Divide and Conquer: Polynomial Multiplication

Version of October 7, 2014

Outline

Outline:

- Introduction
- The polynomial multiplication problem
- An $O(n^2)$ brute force algorithm
- An $O(n^2)$ first divide-and-conquer algorithm
- An improved divide-and-conquer algorithm
- Remarks

Divide-and-Conquer

Already saw some divide and conquer algorithms

Divide

- Divide a given problem into a or more subproblems (ideally of approximately equal size n/b)

Conquer

- Solve each subproblem (directly if small enough or recursively) $a \cdot T(n/b)$

Combine

- Combine the solutions of the subproblems into a global solution $f(n)$

Cost satisfies $\mathbf{T(n) = aT(n/b) + f(n)}$.

Divide and Conquer Examples

- Two major examples so far
  - Maximum Contiguious Subarray
  - Mergesort
- Both satisfied $T(n) = 2T(n/2) + O(n)$
  - $\Rightarrow T(n) = O(n \log n)$
- Also saw Quicksort
  - Divide and Conquer, but unequal size problems

Divide and Conquer Analysis

Main tool is the *Master Theorem* for soving recurrences of form

$$T(n) = aT(n/b) + f(n)$$

where

- $a \geq 1$ and $b \geq 1$ are constants and
- $f(n)$ is a (asymptotically) positive function.
- Note: Initial conditions $T(1), T(2), \ldots T(k)$ for some k. They don't contribute to asymptotic growth
- n/b could be either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$

$$\mathbf{T(n) = aT(n/b) + f(n)}, \quad \mathbf{c} = \log_{\mathbf{b}} \mathbf{a}$$

- If $f(n) = \Theta(n^{c-\epsilon})$ for some $\epsilon > 0$ then $T(n) = \theta(n^c)$

The Master Theorem

$$\mathbf{T(n) = aT(n/b) + f(n), \quad c = \log_b a}$$

- If $f(n) = \Theta(n^{c-\epsilon})$ for some $\epsilon > 0$ then $T(n) = \theta(n^c)$
    - If $T(n) = 4T(n/2) + n$ then $T(n) = \Theta(n^2)$

The Master Theorem

$$\mathbf{T(n) = aT(n/b) + f(n)}, \quad \mathbf{c} = \log_{\mathbf{b}} \mathbf{a}$$

- If $f(n) = \Theta(n^{c-\epsilon})$ for some $\epsilon > 0$ then $T(n) = \theta(n^c)$
  - If $T(n) = 4T(n/2) + n$ then $T(n) = \Theta(n^2)$
  - If $T(n) = 3T(n/2) + n$ then $T(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.58\ldots})$

The Master Theorem

$$\mathbf{T(n) = aT(n/b) + f(n), \quad c = \log_b a}$$

- If $f(n) = \Theta(n^{c-\epsilon})$ for some $\epsilon > 0$ then $T(n) = \theta(n^c)$
  - If $T(n) = 4T(n/2) + n$ then $T(n) = \Theta(n^2)$
  - If $T(n) = 3T(n/2) + n$ then $T(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.58\ldots})$
- If $f(n) = \Theta(n^c)$ then $T(n) = \Theta(n^c \log n)$

The Master Theorem

$$\mathbf{T(n) = aT(n/b) + f(n), \quad c = \log_b a}$$

- If $f(n) = \Theta(n^{c-\epsilon})$ for some $\epsilon > 0$ then $T(n) = \theta(n^c)$
  - If $T(n) = 4T(n/2) + n$ then $T(n) = \Theta(n^2)$
  - If $T(n) = 3T(n/2) + n$ then $T(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.58\ldots})$
- If $f(n) = \Theta(n^c)$ then $T(n) = \Theta(n^c \log n)$
  - If $T(n) = 2T(n/2) + n$ then $T(n) = \Theta(n \log n)$

$$\mathbf{T(n) = aT(n/b) + f(n)}, \quad \mathbf{c = \log_b a}$$

- If $f(n) = \Theta(n^{c-\epsilon})$ for some $\epsilon > 0$ then $T(n) = \theta(n^c)$
  - If $T(n) = 4T(n/2) + n$ then $T(n) = \Theta(n^2)$
  - If $T(n) = 3T(n/2) + n$ then $T(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.58\ldots})$
- If $f(n) = \Theta(n^c)$ then $T(n) = \Theta(n^c \log n)$
  - If $T(n) = 2T(n/2) + n$ then $T(n) = \Theta(n \log n)$
- If $f(n) = \Theta(n^{c+\epsilon})$ for some $\epsilon > 0$
  and if $af(n/b) \leq df(n)$ for some $d < 1$ and large enough n
  then $T(n) = O(f(n))$

