COMP 3711 Design and Analysis of Algorithms

Integer and Matrix Multiplication

Long Multiplication

High School method for multiplying integers

Example: 163 x 97

			1	6	3
		x		9	7
		1	1	4	1
	1	4	6	7	
	1	5	8	1	1

Binary Long Multiplication

Multiply. Given two n-bit integers a and b, compute $a \cdot b$.

- Example: 163 x 97, i.e., 10100011 x 01100001

```
              1 0 1 0 0 0 1 1
            × 0 1 1 0 0 0 0 1
              ---------------
              1 0 1 0 0 0 1 1
            0 0 0 0 0 0 0 0
          0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
      0 0 0 0 0 0 0 0
    1 0 1 0 0 0 1 1
  1 0 1 0 0 0 1 1
0 0 0 0 0 0 0 0
-----------------------------
0 1 1 1 1 0 1 1 1 0 0 0 0 1 1
```

Cost. n binary multiplications to generate each line; we generate n lines. Thus, total cost $\Theta(n^2)$ multiplications (plus $\Theta(n^2)$ binary additions because we summarize n lines)

Binary Multiplication: Break into smaller problems

Goal. Given two n-bit integers a and b, compute: $a \cdot b$.

- Example: 163 x 97, i.e., 10100011 x 01100001 (n=8)

Rewrite numbers. $a = 2^{n/2}a_1 + a_0, \;\; b = 2^{n/2}b_1 + b_0$

a_1=1010, a_0=0011

b_1=0110, b_0=0001

Note: a_1 and b_1 can be thought of having $n/2$ (4 in this example) least significant bits that are equal to 0.

Given rewritten numbers: $a \cdot b = (2^{n/2}a_1 + a_0) \cdot (2^{n/2}b_1 + b_0) = 2^n a_1 b_1 + 2^{n/2}(a_1 b_0 + a_0 b_1) + a_0 b_0$

Observation: Multiplication by 2^k can be done in one time unit by a left shift of k bits.

- Example: 12=00001100. 12x2³=96 is the same as **left shifting** 00001100 by 3 bits
- 00001100 << 3 = 01100000 = 96
- We use << to denote left shifting

Binary Multiplication: Motivation of D&C

Instead of multiplying two n-bit integers *a* and *b* directly with long multiplication:

1] Rewrite numbers: $a = 2^{n/2}a_1 + a_0, \;\; b = 2^{n/2}b_1 + b_0$

a_1=1010, a_0=0011

b_1=0110, b_0=0001

2] The product to be computed becomes:

$a \cdot b$

$= (2^{n/2}a_1 + a_0) \cdot (2^{n/2}b_1 + b_0) = 2^n a_1 b_1 + 2^{n/2}(a_1 b_0 + a_0 b_1) + a_0 b_0$

