sometimes two sub-problems aren’t enough

- More general divide and conquer
  - You’ve broken the problem into $a$ different sub-problems
  - Each has size at most $n/b$
  - The cost of the break-up and recombining the sub-problem solutions is $O(n^k)$

- Recurrence: $T(n) \leq a \cdot T(n/b) + c \cdot n^k$

master divide and conquer recurrence

- If $T(n) \leq a \cdot T(n/b) + c \cdot n^k$ for $n > b$ then
  - if $a > b^k$ then $T(n)$ is $\Theta(n^{\log_b a})$
  - if $a < b^k$ then $T(n)$ is $\Theta(n^k)$
  - if $a = b^k$ then $T(n)$ is $\Theta(n^k \log n)$

- Works even if it is $\left\lfloor \frac{n}{b} \right\rfloor$ instead of $\frac{n}{b}$. 
proving the master recurrence

Problem size \( T(n) = a \cdot T(n/b) + c \cdot n^k \)  \# probs

geometric series

- \( S = t + tr + tr^2 + \ldots + tr^{n-1} \)
- \( rS = tr + tr^2 + \ldots + tr^{n-1} + tr^n \)
- \( (r-1)S = tr^n - t \)
- so \( S = t \frac{r^n - 1}{r - 1} \) if \( r \neq 1 \).

Simple rule

- If \( r \neq 1 \) then \( S \) is a constant times largest term in series

total cost

- Geometric series
  - ratio \( a/b^k \)
  - \( d + 1 = \log_b n + 1 \) terms
  - first term \( cn^k \), last term \( ca^d \)
- If \( a/b^k = 1 \)
  - all terms are equal \( T(n) \) is \( \Theta(n^k \log n) \)
- If \( a/b^k < 1 \)
  - first term is largest \( T(n) \) is \( \Theta(n^k) \)
- If \( a/b^k > 1 \)
  - last term is largest \( T(n) \) is

\[ \Theta(a^d) = \Theta(a^{\log n}) = \Theta(n^{\log a}) \]
proving the master recurrence

Problem size \( T(n) = a \cdot T(n/b) + c \cdot n^k \) # probs

<table>
<thead>
<tr>
<th>n</th>
<th>a</th>
<th>c ( n^k )</th>
</tr>
</thead>
<tbody>
<tr>
<td>n/b</td>
<td>a</td>
<td>( c \cdot n^k \left( \frac{a}{b^k} \right) )</td>
</tr>
<tr>
<td>n/b^2</td>
<td>a^2</td>
<td>( c \cdot n^k \left( \frac{a}{b^k} \right)^2 )</td>
</tr>
<tr>
<td>b</td>
<td>a^d</td>
<td>( c \cdot n^k \left( \frac{a}{b^k} \right)^d )</td>
</tr>
</tbody>
</table>

\( T(1) = c \)

\[ n^3 \text{ multiplications, } n^3 - n^2 \text{ additions} \]
Fastest algorithms theoretically use $O(n \log n)$ time but practical algorithms use $O(n^{2.8})$ which is $O(n^{2.8})$.

- $T(n) = 7(n/2)^2 + cn^2$
- $T(2) = 7(2/2)^2 + c(2)^2$ so $T(n)$ is $O(n^{2.8})$ which is $O(n^{2.8})$.

- Strassen's algorithm has size somewhere between $10$ and $100$.

- Multiply $2 \times 2$ matrices using $7$ instead of $8$ multiplications and lots more than $4$ additions.

- Divide and conquer algorithm.
### the algorithm (yuck)

<table>
<thead>
<tr>
<th>Step</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>$A_{12}(B_{11} + B_{21})$</td>
</tr>
<tr>
<td>$P_2$</td>
<td>$A_{21}(B_{12} + B_{22})$</td>
</tr>
<tr>
<td>$P_3$</td>
<td>$(A_{11} - A_{12})B_{11}$</td>
</tr>
<tr>
<td>$P_4$</td>
<td>$(A_{22} - A_{21})B_{22}$</td>
</tr>
<tr>
<td>$P_5$</td>
<td>$(A_{22} - A_{12})(B_{21} - B_{22})$</td>
</tr>
<tr>
<td>$P_6$</td>
<td>$(A_{11} - A_{21})(B_{12} - B_{11})$</td>
</tr>
<tr>
<td>$P_7$</td>
<td>$(A_{21} - A_{12})(B_{11} + B_{22})$</td>
</tr>
<tr>
<td>$C_{11}$</td>
<td>$P_1 + P_3$</td>
</tr>
<tr>
<td>$C_{12}$</td>
<td>$P_2 + P_3 + P_6 - P_7$</td>
</tr>
<tr>
<td>$C_{21}$</td>
<td>$P_1 + P_4 + P_5 + P_7$</td>
</tr>
<tr>
<td>$C_{22}$</td>
<td>$P_2 + P_4$</td>
</tr>
</tbody>
</table>

### multiplying faster

- If you analyze our usual grade school algorithm for multiplying numbers
  - $O(n^2)$ time
  - On real machines each “digit” is, e.g., 32 bits long but still get $O(n^2)$ running time with this algorithm when run on $n$-bit multiplication
- We can do better!
  - We’ll describe the basic ideas by multiplying polynomials rather than integers
  - Advantage is we don’t get confused by worrying about carries at first

### notes on polynomials

These are just formal sequences of coefficients
- when we show something multiplied by $x^k$ it just means shifted $k$ places to the left – basically no work

