for MergeSort

\[ T(n) = \text{number of comparisons required to mergesort an input of size } n. \]

Merge sort recurrence.

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

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

Assorted proofs. We describe several ways to prove this recurrence. We assume \( n \) is a power of 2 and replace \( \leq \) with = (we only care about the order of magnitude)
Unrolling the recursion

\[
T(n) = \begin{cases} 
  c & \text{if } n = 1 \\
  \frac{2}{n} T\left(\frac{n}{2}\right) + cn & \text{otherwise}
\end{cases}
\]

\[
T(n) = c + \left(\frac{2}{2}\right) T\left(\frac{n}{2}\right) + \left(\frac{2}{4}\right) T\left(\frac{n}{4}\right) + \frac{cn}{2} + \frac{cn}{4}
\]

\[
T(n / 2^k) = T(1)
\]

\[
n / 2^k = 1 \text{ when } k = \log_2 n
\]
Repeated substitution

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

\[
T(n) = \begin{cases} 
    c & \text{if } n = 1 \\
    \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{cn}_{\text{merging}} & \text{otherwise}
\end{cases}
\]

\[
T(n) = \begin{array}{c}
    2T(n/2) + cn \\
    4T(n/4) + cn + 2cn/2 \\
    8T(n/8) + cn + cn + 4cn/4 \\
    \vdots \\
    2^{\log_2 n} T(1) + cn + \cdots + cn \\
\end{array}
\]

This reaches $T(1)$ when $n = 2^{\log_2 n}$ by definition of $\log_2 n$

\[= O(n \log_2 n)\]
Example: Towers of Hanoi, move all disks to third peg without ever placing a larger disk on a smaller one.

What’s the recurrence relation?

Let’s solve it by repeated substitution:

- Unroll the recurrence
- Identify a pattern
- Determine how often the pattern occurs before base case is hit, and sum over all the levels of the recursion
Hanoi by repeated substitution

\[ f_1 = 1 \]
\[ f_n = 2f_{n-1} + 1 = 2(2f_{n-2} + 1) + 1 = 4f_{n-2} + 2 + 1 = 4(2f_{n-3} + 1) + 2 + 1 = \]
\[ = 8f_{n-3} + 4 + 2 + 1 \]
\[ = 2^3f_{n-3} + \sum_{i=0}^{2} 2^i = 2^4f_{n-4} + \sum_{i=0}^{3} 2^i = 2^k f_{n-k} + \sum_{i=0}^{k-1} 2^i \]

After \( n-1 \) substitutions, \( k = n-1, \quad f_{n-(n-1)} = f_1 = 1, \) and then

\[ 2^{n-1}f_1 + \sum_{i=0}^{n-2} 2^i = 2^{n-1} + \sum_{i=0}^{n-2} 2^i = 2^n - 1 = O(2^n) \]

This is a geometric series. What is the recurrence for the series and the formula for its sums?
Repeated substitution for Binary Search

What’s the recurrence relation for binary search?

Apply repeated substitution to solve it.
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?
Solve by repeated substitution

\[
f(n) = 2f(n/2)+1 = 4f(n/4) + 2 + 1 = \ldots 2^k(f(n/2^k)) + 2^{k-1} + 2^{k-2} + \ldots + 1 = 2.2^{k-1} = 2n-1
\]

when \( k = \log_2 n \) \( n=2^k \) and \( f(n/2^k) = f(1)=1 \)

STUDY YOUR LOGs (see Orders of magnitude lecture notes)

Useful trick: \( y^{\log x} = x^{\log y} \)

Again: \( x^0 + x^1 + \ldots + x^n = (x^{n+1}-1)/(x-1) \) (geometric series)
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, $a \geq 1$, $b$ is an integer $> 1$, and $c$ and $d$ are real numbers with $c$ positive and $d$ nonnegative. Then

$$f(n) = \begin{cases} 
O(n^d) & \text{if } a < b^d \\
O(n^d \log n) & \text{if } a = b^d \\
O(n^{\log_b a}) & \text{if } a > b^d 
\end{cases}$$

*From section 7.3 in Rosen*
mergesort: Recurrence Analysis

\[ f(n) = a \cdot f\left(\frac{n}{b}\right) + cn^d \]

\[
a = \\
b = \\
d = \\
O(?)
\]

\[
f(n) = \begin{cases} 
O(n^d) & \text{if } a < b^d \\
O(n^d \log n) & \text{if } a = b^d \\
O(n^{\log_b a}) & \text{if } a > b^d 
\end{cases}
\]

Now do WA2