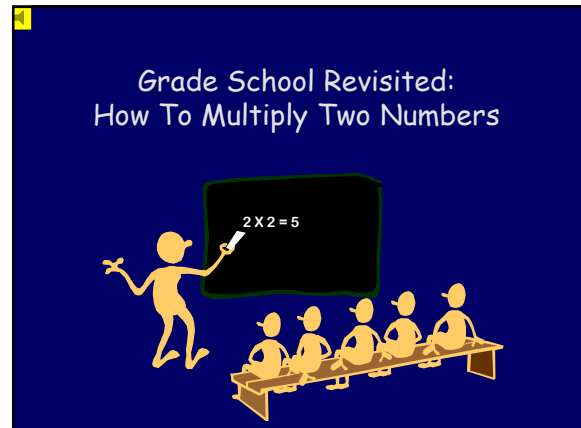


Announcements

For any questions, email us:
ryanw@ias.edu and virgi@ias.edu

Handouts from our lectures can be found at:
<http://www.math.ias.edu/~virgi/greatcs/handouts.html>



Complex Numbers

Multiplying 2 complex numbers: ($i^2 = -1$)

$$(a+bi)(c+di) = [ac - bd] + [ad + bc]i$$

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

Suppose a real multiplication costs \$1 and an addition/subtraction costs \$0.01.

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: $ac - bd, ad + bc$

$$m_1 = ac \quad \$1$$

$$m_2 = bd \quad \$1$$

$$A_1 = m_1 - m_2 = ac - bd \quad \leftarrow \$0.01$$

$$m_3 = (a+b)(c+d) = ac + ad + bc + bd$$

$$A_2 = m_3 - m_1 - m_2 = ad + bc$$

\$1.02 \$0.02

Moral

The most obvious solution might not be the best one!

Often there are very clever ways to save work...

Question:

The Gauss trick saves one multiplication out of four. It requires 25% less work.

Could there be a context where performing 3 multiplications for every 4 provides a more dramatic savings?

How to add two n digit numbers.

$$\begin{array}{r}
 + \quad * * * * * * * * * * \\
 \quad * * * * * * * * * * \\
 \hline
 \end{array}$$

How to add two n digit numbers.

$$\begin{array}{r}
 + \quad * * * * * * * * * * * * \\
 \quad * * * * * * * * * * * * \\
 \hline
 \end{array}$$

A yellow box highlights the last two digits of both numbers. An arrow points from the top-right corner of the box to the digit in the second row, and another arrow points from the bottom-right corner of the box to the digit in the first row.

How to add two n digit numbers.

$$\begin{array}{r}
 + \quad * * * * * * * * * * * * \\
 \quad * * * * * * * * * * * * \\
 \hline
 \end{array}$$

A yellow box highlights the last two digits of both numbers. An arrow points from the top-right corner of the box to the digit in the second row, and another arrow points from the bottom-right corner of the box to the digit in the first row.

How to add two n digit numbers.

$$\begin{array}{r}
 + \quad * * * * * * * * * * * * * * \\
 \quad * * * * * * * * * * * * * * \\
 \hline
 \end{array}$$

A yellow box highlights the last two digits of both numbers. An arrow points from the top-right corner of the box to the digit in the second row, and another arrow points from the bottom-right corner of the box to the digit in the first row.

How to add two n digit numbers.

$$\begin{array}{r}
 + \quad * * * * * * * * * * * * * * \\
 \quad * * * * * * * * * * * * * * \\
 \hline
 \end{array}$$

A yellow box highlights the last two digits of both numbers. An arrow points from the top-right corner of the box to the digit in the second row, and another arrow points from the bottom-right corner of the box to the digit in the first row.

How to add two n digit numbers.

$$\begin{array}{r}
 + \quad * * * * * * * * * * * * * * * * \\
 \quad * * * * * * * * * * * * * * * * \\
 \hline
 \end{array}$$

A yellow box highlights the last two digits of both numbers. An arrow points from the top-right corner of the box to the digit in the second row, and another arrow points from the bottom-right corner of the box to the digit in the first row.