### Usual polynomial multiplication

<table>
<thead>
<tr>
<th>Coefficients</th>
<th>Polynomial</th>
</tr>
</thead>
<tbody>
<tr>
<td>4x² + 2x + 2</td>
<td>$x^2 - 3x + 1$</td>
</tr>
<tr>
<td>x² - 3x + 1</td>
<td>4x² + 2x + 2</td>
</tr>
<tr>
<td>-12x³ - 6x² - 6x</td>
<td>4x⁴ + 2x³ + 2x²</td>
</tr>
<tr>
<td>4x⁴ + 2x³ + 2x²</td>
<td>4x⁴ - 10x³ + 0x² - 4x + 2</td>
</tr>
</tbody>
</table>

### polynomial multiplication

- **Given:**
  - Degree $n-1$ polynomials $P$ and $Q$
    - $P = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-2} x^{n-2} + a_{n-1} x^{n-1}$
    - $Q = b_0 + b_1 x + b_2 x^2 + \ldots + b_{n-2} x^{n-2} + b_{n-1} x^{n-1}$
- **Compute:**
  - Degree $2n-2$ Polynomial $P \cdot Q$
    - $P \cdot Q = a_0 b_0 + (a_0 b_2 + a_1 b_1) x + (a_0 b_2 + a_1 b_2 + a_2 b_0) x^2 + \ldots + (a_{2n-3} b_{n-1} + a_{2n-2} b_n) x^{2n-3} + a_{2n-1} b_{n-1} x^{2n-2}$
- **Obvious Algorithm:**
  - Compute all $a_i b_j$ and collect terms
  - $O(n^2)$ time
naive divide and conquer

- Assume \( n = 2k \)
  \[
  P = (a_0 + a_1 x + a_2 x^2 + \ldots + a_{k-2} x^{k-2} + a_{k-1} x^{k-1}) + \]
  \[
  (a_k + a_{k+1} x + \ldots + a_n x^{n-k}) x^k
  \]
  \[
  = P_0 + P_1 x^k
  \]
  where \( P_0 \) and \( P_1 \) are degree \( k-1 \) polynomials
- Similarly \( Q = Q_0 + Q_1 x^k \)
- \( PQ = (P_0 + P_1 x^k)(Q_0 + Q_1 x^k) \)
  \[
  = P_0 Q_0 + (P_1 Q_0 + P_0 Q_1) x^k + P_1 Q_1 x^{2k}
  \]

Karatsuba's algorithm

A better way to compute the terms

\[
 PQ = (P_0 + P_1 x^k)(Q_0 + Q_1 x^k)
\]

\[
 = P_0 Q_0 + (P_1 Q_0 + P_0 Q_1)x^k + P_1 Q_1 x^{2k}
\]

- Compute
  \[
  A \leftarrow P_0 Q_0
  \]
  \[
  B \leftarrow P_1 Q_1
  \]
  \[
  C \leftarrow (P_0 + P_1)(Q_0 + Q_1) = P_0 Q_0 + P_1 Q_0 + P_0 Q_1 + P_1 Q_1
  \]
- Then
  \[
  P_0 Q_1 + P_1 Q_0 = C - A - B
  \]
  So \( PQ = A + (C - A - B)x^k + B x^{2k} \)

Karatsuba: details

PolyMul(P, Q):

- \( P, Q \) are length \( n = 2k \) vectors, with \( P[0], Q[0] \) being the coefficient of \( x^0 \) in polynomials \( P, Q \) respectively.
- Let \( P_{zero} \) be elements \( 0..k-1 \) of \( P \); \( P_{one} \) be elements \( k..n-1 \)
- \( Q_{zero}, Q_{one} \) : similar
- If \( n = 1 \) then Return(\( P[0] Q[0] \))
- else
  \[
  A \leftarrow\text{PolyMul}(P_{zero}, Q_{zero}); \quad \text{result is a (2k-1)-vector}
  \]
  \[
  B \leftarrow\text{PolyMul}(P_{one}, Q_{one}); \quad \text{ditto}
  \]
  \[
  P_{sum} \leftarrow P_{zero} + P_{one}; \quad \text{add corresponding elements}
  \]
  \[
  Q_{sum} \leftarrow Q_{zero} + Q_{one}; \quad \text{ditto}
  \]
  \[
  C \leftarrow\text{PolyMul}(P_{sum}, Q_{sum}); \quad \text{another (2k-1)-vector}
  \]
  \[
  Mid \leftarrow C - A - B; \quad \text{subtract corresponding elements}
  \]
  \[
  R \leftarrow A + \text{Shift}(\text{Mid} \cdot n/2) + \text{Shift}(B.n) \quad \text{a (2n-1)-vector}
  \]
  Return(\( R) \)

Multiplication

- Polynomials
  - Naive: \( \Theta(n^2) \)
  - Karatsuba: \( \Theta(n^{1.585}) \)
  - Best known: \( \Theta(n \log n) \)
    "Fast Fourier Transform"
    FFT widely used for signal processing

- Integers
  - Similar, but some ugly details re: carries, etc. due to Schonhage-Strassen in 1971 gives \( \Theta(n \log n \log \log n) \)
  - Improvement in 2007 due to Furer gives \( \Theta(n \log n \ 2^{\log^* n}) \)
  - Used in practice in symbolic manipulation systems like Maple