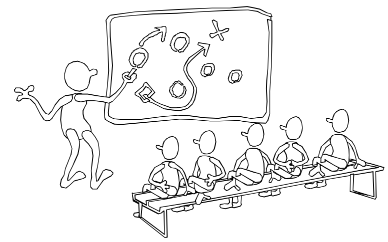
15-251

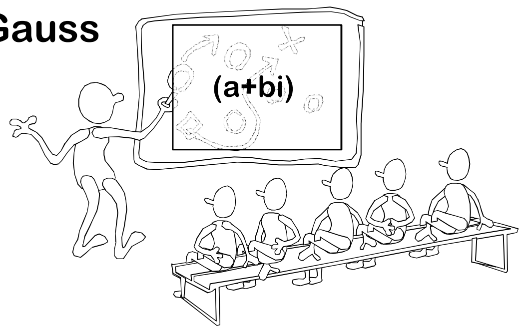
Great Theoretical Ideas in Computer Science

Grade School Revisited: How To Multiply Two Numbers

Lecture 23 (November 13, 2007)



Gauss



Gauss' Complex Puzzle

Remember how to multiply two complex numbers a + bi and c + di?

(a+bi)(c+di) = [ac –bd] + [ad + bc] i

Input: a,b,c,d
Output: ac-bd, ad+bc

If multiplying two real numbers costs $1 and adding them costs a penny, what is the cheapest way to obtain the output from the input?

Can you do better than $4.02?

Gauss' $3.05 Method

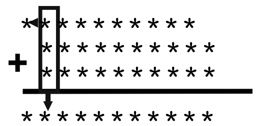
Input: a,b,c,d
Output: ac-bd, ad+bc

c	$X_1 = a + b$	
c	$X_2 = c + d$	
\$	$X_3 = X_1 X_2$	$= ac + ad + bc + bd$
\$	$X_4 = ac$	
\$	$X_5 = bd$	
c	$X_6 = X_4 – X_5$	$= ac - bd$
cc	$X_7 = X_3 – X_4 – X_5$	$= bc + ad$

The Gauss optimization saves one multiplication out of four.

It requires 25% less work.

Time complexity of grade school addition

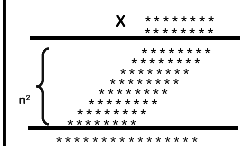


T(n) = amount of time grade school addition uses to add two n-bit numbers

We saw that T(n) was linear

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

Time complexity of grade school multiplication

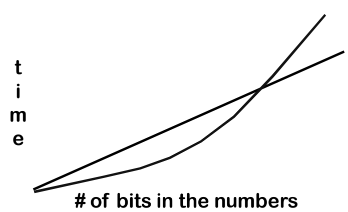


T(n) = The amount of time grade school multiplication uses to add two n-bit numbers

We saw that T(n) was quadratic

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

Grade School Addition: Linear time
Grade School Multiplication: Quadratic time



No matter how dramatic the difference in the constants, the quadratic curve will eventually dominate the linear curve

Is there a sub-linear time method for addition?

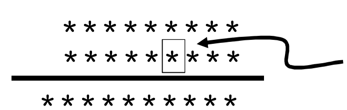
Any addition algorithm takes $\Omega(n)$ time

Claim: Any algorithm for addition must read all of the input bits

Proof: Suppose there is a mystery algorithm A that does not examine each bit

Give A a pair of numbers. There must be some unexamined bit position i in one of the numbers

Any addition algorithm takes $\Omega(n)$ time



A did not read this bit at position i

If A is not correct on the inputs, we found a bug

If A is correct, flip the bit at position i and give A the new pair of numbers. A gives the same answer as before, which is now wrong.

Grade school addition can't be improved upon by more than a constant factor

Grade School Addition: $\Theta(n)$ time. Furthermore, it is optimal

Grade School Multiplication: $\Theta(n^2)$ time

Is there a clever algorithm to multiply two numbers in linear time?

Despite years of research, no one knows! If you resolve this question, Carnegie Mellon will give you a PhD!

