Projeto e Análise de Algoritmos

Prof. Eduardo Laber

www-di.inf.puc-rio.br/~laber

laber@inf.puc-rio.br

Thinking about Algorithms Abstractly

Introduction

- So you want to be a computer scientist?
- Grade School Revisited: How To Multiply Two Numbers

Jeff Edmonds

York University

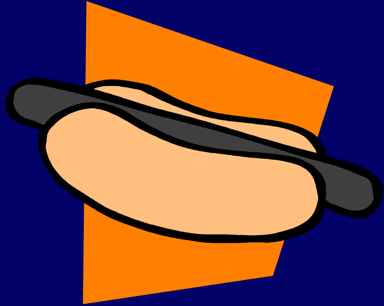
Lecture 1

So you want to be a computer scientist?

Is your goal to be a mundane programmer?

Or a great leader and thinker?

Original Thinking



Boss assigns task:

– Given today's prices of pork, grain, sawdust, …
– Given constraints on what constitutes a hotdog.
– Make the cheapest hotdog.

Everyday industry asks these questions.

Your answer:

- Um? Tell me what to code.

With more suffocated software engineering systems, the demand for mundane programmers will diminish.

Your answer:

- I learned this great algorithm that will work.

Soon all known algorithms will be available in libraries.

Your answer:

- I can develop a new algorithm for you.

Great thinkers
will always be needed.

The future belongs to the computer scientist who has

- Content: An up to date grasp of fundamental problems and solutions
- Method: Principles and techniques to solve the vast array of unfamiliar problems that arise in a rapidly changing field

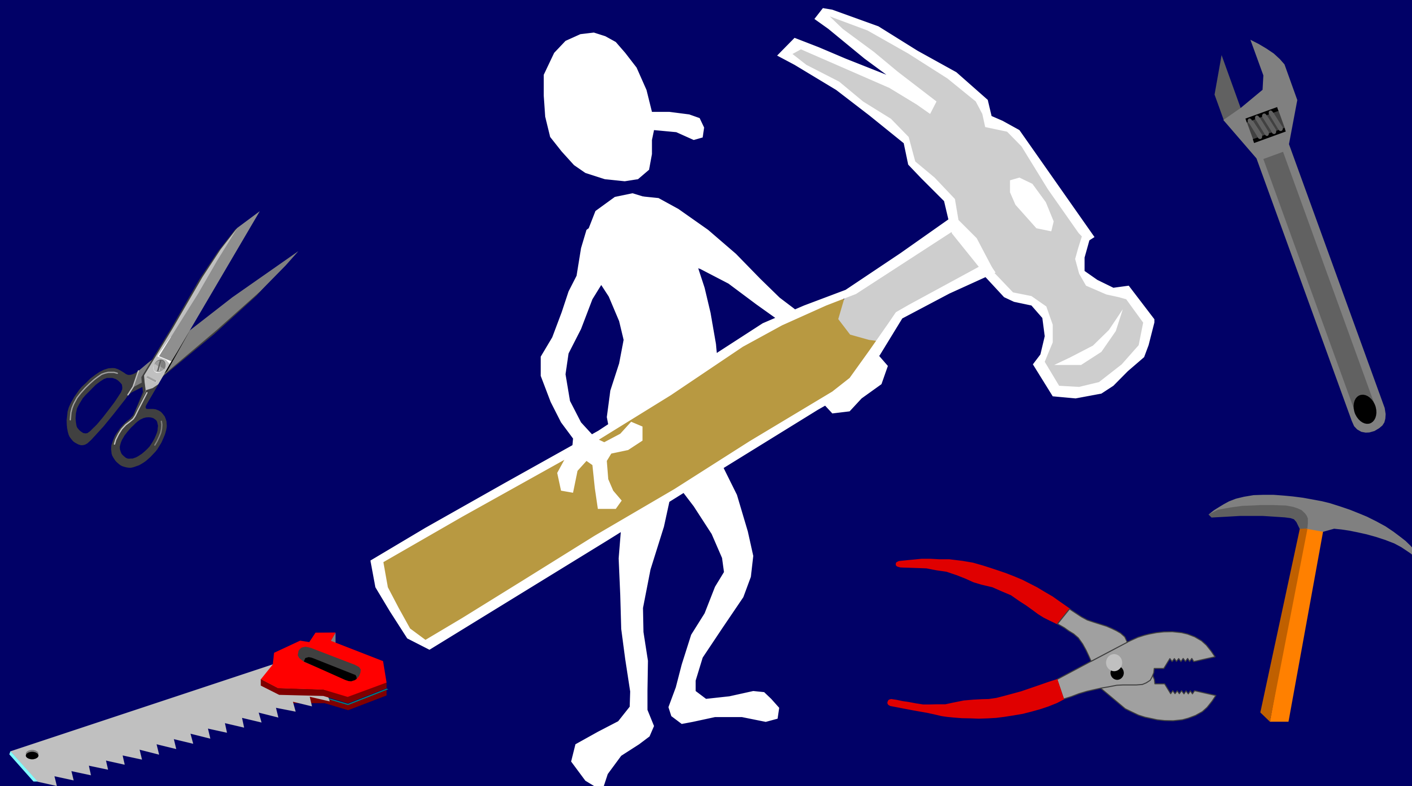
Course Content

- A list of algoirthms.
  - Learn their code.
  - Trace them until you are convenced that they work.
  - Impliment them.

```
class InsertionSortAlgorithm extends SortAlgorithm
{
  void sort(int a[]) throws Exception {
        for (int i = 1; i < a.length; i++) {
          int j = i;
          int B = a[i];
          while ((j > 0) && (a[j-1] > B)) {
            a[j] = a[j-1];
            j--; }
          a[j] = B;
       }}
```

Course Content

- A survey of algorithmic design techniques.
- Abstract thinking.
- How to develop new algorithms for any problem that may arise.

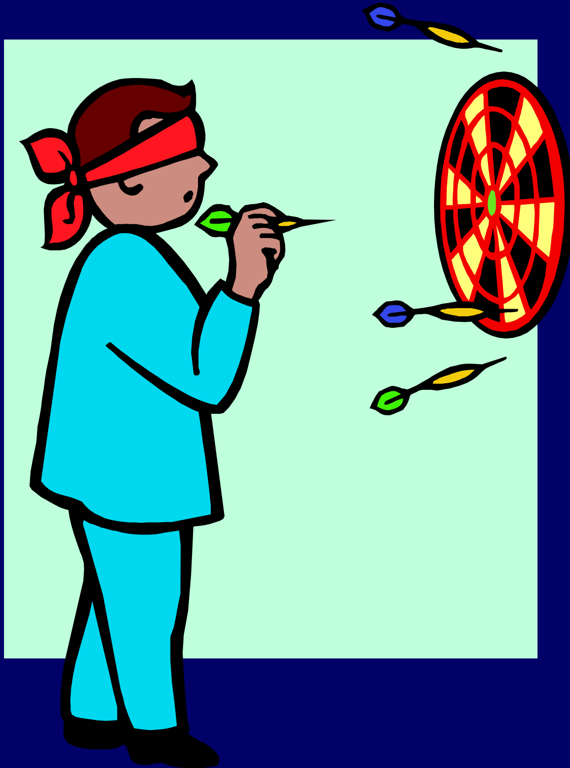


Study:

- Many experienced programmers were asked to code up binary search.

Study:

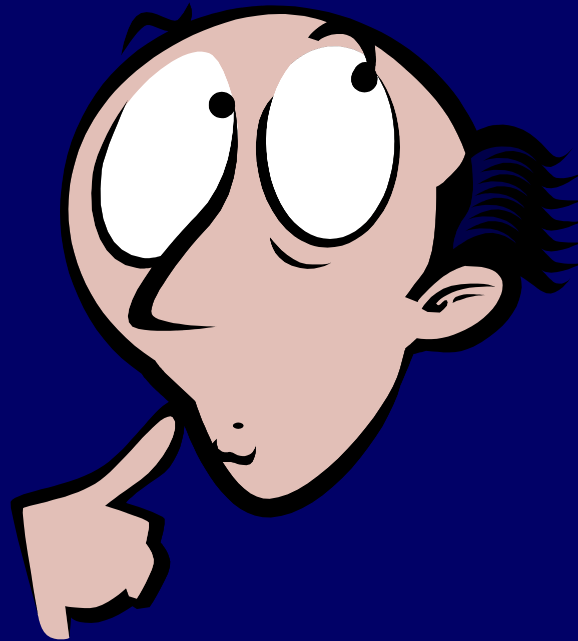
- Many experienced programmers were asked to code up binary search.



80% got it wrong

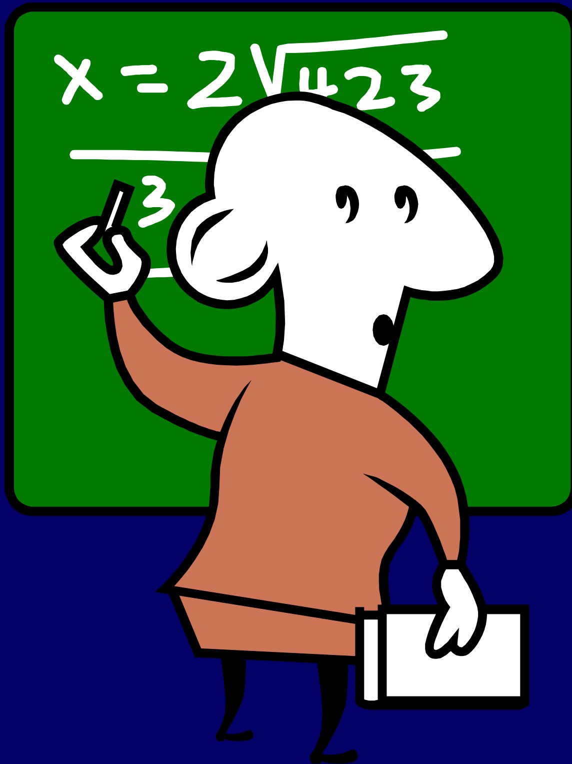
Good thing is was not for a nuclear power plant.

What did they lack?



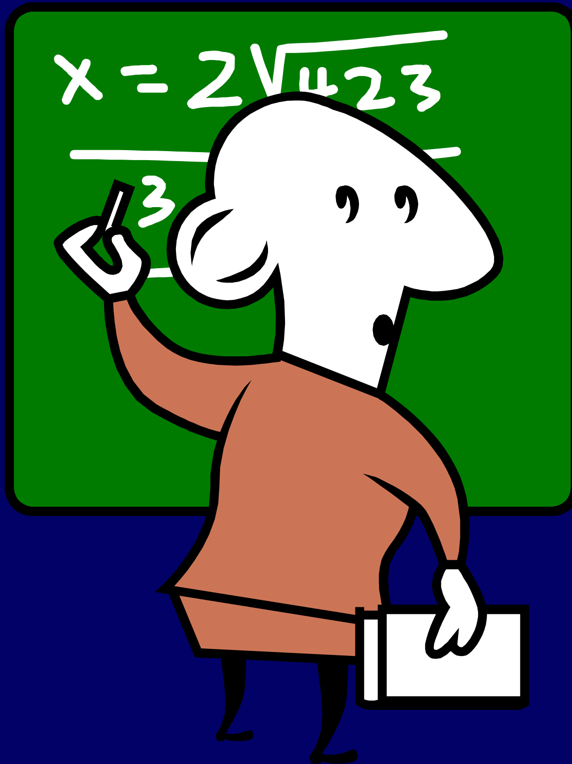
What did they lack?

- Formal proof methods?



What did they lack?

- Formal proof methods?



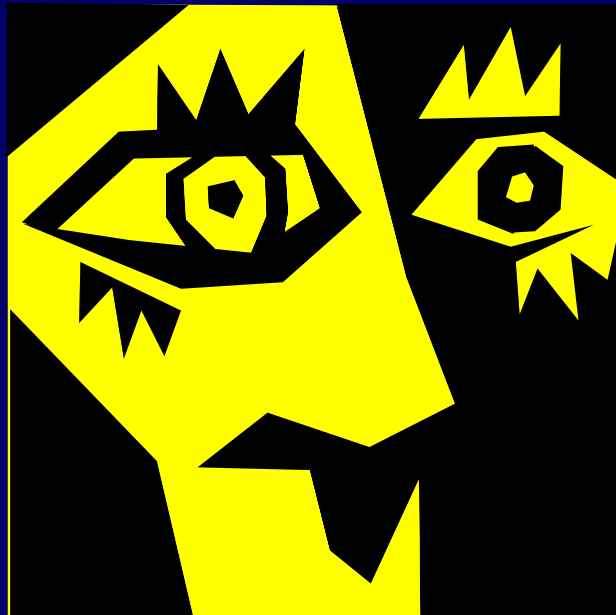
Yes, likely

Industry is starting to realize that formal methods are important.