Time complexity of grade school addition

```

+ * * * * *
  * * * * *
  * * * * *
  * * * * *
  * * * * *
            
```

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

What do you mean by "time"?

Roadblock ???

A given algorithm will take different amounts of time on the same inputs depending on such factors as:

- Processor speed
- Instruction set
- Disk speed
- Brand of compiler

Our Goal

We want to define TIME in a sense that transcends implementation details and allows us to make assertions about grade school addition in a very general way.


Hold on! You just admitted that it makes no sense to measure the time, $T(n)$, taken by the method of grade school addition since the time depends on the implementation details. We will have to speak of the time taken by a particular implementation, as opposed to the time taken by the method in the abstract.

Don't jump to conclusions! Your objections are serious, but not insurmountable. There is a very nice sense in which we can analyze grade school addition without ever having to worry about implementation details.


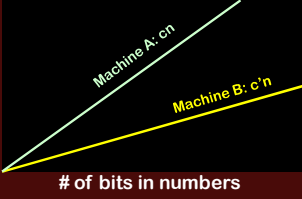
Here is how it works . . .

On any reasonable computer adding 3 bits and writing down the two bit answer can be done in constant time. Pick any particular computer A and define c to be the time it takes to perform on that computer.


Total time to add two n -bit numbers using grade school addition: cn [c time for each of n columns]

Implemented on another computer B the running time will be $c'n$ where c' is the time it takes to perform  on that computer.

Total time to add two n -bit numbers using grade school addition: $c'n$ [c' time for each of n columns]





The fact that we get a line is **invariant** under changes of implementations. Different machines result in different slopes, but time grows linearly as input size increases.

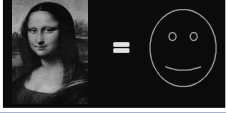


Thus we arrive at an implementation independent insight: Grade School Addition is a linear time algorithm.


Determining the growth rate of the resource curve as the problem size increases is one of the fundamental ideas of computer science.



Abstraction:
Abstract away the inessential features of a problem or solution



I see! We can define away the details of the world that we do not wish to currently study, in order to recognize the similarities between seemingly different things..




TIME vs INPUT SIZE

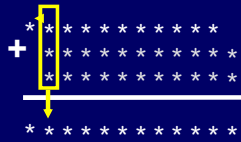
For any algorithm, define
INPUT SIZE = # of bits/digits to specify inputs,

Define
 $T(n)$ = the worst-case amount of time used on inputs of size n .

We Often Ask:
What is the GROWTH RATE of $T(n)$?



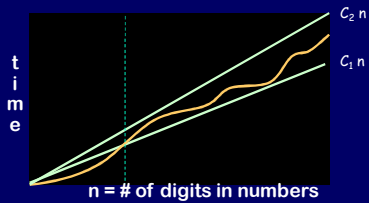
Time complexity of grade school addition



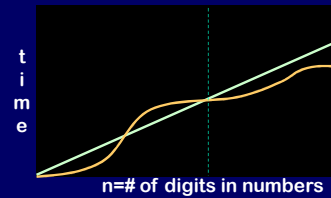
On any reasonable computer adding 3 digits can be done in *constant* time.

$T(n)$ = The amount of time grade school addition uses to add two n digit numbers
= $\theta(n)$ = linear time.

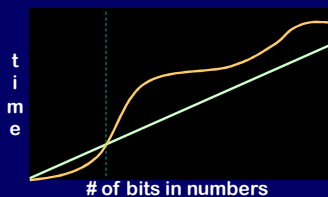
$f = \theta(n)$ means that f can be sandwiched between two lines for all large n



$f = O(n)$ means that there is a line that can be drawn that stays above f from some point on



$f = \Omega(n)$ means that there is a line that can be drawn that stays below f from some point on



$$f(n) = \theta(n)$$

So $f(n) = \theta(n)$ if and only if

- $f(n) = O(n)$, and
- $f(n) = \Omega(n)$

Please feel free to ask questions!