Can we even break the quadratic time barrier?

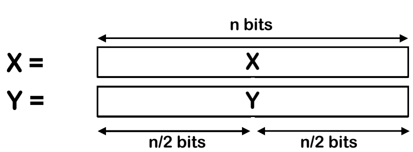
In other words, can we do something very different than grade school multiplication?

Divide And Conquer

An approach to faster algorithms:

- DIVIDE a problem into smaller subproblems
- CONQUER them recursively
- GLUE the answers together so as to obtain the answer to the larger problem

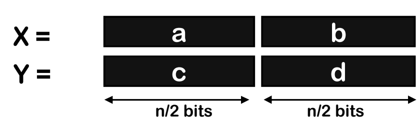
Multiplication of 2 n-bit numbers



$$X = a\,2^{n/2} + b \qquad Y = c\,2^{n/2} + d$$

$$X \times Y = ac\,2^n + (ad + bc)\,2^{n/2} + bd$$

Multiplication of 2 n-bit numbers



$$X \times Y = ac\,2^n + (ad + bc)\,2^{n/2} + bd$$

```
MULT(X,Y):
    If |X| = |Y| = 1 then return XY
    else  break X into a;b and Y into c;d
          return MULT(a,c) 2^n + (MULT(a,d)
          + MULT(b,c)) 2^(n/2) + MULT(b,d)
```

Same thing for numbers in decimal!

n digits

X = a | b

Y = c | d

n/2 digits | n/2 digits

$X = a\,10^{n/2} + b \quad Y = c\,10^{n/2} + d$

$X \times Y = ac\,10^{n} + (ad + bc)\,10^{n/2} + bd$

Multiplying (Divide & Conquer style)

12345678 * 21394276

1234*2139 1234*4276 5678*2139 5678*4276

12*21 12*39 34*21 34*39

1*2 1*1 2*2 2*1

2 1 4 2

Hence: $12*21 = 2*10^2 + (1 + 4)10^1 + 2 = 252$

a | b

c | d

Multiplying (Divide & Conquer style)

12345678 * 21394276

1234*2139 1234*4276 5678*2139 5678*4276

$\boxed{252}\,*10^4 + \boxed{468}\,*10^2 + \boxed{714}\,*10^2 + \boxed{1326}\,*1 = 2639526$

a | b

c | d

Multiplying (Divide & Conquer style)

12345678 * 21394276

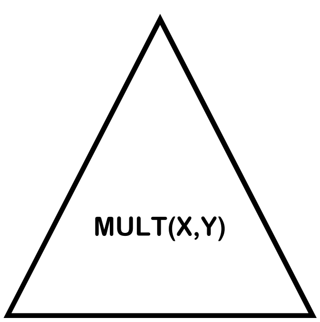
$\boxed{2639526}\,*10^8 + \boxed{5276584}\,*10^4 + \boxed{12145242}\,*10^4 + \boxed{24279128}\,*1$