But even without formal methods …. ?

What did they lack?

- Fundamental understanding of the algorithmic design techniques.
- Abstract thinking.



Course Content

Notations, analogies, and abstractions
for developing,
thinking about,
and describing algorithms
so correctness is transparent

Please feel free to ask questions!

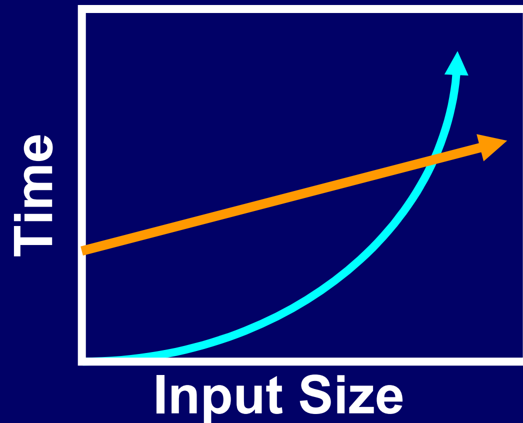
A survey of fundamental ideas and algorithmic design techniques

For example . . .

Start With Some Math

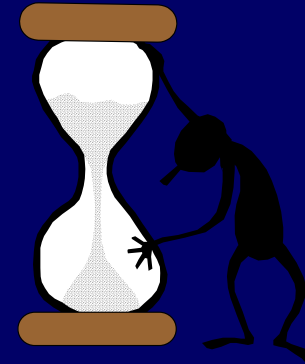
Classifying Functions

$$f(i) = n^{\Theta(n)}$$



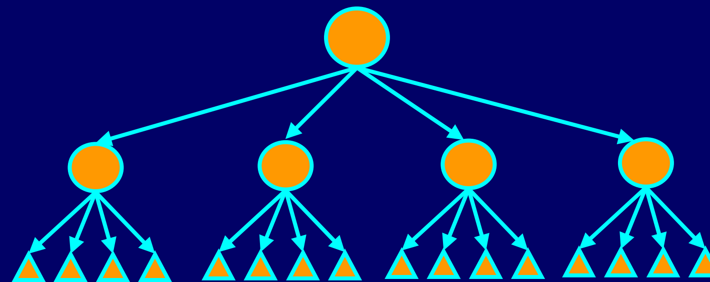
Time Complexity

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



Recurrence Relations

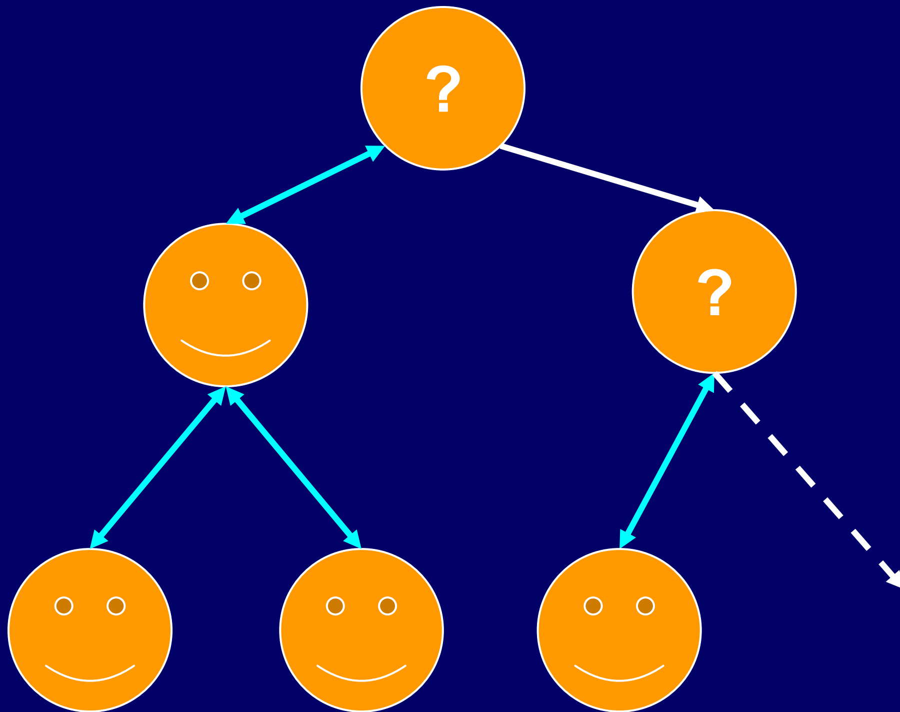
$$T(n) = a\,T(n/b) + f(n)$$

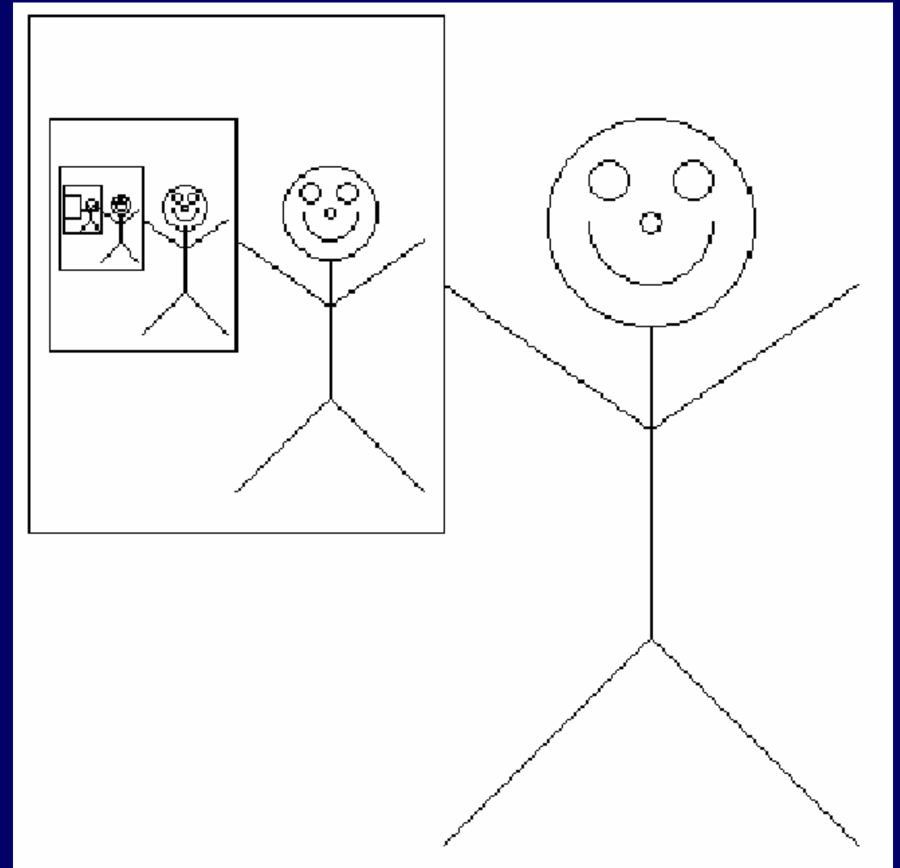


Graph Search Algorithms

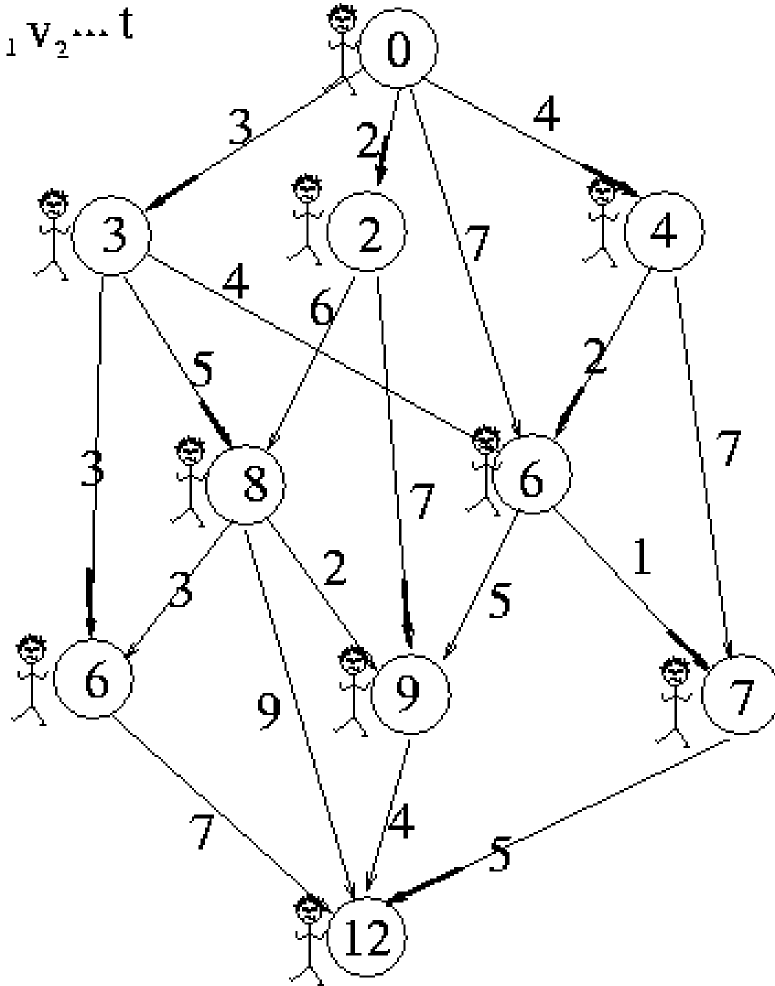
Greedy Algorithms

Recursive Algorithms



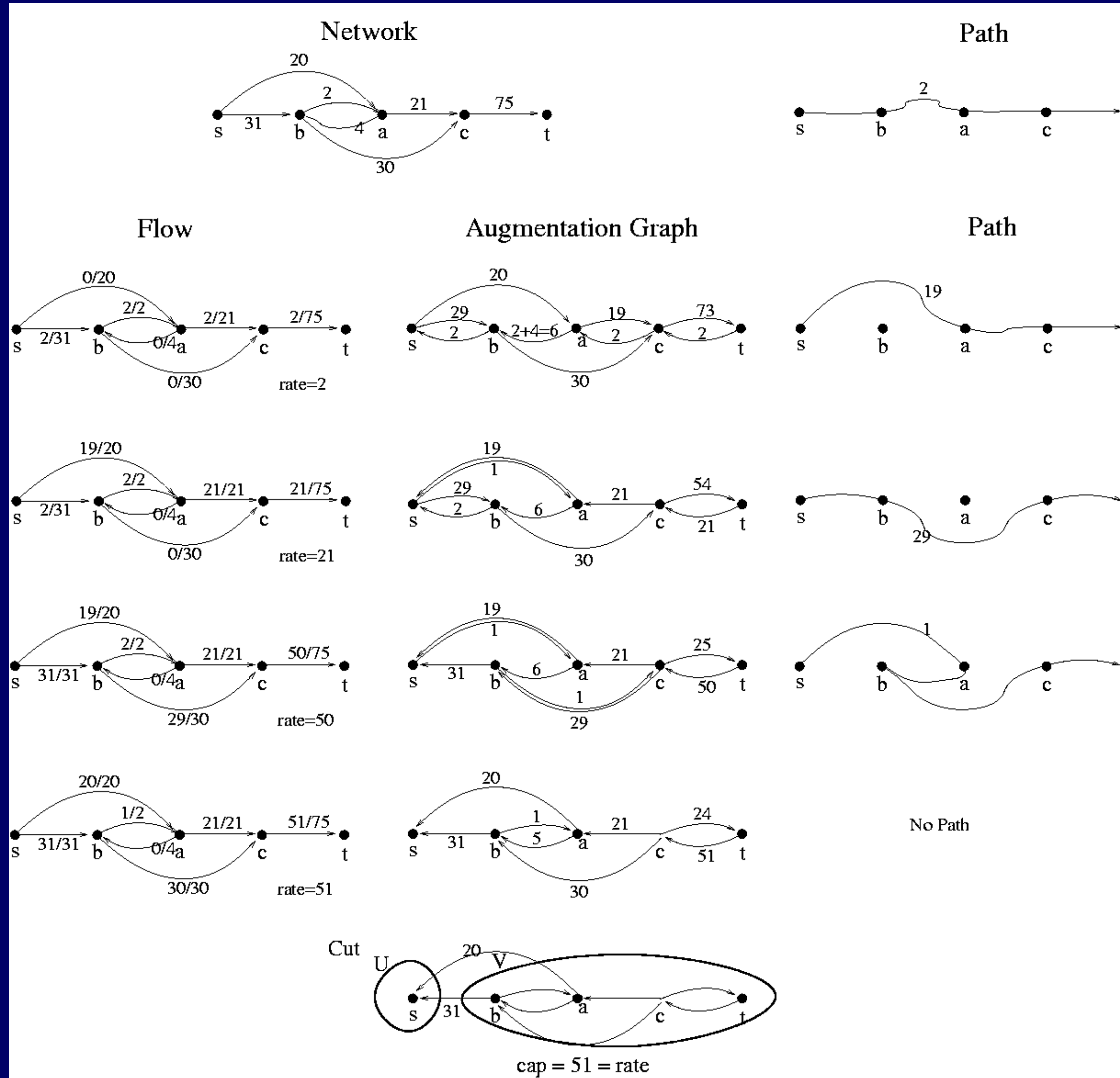


Dynamic Programing

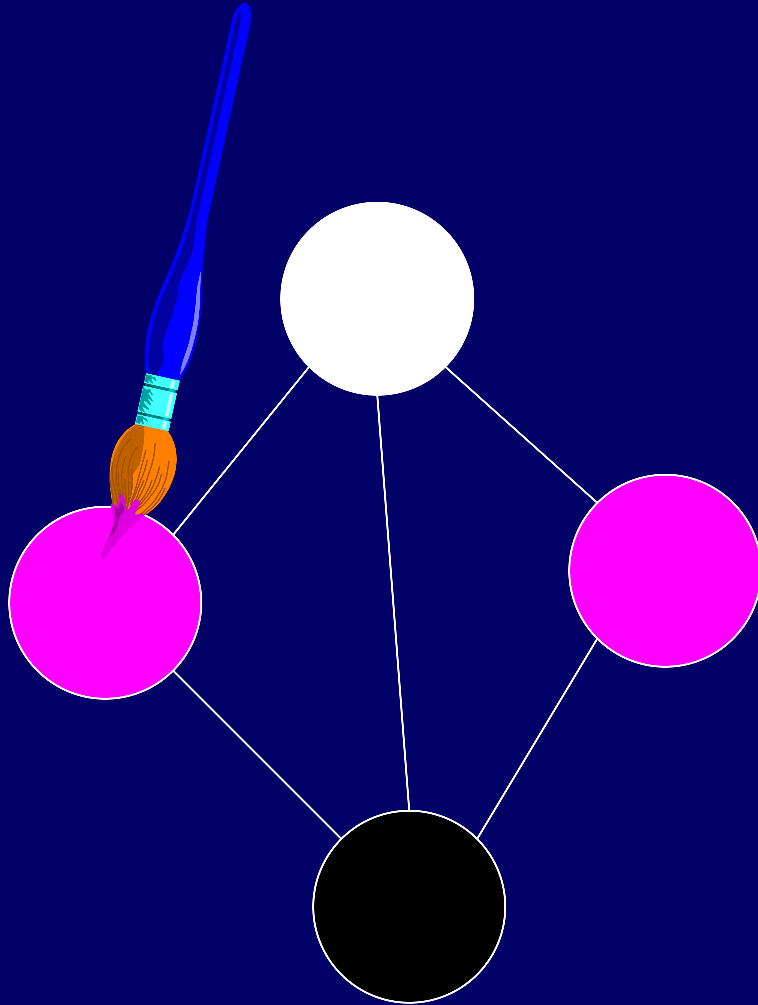


i	x_i \ y_j (j)	0 (j=0)	0 (j=1)	1 (j=2)	0 (j=3)	1 (j=4)	1 (j=5)	0 (j=6)	1 (j=7)	0 (j=8)
0		0	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	1	1	1	1
2	0	0	1	1	2	2	2	2	2	2
3	0	0	1	1	2	2	2	3	3	3
4	1	0	1	2	2	3	3	3	4	4
5	0	0	1	2	3	3	3	4	4	5
6	1	0	1	2	3	4	4	4	5	5
7	0	0	1	2	3	4	4	5	5	6

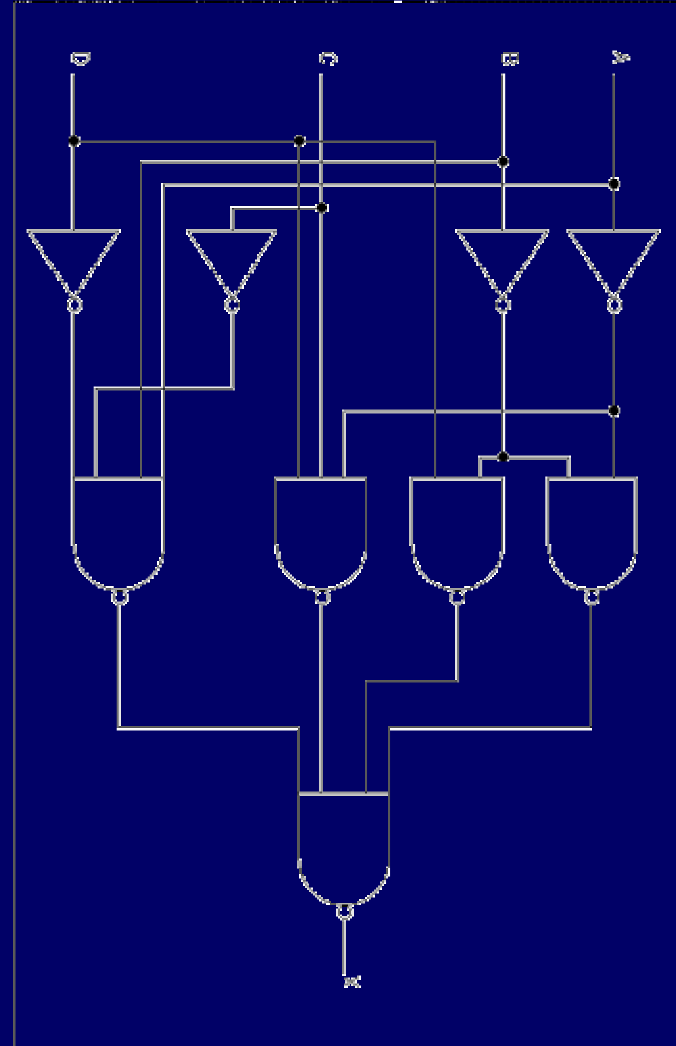
Network Flows



Reduction



=



Rudich www.discretemath.com