Is there a faster way to add?

QUESTION: Is there an algorithm to add two n digit numbers whose time $T(n)$ grows *sub-linearly* in n ?

For any line cn , for all sufficiently large n , $T(n)$ is always below cn .

This would mean we do not read all the digits! (since then $T(n) \geq n$)

Any correct algorithm for addition must read all of the input digits

- Suppose there is a mystery algorithm ALGO that does not examine each digit
- Give ALGO two numbers X and Y.
- There must be some *unexamined* digit in one of the numbers, say X_i – the i -th digit of X.
- If ALGO does not return $X+Y$, we found a bug!
- If ALGO outputs $X+Y$, change X_i to something else and give ALGO Y and the **new X'**. Note $X'+Y$ is different from $X+Y$!
- Since ALGO does not look at the i -th digit of X' , it will return $X+Y$ again, and not $X'+Y$!
- ALGO is **incorrect**!

So any algorithm for addition must use time at least linear in the size of the numbers.

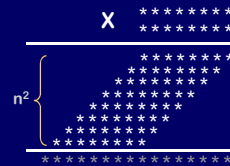
Grade school addition is essentially as good as it can be.



How to multiply two n digit numbers.



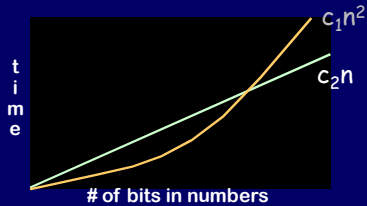
How to multiply two n-digit numbers.



I get it! The total time is bounded by cn^2 .



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



No matter how dramatic the difference in c_1 and c_2 the quadratic curve will eventually dominate the linear curve


Neat! We have demonstrated that as things scale multiplication is a harder problem than addition.

Mathematical confirmation of our common sense.




Don't jump to conclusions!
 We have argued that grade school multiplication uses more time than grade school addition. This is a comparison of the complexity of two algorithms.

To argue that multiplication is an inherently harder problem than addition we would have to show that no possible multiplication algorithm runs in linear time.



Grade School Addition: $\Theta(n)$ time
 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, Rutgers/Princeton/MIT... will give you a PhD!



Is there a faster way to multiply two numbers than the way you learned in grade school?



Divide And Conquer
 (an approach to faster algorithms)

DIVIDE my instance to the problem into smaller instances to the same problem.

Have a friend solve them.
 Do not worry about it yourself.

GLUE the answers together so as to obtain the answer to your larger instance.

Example: Find the Minimum among n numbers

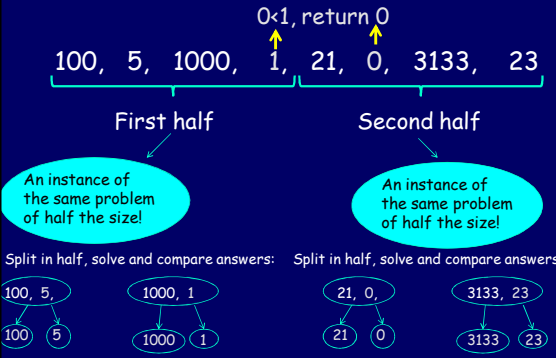
100, 5, 1000, 1, 21, 0, 3133, 23

0 < 1, return 0

First half Second half

An instance of the same problem of half the size! An instance of the same problem of half the size!

Split in half, solve and compare answers: Split in half, solve and compare answers:



$T(n)$ = Runtime for finding the minimum of n numbers

When the blob has one element, it returns it in constant time: $T(1) = 1$.

Each larger blob introduces two new blobs of half the size: they get solved in $T(n/2)$ time each, and then the blob does one comparison: constant amount of work, say 1.