= 264126842539128

a | b

c | d

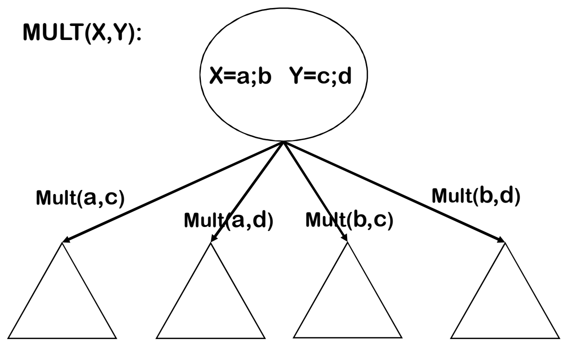
Divide, Conquer, and Glue



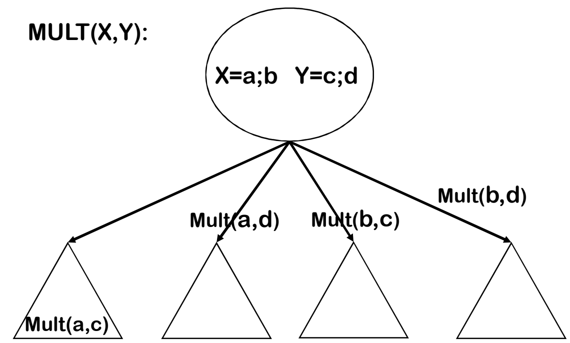
Divide, Conquer, and Glue

MULT(X,Y): if |X| = |Y| = 1
then return XY,
else…

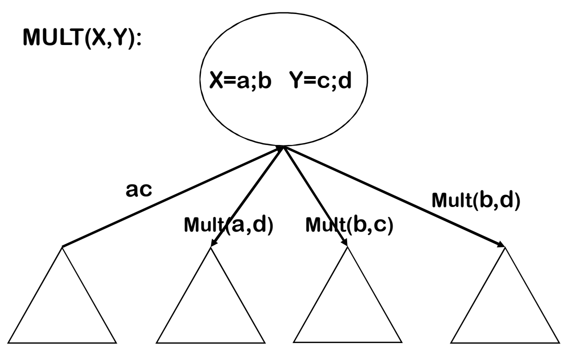
Divide, Conquer, and Glue



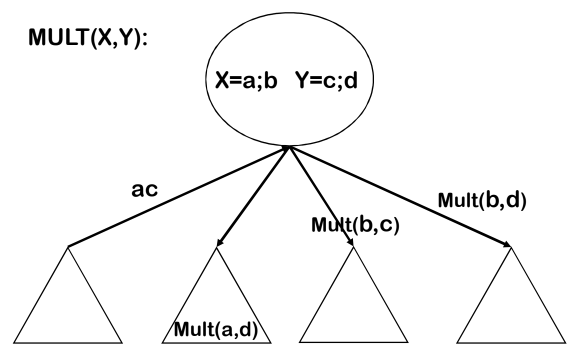
Divide, Conquer, and Glue



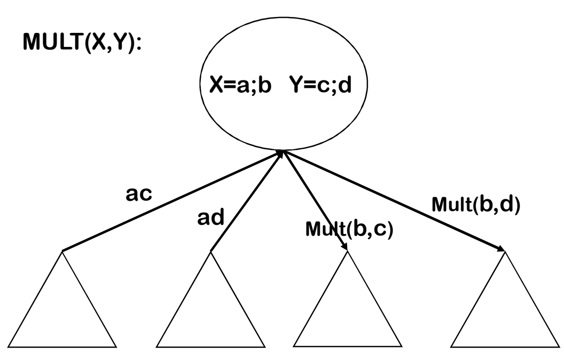
Divide, Conquer, and Glue



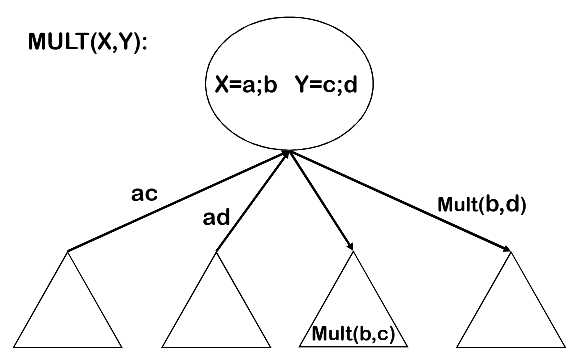
Divide, Conquer, and Glue



Divide, Conquer, and Glue

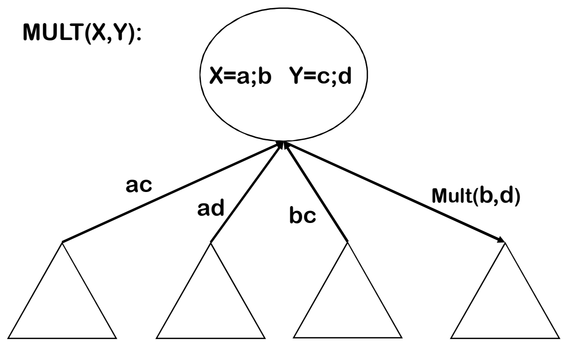


Divide, Conquer, and Glue



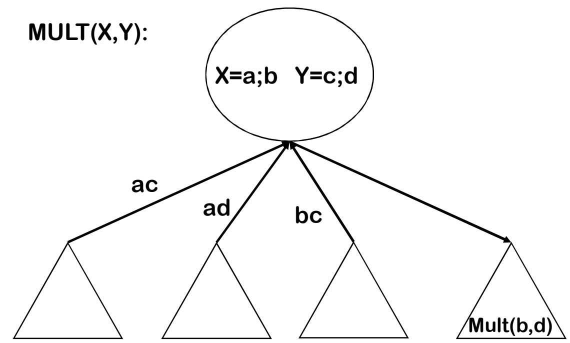
Divide, Conquer, and Glue

MULT(X,Y):



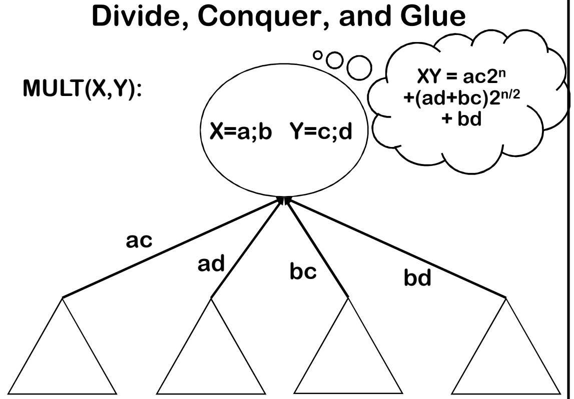
Divide, Conquer, and Glue

MULT(X,Y):



Divide, Conquer, and Glue