Useful Learning Techniques

Read Ahead

You are expected to read the lecture notes **before** the lecture.

This will facilitate more productive discussion during class.

Be Creative

- Ask questions.
- Why is it done this way and not that way?

Guesses and Counter Examples

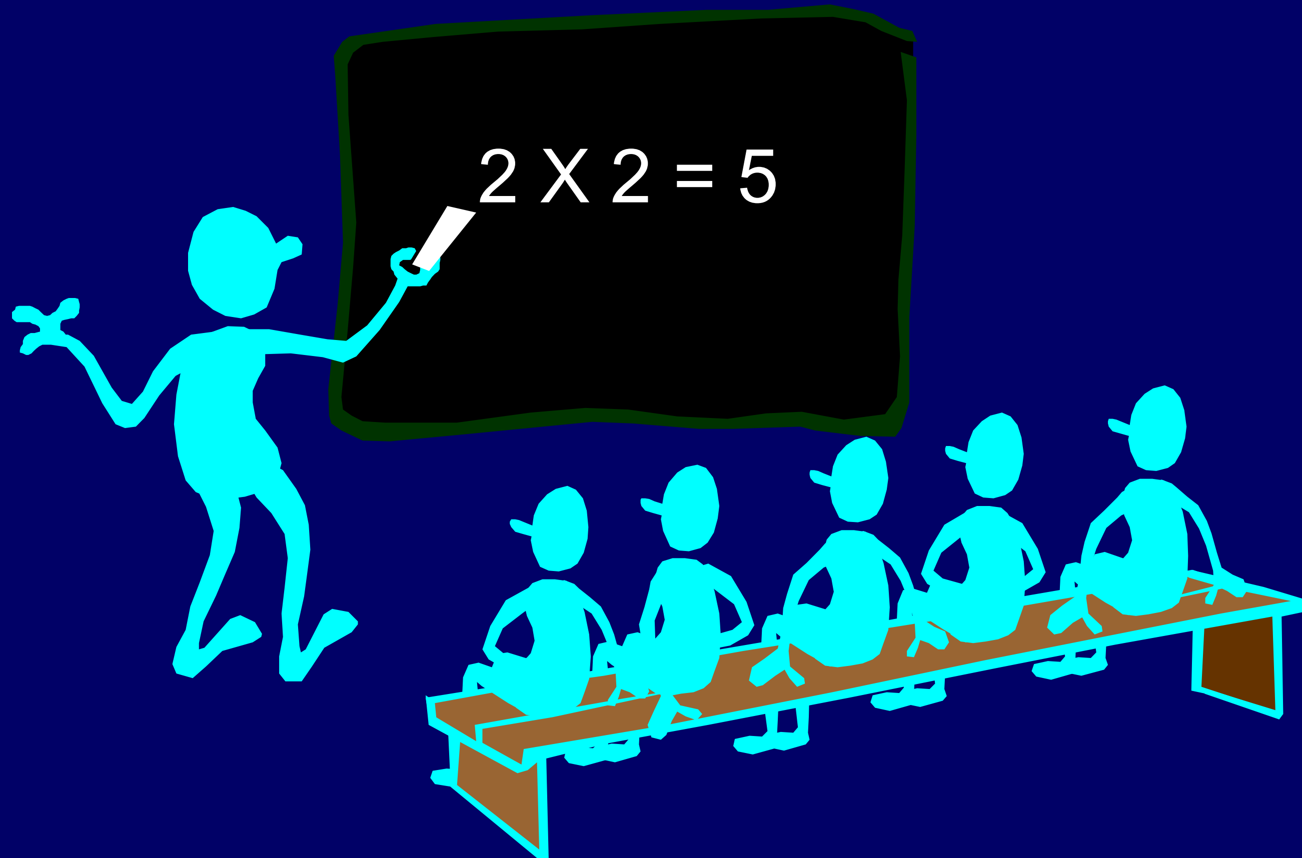
- Guess at potential algorithms for solving a problem.
- Look for input instances for which your algorithm gives the wrong answer.

Refinement:

The best solution comes from a process of repeatedly refining and inventing alternative solutions

A Few Example Algorithms

Grade School Revisited: How To Multiply Two Numbers



Slides in this next section produced by

Steven Rudich

from Carnegie Mellon University

Individual Slides will be marked

Rudich www.discretemath.com

Complex Numbers

- Remember how to multiply 2 complex numbers?
- $(a+bi)(c+di) = [ac - bd] + [ad + bc]\, i$
- Input: a,b,c,d   Output: ac-bd, ad+bc
- If a real multiplication costs $1 and an addition cost 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:

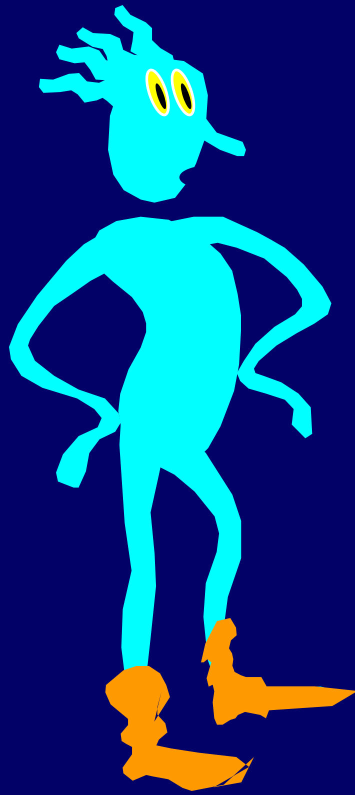
Input: a,b,c,d   Output: ac-bd, ad+bc

- $m_1 = ac$
- $m_2 = bd$
- $A_1 = m_1 - m_2 = ac\text{-}bd$
- $m_3 = (a+b)(c+d) = ac + ad + bc + bd$
- $A_2 = m_3 - m_1 - m_2 = ad+bc$

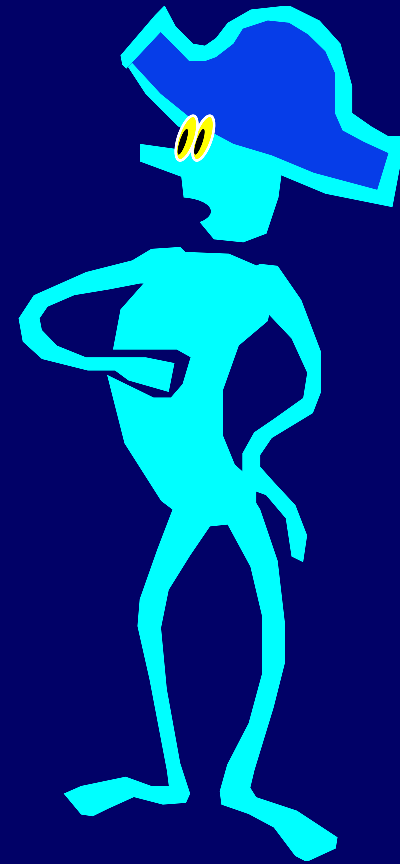
Question:

- The Gauss “hack” 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?