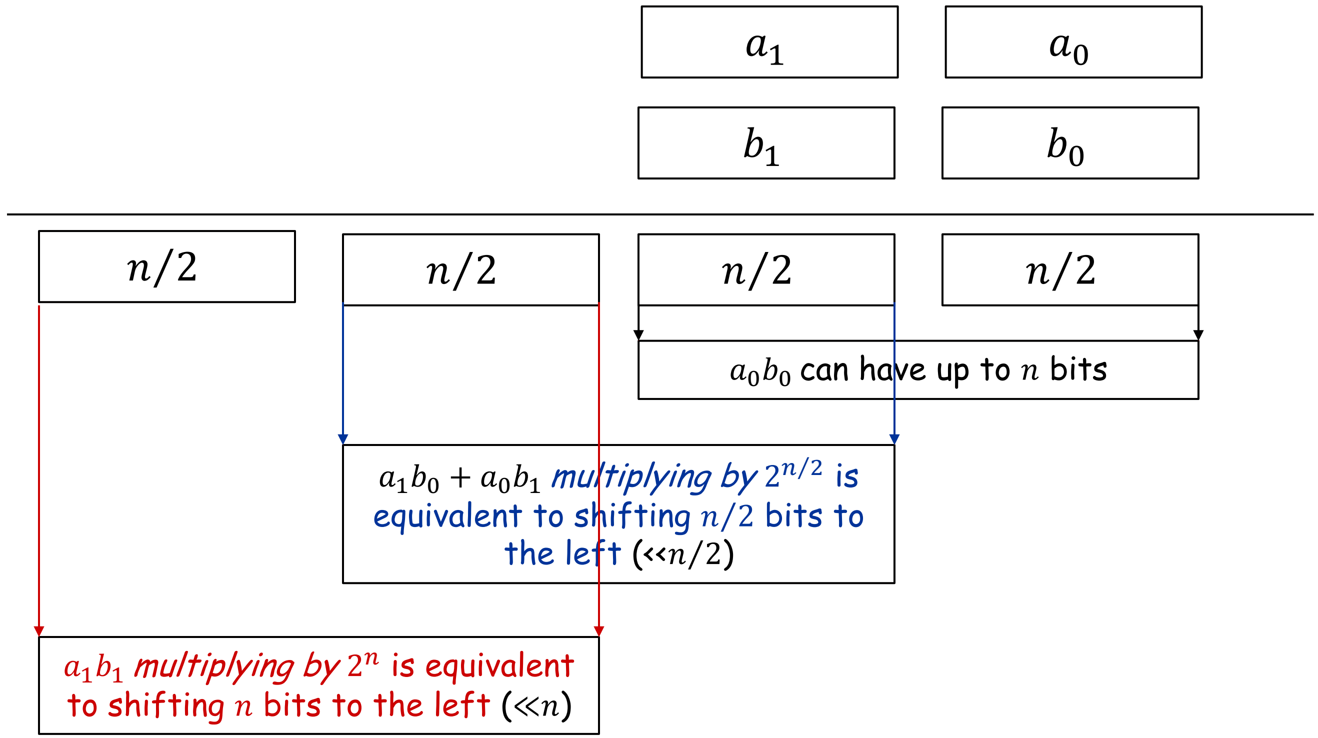
3] The new computation requires 4 products of integers, each with n/2 bits:

$$a_1 b_1, a_1 b_0, a_0 b_1, a_0 b_0$$

4] Apply D&C by splitting a problem of size n, to 4 problems of size $n/2$.

D&C Binary Multiplication: Visualization

$$a \cdot b = 2^n a_1 b_1 + 2^{n/2}(a_1 b_0 + a_0 b_1) + a_0 b_0$$



The first divide-and-conquer algorithm for integer multiplication

Suppose the bits are stored in arrays $A[1..n]$ and $B[1..n]$, $A[1]$ and $B[1]$ are the least significant bits

Multiply(A, B)**:**
$n \leftarrow$ **size of** A
if $n = 1$ **then return** $A[1] \cdot B[1]$
$mid \leftarrow \lfloor n/2 \rfloor$
$U \leftarrow$ **Multiply**$(A[mid+1..n], B[mid+1..n])$ // $a_1 b_1$
$V \leftarrow$ **Multiply**$(A[mid+1..n], B[1..mid])$ // $a_1 b_0$
$W \leftarrow$ **Multiply**$(A[1..mid], B[mid+1..n])$ // $a_0 b_1$
$Z \leftarrow$ **Multiply**$(A[1..mid], B[1..mid])$ // $a_0 b_0$
$M[1..2n] \leftarrow 0$
$M[1..n] \leftarrow Z$ // $a_0 b_0$
$M[mid+1..] \leftarrow M[mid+1..] \oplus V \oplus W$ // + $(a_1 b_0 + a_0 b_1) \ll n/2$
$M[2mid+1..] \leftarrow M[2mid+1..] \oplus U$ // + $a_1 b_1 \ll n$
return M

$\oplus$: denotes the integer addition algorithm

Analysis with Expansion Method

Recurrence.

For, $n > 1$, $T(n) = 4T(n/2) + n$. $T(1) = 1$

$$T(n) = 4\,T\left(\frac{n}{2}\right) + n$$

$$= 4\left(4T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n \qquad = 4^2\ T\left(\frac{n}{2^2}\right) + 2n + n$$

$$= 4^2\ \left(4T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right) + 2n + n \qquad = 4^3\ T\left(\frac{n}{2^3}\right) + 2^2n + 2n + n$$

$$= 4^3\ \left(4T\left(\frac{n}{2^4}\right) + \frac{n}{2^3}\right) + 2^2n + 2n + n \qquad = 4^4\ T\left(\frac{n}{2^4}\right) + (2^3 + 2^2 + 2 + 1)n$$

....

$$= 4^i\ T\left(\frac{n}{2^i}\right) + \left(2^{i-1} + \ \cdots + 2 + 1\right)n$$