MULT(X,Y):



Time required by MULT

T(n) = time taken by MULT on two n-bit numbers

What is T(n)? What is its growth rate?

Big Question: Is it $\Theta(n^2)$?

$$T(n) = 4\,T(n/2) + (k'n + k'')$$

conquering time

divide and glue

Recurrence Relation

$T(1) = k$ for some constant k

$T(n) = 4\,T(n/2) + k'n + k''$ for constants k' and k''

MULT(X,Y):
  If |X| = |Y| = 1 then return XY
  else break X into a;b and Y into c;d
    return MULT(a,c) 2^n + (MULT(a,d) + MULT(b,c)) $2^{n/2}$ + MULT(b,d)

Recurrence Relation

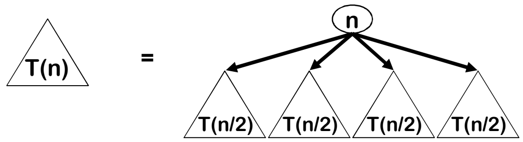
$T(1) = 1$

$T(n) = 4\,T(n/2) + n$

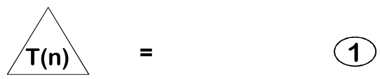
MULT(X,Y):
  If |X| = |Y| = 1 then return XY
  else break X into a;b and Y into c;d
    return MULT(a,c) 2^n + (MULT(a,d) + MULT(b,c)) $2^{n/2}$ + MULT(b,d)