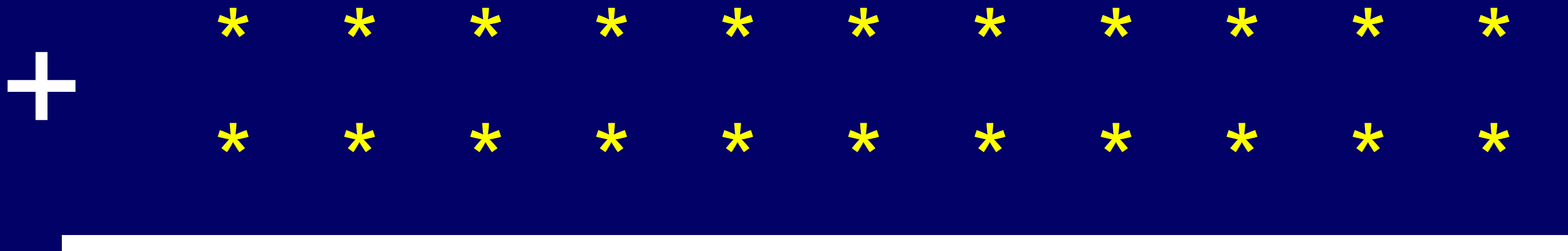


Odette

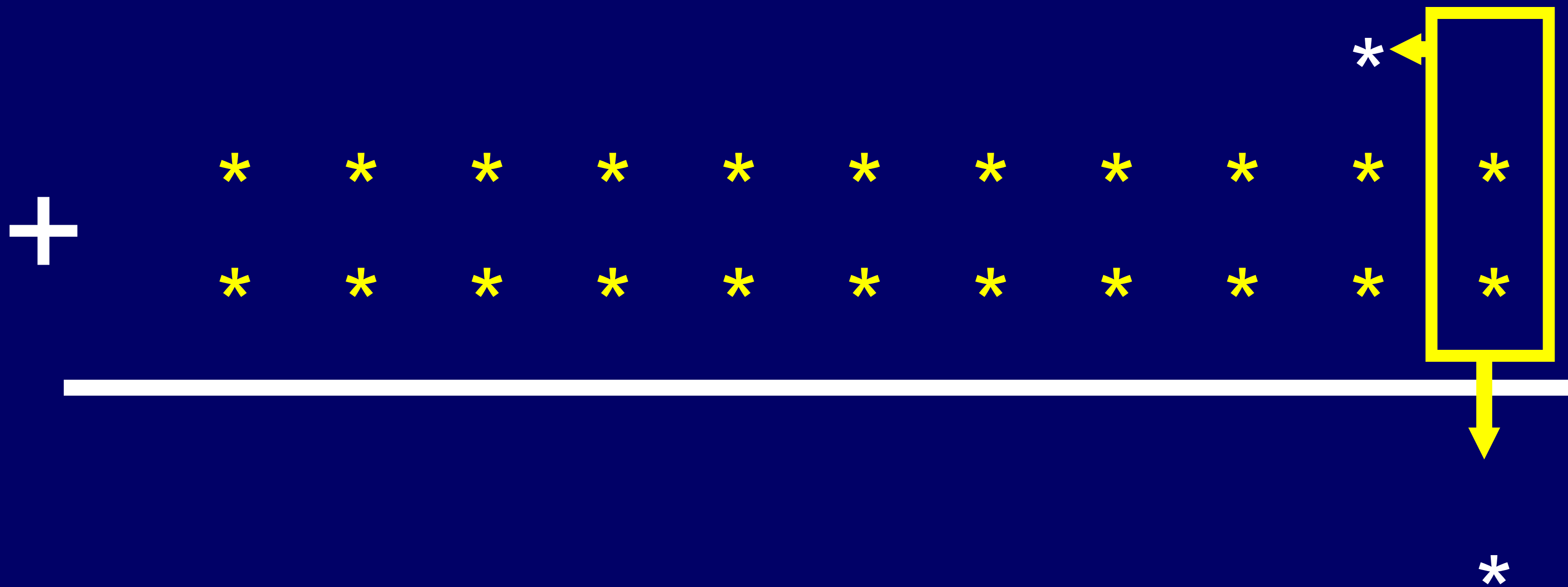


Bonzo

How to add 2 n-bit numbers.



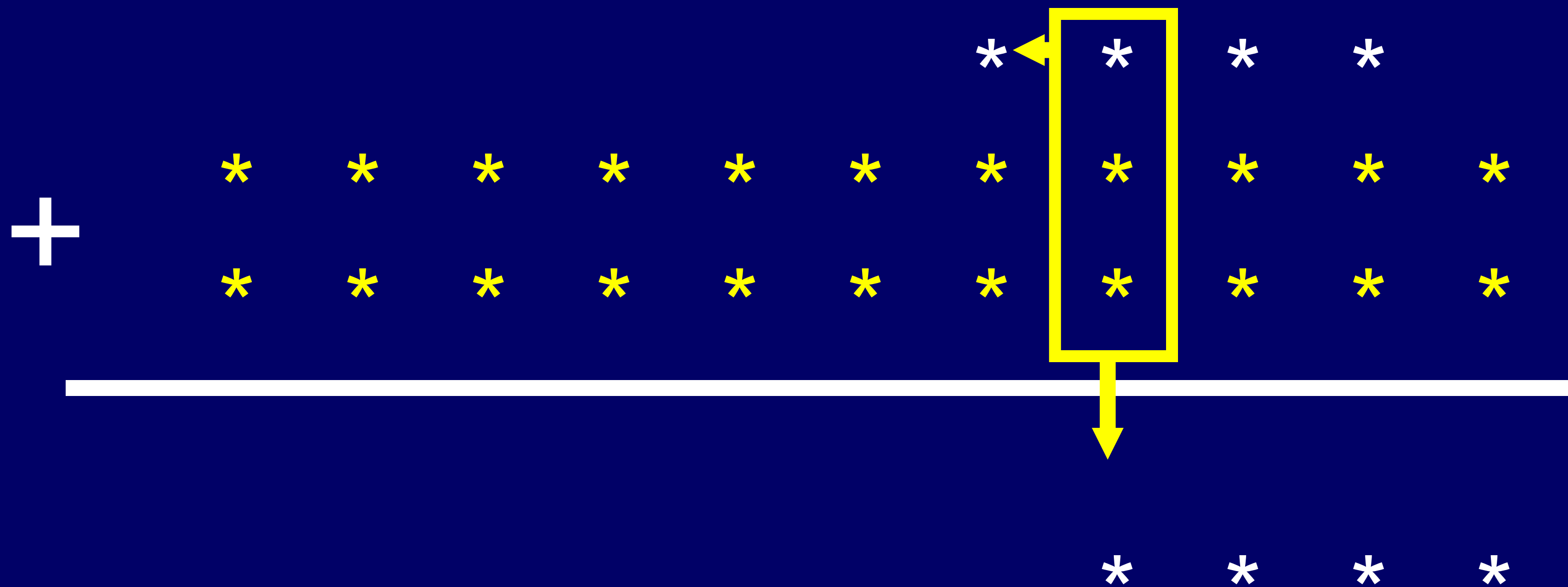
How to add 2 n-bit numbers.



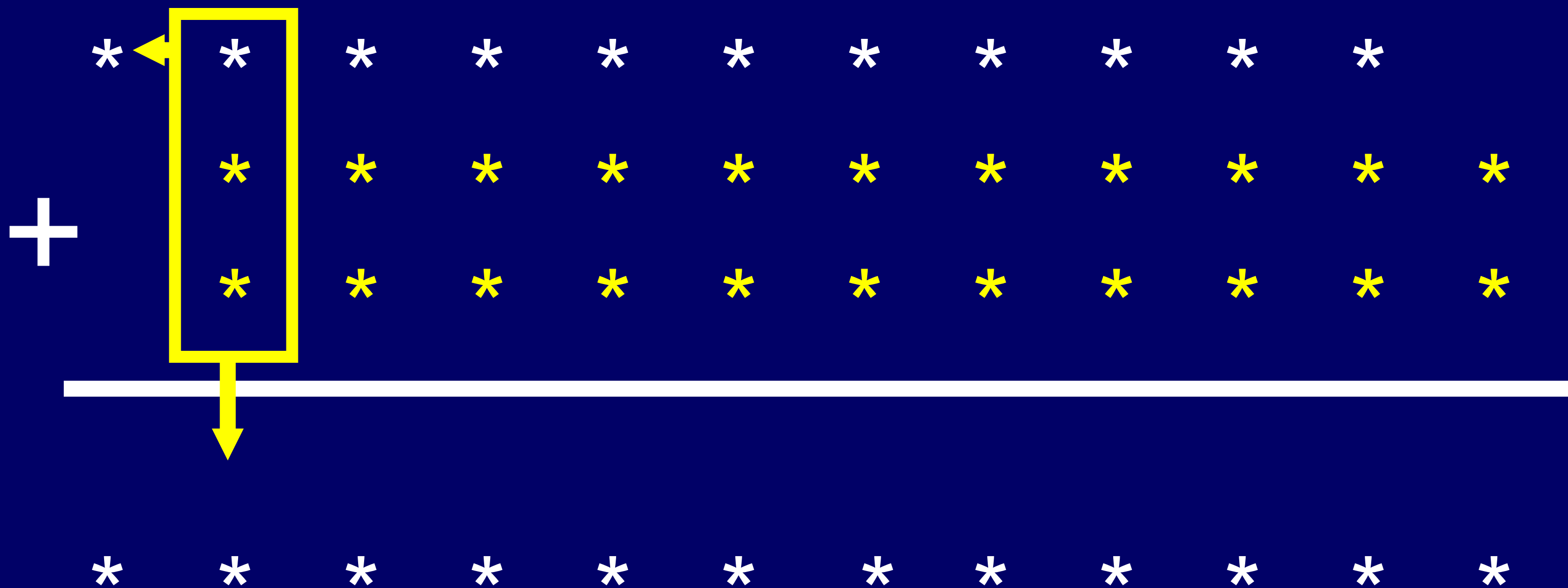
How to add 2 n-bit numbers.

How to add 2 n-bit numbers.

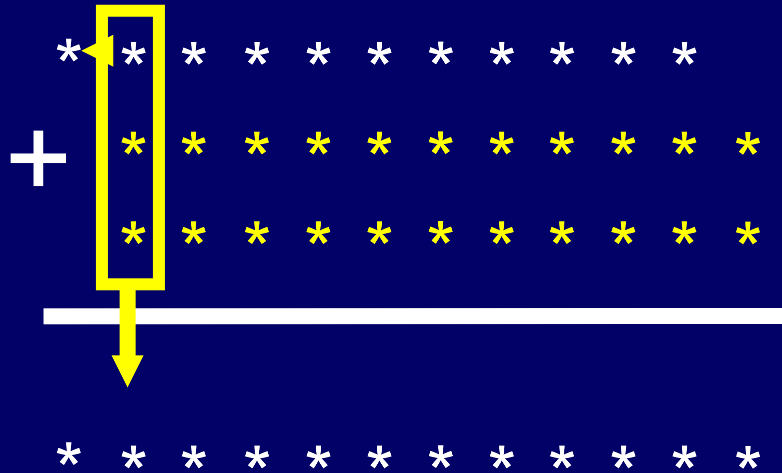
How to add 2 n-bit numbers.



How to add 2 n-bit numbers.