Recurrence relation: $T(n) = 2T(n/2) + 1, T(1) = 1$

$T(n) = 2T(n/2) + 1$

Claim: $T(n) = 2n - 1$

Induction on n :

- Base case: $T(1) = 2 \cdot 1 - 1 = 1$
- Induction: If $T(n/2) = 2 \cdot (n/2) - 1 = n - 1$, then

$T(n) = 2 \cdot (n - 1) + 1 = 2n - 1$.

Guess and verify approach

Multiplication of two n digit numbers

$X = \begin{matrix} \boxed{a} & \boxed{b} \\ \boxed{c} & \boxed{d} \end{matrix}$
 $Y = \begin{matrix} \boxed{c} & \boxed{d} \end{matrix}$

$n/2$ digits $n/2$ digits

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

$XY = (a \cdot 10^{n/2} + b)(c \cdot 10^{n/2} + d) = ac \cdot 10^n + (ad+bc) \cdot 10^{n/2} + bd$

Multiplication of two n digit numbers

$X = \begin{matrix} \boxed{a} & \boxed{b} \\ \boxed{c} & \boxed{d} \end{matrix}$
 $Y = \begin{matrix} \boxed{c} & \boxed{d} \end{matrix}$

$XY = ac \cdot 10^n + (ad+bc) \cdot 10^{n/2} + bd$

MULT(X,Y):
 If $|X| = |Y| = 1$ then
 RETURN XY
 Break X into a;b and Y into c;d
 RETURN
 $MULT(a,c) \cdot 10^n + (MULT(a,d) + MULT(b,c)) \cdot 10^{n/2} + MULT(b,d)$

Time required by MULT

$T(n)$ = time taken by MULT on two n digit numbers

What is $T(n)$?
 Is it $\theta(n^2)$?

Recurrence Relation

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

4 calls on half the size Break X and Y into two Combine results of 4 calls

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

MULT(X,Y):
 If $|X| = |Y| = 1$ then
 RETURN XY
 Break X into a;b and Y into c;d
 RETURN
 $MULT(a,c) \cdot 10^n + (MULT(a,d) + MULT(b,c)) \cdot 10^{n/2} + MULT(b,d)$

Let's be concrete

$$T(1) = 1$$

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

How do we unravel $T(n)$ so that we can determine its growth rate?



Technique 1 Guess and Verify

Recurrence Relation:

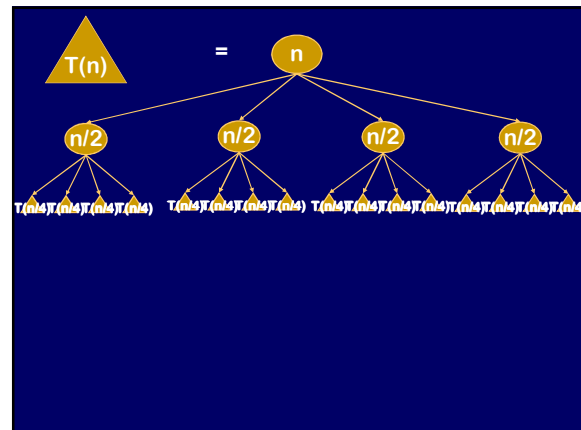
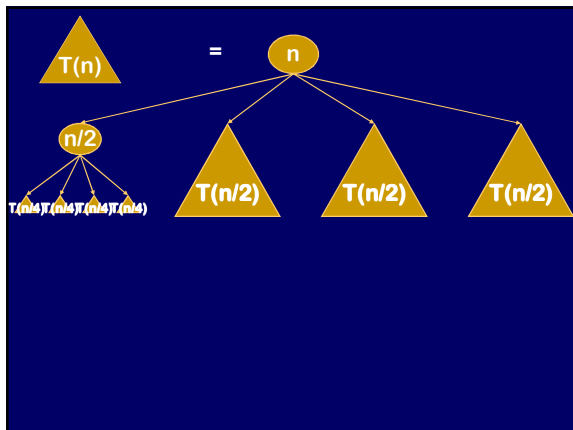
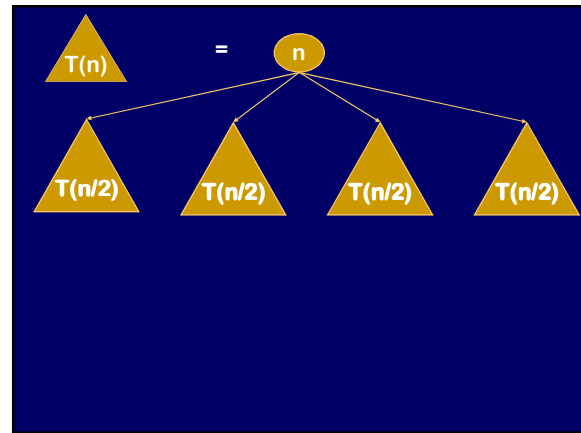
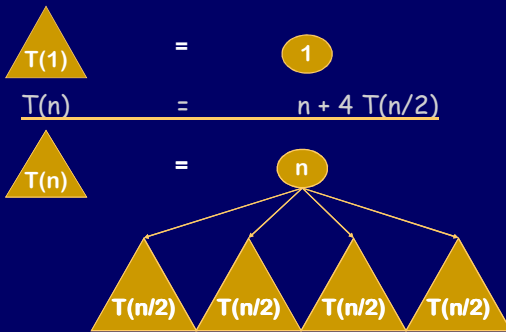
$$T(1) = 1 \text{ \& } T(n) = 4T(n/2) + n$$