Analysis with Expansion Method (cont)

When we reach level i we have (total) cost:

$$4^i\, T\left(\frac{n}{2^i}\right) + \left(2^{i-1} + \ \ldots + 2 + 1\right)n$$

We stop when the problem size becomes 1, i.e., when we reach level *i,* such that: $n/2^i = 1 \Longrightarrow n = 2^i \Longrightarrow i = \log_2 n$. Thus, $4^i = 4^{\log_2 n} = n^{\log_2 4} = n^2$ and T(1)=1. The total cost becomes

$$n^2$$

$$+ n\left(2^{i-1} + \ \ldots + 2 + 1\right) = n^2 + n \sum_{i=0}^{\log_2 n - 1} 2^i = n^2 + n\,\frac{2^{\log_2 n} - 1}{2 - 1} =$$

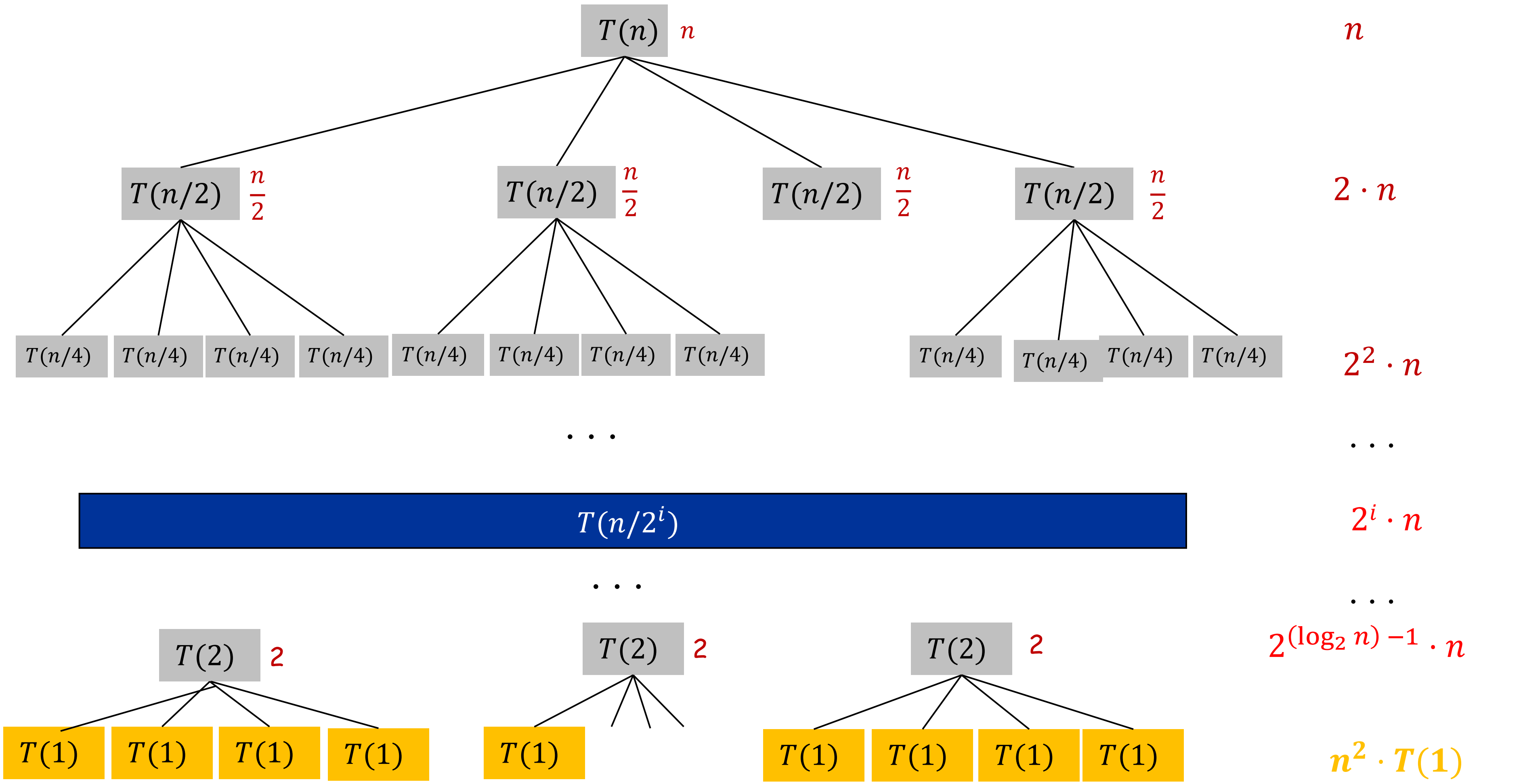
$$= n^2 + n(n-1) = \Theta(n^2)$$

Analysis with Recursion Tree Method

Recurrence:

$$T(n) = 4T(n/2) + n; \quad T(1) = 1$$

Solve the recurrence:



Analysis with Recursion Tree Method (cont)

$$n + 2n + 2^2 n + \cdots + \ +2^{(\log_2 n)-1} n \ + 4^{\log_2 n}\, T(1)$$
$$=$$
$$n\left(1 + 2 + 2^2 \ + \ 2^3 \ + \cdots + \ 2^{(\log_2 n)-1}\right) + n^2 =$$
$$n\ \left(\frac{2^{\log_2 n} - 1}{2-1}\right) + \ n^2 \ = \ n(n-1) + \ n^2 = \Theta(n^2)$$

- The divide-and-conquer algorithm is as bad as the primary school method
- Essentially, the algorithm still multiplies every bit of A with every bit of B.
- Compared with merge sort, the key difference is that one problem generates 4 subproblems of size $\boldsymbol{n/2}$.

Karatsuba Multiplication

- Let $a = a_1 2^{n/2} + a_0$, and $b = b_1 2^{n/2} + b_0$
  where a_1, a_0, b_1, b_0 are all $(n/2)$-bit integers.

- We already saw

$$ab = a_1\, b_1\, 2^n + (\boldsymbol{a_1 b_0} + \boldsymbol{a_0 b_1})2^{n/2} + a_0\, b_0$$

- Observation: We do not need the individual products $\boldsymbol{a_1 b_0}, \boldsymbol{a_0 b_1}$. Only their sum $\boldsymbol{a_1 b_0} + \boldsymbol{a_0 b_1}$.

- But given that we compute $a_1 b_1$, $a_0 b_0$ anyway, this sum requires only one additional multiplication:

$$\boldsymbol{a_1 b_0} + \boldsymbol{a_0 b_1} = (a_1 + a_0)(b_1 + b_0) - a_1 b_1 - a_0 b_0$$

Calculating ab now only requires performing 3 multiplication subproblems of size $n/2$!

Karatsuba's multiplication algorithm

```
Multiply(A,B):
n ← size of A
if n = 1 then return A[1]·B[1]
mid ← ⌊n/2⌋
U ← Multiply(A[mid+1..n], B[mid+1..n])          //a_1 b_1
Z ← Multiply(A[1..mid], B[1..mid])              // a_0 b_0
A' ← A[mid+1..n] ⊕ A[1..mid]                     // a_1+a_0
B' ← B[mid+1..n] ⊕ B[1..mid]                     //b_1 +b_0
Y ← Multiply(A', B')                             // (a_1 + a_0)(b_1 + b_0)
M[1..2n] ← 0
M[1..n] ← M[1..n] ⊕ Z                            // a_0 b_0
M[mid+1..] ← M[mid+1..] ⊕ Y ⊖ U ⊖ Z              //+(a_1 b_0 + a_0 b_1) ≪ n/2
M[2mid+1..] ← M[2mid+1..] ⊕ U                    // + a_1 b_1 ≪ n
return M
```

$\oplus$ $\ominus$: denotes the integer addition/subtraction algorithm

Analysis with Expansion Method

Recurrence.

For, $n > 1$, $T(n) = 3T(n/2) + n$. $T(1) = 1$

$$T(n) = 3\,T\left(\frac{n}{2}\right) + n$$

$$= 3\left(3T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n \qquad = 3^2\ T\left(\frac{n}{2^2}\right) + \frac{3}{2}n + n$$

$$= 3^2\ \left(3T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right) + \frac{3}{2}n + n \qquad = 3^3\ T\left(\frac{n}{2^3}\right) + \frac{3^2}{2^2}n + \frac{3}{2}n + n$$

$$= 3^3\ \left(3T\left(\frac{n}{2^4}\right) + \frac{n}{2^3}\right) + \frac{3^2}{2^2}n + \frac{3}{2}n + n \qquad = 3^4\ T\left(\frac{n}{2^4}\right) + \left(\frac{3^3}{2^3} + \frac{3^2}{2^2} + \frac{3}{2} + 1\right)n$$