Time complexity of grade school addition

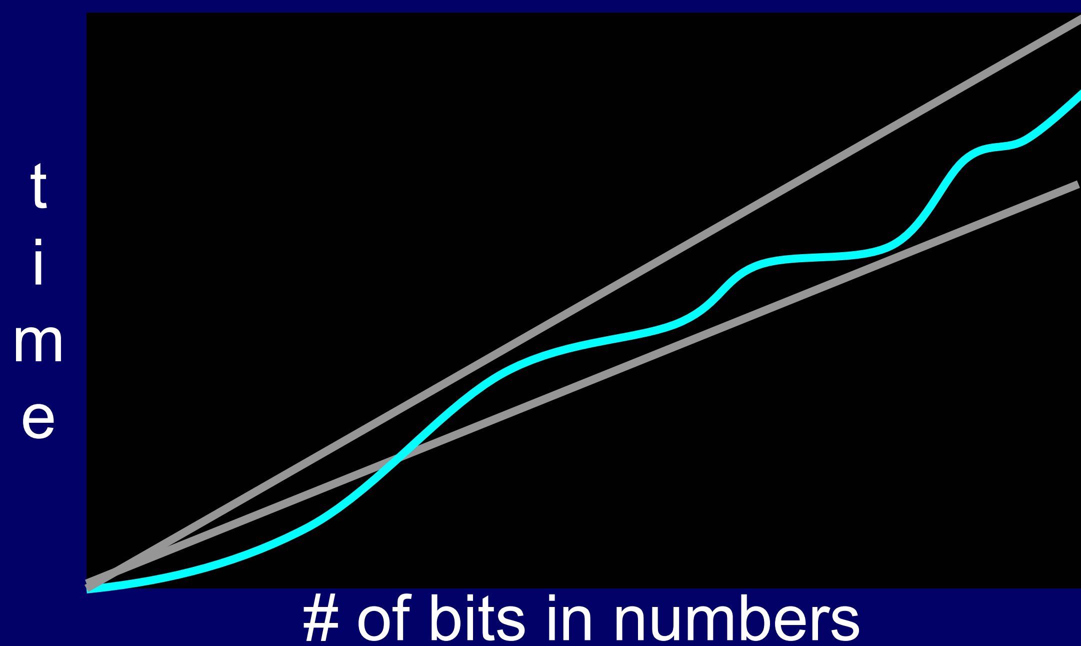


On any reasonable computer adding 3 bits can be done in constant time.

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

= $\theta(n)$ = linear time.

f = θ(n) means that f can be sandwiched between two lines



Please feel free
to ask questions!

Is there a faster way to add?

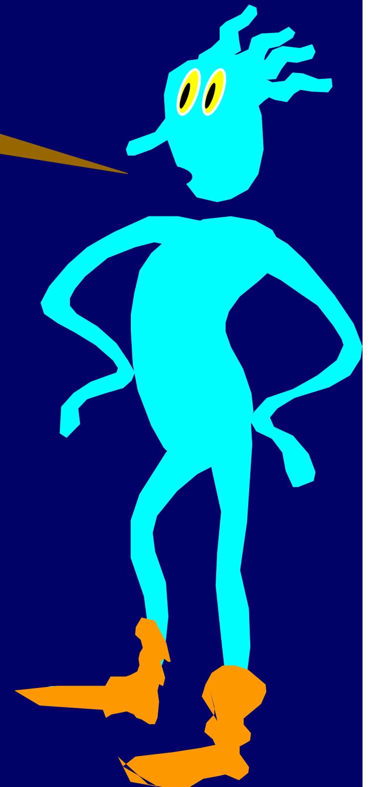
- **QUESTION:** Is there an algorithm to add two n-bit numbers whose time grows sub-linearly in n?

Any algorithm for addition must read all of the input bits

- Suppose there is a mystery algorithm that does not examine each bit
- Give the algorithm a pair of numbers. There must be some unexamined bit position i in one of the numbers
- If the algorithm is not correct on the numbers, we found a bug
- If the algorithm is correct, flip the bit at position i and give the algorithm the new pair of numbers. It give the same answer as before so it must be wrong since the sum has changed

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.

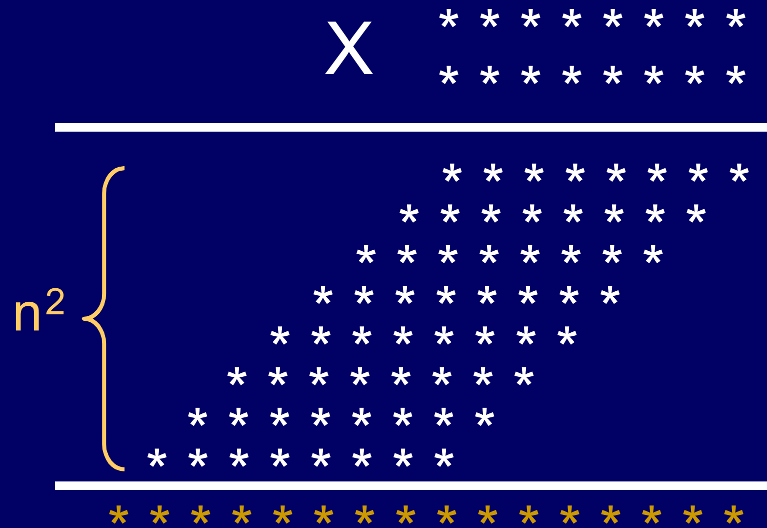


How to multiply 2 n-bit numbers.

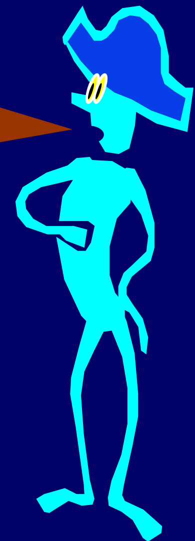
```
               * * * * * * * *
         X     * * * * * * * *
   ---------------------------------
                * * * * * * * *
              * * * * * * * *
            * * * * * * * *
   n^2    * * * * * * * *
        * * * * * * * *
      * * * * * * * *
    * * * * * * * *
  * * * * * * * *
   ---------------------------------
 * * * * * * * * * * * * * * * *
```

n^2

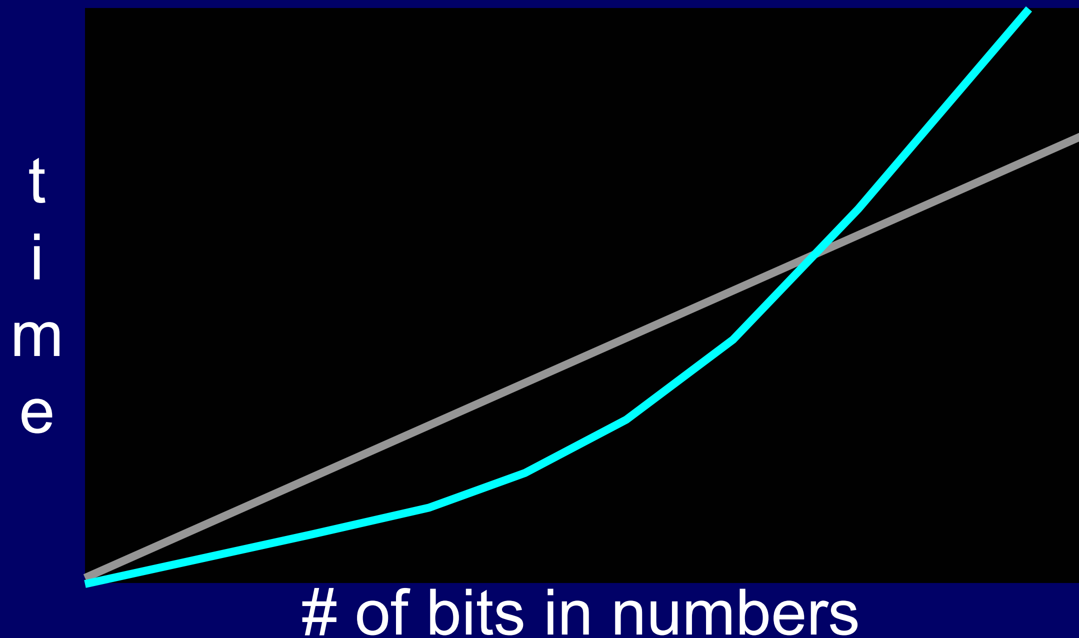
How to multiply 2 n-bit numbers.



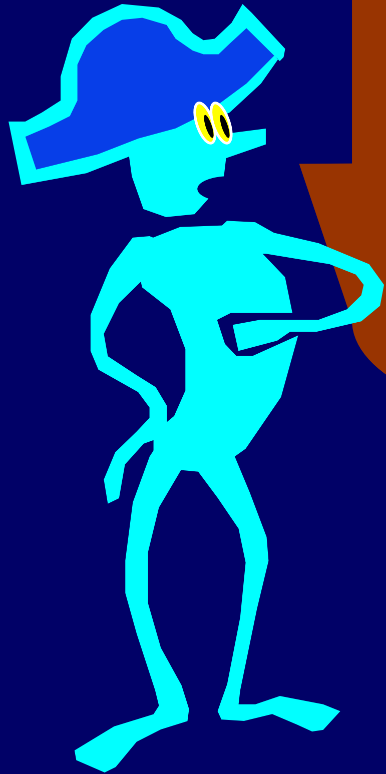
I get it! The total time is bounded by cn^2.



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



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



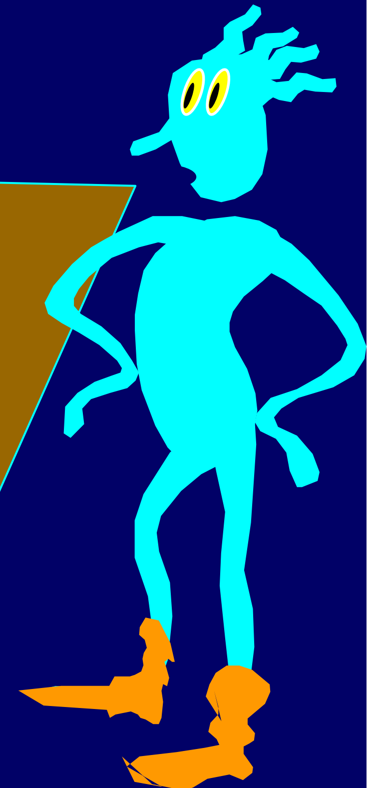
Neat! We have demonstrated that as things scale multiplication is a harder problem than addition.

Mathematical confirmation of our common sense.

Don't jump to conclusions!

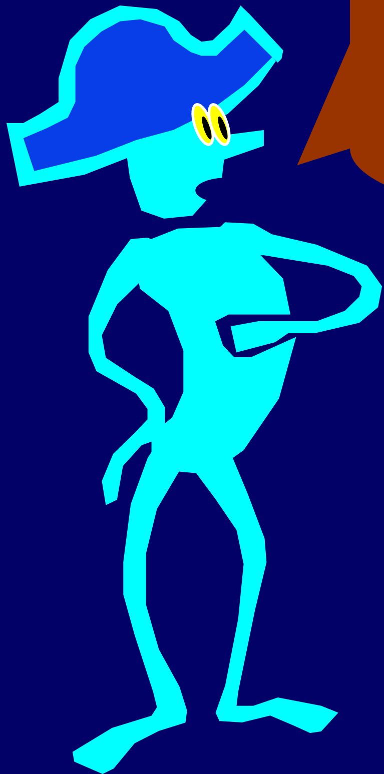
We have argued that grade school multiplication uses more time than grade school addition. This is a comparison of the complexity of two algorithms.

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.