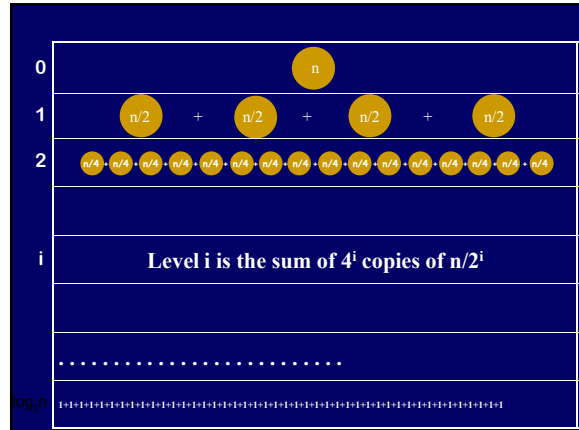
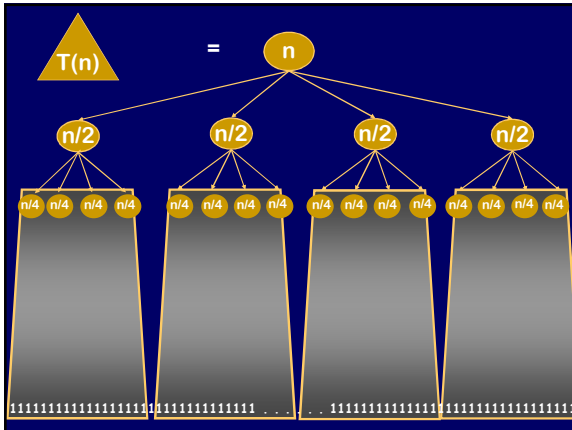
Guess: $T(n) = 2n^2 - n$

Verify:	Left Hand Side	Right Hand Side
	$T(1) = 2(1)^2 - 1$	1
	$T(n) = 2n^2 - n$	$4T(n/2) + n$ $= 4 [2(n/2)^2 - (n/2)] + n$ $= 2n^2 - n$

Technique 2: Decorate The Tree

$$T(1) = 1$$





A diagram showing the levels of a recursion tree with associated mathematical calculations. Level 0: n . Level 1: $n/2 + n/2 + n/2 + n/2$. Level 2: $n/4 + n/4 + n/4 + n/4 + n/4 + n/4 + n/4 + n/4$. Level i : "Level i is the sum of 4^i copies of $n/2^i$ ". The bottom of the diagram is labeled $\log_4 n$. To the right of the diagram, the following calculations are shown:
 $= 1 \cdot n$
 $= 4 \cdot n/2$
 $= 16 \cdot n/4$
 $= 4^i \cdot n/2^i$
 $= 4^{\log_4 n} \cdot n/2^{\log_4 n}$
 $= n^{\log_4 4} \cdot 1$
Total: $\theta(n^{\log_4 4}) = \theta(n^2)$

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

All that work for nothing!

