Recursive Algorithms

Introduction

Applications to Numeric Computation

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' Method:

Johann Carl Friedrich Gauss
(* 30. April 1777 in Braunschweig
† 23. Februar 1855 in Göttingen)

- Input: a,b,c,d   Output: ac-bd, ad+bc
- $m_1 = ac$
- $m_2 = bd$
- $A_1 = m_1 - m_2 = ac\text{-}bd$
- $m_3 = (a+b)(c+d) = \cancel{ac} + \boxed{ad + bc} + \cancel{bd}$
- $A_2 = m_3 - m_1 - m_2 = ad+bc$

Total Cost?

\$3.05!

Question

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

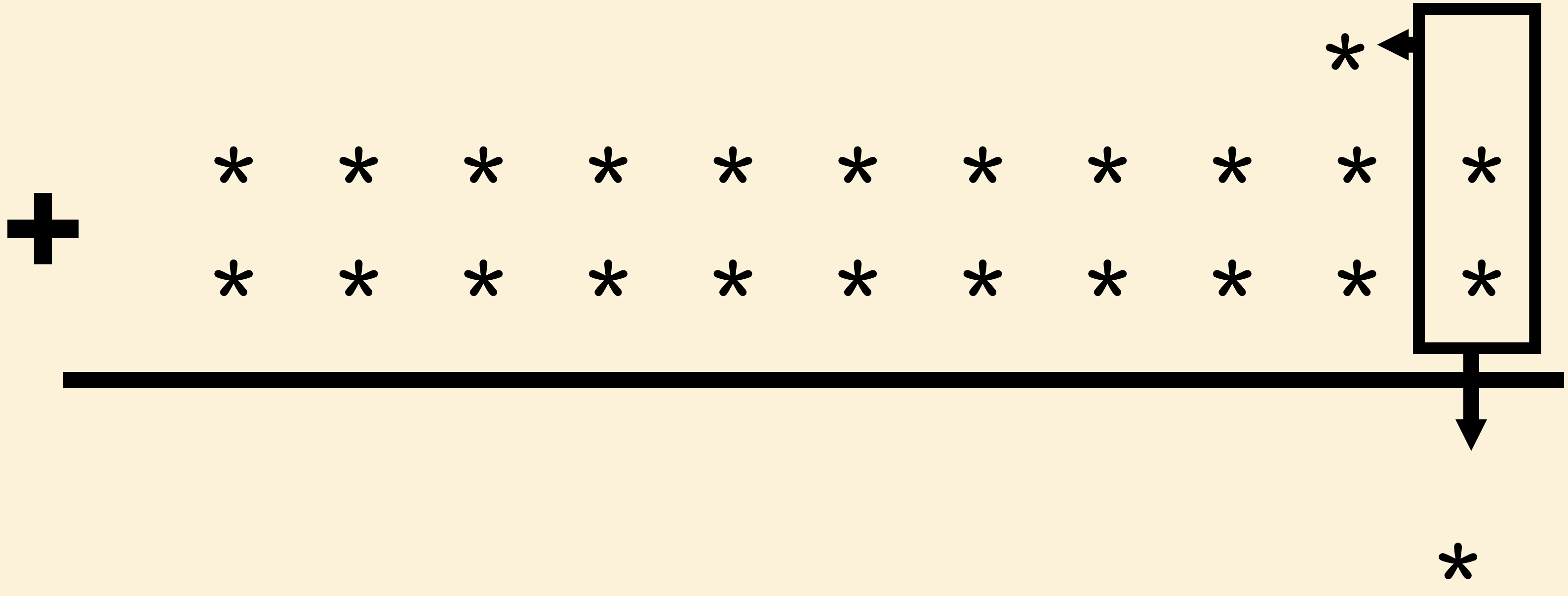
- Could there be a context where performing 3 multiplications for every 4 provides a more dramatic savings?

- Let’s back up a bit.