....

$$= 3^i\ T\left(\frac{n}{2^i}\right) + \left(\frac{3^{i-1}}{2^{i-1}} + \frac{3^{i-2}}{2^{i-2}} + \cdots + \frac{3}{2} + 1\right)n$$

Analysis with Expansion Method (cont)

When we reach level i we have (total) cost:

$$3^i\, T\left(\frac{n}{2^i}\right) + \left(\left(\frac{3}{2}\right)^{i-1} + \ \dots + \left(\frac{3}{2}\right) + 1\right) n$$

We stop when the problem size becomes 1, i.e., when we reach level *i*, such that: $n/2^i = 1 \Longrightarrow n = 2^i \Longrightarrow i = \log_2 n$. Thus, $3^i = 3^{\log_2 n} = n^{\log_2 3} = n^{1.585}$ and T(1)=1. The total cost is:

$$n^{\log_2 3} \ + n\left(\left(\frac{3}{2}\right)^{i-1} + \ \dots\ \frac{3}{2} + 1\right) =$$

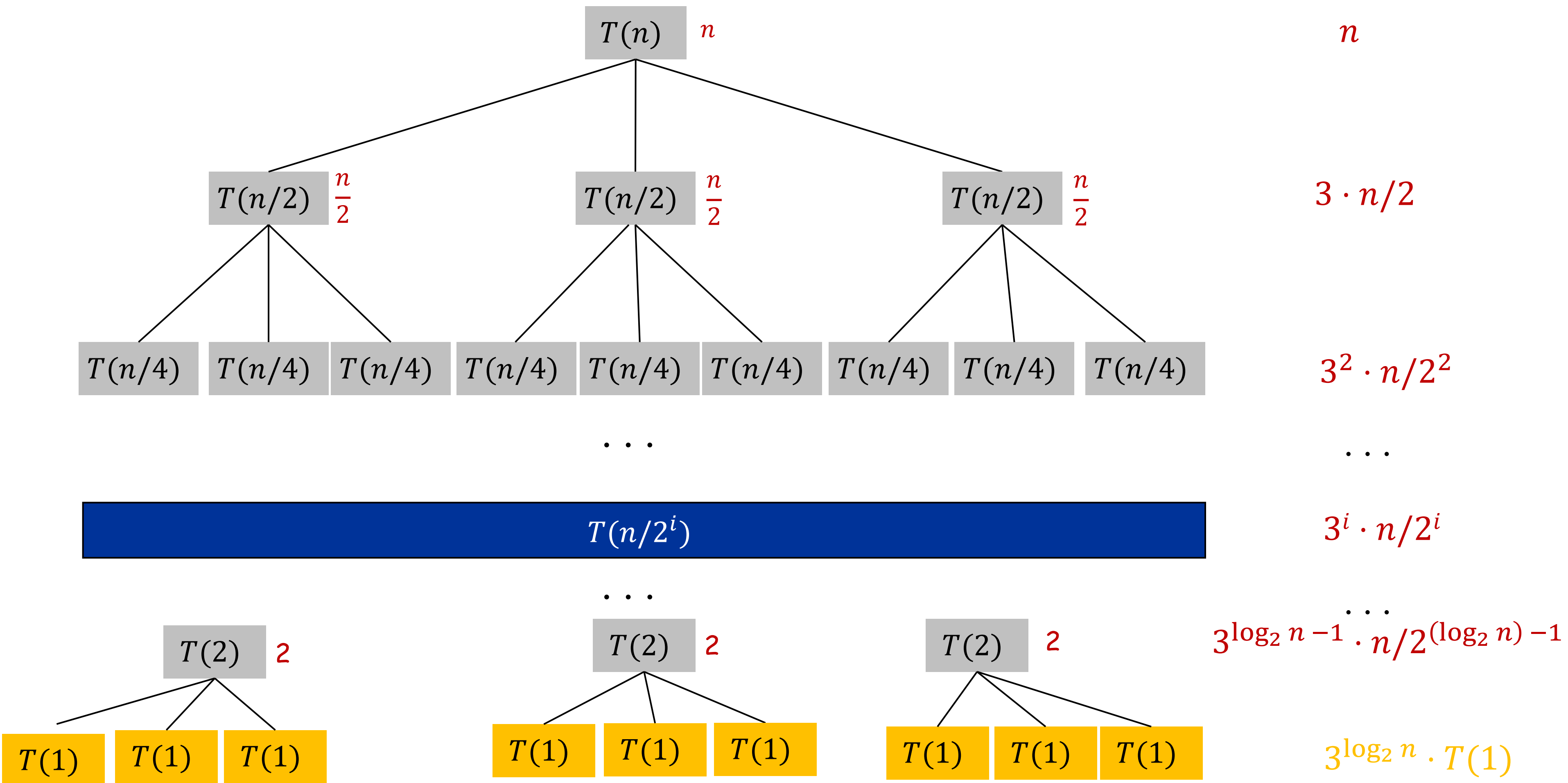
$$\boldsymbol{\Theta}(n^{\log_2 3}) \ = \boldsymbol{\Theta}\left(\boldsymbol{n}^{\mathbf{1.585\dots}}\right)$$