The Master Theorem

$$\mathbf{T(n) = aT(n/b) + f(n)}, \quad \mathbf{c} = \log_{\mathbf{b}} \mathbf{a}$$

- If $f(n) = \Theta(n^{c-\epsilon})$ for some $\epsilon > 0$ then $T(n) = \theta(n^c)$
  - If $T(n) = 4T(n/2) + n$ then $T(n) = \Theta(n^2)$
  - If $T(n) = 3T(n/2) + n$ then $T(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.58\ldots})$
- If $f(n) = \Theta(n^c)$ then $T(n) = \Theta(n^c \log n)$
  - If $T(n) = 2T(n/2) + n$ then $T(n) = \Theta(n \log n)$
- If $f(n) = \Theta(n^{c+\epsilon})$ for some $\epsilon > 0$
  and if $af(n/b) \leq df(n)$ for some $d < 1$ and large enough n
  then $T(n) = O(f(n))$
  - If $T(n) = T(n/2) + n$ then $T(n) = \Theta(n)$

More Master Theorem

There are many variations of the Master Theorem.
Here's one...

- If $T(n) = T(3n/4) + T(n/5) + n$ then $T(n) = \Theta(n)$

There are many variations of the Master Theorem.
Here's one...

- If $T(n) = T(3n/4) + T(n/5) + n$ then $T(n) = \Theta(n)$
- More generally, given constants $\alpha_i > 0$ with $\sum_i \alpha_i < 1$,

  if $T(n) = n + \sum_{i=1}^{k} T(\alpha_i n)$

  then $T(n) = \Theta(n)$.

Objective and Outline

Outline:

- Introduction
- The polynomial multiplication problem
- An $O(n^2)$ brute force algorithm
- An $O(n^2)$ first divide-and-conquer algorithm
- An improved divide-and-conquer algorithm
- Remarks

The Polynomial Multiplication Problem

Definition (Polynomial Multiplication Problem)

Given two polynomials

$$\begin{aligned} A(x) &= a_0 + a_1 x + \cdots + a_n x^n \\ B(x) &= b_0 + b_1 x + \cdots + b_m x^m \end{aligned}$$

Compute the product $A(x)B(x)$

The Polynomial Multiplication Problem

Definition (Polynomial Multiplication Problem)

Given two polynomials

$$\begin{aligned} A(x) &= a_0 + a_1 x + \cdots + a_n x^n \\ B(x) &= b_0 + b_1 x + \cdots + b_m x^m \end{aligned}$$

Compute the product $A(x)B(x)$

Example

$$\begin{aligned} A(x) &= 1 + 2x + 3x^2 \\ B(x) &= 3 + 2x + 2x^2 \\ A(x)B(x) &= 3 + 8x + 15x^2 + 10x^3 + 6x^4 \end{aligned}$$

The Polynomial Multiplication Problem

Definition (Polynomial Multiplication Problem)

Given two polynomials

$$\begin{aligned} A(x) &= a_0 + a_1 x + \cdots + a_n x^n \\ B(x) &= b_0 + b_1 x + \cdots + b_m x^m \end{aligned}$$

Compute the product $A(x)B(x)$

Example

$$\begin{aligned} A(x) &= 1 + 2x + 3x^2 \\ B(x) &= 3 + 2x + 2x^2 \\ A(x)B(x) &= 3 + 8x + 15x^2 + 10x^3 + 6x^4 \end{aligned}$$

- Assume that the coefficients a_i and b_i are stored in arrays $A[0 \ldots n]$ and $B[0 \ldots m]$

The Polynomial Multiplication Problem

Definition (Polynomial Multiplication Problem)

Given two polynomials

$$\begin{aligned} A(x) &= a_0 + a_1 x + \cdots + a_n x^n \\ B(x) &= b_0 + b_1 x + \cdots + b_m x^m \end{aligned}$$

Compute the product $A(x)B(x)$

Example

$$\begin{aligned} A(x) &= 1 + 2x + 3x^2 \\ B(x) &= 3 + 2x + 2x^2 \\ A(x)B(x) &= 3 + 8x + 15x^2 + 10x^3 + 6x^4 \end{aligned}$$

- Assume that the coefficients a_i and b_i are stored in arrays $A[0 \ldots n]$ and $B[0 \ldots m]$
- Cost: number of scalar multiplications and additions

What do we need to compute exactly?