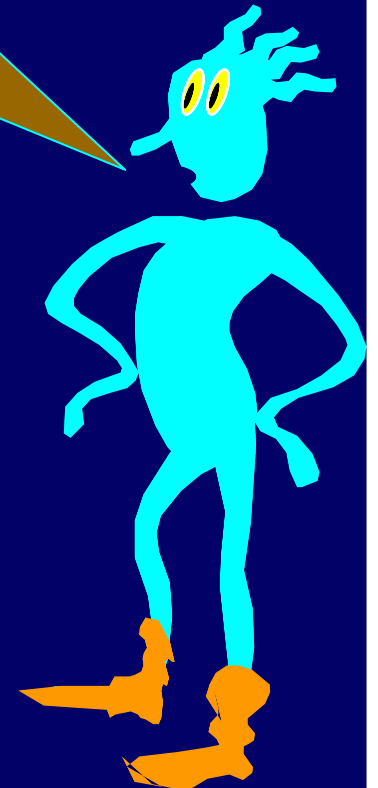
Grade School Addition: $\theta(n)$ time

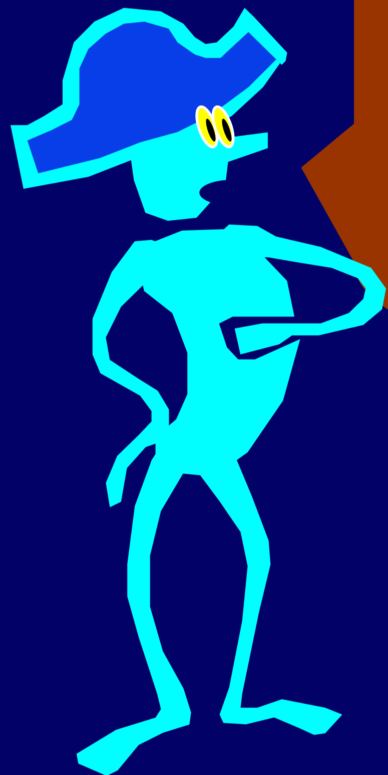
Grade School Multiplication: $\theta(n^2)$ time



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

Despite years of research, no one knows! If you resolve this question, PUC 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** 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 | b |
- Y = | c | d |

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

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

Multiplication of 2 n-bit numbers

- X = | a | b |
- Y = | c | d |
- XY = $ac\ 2^n + (ad+bc)\ 2^{n/2} + bd$

MULT(X,Y):

If |X| = |Y| = 1 then RETURN XY

Break X into a;b and Y into c;d

RETURN

MULT(a,c) 2^n + (MULT(a,d) + MULT(b,c)) $2^{n/2}$ + MULT(b,d)

Time required by MULT

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

- What is $T(n)$?
  What is its growth rate?
  Is it $\theta(n^2)$?

