Next topic: Divide & Conquer

Classic example: Merge Sort

Input: array $A = [a_1, \ldots, a_n]$ of $n$ numbers

Output: sorted array

Idea: split into 2 sublists,
recursively sort each sublist,
and then merge the sorted sublists.

\[
\text{MergeSort}(A): \quad \begin{cases}
\text{if } n=1, \text{ return } (A) \\
\text{if } n>1: \\
B = [a_1, \ldots, a_{n/2}], \quad C = [a_{n/2+1}, \ldots, a_n] \\
D = \text{MergeSort}(B) \\
E = \text{MergeSort}(C) \\
F = \text{Merge}(D, E) \\
\text{Return}(F)
\end{cases}
\]
Merge: takes two sorted arrays \( X \) & \( Y \) and outputs their sorted union \( Z \).

\( X_1 \) is smallest of \( X \), & \( Y_1 \) is smallest of \( Y \)

So \( \text{min}(X_1, Y_1) \) is smallest of \( Z \).

Compare \( X_1 \) & \( Y_1 \), paste smaller at beginning of \( Z \), then recurse on remaining lists.

\[ \text{Merge}(X, Y) \]

input: \( X = [x_1, \ldots, x_k] \) & \( Y = [y_1, \ldots, y_l] \)

which are sorted, so \( x_1 \leq x_2 \leq \ldots \leq x_k \)
& \( y_1 \leq y_2 \leq \ldots \leq y_l \)

output: \( Z = X U Y \) in sorted order.

if \( k = 0 \), return \( Y \)
if \( l = 0 \), return \( X \)

if \( x_1 \leq y_1 \)
then \( Z = [x_1, \text{Merge}([x_2, \ldots, x_k], Y)] \)
else \( Z = [y_1, \text{Merge}(X, [y_2, \ldots, y_l])] \)

Return \( Z \)
Running time:

Merge: takes $O(1)$ time per entry

\[ \Rightarrow O(k+l) \text{ total time.} \]

For MergeSort,

let $T(n)$ = running time of MergeSort on worst case input for $n$ numbers.

Then,

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

Base case, $T(1) = O(1)$.

We'll see: this solves to

\[ T(n) = \Theta(n \log n) \]
Dynamic Programming:
Write recursive formula, but often use slightly smaller subproblems.
Example: express $T(i)$ in terms of $T(i-1)$ and $T(i-2)$.
So a recursive algorithm will blow-up small subproblems will be solved too many times.
Hence, use iterative algorithm to solve bottom-up.

Divide & Conquer:
Express solution to problem of size $n$ in terms of subproblems of size $cn$ for $c<1$.
Example: Mergesort used $\frac{n}{2}$-sized subproblems, even $\frac{n}{\log n}$ is ok.
Use recursion to solve subproblems.
Then "Combine" solutions to subproblems to get solution to original problem of size $n$. 
Manipulating logs:

$$\log_b n = O(\log n)$$ \text{ when base not specified} \ 
\text{its base } 2.

base e \rightarrow

$$\ln n = O(\log n)$$

$$2^{\log n} = n$$

But is $$2^{\log_3 n} = O(n)$$? \text{No!} \ 
\text{in the exponent the base matters.}

How to reexpress $$2^{\log_3 n}$$ as a polynomial
\text{means in the form } n^c \ \ 
\text{exponent is a constant, non-zero, like } 1, 2, 1.3, \text{ etc.}

\text{idea: reexpress so this number}
\text{matches this one}

Key: \(2 = 3^{\log_3 2}\ \ 
(x^y)^z = x^{yz} = (x^z)^y \approx 1.585\)

$$2^{\log_3 n} = (3^{\frac{\log_3 n}{\log_3 2}}) = \left(3^{\log_3 n}\right)^{\log_2 3} = n$$