Define

- $A(x) = \sum_{i=0}^{n} a_i x^i$
- $B(x) = \sum_{i=0}^{m} b_i x^i$

What do we need to compute exactly?

Define

- $A(x) = \sum_{i=0}^{n} a_i x^i$
- $B(x) = \sum_{i=0}^{m} b_i x^i$
- $C(x) = A(x)B(x) = \sum_{k=0}^{n+m} c_k x^k$

What do we need to compute exactly?

Define

- $A(x) = \sum_{i=0}^{n} a_i x^i$
- $B(x) = \sum_{i=0}^{m} b_i x^i$
- $C(x) = A(x)B(x) = \sum_{k=0}^{n+m} c_k x^k$

Then

$$c_k = \sum_{0 \le i \le n,\ 0 \le j \le m,\ i+j=k} a_i b_j \quad \text{for all } 0 \le k \le m+n$$

Definition

The vector $(c_0, c_1, \ldots, c_{m+n})$ is the convolution of the vectors $(a_0, a_1, \ldots, a_n)$ and $(b_0, b_1, \ldots, b_m)$

What do we need to compute exactly?

Define

- $A(x) = \sum_{i=0}^{n} a_i x^i$
- $B(x) = \sum_{i=0}^{m} b_i x^i$
- $C(x) = A(x)B(x) = \sum_{k=0}^{n+m} c_k x^k$

Then

$$c_k = \sum_{0 \le i \le n,\ 0 \le j \le m,\ i+j=k} a_i b_j \quad \text{for all } 0 \le k \le m+n$$

Definition

The vector $(c_0, c_1, \ldots, c_{m+n})$ is the convolution of the vectors $(a_0, a_1, \ldots, a_n)$ and $(b_0, b_1, \ldots, b_m)$

While polynomial multiplication is interesting, real goal is to calculate convolutions. *Major* subroutine in digital signal processing

Objective and Outline

Outline:

- Introduction
- The polynomial multiplication problem
- An $O(n^2)$ brute force algorithm
- An $O(n^2)$ first divide-and-conquer algorithm
- An improved divide-and-conquer algorithm
- Remarks

Direct (Brute Force) Approach

To ease analysis, assume $n = m$.

Direct (Brute Force) Approach

To ease analysis, assume $n = m$.

- $A(x) = \sum_{i=0}^{n} a_i x^i$ and $B(x) = \sum_{i=0}^{n} b_i x^i$
- $C(x) = A(x)B(x) = \sum_{k=0}^{2n} c_k x^k$ with

$$c_k = \sum_{0 \le i,j \le n, i+j=k} a_i b_j, \quad \text{for all } 0 \le k \le 2n$$

Direct (Brute Force) Approach

To ease analysis, assume $n = m$.

- $A(x) = \sum_{i=0}^{n} a_i x^i$ and $B(x) = \sum_{i=0}^{n} b_i x^i$
- $C(x) = A(x)B(x) = \sum_{k=0}^{2n} c_k x^k$ with

$$c_k = \sum_{0 \le i,j \le n, i+j=k} a_i b_j, \quad \text{for all } 0 \le k \le 2n$$

Direct approach:

Direct (Brute Force) Approach

To ease analysis, assume $n = m$.

- $A(x) = \sum_{i=0}^{n} a_i x^i$ and $B(x) = \sum_{i=0}^{n} b_i x^i$
- $C(x) = A(x)B(x) = \sum_{k=0}^{2n} c_k x^k$ with

$$c_k = \sum_{0 \le i,j \le n, i+j=k} a_i b_j, \quad \text{for all } 0 \le k \le 2n$$

Direct approach: Compute all c_k's using the formula above

Direct (Brute Force) Approach

To ease analysis, assume $n = m$.