A cartoon character with a blue hat and a yellow body, standing with one hand on its hip. A speech bubble points to the text "All that work for nothing!".

MULT revisited

MULT(X,Y):
 If $|X| = |Y| = 1$ then
 RETURN XY
 Break X into a;b and Y into c;d
 RETURN
 MULT(a,c) 10^a + (MULT(a,d) + MULT(b,c)) 10^{a/2} + MULT(b,d)

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

A cartoon character with a yellow body and a blue hat, standing with one hand on its hip.

Gauss' Hack:
 Input: a,b,c,d Output: ac, ad+bc, bd

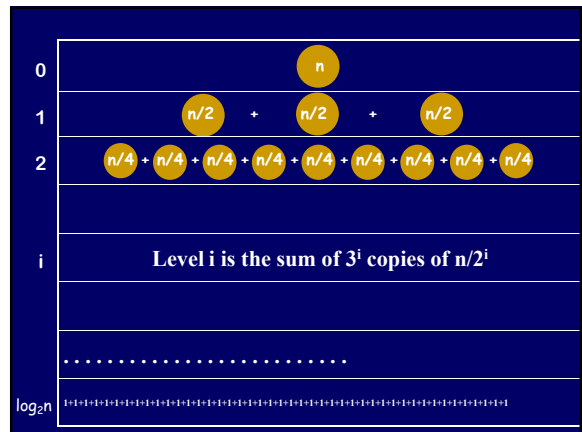
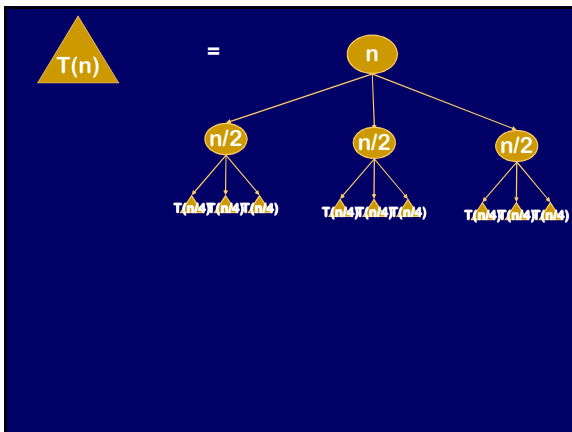
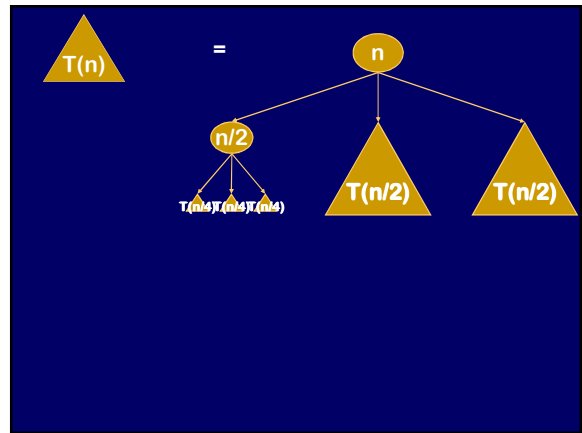
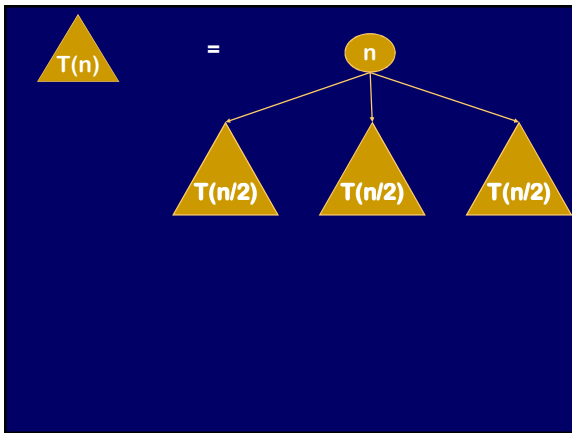
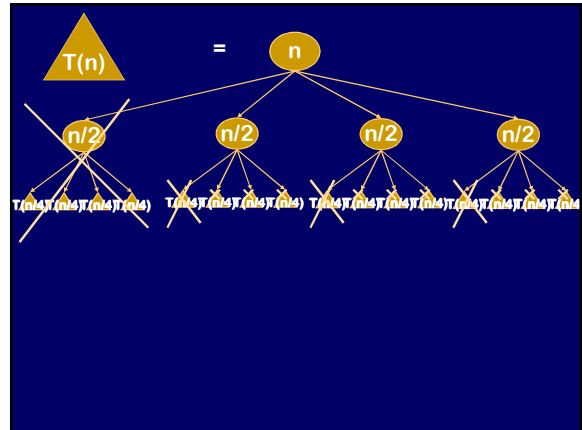
$A_1 = ac$
 $A_3 = bd$
 $m_3 = (a+b)(c+d) = \cancel{ac} + \cancel{ad} + \cancel{bc} + bd$
 $A_2 = m_3 - A_1 - A_3 = ad + bc$