Recurrence Relation

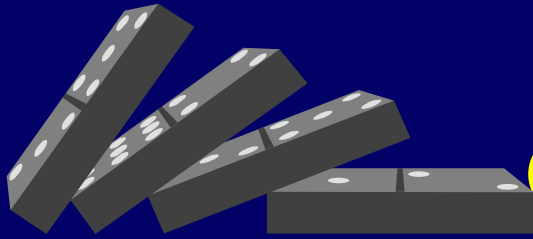
- $T(1) = k$ for some constant k
- $T(n) = 4\,T(n/2) + k'\,n + k''$ for some constants k' and k''

```
MULT(X,Y):
 If |X| = |Y| = 1 then RETURN XY
 Break X into a;b and Y into c;d
 RETURN
   MULT(a,c) 2^n + (MULT(a,d) + MULT(b,c)) 2^(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:**

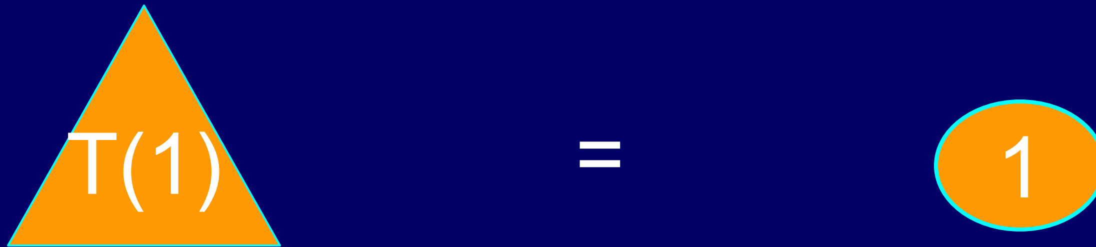
$$T(1) = 1 \;\&\; T(n) = 4T(n/2) + n$$

- **Guess:** $G(n) = 2n^2 - n$
- **Verify:**

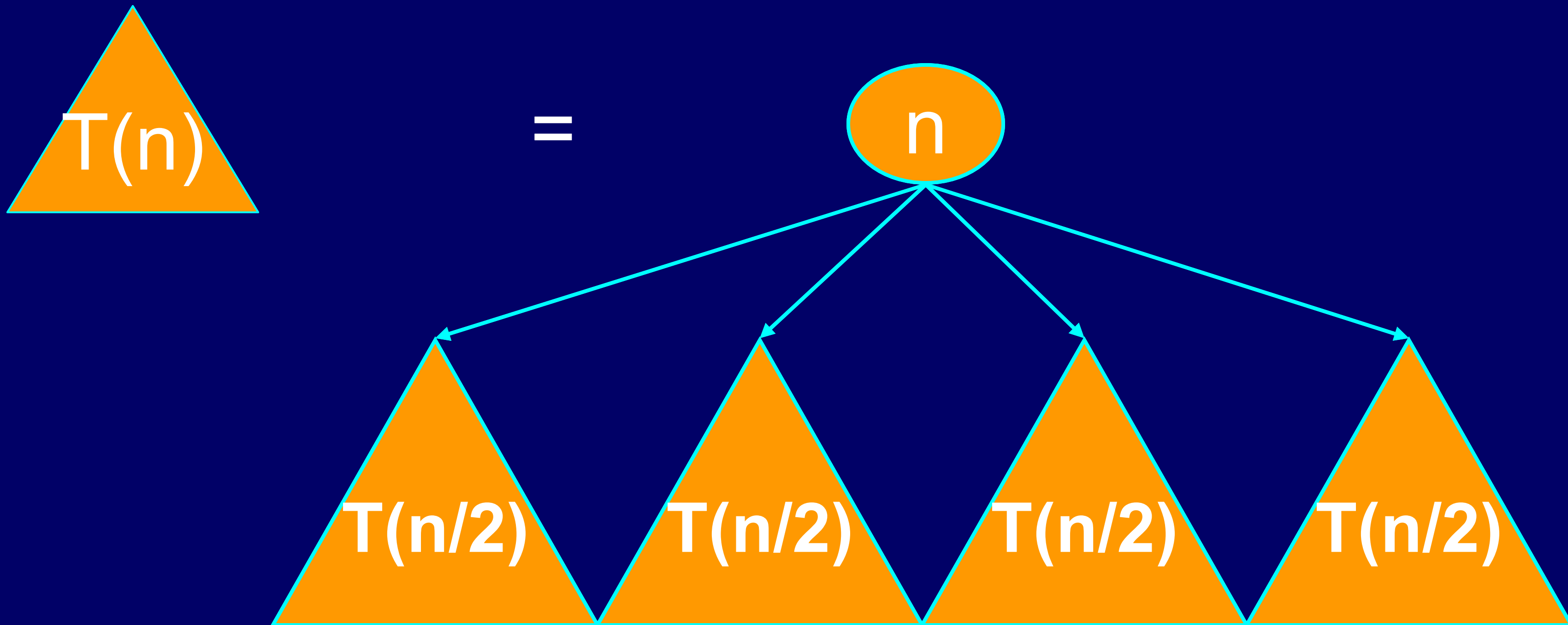
Left Hand Side	Right Hand Side
$\mathbf{T(1)} = 2(1)^2 - 1$	$\mathbf{1}$
$\mathbf{T(n)}$	$\mathbf{4T(n/2) + n}$
$= 2n^2 - n$	$= 4\,[2(^n/_2)^2 - (^n/_2)] + n$
	$= 2n^2 - n$

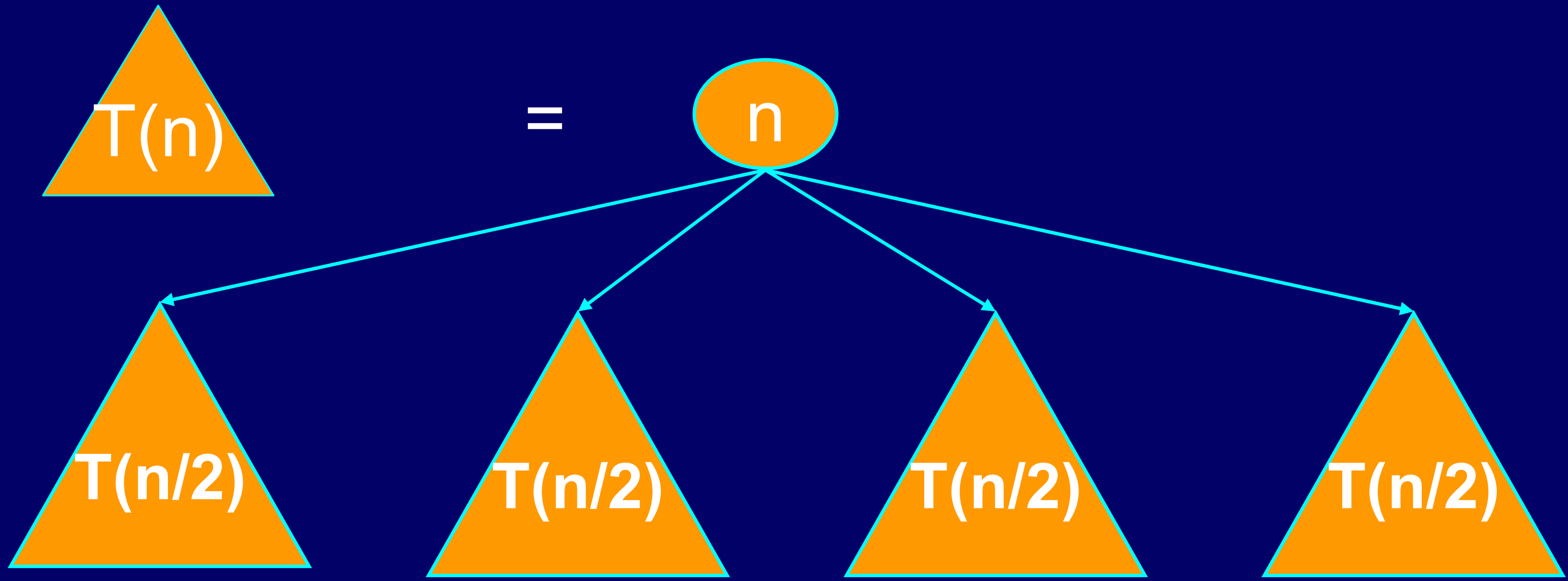
Technique 2: Decorate The Tree

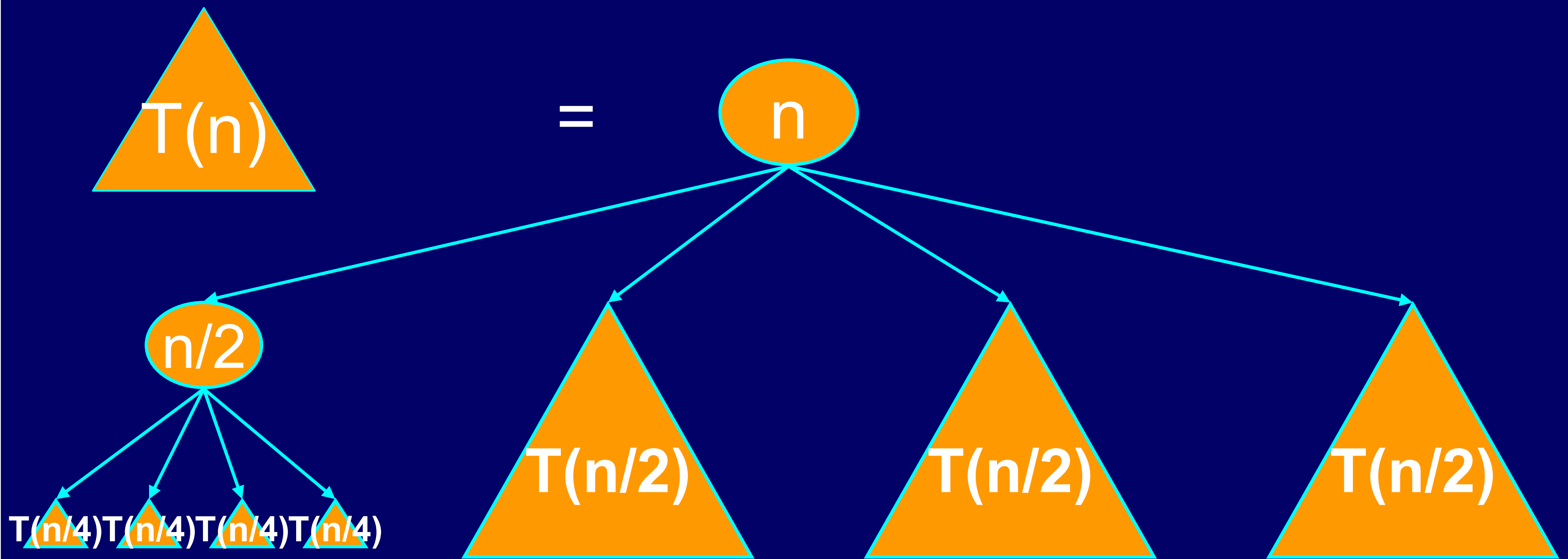
- $T(1) = 1$

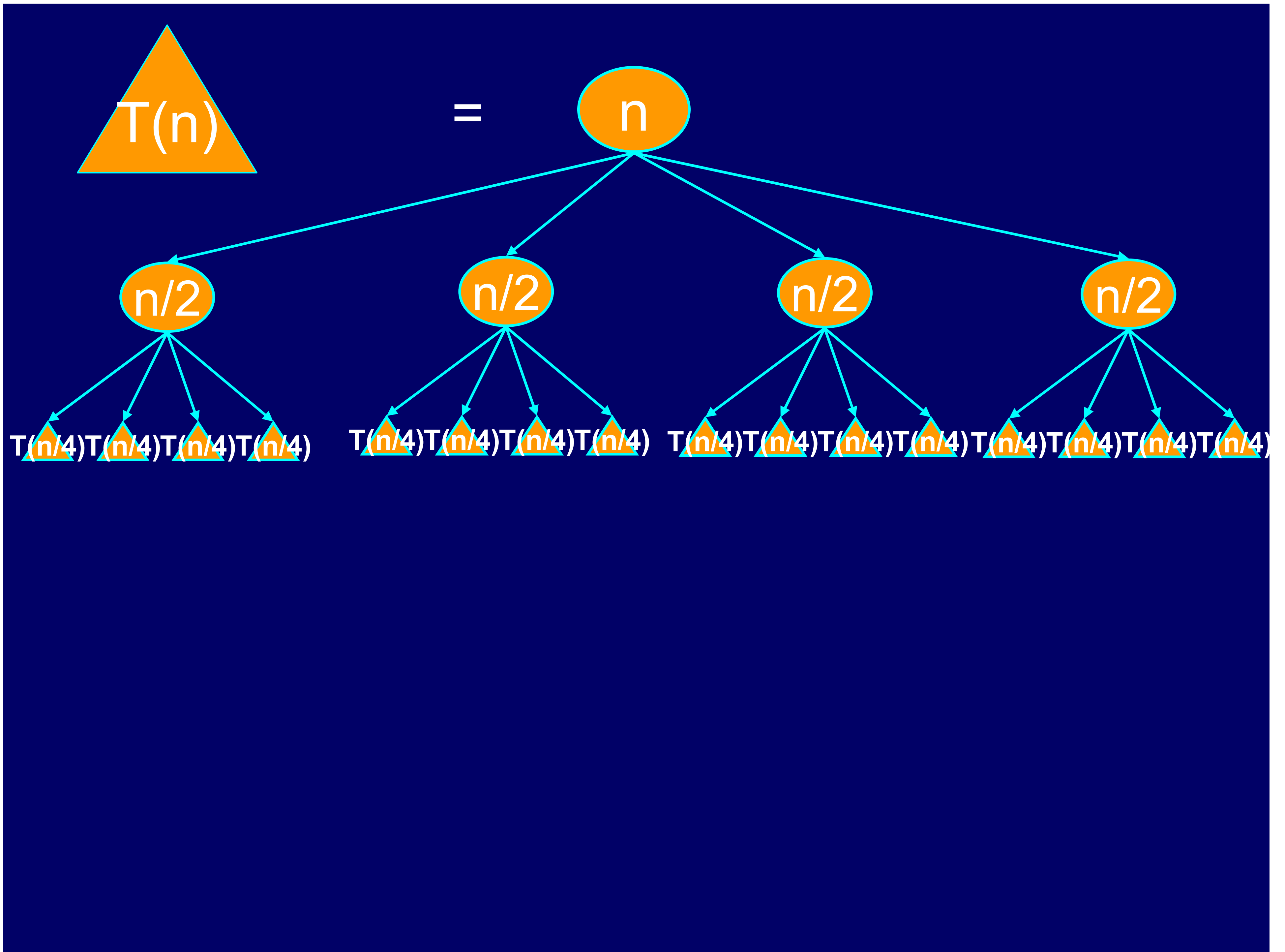


- $T(n) = n + 4\,T(n/2)$




T(n)
=
n
T(n/2)
T(n/2)
T(n/2)
T(n/2)


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


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

T(n) = n

n/2 n/2 n/2 n/2

n/4 n/4 n/4 n/4 n/4 n/4 n/4 n/4 n/4 n/4 n/4 n/4 n/4 n/4 n/4 n/4

11111111111111111111111111111111 111111111111111111111111111111111

0	n
1	n/2 + n/2 + n/2 + n/2
2	n/4 + n/4 + n/4 + n/4 + n/4 + n/4 + n/4 + n/4 + n/4 + n/4 + n/4 + n/4 + n/4 + n/4 + n/4 + n/4
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+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1

Level		
0	n	$=1 \cdot n$
1	$n/2 + n/2 + n/2 + n/2$	$= 4 \cdot n/2$
2	$n/4 + n/4 + n/4 + n/4 + n/4 + n/4 + n/4 + n/4 + n/4 + n/4 + n/4 + n/4 + n/4 + n/4 + n/4 + n/4$	$= 16 \cdot n/4$
i	Level i is the sum of 4^i copies of $n/2^i$	$= 4^i \cdot n/2^i$
	. . . .	
$\log_2 n$	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+1+1+1+1+1	$= 4^{\log n} \cdot n/2^{\log n}$

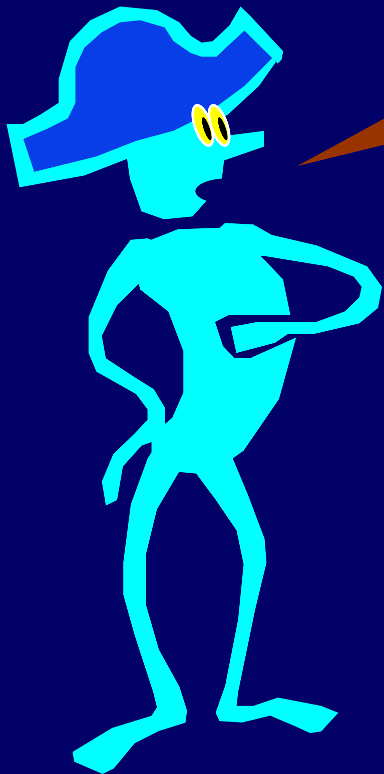
$=n^{\log 4} \cdot 1$

Total: $\theta(n^{\log 4}) = \theta(n^2)$

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