- $A(x) = \sum_{i=0}^{n} a_i x^i$ and $B(x) = \sum_{i=0}^{n} b_i x^i$
- $C(x) = A(x)B(x) = \sum_{k=0}^{2n} c_k x^k$ with

$$c_k = \sum_{0 \le i,j \le n, i+j=k} a_i b_j, \quad \text{for all } 0 \le k \le 2n$$

Direct approach: Compute all c_k's using the formula above

- Total number of multiplications: $\Theta(n^2)$
- Total number of additions: $\Theta(n^2)$
- Complexity: $\Theta(n^2)$

Outline:

- Introduction
- The polynomial multiplication problem
- An $O(n^2)$ brute force algorithm
- An $O(n^2)$ first divide-and-conquer algorithm
- An improved divide-and-conquer algorithm
- Remarks

Assume n is a power of 2
Define

$$
\begin{aligned}
A_0(x) &= a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1} \\
A_1(x) &= a_{\frac{n}{2}} + a_{\frac{n}{2}+1} x + \cdots + a_n x^{\frac{n}{2}} \\
A(x) &= A_0(x) + A_1(x) x^{\frac{n}{2}}
\end{aligned}
$$

Divide-and-Conquer: Divide

Assume n is a power of 2
Define

$$\begin{aligned} A_0(x) &= a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1} \\ A_1(x) &= a_{\frac{n}{2}} + a_{\frac{n}{2}+1} x + \cdots + a_n x^{\frac{n}{2}} \\ A(x) &= A_0(x) + A_1(x) x^{\frac{n}{2}} \end{aligned}$$

Similarly, define $B_0(x)$ and $B_1(x)$ such that

$$B(x) = B_0(x) + B_1(x) x^{\frac{n}{2}}$$

Assume n is a power of 2
Define

$$
\begin{aligned}
A_0(x) &= a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1} \\
A_1(x) &= a_{\frac{n}{2}} + a_{\frac{n}{2}+1} x + \cdots + a_n x^{\frac{n}{2}} \\
A(x) &= A_0(x) + A_1(x) x^{\frac{n}{2}}
\end{aligned}
$$

Similarly, define $B_0(x)$ and $B_1(x)$ such that

$$B(x) = B_0(x) + B_1(x) x^{\frac{n}{2}}$$

$A(x)B(x) =$

Divide-and-Conquer: Divide

Assume n is a power of 2
Define

$$\begin{aligned} A_0(x) &= a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1} \\ A_1(x) &= a_{\frac{n}{2}} + a_{\frac{n}{2}+1} x + \cdots + a_n x^{\frac{n}{2}} \\ A(x) &= A_0(x) + A_1(x) x^{\frac{n}{2}} \end{aligned}$$

Similarly, define $B_0(x)$ and $B_1(x)$ such that

$$B(x) = B_0(x) + B_1(x) x^{\frac{n}{2}}$$

$A(x)B(x) = A_0(x)B_0(x) +$

Divide-and-Conquer: Divide

Assume n is a power of 2
Define

$$\begin{aligned} A_0(x) &= a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1} \\ A_1(x) &= a_{\frac{n}{2}} + a_{\frac{n}{2}+1} x + \cdots + a_n x^{\frac{n}{2}} \\ A(x) &= A_0(x) + A_1(x) x^{\frac{n}{2}} \end{aligned}$$

Similarly, define $B_0(x)$ and $B_1(x)$ such that

$$B(x) = B_0(x) + B_1(x) x^{\frac{n}{2}}$$

$A(x)B(x) = A_0(x)B_0(x) + A_0(x)B_1(x)x^{\frac{n}{2}} +$

Divide-and-Conquer: Divide

Assume n is a power of 2
Define

$$\begin{aligned} A_0(x) &= a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1} \\ A_1(x) &= a_{\frac{n}{2}} + a_{\frac{n}{2}+1} x + \cdots + a_n x^{\frac{n}{2}} \\ A(x) &= A_0(x) + A_1(x) x^{\frac{n}{2}} \end{aligned}$$

Similarly, define $B_0(x)$ and $B_1(x)$ such that

$$B(x) = B_0(x) + B_1(x) x^{\frac{n}{2}}$$

$$A(x)B(x) = A_0(x)B_0(x) + A_0(x)B_1(x)x^{\frac{n}{2}} + A_1(x)B_0(x)x^{\frac{n}{2}} +$$

Divide-and-Conquer: Divide

Assume n is a power of 2
Define

$$\begin{aligned} A_0(x) &= a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1} \\ A_1(x) &= a_{\frac{n}{2}} + a_{\frac{n}{2}+1} x + \cdots + a_n x^{\frac{n}{2}} \\ A(x) &= A_0(x) + A_1(x) x^{\frac{n}{2}} \end{aligned}$$

Similarly, define $B_0(x)$ and $B_1(x)$ such that

$$B(x) = B_0(x) + B_1(x) x^{\frac{n}{2}}$$

$$A(x)B(x) = A_0(x)B_0(x) + A_0(x)B_1(x)x^{\frac{n}{2}} + A_1(x)B_0(x)x^{\frac{n}{2}} + A_1(x)B_1(x)x^n$$

