Divide and Conquer
Divide and Conquer:
a (somewhat) general theorem

The setup: a recursive algorithm that on inputs of size $n \geq n_0$, recursively solves

- $\leq a$ smaller sub-problems,
- each of size $\leq n/b + c$,
- with a “local” running time $\leq dn^e$

where $n_0, a, b, c, d, e$ are constants

"$T(n) \leq aT(n/b + c) + O(n^e)$"

**Simplification:** assume $c = 0$

**General case:** exercise
Recursion tree analysis

At level 1, size ≤ \( n/b \)

At level 2, size ≤ \( n/b^2 \)

... 

At level \( j \), size ≤ \( n/b^j \)

At level \( j \), there are ≤ \( a^j \) nodes

Set \( k := \lfloor \log_b n \rfloor \), so \( n ≤ b^k < bn \)

Level \( k \) is the last level in the tree

Let \( w = \) sum of costs at levels \( 0, \ldots, k \)

For each \( j = 0 \ldots k \), sum of costs at level \( j \) is

\[ \leq a^j \cdot d(n/b^j)^e = d \cdot n^e(a/b^e)^j \]
Therefore,

\[ w \leq d \cdot n^e \sum_{j=0}^{k} \delta^j, \]

where \( \delta := a/b^e \)

**Case 1: \( \delta < 1 \)**

\[ \sum_{j=0}^{\infty} \delta^j = 1/(1 - \delta) \implies w \leq (d/(1 - \delta))n^e \]

Total running time = \( O(n^e) \)

**Case 2: \( \delta = 1 \)**

\[ \sum_{j=0}^{k} \delta^j = (k + 1) \implies w \leq d(k + 1)n^e \]

Total running time = \( O(n^e \log n) \)
Case 3: $\delta > 1$

$$\sum_{j=0}^{k} \delta^j = \frac{\delta^{k+1} - 1}{\delta - 1}$$

and so for some constant $C$, we have

$$w \leq Cn^e \delta^k = Cn^e a^k / (b^k)^e \leq Ca^k$$

$$\leq Ca^{\log_b n + 1} = Ca \cdot a^{\log_b n}$$

$$= Ca \cdot b^{\log_b a \cdot \log_b n}$$

$$= Ca \cdot n^{\log_b a}$$

Total running time $= O(n^{\log_b a})$
Summarizing — the “Master Theorem”

Let $f := \log_b a$

**Case 1:** $e > f \implies O(n^e)$

**Case 2:** $e = f \implies O(n^e \log n)$

**Case 3:** $e < f \implies O(n^f)$
Application: faster multiplication

Problem: multiply two $n$-bit integers

An “$n$-bit integer” is an integer $a$ such that $0 \leq a < 2^n$

An $n$-bit integer can be represented using an array of $n$ bits

In practice, one packs several bits into a “word”
Addition of $n$-bit integers

The sum of two $n$-bit integers is an $(n + 1)$-bit integer, and can be computed in time $O(n)$

input: $a = (a_{n-1}, \ldots, a_0)$, $b = (b_{n-1}, \ldots, b_0)$
output: $c = (c_n, c_{n-1}, \ldots, c_0)$

$carry \leftarrow 0$
for $i$ in $[0..n)$ do
    $t \leftarrow a_i + b_i + carry$
    $c_i \leftarrow t \mod 2$
    $carry \leftarrow \lfloor t/2 \rfloor$
$c_n \leftarrow carry$
Multiplication of \( n \)-bit integers

The product of two \( n \)-bit integers is a \((2n)\)-bit integer, and can be computed in time \( O(n^2) \)

input: \( a = (a_{n-1}, \ldots, a_0) \), \( b = (b_{n-1}, \ldots, b_0) \)
output: \( c = (c_{2n-1}, \ldots, c_0) \)

initialize \( c_i \leftarrow 0 \) for \( i \) in \([0..2n)\)

for \( i \) in \([0..n)\) do

    // \( c \leftarrow c + 2^i b_i \cdot a \)
    carry \leftarrow 0

    for \( j \) in \([0..n)\) do

        \( t \leftarrow c_{i+j} + b_i \cdot a_j + \text{carry} \)
        \( c_{i+j} \leftarrow t \mod 2 \)
        \( \text{carry} \leftarrow \lfloor t/2 \rfloor \)

    \( c_{i+n} \leftarrow \text{carry} \)
Karatsuba’s multiplication algorithm

Input: two $n$-bit integers, $a$ and $b$

If $n$ is “very small”, use the naive algorithm.

Otherwise, divide each number into two pieces:

$$a = a_{hi}2^k + a_{lo}$$

$$b = b_{hi}2^k + b_{lo},$$

where $k := \lfloor n/2 \rfloor$

<table>
<thead>
<tr>
<th>$a$:</th>
<th>$a_{hi}$</th>
<th>$a_{lo}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$:</td>
<td>$b_{hi}$</td>
<td>$b_{lo}$</td>
</tr>
</tbody>
</table>
\[ ab = a_{hi}b_{hi}2^{2k} + (a_{hi}b_{lo} + a_{lo}b_{hi})2^k + a_{lo}b_{lo} \]

\[ \approx n \]

\[ a_{hi}b_{hi} \]

\[ a_{hi}b_{lo} + a_{lo}b_{hi} \]

\[ a_{lo}b_{lo} \]

\[ ab \]

\[ \approx 2n \]
One idea:
Recursively compute the four sub-products
\[ a_{hi}b_{hi}, \ a_{hi}b_{lo}, \ a_{lo}b_{hi}, \ a_{lo}b_{lo} \]
Case 3 of Master Theorem: \( e = 1, f = \log_2 4 = 2 \)
\[ \implies \text{another } O(n^2) \text{ algorithm} \]

A better idea:
Compute \( A \leftarrow a_{hi} + a_{lo}, \ B \leftarrow b_{hi} + b_{lo} \)
Recursively compute three products:
\[ H \leftarrow a_{hi}b_{hi}, \ L \leftarrow a_{lo}b_{lo}, \ F \leftarrow AB \]
Observations:
\[ F = a_{hi}b_{hi} + a_{hi}b_{lo} + a_{lo}b_{hi} + a_{lo}b_{lo} \]
\[ M := F - (H + L) = a_{hi}b_{lo} + a_{lo}b_{hi} \]
\[ P := H2^{2k} + M2^k + L = ab \]
Case 3 of Master Theorem: \( e = 1, f = \log_2 3 \approx 1.585 \)
Running time is \( O(n^{\log_2 3}) \)
Notes:

• Karatsuba is *not* the fastest method: using the Fast Fourier Transform, one can multiply two $n$-bit integers in time $O(n \log n \log \log n)$

• For $n$ (roughly) in the range 500–10,000, Karatsuba is the fastest

• You use it every time you buy something from amazon.com, or use ssh — it’s used to implement public-key cryptosystems