Divide & Conquer Algorithms.

Often we solve problems by
- breaking up into subproblems of some kind
- solving each subproblem
- combining results to get final answer

This strategy => 'Divide and Conquer Algorithm'

q: Binary search

\[ T(n) = T\left(\frac{n}{2}\right) + 1 \]

\[ T(n) = \text{work required to sort the list of } n \text{ elements} \]

A recurrence relation.

- \( T(n) \) is work required to sort a list of size \( n \).
- \( T(n) = T\left(\frac{n}{2}\right) + 1 \) for \( n > 1 \), base case when \( n = 1 \).

In general, D&C algorithms have recurrences like

\[ T(n) = a T\left(\frac{n}{b}\right) + c n^i \]

In general, \( T(n) \) is a complicated function of \( n \), but behaves roughly if \( n \rightarrow \infty \).

- We need enough to find the asymptotic complexity.

If we define

\[ T(0) = T(1) = t_j \]

\[ T(n) = T(\frac{n}{2}) + t_j \]

\[ t_j = a t_{j-1} + c b^i \]

Where: const. coeff., order linear difference eq.

From above,

\[ t_0 = a \]

\[ t_j = \frac{a^j + c b^i j}{a^j - c b^i} \]

Suppose \( a > b \) so \( 1 + x + x^2 + \ldots + x^{a-1} = \frac{1 - x^a}{1 - x} \)

\[ t_j = a^j + c b^i \frac{a^j - c b^i}{a^j - c b^i} \]

\[ = \frac{a^j + c b^i (a^j - b^i)}{a - b^i} \]
1. \( T(n) = T(n/2) + c \) 
\[ b = 2, a = 1, c = 0 \] 
\[ \Rightarrow a + 1 = b \cdot c = 2 \] 
\[ \Rightarrow T(n) \sim n \log_2 n \]

2. \( \max/\min \) 
\[ T(n) = 2 \cdot T(n/2) + c \] 
\[ b = 2, a = 2, c = 0 \] 
\[ \Rightarrow a > b \] 
\[ \Rightarrow T(n) \sim n \log_2 n \]

3. Merge sort 
\[ T(n) = 2 \cdot T(n/2) + n \] 
\[ b = 2, a = 2, c = 1 \] 
\[ \Rightarrow a = b \] 
\[ \Rightarrow T(n) \sim n \log_2 n \]

\[ \text{Summary (w/ other cases)} \]
\[ a = b \] 
\[ T(n) \sim n \text{ undefined} \]
\[ a < b \] 
\[ T(n) \sim n \]
\[ a > b \] 
\[ T(n) \sim n \log_2 n \]

\[ \text{Matrix multiplication} \]

Given two \( n \times n \) matrices \( A, B \)

Form \( C = AB \)

Standard algorithm 
\[ C_{ij} = \sum_k A_{ik} B_{kj} \]
\[ \sim n^2 \text{ elements} \]
\[ \sim n^3 \text{ multiplications} \]

To avoid \( \Theta(n^3) \) algorithm? 

Split each matrix into submatrices of size \( \frac{\sqrt{n}}{2} \times \frac{\sqrt{n}}{2} \)

\[ A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \quad C = \ldots \]

\[ \begin{align*}
C_{11} &= A_{11} B_{11} + A_{12} B_{21} \\
C_{12} &= A_{11} B_{12} + A_{12} B_{22} \\
C_{21} &= A_{21} B_{11} + A_{22} B_{21} \\
C_{22} &= A_{21} B_{12} + A_{22} B_{22} 
\end{align*} \]

\[ \Rightarrow \text{need } 8 \times \frac{\sqrt{n}}{2} = \frac{2n^{3/2}}{2} \text{ adds to form } C \]

\[ \text{However, } \]
\[ G(7) = \begin{align*}
C_{11} &= P + S - T + V \\
C_{12} &= R + T \\
C_{21} &= Q + S \\
C_{22} &= P + R - Q + U
\end{align*} \]
so now
\[ T(n) = \frac{1}{7} T\left(\frac{n}{7}\right) + Cn^2 \]
\[ b = 7, \ a = 7, \ c = 2 \]
\[ \alpha > \beta^2 \]
\[ \Rightarrow T(n) \sim n^{\log_7 2} = n^{0.78} = 2.81 \]
so asymptotically (for large n)

This will be faster than the standard way

\[ \lim_{n \to \infty} \frac{Cn^2}{n^{2.81}} = 0 \quad \forall \ C > 0 \]

Notes
1. Can prove that you can't enforce on this by solving the subproblems.
2. Current world record in
\[ T(n) = Cn \quad n \approx 1.496 \]
but a C is too large it's not a practical algorithm
3. Need to know the constant with the best test if algorithm is useful