Assume n is a power of 2
Define

$$\begin{aligned} A_0(x) &= a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1} \\ A_1(x) &= a_{\frac{n}{2}} + a_{\frac{n}{2}+1} x + \cdots + a_n x^{\frac{n}{2}} \\ A(x) &= A_0(x) + A_1(x) x^{\frac{n}{2}} \end{aligned}$$

Similarly, define $B_0(x)$ and $B_1(x)$ such that

$$B(x) = B_0(x) + B_1(x) x^{\frac{n}{2}}$$

$$A(x)B(x) = A_0(x)B_0(x) + A_0(x)B_1(x)x^{\frac{n}{2}} + A_1(x)B_0(x)x^{\frac{n}{2}} + A_1(x)B_1(x)x^n$$

The original problem (of size n) is divided into
4 problems of input size $n/2$

Example

$$
\begin{aligned}
A(x) &= 2 + 5x + 3x^2 + x^3 - x^4 \\
B(x) &= 1 + 2x + 2x^2 + 3x^3 + 6x^4 \\
A(x)B(x) &= 2 + 9x + 17x^2 + 23x^3 + 34x^4 + 39x^5 \\
&\quad +19x^6 + 3x^7 - 6x^8
\end{aligned}
$$

$A_0(x) = 2 + 5x, A_1(x) = 3 + x - x^2, A(x) = A_0(x) + A_1(x)x^2$

Example

$$\begin{aligned} A(x) &= 2+5x+3x^2+x^3-x^4 \\ B(x) &= 1+2x+2x^2+3x^3+6x^4 \\ A(x)B(x) &= 2+9x+17x^2+23x^3+34x^4+39x^5 \\ &\quad +19x^6+3x^7-6x^8 \end{aligned}$$

$$A_0(x)=2+5x, A_1(x)=3+x-x^2, A(x)=A_0(x)+A_1(x)x^2$$

$$B_0(x)=1+2x, B_1(x)=2+3x+6x^2, B(x)=B_0(x)+B_1(x)x^2$$

Example

$$
\begin{aligned}
A(x) &= 2+5x+3x^2+x^3-x^4 \\
B(x) &= 1+2x+2x^2+3x^3+6x^4 \\
A(x)B(x) &= 2+9x+17x^2+23x^3+34x^4+39x^5 \\
&\quad +19x^6+3x^7-6x^8
\end{aligned}
$$

$$A_0(x)=2+5x, A_1(x)=3+x-x^2, A(x)=A_0(x)+A_1(x)x^2$$
$$B_0(x)=1+2x, B_1(x)=2+3x+6x^2, B(x)=B_0(x)+B_1(x)x^2$$

$$
\begin{aligned}
A_0(x)B_0(x) &= 2+9x+10x^2 \\
A_1(x)B_1(x) &= 6+11x+19x^2+3x^3-6x^4 \\
A_0(x)B_1(x) &= 4+16x+27x^2+30x^3 \\
A_1(x)B_0(x) &= 3+7x+x^2-2x^3
\end{aligned}
$$

Example

$$\begin{aligned} A(x) &= 2 + 5x + 3x^2 + x^3 - x^4 \\ B(x) &= 1 + 2x + 2x^2 + 3x^3 + 6x^4 \\ A(x)B(x) &= 2 + 9x + 17x^2 + 23x^3 + 34x^4 + 39x^5 \\ &\quad +19x^6 + 3x^7 - 6x^8 \end{aligned}$$

$$A_0(x) = 2 + 5x, A_1(x) = 3 + x - x^2, A(x) = A_0(x) + A_1(x)x^2$$
$$B_0(x) = 1 + 2x, B_1(x) = 2 + 3x + 6x^2, B(x) = B_0(x) + B_1(x)x^2$$

$$\begin{aligned} A_0(x)B_0(x) &= 2 + 9x + 10x^2 \\ A_1(x)B_1(x) &= 6 + 11x + 19x^2 + 3x^3 - 6x^4 \\ A_0(x)B_1(x) &= 4 + 16x + 27x^2 + 30x^3 \\ A_1(x)B_0(x) &= 3 + 7x + x^2 - 2x^3 \\ A_0(x)B_1(x) + A_1(x)B_0(x) &= 7 + 23x + 28x^2 + 28x^3 \end{aligned}$$