Analysis with Recursion Tree Method

Recurrence:

$$T(n) = 3T(n/2) + n$$

Solve the recurrence:



Analysis (continued)

Recurrence For First D&C Algorithm

$$T(n) = 4T(n/2) + n; \quad T(1) = 1$$

Solution: $T(n) = \Theta(n^2)$

Recurrence For Karatsuba Multiplication

$$T(n) = 3T(n/2) + n; \quad T(1) = 1$$

Solution: $T(n) = \Theta(n^{1.585\ldots})$

Analysis (continued)

Karatsuba Multiplication:

- Dividing each integer into 2 parts, and solve 3 subproblems
  - $T(n) = 3T(n/2) + n$, $T(n) = \Theta\left(n^{\log_2 3}\right) = \Theta(n^{1.585\ldots})$

Progressive improvements:

- Dividing each integer into 3 parts, and solve 5 subproblems
  - $T(n) = 5T(n/3) + n$, $T(n) = \Theta\left(n^{\log_3 5}\right) = \Theta(n^{1.465})$
- Dividing each integer into 4 parts, and solve 7 subproblems
  - $T(n) = 7T(n/4) + n$, $T(n) = \Theta\left(n^{\log_4 7}\right) = \Theta(n^{1.404})$
- …
- An $\Theta(n \log n \log\log n)$ algorithm (based on Fast Fourier Transform)
- An $\Theta(n \log n \log\log\log n)$ algorithm
- An $\Theta\left(n \log n \, 2^{\Theta(log^* n)}\right)$ algorithm ($log^* n$ is a VERY slow growing function)
- The conjecture was that the problem can be solved in $O(n \log n)$ time.

Conjecture Proven (2019)

- $\Theta(n \log n)$ time algorithm found

Matrix Multiplication

Matrix multiplication. Given two n-by-n matrices A and B, compute $C = AB$.

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

$$\begin{bmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nn} \end{bmatrix}$$

Brute force. $\Theta(n^3)$ time.

Fundamental question. Can we improve upon brute force?

Matrix Multiplication: First Attempt

Divide-and-conquer.

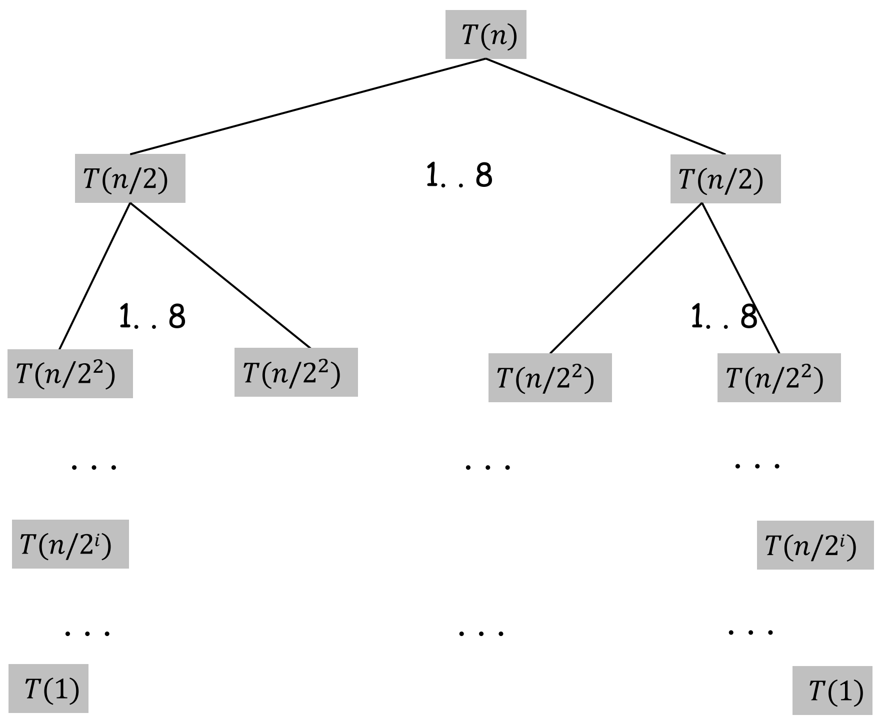
- Divide: partition A and B into $\frac{1}{2}n$-by-$\frac{1}{2}n$ blocks.
- Conquer: multiply 8 $\frac{1}{2}n$-by-$\frac{1}{2}n$ submatrices recursively.
- Combine: add appropriate products using 4 matrix additions.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\ C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\ C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\ C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22}) \end{aligned}$$