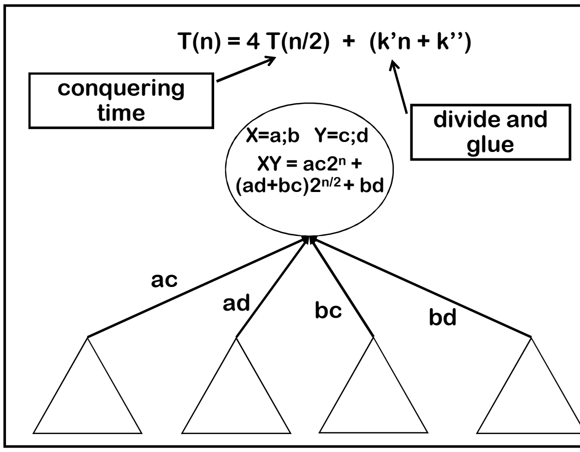
Technique: Labeled Tree Representation

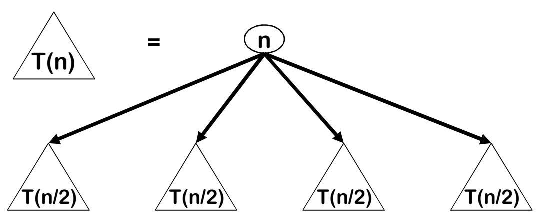
T(n)	=	n + 4 T(n/2)

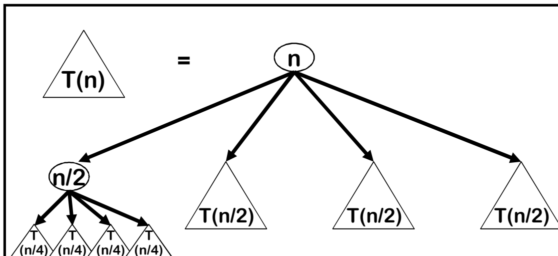


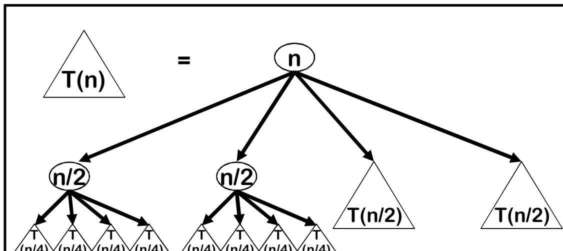
T(1)	=	1











0	n
1	n/2 + n/2 + n/2 + n/2
2	
i	Level i is the sum of 4^i copies of $n/2^i$
	. . . .
$\log_2(n)$	1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1

$1n =$	n
$2n =$	n/2 + n/2 + n/2 + n/2
$4n =$	
$2^i n =$	Level i is the sum of 4^i copies of $n/2^i$
	. . . .
$(n)n =$	1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1

$n(1+2+4+8+ \ldots +n) = n(2n-1) = 2n^2-n$

Divide and Conquer MULT: $\Theta(n^2)$ time
Grade School Multiplication: $\Theta(n^2)$ time

MULT revisited

```
MULT(X,Y):
    If |X| = |Y| = 1 then return XY
    else  break X into a;b and Y into c;d
          return MULT(a,c) 2^n + (MULT(a,d)
          + MULT(b,c)) 2^(n/2) + MULT(b,d)
```

MULT calls itself 4 times. Can you see a way to reduce the number of calls?

Gauss' optimization

Input: a,b,c,d
Output: ac-bd, ad+bc

c	$X_1 = a + b$	
c	$X_2 = c + d$	
\$	$X_3 = X_1 X_2$	$= ac + ad + bc + bd$
\$	$X_4 = ac$	
\$	$X_5 = bd$	
c	$X_6 = X_4 - X_5$	$= ac - bd$
cc	$X_7 = X_3 - X_4 - X_5$	$= bc + ad$

Karatsuba, Anatolii Alexeevich (1937-)

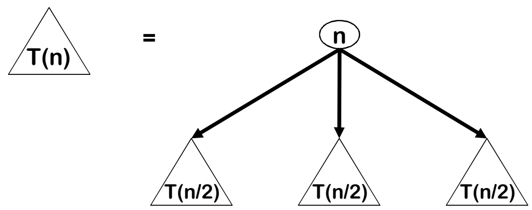
Sometime in the late 1950's Karatsuba had formulated the first algorithm to break the n^2 barrier!