$$A_0(x)B_0(x) + (A_0(x)B_1(x) + A_1(x)B_0(x))x^2 + A_1(x)B_1(x)x^4$$

Example

$$
\begin{aligned}
A(x) &= 2+5x+3x^2+x^3-x^4\\
B(x) &= 1+2x+2x^2+3x^3+6x^4\\
A(x)B(x) &= 2+9x+17x^2+23x^3+34x^4+39x^5\\
&\quad +19x^6+3x^7-6x^8
\end{aligned}
$$

$$A_0(x)=2+5x, A_1(x)=3+x-x^2, A(x)=A_0(x)+A_1(x)x^2$$
$$B_0(x)=1+2x, B_1(x)=2+3x+6x^2, B(x)=B_0(x)+B_1(x)x^2$$

$$
\begin{aligned}
A_0(x)B_0(x) &= 2+9x+10x^2\\
A_1(x)B_1(x) &= 6+11x+19x^2+3x^3-6x^4\\
A_0(x)B_1(x) &= 4+16x+27x^2+30x^3\\
A_1(x)B_0(x) &= 3+7x+x^2-2x^3\\
A_0(x)B_1(x)+A_1(x)B_0(x) &= 7+23x+28x^2+28x^3
\end{aligned}
$$

$$A_0(x)B_0(x)+(A_0(x)B_1(x)+A_1(x)B_0(x))x^2+A_1(x)B_1(x)x^4$$
$$=2+9x+17x^2+23x^3+34x^4+39x^5+19x^6+3x^7-6x^8$$

Conquer: Solve the four subproblems

- compute

$$A_0(x)B_0(x),\quad A_0(x)B_1(x),\quad A_1(x)B_0(x),\quad A_1(x)B_1(x)$$

Conquer: Solve the four subproblems

- compute

$$A_0(x)B_0(x), \quad A_0(x)B_1(x), \quad A_1(x)B_0(x), \quad A_1(x)B_1(x)$$

by recursively calling the algorithm 4 times

Divide-and-Conquer: Conquer

Conquer: Solve the four subproblems

- compute

$$A_0(x)B_0(x), \quad A_0(x)B_1(x), \quad A_1(x)B_0(x), \quad A_1(x)B_1(x)$$

by recursively calling the algorithm 4 times

Combine

Conquer: Solve the four subproblems

- compute

$$A_0(x)B_0(x),\quad A_0(x)B_1(x),\quad A_1(x)B_0(x),\quad A_1(x)B_1(x)$$

by recursively calling the algorithm 4 times

Combine

- adding the following four polynomials

$$A_0(x)B_0(x)+A_0(x)B_1(x)x^{\frac{n}{2}}+A_1(x)B_0(x)x^{\frac{n}{2}}+A_1(x)B_1(x)x^n$$

- takes $O(n)$ operations (Why?)

The First Divide-and-Conquer Algorithm

PolyMulti1(A(x), B(x))

begin

$$A_0(x) = a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1};$$
$$A_1(x) = a_{\frac{n}{2}} + a_{\frac{n}{2}+1} x + \cdots + a_n x^{\frac{n}{2}};$$
$$B_0(x) = b_0 + b_1 x + \cdots + b_{\frac{n}{2}-1} x^{\frac{n}{2}-1};$$
$$B_1(x) = b_{\frac{n}{2}} + b_{\frac{n}{2}+1} x + \cdots + b_n x^{\frac{n}{2}};$$

PolyMulti1(A(x), B(x))

begin

$A_0(x) = a_0 + a_1x + \cdots + a_{\frac{n}{2}-1}x^{\frac{n}{2}-1};$
$A_1(x) = a_{\frac{n}{2}} + a_{\frac{n}{2}+1}x + \cdots + a_nx^{\frac{n}{2}};$
$B_0(x) = b_0 + b_1x + \cdots + b_{\frac{n}{2}-1}x^{\frac{n}{2}-1};$
$B_1(x) = b_{\frac{n}{2}} + b_{\frac{n}{2}+1}x + \cdots + b_nx^{\frac{n}{2}};$
$U(x) = PolyMulti1(A_0(x), B_0(x));$
$V(x) = PolyMulti1(A_0(x), B_1(x));$
$W(x) = PolyMulti1(A_1(x), B_0(x));$
$Z(x) = PolyMulti1(A_1(x), B_1(x));$