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

All that work for nothing!



MULT revisited

MULT(X,Y):

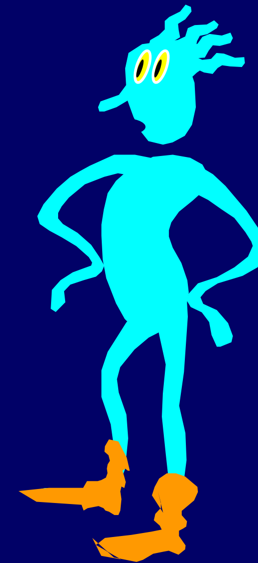
If |X| = |Y| = 1 then RETURN XY

Break X into a;b and Y into c;d

RETURN

$$\text{MULT}(a,c)\, 2^n + (\text{MULT}(a,d) + \text{MULT}(b,c))\, 2^{n/2} + \text{MULT}(b,d)$$

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



Gauss' Hack:

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

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

Gaussified MULT (Karatsuba 1962)

MULT(X,Y):

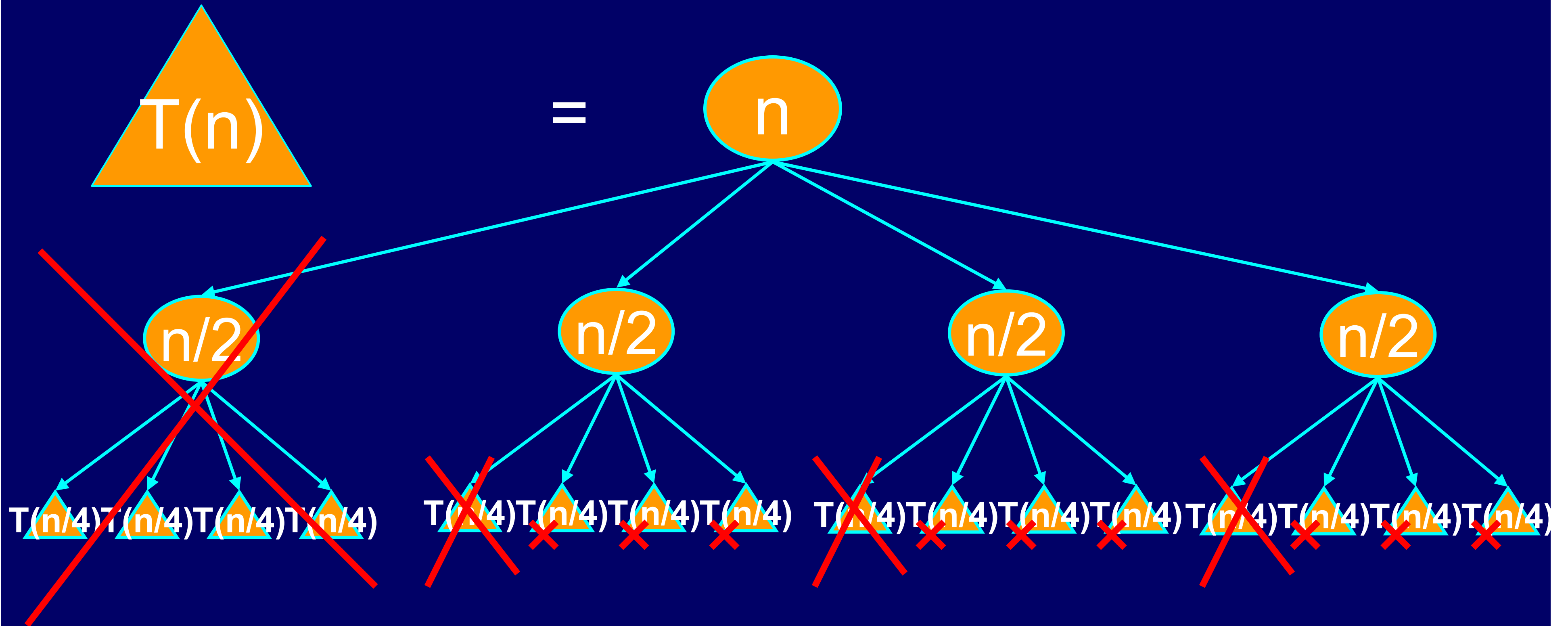
If $|X| = |Y| = 1$ then RETURN XY

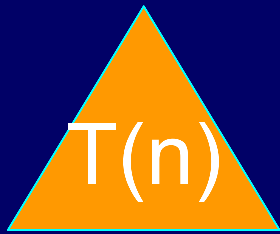
Break X into a;b and Y into c;d

e = MULT(a,c) and f =MULT(b,d)

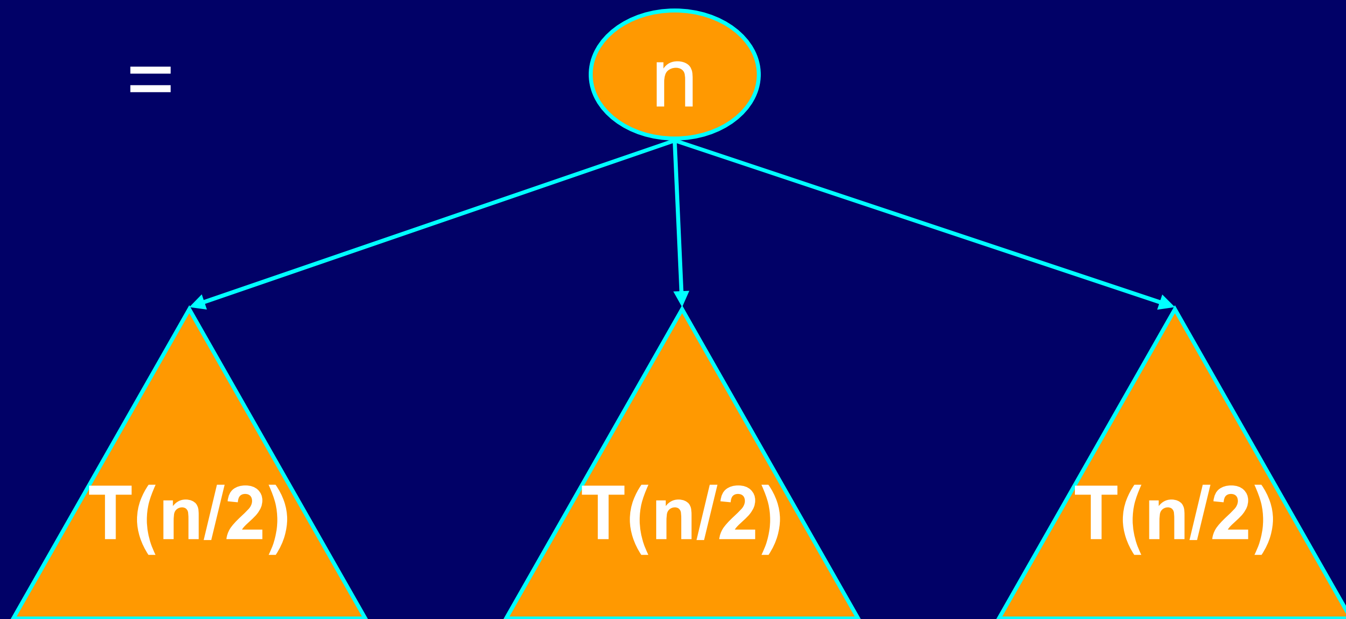
RETURN $e2^n$ + (MULT(a+b, c+d) – e - f) $2^{n/2}$ + f

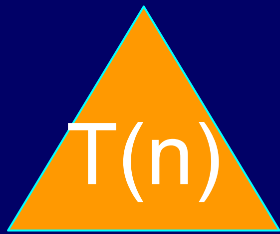
- $T(n) = 3\ T(n/2) + n$
- *Actually:* $T(n) = 2\ T(n/2) + T(n/2 + 1) + kn$


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

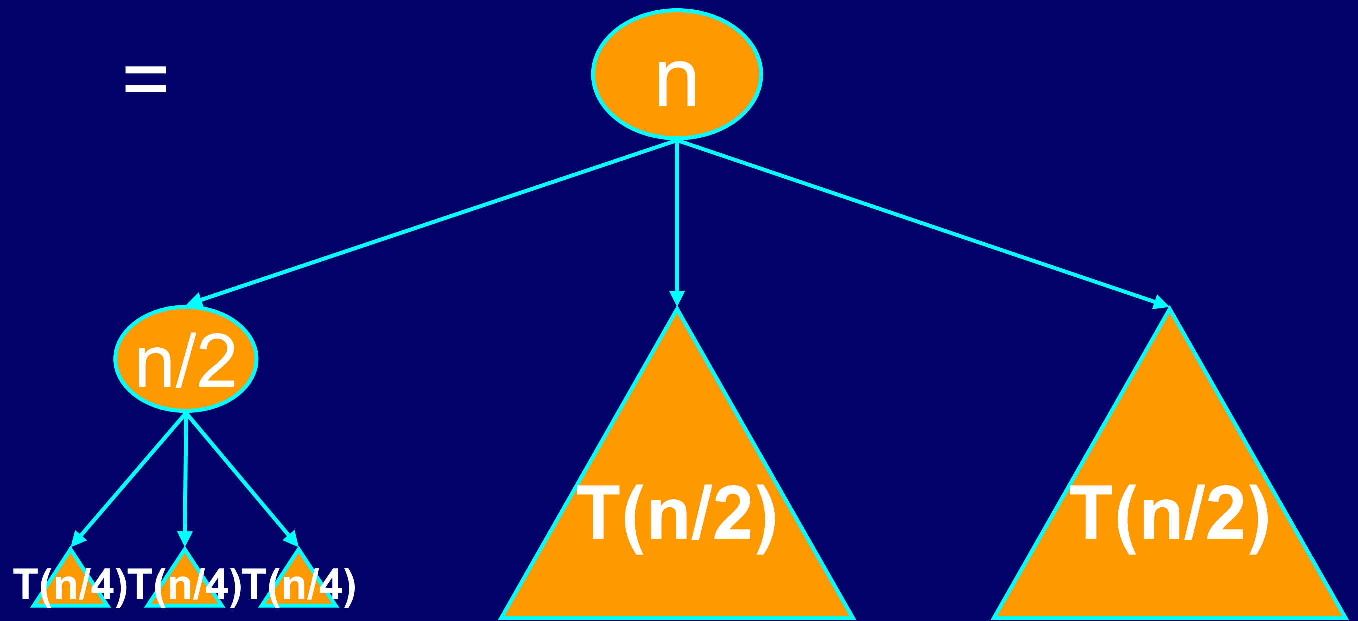

T(n)

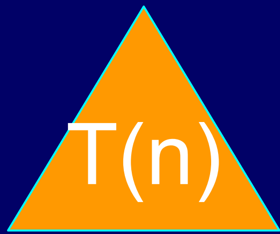
=


n
T(n/2)
T(n/2)
T(n/2)

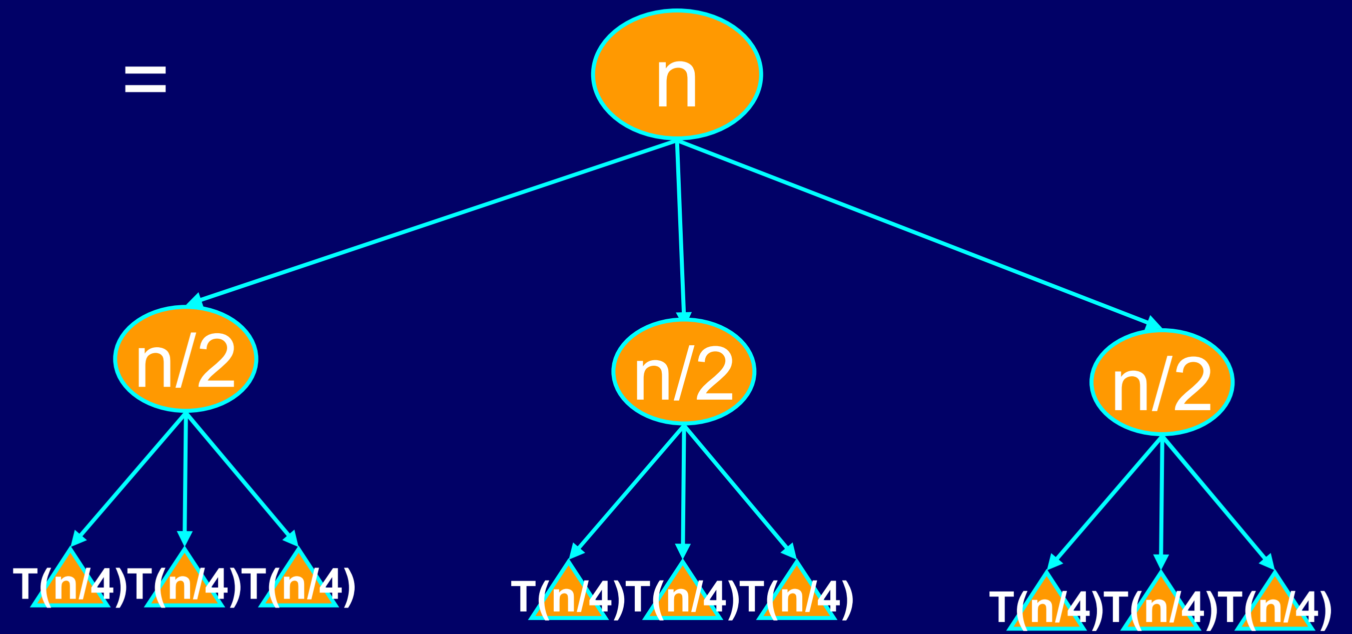

T(n)

=


n
n/2
T(n/2)
T(n/2)
T(n/4)T(n/4)T(n/4)


T(n)

=


n
n/2
n/2
n/2
T(n/4)T(n/4)T(n/4)
T(n/4)T(n/4)T(n/4)
T(n/4)T(n/4)T(n/4)

0 n

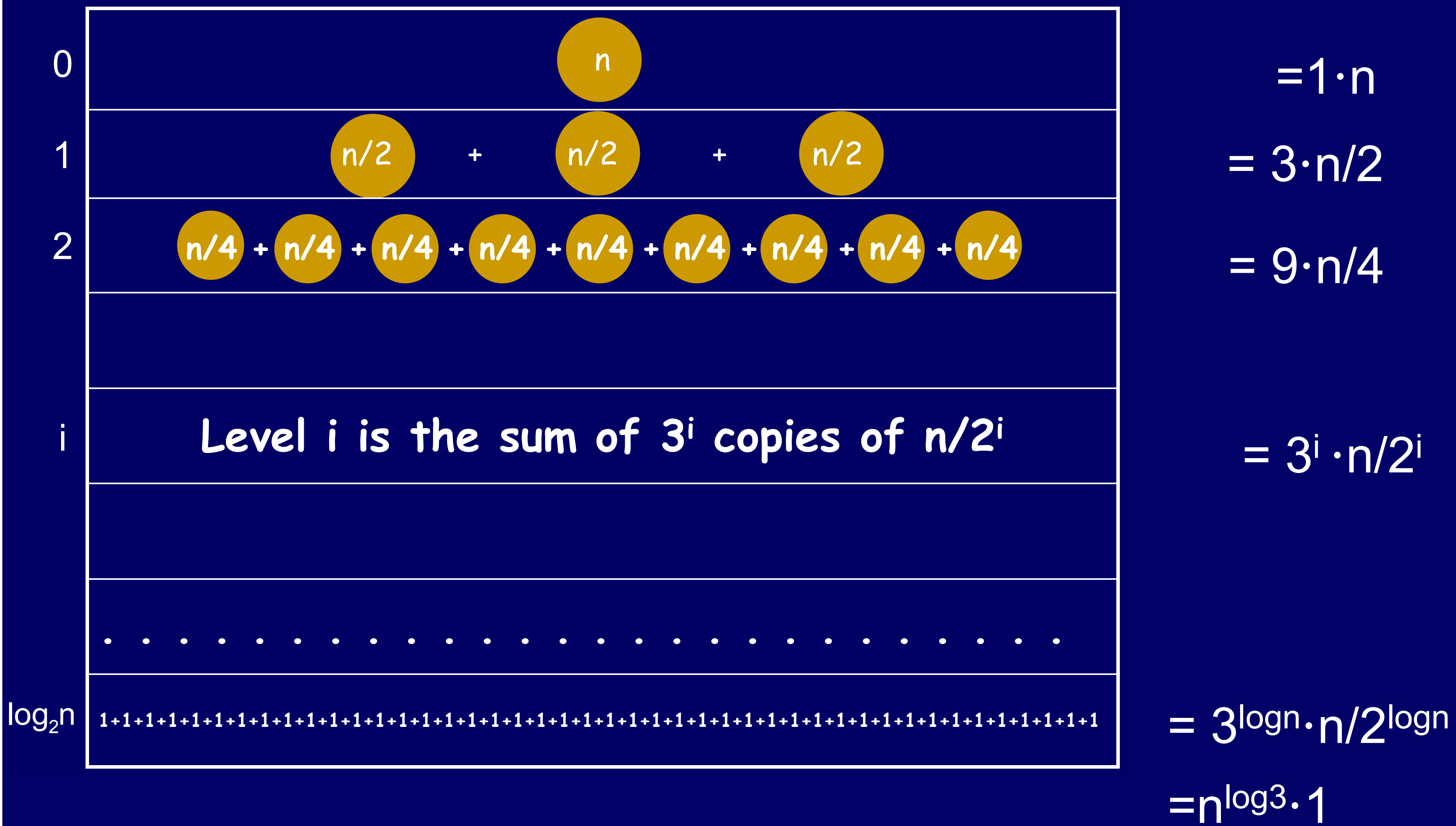
1 n/2 + n/2 + n/2

2 n/4 + n/4 + n/4 + n/4 + n/4 + n/4 + n/4 + n/4 + n/4

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+1+1+1+1+1



Total: $\theta(n^{\log 3}) = \theta(n^{1.58..})$

Dramatic improvement for large n

Not just a 25% savings!

$\theta(n^2)$ vs $\theta(n^{1.58..})$

Multiplication Algorithms

Grade School	n^2
Karatsuba	$n^{1.58\ldots}$
Fastest Known	n logn loglogn

Not interested, Bonzo.
I took the initiative and asked out a guy in my Databases class.

End

Informações

Livro Texto: Algorithms Design
Eva Tardos and Jon Kleinberg

www-di/inf.puc-rio.br/~laber