$$T(n) \;=\; \underbrace{8\mathrm{T}(n/2)}_{\text{Recursive calls}} + \underbrace{\mathrm{O}(n^2)}_{\text{Add, form submatrices}} \quad \Longrightarrow \quad T(n) = \mathrm{O}(n^3)$$

Solving the Recurrence $T(n) = 8T\left(\frac{n}{2}\right) + n^2$,



Lv	#pr	work/pr	work/lv
0	1	n^2	n^2
1	8	$(n/2)^2$	$2n^2$
2	8^2	$(n/2^2)^2$	2^2n^2
i	8^i	$(n/2^i)^2$	2^in^2
$\log_2 n$	$8^{\log_2 n} = n^{\log_2 8}$ $= n^3$	1	n^3

Strassen's Matrix Multiplication Algorithm

Key idea. multiply 2-by-2 block matrices with only 7 multiplications.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

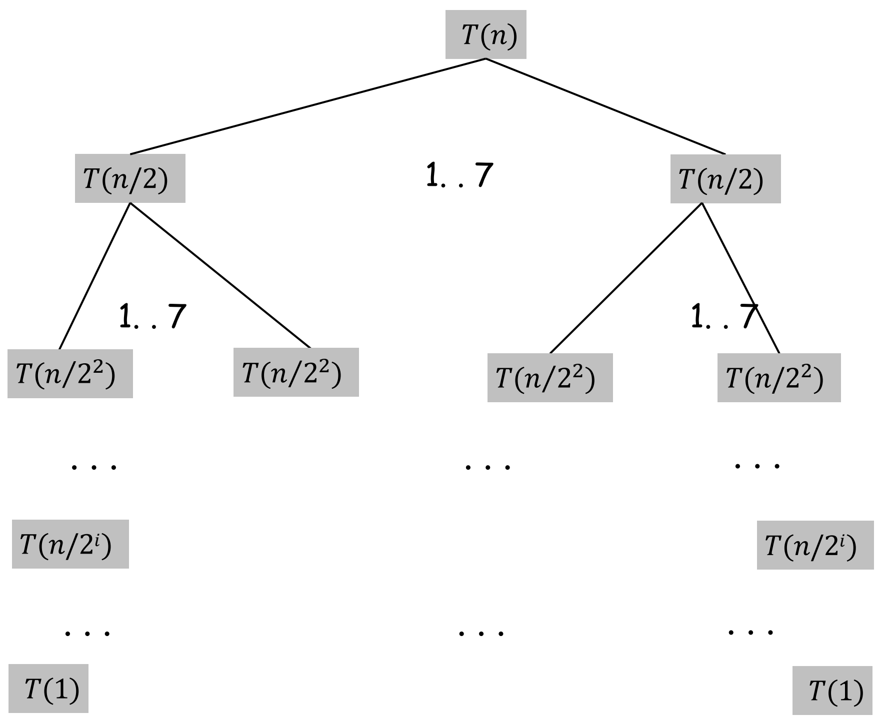
$$C_{11} = P_5 + P_4 - P_2 + P_6$$
$$C_{12} = P_1 + P_2$$
$$C_{21} = P_3 + P_4$$
$$C_{22} = P_5 + P_1 - P_3 - P_7$$