PolyMulti1(A(x), B(x))

begin

$A_0(x) = a_0 + a_1x + \cdots + a_{\frac{n}{2}-1}x^{\frac{n}{2}-1};$

$A_1(x) = a_{\frac{n}{2}} + a_{\frac{n}{2}+1}x + \cdots + a_nx^{\frac{n}{2}};$

$B_0(x) = b_0 + b_1x + \cdots + b_{\frac{n}{2}-1}x^{\frac{n}{2}-1};$

$B_1(x) = b_{\frac{n}{2}} + b_{\frac{n}{2}+1}x + \cdots + b_nx^{\frac{n}{2}};$

$U(x) = PolyMulti1(A_0(x), B_0(x));$

$V(x) = PolyMulti1(A_0(x), B_1(x));$

$W(x) = PolyMulti1(A_1(x), B_0(x));$

$Z(x) = PolyMulti1(A_1(x), B_1(x));$

return $\left(U(x) + [V(x) + W(x)]x^{\frac{n}{2}} + Z(x)x^n\right)$

end

Assume that n is a power of 2

$$T(n) = \begin{cases} 4T(n/2) + n, & \text{if } n > 1, \\ 1, & \text{if } n = 1. \end{cases}$$

By the Master Theorem for recurrences

$$T(n) = \Theta(n^2).$$

Analysis of Running Time

Assume that n is a power of 2

$$T(n) = \begin{cases} 4T(n/2) + n, & \text{if } n > 1, \\ 1, & \text{if } n = 1. \end{cases}$$

By the Master Theorem for recurrences

$$T(n) = \Theta(n^2).$$

Same order as the brute force approach!
No improvement!

Objective and Outline

Outline:

- Introduction
- The polynomial multiplication problem
- An $O(n^2)$ brute force algorithm
- An $O(n^2)$ first divide-and-conquer algorithm
- An improved divide-and-conquer algorithm
- Remarks

Two Observations

Observation 1:

We said that we need the 4 terms:

$$A_0 B_0,\ A_0 B_1,\ A_1 B_0,\ A_1 B_.$$

What we really *need are the 3 terms:*

$$A_0 B_0,\ A_0 B_1 + A_1 B_0,\ A_1 B_1!$$

Two Observations

Observation 1:

We said that we need the 4 terms:

$$A_0B_0,\ A_0B_1,\ A_1B_0,\ A_1B_.$$

What we really *need are the 3 terms:*

$$A_0B_0,\ A_0B_1 + A_1B_0,\ A_1B_1!$$

Observation 2:

The three terms can be obtained using only 3 multiplications:

Two Observations

Observation 1:

We said that we need the 4 terms:

$$A_0B_0,\ A_0B_1,\ A_1B_0,\ A_1B_.$$

What we really *need are the 3 terms:*

$$A_0B_0,\ A_0B_1 + A_1B_0,\ A_1B_1!$$

Observation 2:

The three terms can be obtained using only 3 multiplications:

$$\begin{aligned} Y &= (A_0 + A_1)(B_0 + B_1) \\ U &= A_0B_0 \\ Z &= A_1B_1 \end{aligned}$$

Two Observations

Observation 1:

We said that we need the 4 terms:

$$A_0B_0,\ A_0B_1,\ A_1B_0,\ A_1B_.$$

What we really *need are the 3 terms:*

$$A_0B_0,\ A_0B_1 + A_1B_0,\ A_1B_1!$$

Observation 2:

The three terms can be obtained using only 3 multiplications:

$$\begin{aligned} Y &= (A_0 + A_1)(B_0 + B_1) \\ U &= A_0B_0 \\ Z &= A_1B_1 \end{aligned}$$

- *We need U and Z and*

Two Observations

Observation 1:

We said that we need the 4 terms:

$$A_0 B_0, \; A_0 B_1, \; A_1 B_0, \; A_1 B_.$$

What we really *need are the 3 terms:*

$$A_0 B_0, \; A_0 B_1 + A_1 B_0, \; A_1 B_1!$$

Observation 2:

The three terms can be obtained using only 3 multiplications:

$$\begin{aligned} Y &= (A_0 + A_1)(B_0 + B_1) \\ U &= A_0 B_0 \\ Z &= A_1 B_1 \end{aligned}$$

- *We need U and Z and*
- $A_0 B_1 + A_1 B_0 =$

Two Observations

Observation 1:

We said that we need the 4 terms:

$$A_0 B_0,\ A_0 B_1,\ A_1 B_0,\ A_1 B_.$$

What we really *need are the 3 terms:*

$$A_0 B_0,\ A_0 B_1 + A_1 B_0,\ A_1 B_1!$$

Observation 2:

