Another Example of Divide and Conquer.

The Maximum Sum Sub array Problem (MSSP): Given an array $A$ of $n$ numbers find a segment $A[i, j]$ so that the sum of numbers in this segment is the largest among all array segments. In an easier version of MSSP, we are given $A$, and an index $k$, and are asked to find a segment $A[i, j]$ of maximum sum so that $i \leq k \leq j$. So we are forcing $A[k]$ to be in the solution. We refer to this problem as FMSSP. Note that FMSSP can be solved in linear time.
A divide and conquer Algorithm for MSSP

MSSP(L,H)
If L < H Then
M:=\((L+H)\text{Div }2\)
X ← MSSP(L, M)
Y ← MSSP(M + 1, H)
Z ← FMSSP(L, M, H)
MSSP ← Maximum\{X, Y, Z\}
EndIf
Else MSSP ← A[L]
End.

Time complexity:

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

\[T(n) = \Theta(n \log n)\]
Recursive matrix mult.

\[ C = A \times B \]

\[ C_{11} = A_{11}B_{11} + A_{12}B_{21} \]
\[ C_{12} = A_{11}B_{12} + A_{12}B_{22} \]
\[ C_{21} = A_{21}B_{11} + A_{22}B_{21} \]
\[ C_{22} = A_{21}B_{12} + A_{22}B_{22} \]

Time complexity?

\[
T(n) = 8T(n/2) + cn^2, \ n > 2, \ T(2) = b
\]

\[ T(n) = O(n^3) \]
Assume that we have an algorithm for multiply two $2 \times 2$ matrices using only 7 scalar multiplications. Use this algorithm to Design an algorithm for multiplying $2 \times n \times n$ matrices in $O()$ time.

Let $A$ and $B$ the $n \times n$ matrices. We partition $A$ and $B$ each to 4 matrices as above. We using the given algorithm to compute $C_{11}$, $C_{12}$, $C_{21}$ and $C_{22}$ using only 7 matrix multiplications.

Then,

$$T(n) = 7T(n/2) + O(n^2).$$

The answer is

$$T(n) = O(n^{\log 7}).$$
Greedy Algorithms

The Activity Selection Problem.
Assume that we have a set of \( n \) activities. Activity \( i, i = 1, 2, 3, n \) has the start time \( s_i \) and finish time \( f_i \), and we want to find the largest set of mutually disjoint (compatible) activities. The algorithm first sort the activities with respect to finish time, and then applies the greedy strategy. That is, the algorithm first picks the activity with smallest finish time, then the next activity (in the sorted list) that is compatible with the first, and so on.
Greedy Algorithms

Lemma

Let $a_1$ be the first activity selected by the greedy algorithm, then there is an optimal solution $O^1$ that contains $a_1$.

Proof. Let $O$ be an optimal solution, with the first activity $a_j, j \geq 1$. Define

$$O^1 = (O - \{a_j\}) \cup a_1$$

Let $A$ be the set of activities that the greedy algorithm computes. We can prove that the greedy algorithm is computing an optimal solution by induction on $|A|$ and employing the Lemma.