$$P_1 = A_{11} \times (B_{12} - B_{22})$$
$$P_2 = (A_{11} + A_{12}) \times B_{22}$$
$$P_3 = (A_{21} + A_{22}) \times B_{11}$$
$$P_4 = A_{22} \times (B_{21} - B_{11})$$
$$P_5 = (A_{11} + A_{22}) \times (B_{11} + B_{22})$$
$$P_6 = (A_{12} - A_{22}) \times (B_{21} + B_{22})$$
$$P_7 = (A_{11} - A_{21}) \times (B_{11} + B_{12})$$

- 7 multiplications of $\left(\frac{1}{2}n\right)$-by-$\left(\frac{1}{2}n\right)$ submatrices.
- $\Theta(n^2)$ additions and subtractions.
- $T(n) = 7T(n/2) + n^2 \implies T(n) = \Theta\left(n^{\log_2 7}\right) = \Theta(n^{2.807})$

In practice: Used to multiply large matrices (e.g., $n > 100$)

Solving the Recurrence $T(n) = 7T\left(\frac{n}{2}\right) + n^2$,



Lv	#pr	work/pr	work/lv
0	1	n^2	n^2
1	7	$(n/2)^2$	$(7/4)n^2$
2	7^2	$(n/2^2)^2$	$(7/4)^2n^2$
i	7^i	$(n/2^i)^2$	$(7/4)^in^2$
$\log_2 n$	$7^{\log_2 n} = n^{\log_2 7}$	1	$n^{\log_2 7}$

Fast Matrix Multiplication in Theory

Q. Multiply two 2-by-2 matrices with only 7 multiplications?
A. Yes! $\Theta(n^{2.807})$ [Strassen, 1969]

Q. Multiply two 2-by-2 matrices with only 6 multiplications?
A. Impossible.

Q. Two 3-by-3 matrices with only 21 multiplications?
A. Also impossible.

Q. Two 70-by-70 matrices with only 143,640 multiplications?
A. Yes! $\Theta(n^{2.795})$

The competition continues…

- $\Theta(n^{2.376})$ [Coppersmith-Winograd, 1990.]
- $\Theta(n^{2.374})$ [Stothers, 2010.]
- $\Theta(n^{2.3728642})$ [Williams, 2011.]
- $\Theta(n^{2.3728639})$ [Le Gall, 2014.]
- Conjecture: close to $\Theta(n^2)$

Exercise on Exponentiation

Recursive algorithm for computing c^n:

SlowPower(c, n)

If $n = 1$ Then Return c

Return SlowPower$(c, n-1) \cdot c$

How many multiplications SlowPower requires for computing c^{15};

Recursive calls:

SL$(c, 15)$

SL$(c, 14) \cdot c$

SL$(c, 13) \cdot c$

...

SL$(c, 1) \cdot c$

c (0 multiplications)

$c^2 \leftarrow c \cdot c$ (1 multiplication)

$c^{14} \leftarrow c^{13} \cdot c$ (1 multiplication)

$c^{15} \leftarrow c^{14} \cdot c$ (1 multiplication)

Total: 14 multiplications

What is the recurrence and running time for SlowPower(c, n) as a function of the number of multiplications?

Exercise on Exponentiation (cont)

SquaringPower(c, n)
 If $n = 1$ Then Return c
 T = SquaringPower$(c, \lfloor n/2 \rfloor)$
 If n is even Then Return $T \times T$
  Else Return $T \times T \times c$

What is the recurrence and running time for SquaringPower(c, n)?
How many multiplications SquaringPower requires for computing c^{15};
Recursive calls:
SP$(c, 15)$
  SP$(c, 7)$; $T \times T \times c$ $(T \leftarrow c^7)$
   SP$(c, 3)$; $T \times T \times c$ $(T \leftarrow c^3)$
    SP$(c, 1)$; $T \times T \times c$ $(T \leftarrow c)$

Total: 6 multiplications
What is the minimum number of multiplications for computing c^{15}?
5: $c^2 \leftarrow c \cdot c,\ c^3 \leftarrow c^2 \cdot c,\ c^5 \leftarrow c^3 \cdot c^2,\ c^{10} \leftarrow c^5 \cdot c^5,\ c^{15} \leftarrow c^{10} \cdot c^5$