The three terms can be obtained using only 3 multiplications:

$$\begin{aligned} Y &= (A_0 + A_1)(B_0 + B_1) \\ U &= A_0 B_0 \\ Z &= A_1 B_1 \end{aligned}$$

- *We need U and Z and*
- $A_0 B_1 + A_1 B_0 = Y - U - Z$

PolyMulti2(A(x), B(x))

begin

$A_0(x) = a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1};$
$A_1(x) = a_{\frac{n}{2}} + a_{\frac{n}{2}+1} x + \cdots + a_n x^{n-\frac{n}{2}};$
$B_0(x) = b_0 + b_1 x + \cdots + b_{\frac{n}{2}-1} x^{\frac{n}{2}-1};$
$B_1(x) = b_{\frac{n}{2}} + b_{\frac{n}{2}+1} x + \cdots + b_n x^{n-\frac{n}{2}};$

PolyMulti2(A(x), B(x))

begin

$A_0(x) = a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$;

$A_1(x) = a_{\frac{n}{2}} + a_{\frac{n}{2}+1} x + \cdots + a_n x^{n-\frac{n}{2}}$;

$B_0(x) = b_0 + b_1 x + \cdots + b_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$;

$B_1(x) = b_{\frac{n}{2}} + b_{\frac{n}{2}+1} x + \cdots + b_n x^{n-\frac{n}{2}}$;

$Y(x) = PolyMulti2(A_0(x) + A_1(x), B_0(x) + B_1(x))$;

$U(x) = PolyMulti2(A_0(x), B_0(x))$;

$Z(x) = PolyMulti2(A_1(x), B_1(x))$;

return $\left(U(x) + [Y(x) - U(x) - Z(x)]x^{\frac{n}{2}} + Z(x)x^{2\frac{n}{2}}\right)$

end

Running Time of the Modified Algorithm

$$T(n) = \begin{cases} 3T(n/2) + n, & \text{if } n > 1, \\ 1, & \text{if } n = 1. \end{cases}$$

Running Time of the Modified Algorithm

$$T(n) = \begin{cases} 3T(n/2) + n, & \text{if } n > 1, \\ 1, & \text{if } n = 1. \end{cases}$$

By the Master Theorem for recurrences

$$T(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.58\dots}).$$

Running Time of the Modified Algorithm

$$T(n) = \begin{cases} 3T(n/2) + n, & \text{if } n > 1, \\ 1, & \text{if } n = 1. \end{cases}$$

By the Master Theorem for recurrences

$$T(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.58\ldots}).$$

Much better than previous $\Theta(n^2)$ algorithms!

Objective and Outline

Outline:

- Introduction
- The polynomial multiplication problem
- An $O(n^2)$ brute force algorithm
- An $O(n^2)$ first divide-and-conquer algorithm
- An improved divide-and-conquer algorithm
- Remarks

Remarks

- This algorithm can also be used for (long) integer multiplication
  - Really designed by Karatsuba (1960, 1962) for that purpose.
  - Response to conjecture by Kolmogorov, founder of modern probability, that this would require $\Theta(n^2)$.
- Similar to technique deeloped by Strassen a few years later to multiply 2 $n \times n$ matrices in $O(n^{\log_2 7})$ operations, instead of the $\Theta(n^3)$ that a straightforward algorithm would use.
- Takeaway from this lesson is that divide-and-conquer doesn't always give you faster algorithm. Sometimes, you need to be more clever.
- Coming up. An $O(n \log n)$ solution to the polynomial multiplication problem

Remarks

- This algorithm can also be used for (long) integer multiplication
  - Really designed by Karatsuba (1960, 1962) for that purpose.
  - Response to conjecture by Kolmogorov, founder of modern probability, that this would require $\Theta(n^2)$.
- Similar to technique deeloped by Strassen a few years later to multiply 2 $n \times n$ matrices in $O(n^{\log_2 7})$ operations, instead of the $\Theta(n^3)$ that a straightforward algorithm would use.
- Takeaway from this lesson is that divide-and-conquer doesn't always give you faster algorithm. Sometimes, you need to be more clever.
- Coming up. An $O(n \log n)$ solution to the polynomial multiplication problem
  - It involves strange recasting of the problem and solution using the Fast Fourier Transform algorithm as a subroutine
  - The FFT is another classic D & C algorithm that we will learn soon.