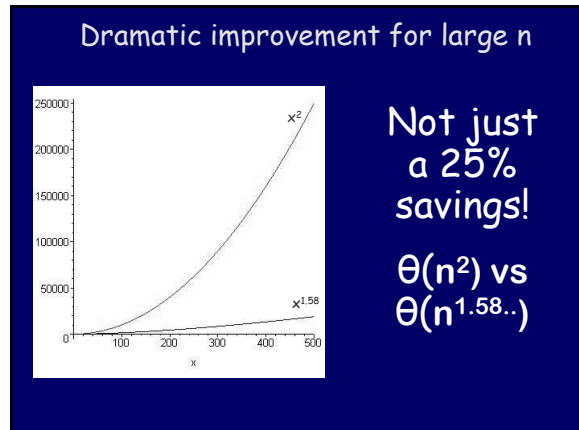
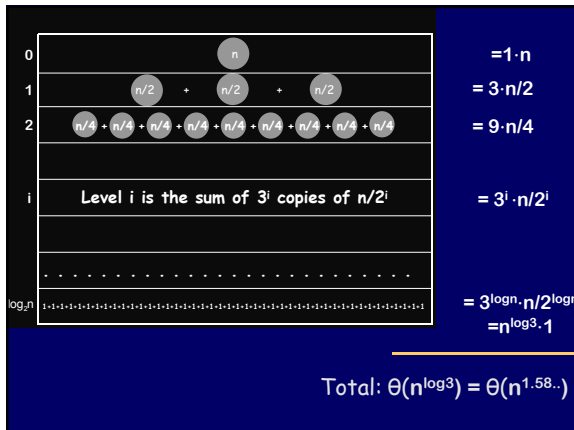
Gaussified MULT
 (Karatsuba 1962)

MULT(X,Y):
 If $|X| = |Y| = 1$ then RETURN XY
 Break X into a;b and Y into c;d
 $e = \text{MULT}(a,c)$ and $f = \text{MULT}(b,d)$
 RETURN $e10^n + (\text{MULT}(a+b, c+d) - e - f) 10^{n/2} + f$

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

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





Multiplication Algorithms

Kindergarten ? $3 \cdot 4 = 3 + 3 + 3$	n^2 Homework
Grade School	n^2
Karatsuba	$n^{1.58..}$
Fastest Known	$n \log n 2^{O(\log^* n)}$ Fürer 2007