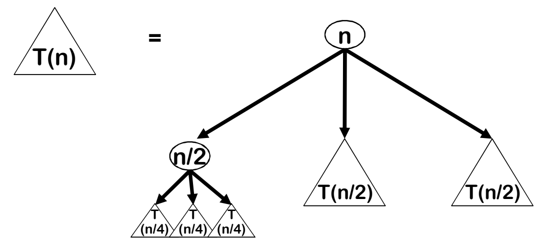
Gaussified MULT (Karatsuba 1962)

```
MULT(X,Y):
    If |X| = |Y| = 1 then return XY
    else  break X into a;b and Y into c;d
                e : = MULT(a,c)
                f  := MULT(b,d)
    return
    e 2^n + (MULT(a+b,c+d) – e - f) 2^(n/2) + f
```

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

Actually: $T(n) = 2\ T(n/2) + T(n/2 + 1) + kn$





0	n
1	n/2 + n/2 + n/2
2	
i	Level i is the sum of 3^i copies of $n/2^i$
	. . . .
$\log_2(n)$	1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1

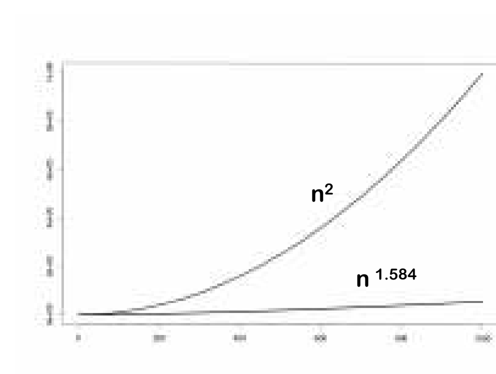
1n =	n
3/2n =	n/2 + n/2 + n/2
9/4n =	
$(3/2)^i n$ =	Level i is the sum of 3^i copies of $n/2^i$
	. . . .
$(3/2)^{\log n} n$ =	1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1

$$n(1+3/2+(3/2)^2+ \ldots + (3/2)^{\log_2 n}) \quad = \quad 3n^{1.58\ldots} - 2n$$

Dramatic Improvement for Large n

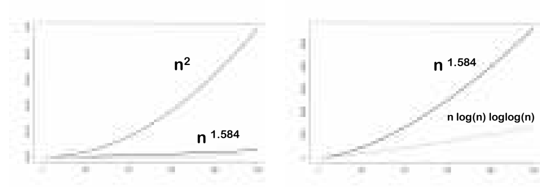
$$\begin{aligned} T(n) &= 3n^{\log_2 3} - 2n \\ &= \Theta(n^{\log_2 3}) \\ &= \Theta(n^{1.58\ldots}) \end{aligned}$$

A huge savings over $\Theta(n^2)$ when n gets large.



Multiplication Algorithms

Kindergarten	$n2^n$
Grade School	n^2
Karatsuba	$n^{1.58...}$
Fastest Known	n logn loglogn



A short digression on parallel algorithms

Adding n numbers

For the next two slides, assume that the CPU can access any number, and add/mult/subtract any two numbers in unit time.

Given n numbers $a_1, a_2, \ldots, a_n$
How much time to add them all up using 1 CPU?

$\Omega(n)$
The CPU must at least look at all the numbers.

Adding n numbers (in parallel)

Given n numbers $a_1, a_2, \ldots, a_n$
How much time to add them all up using as many CPUs as you want?

Think of this as getting a group of people together to add the n numbers.

Not clear if any one CPU must look at all numbers so $\Omega(n)$ lower does not hold any more.
In fact, we can do it in O(log n) time.

Addition in the old model?

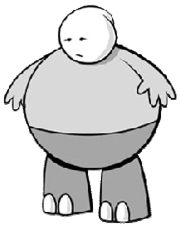
How do CPUs add n-bit numbers?

```
 * * * * * * * * * *
   * * * * * * * * * *
   * * * * * * * * * *
 -----------------------
 * * * * * * * * * * * *
```

The k-th carry bit depends on the partial sum to the right of it

If we had all the carry bits, we could compute the sum fast.

How do we compute all the carry bits?



Here's What
You Need to
Know…

- Gauss's Multiplication Trick
- Proof of Lower bound for addition
- Divide and Conquer
- Solving Recurrences
- Karatsuba Multiplication