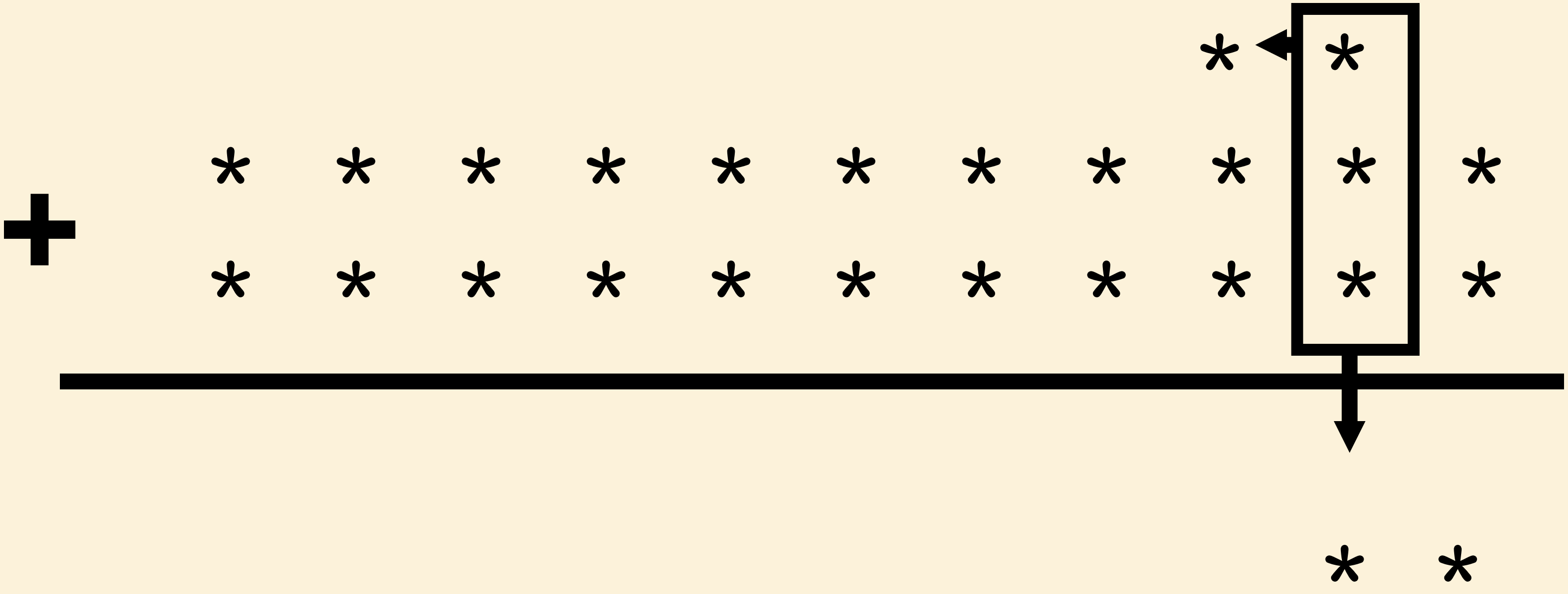
How to add 2 n-bit numbers.

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

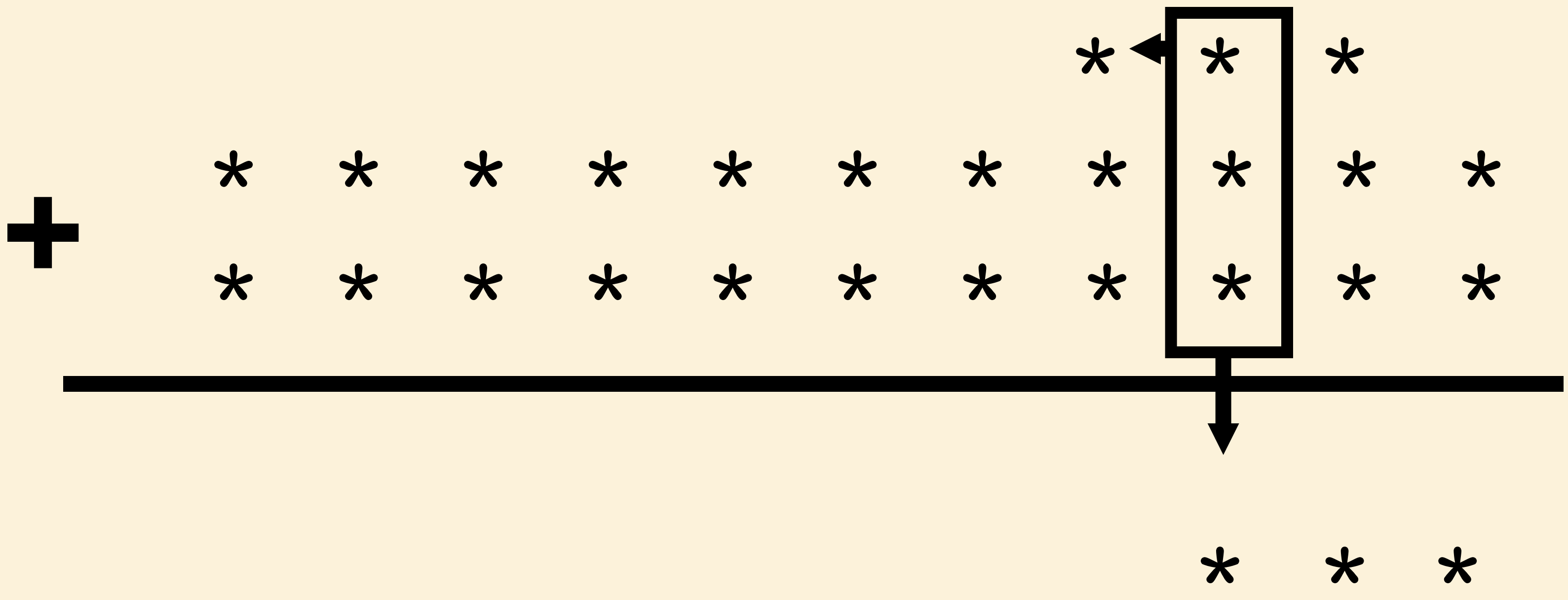
How to add 2 n-bit numbers.



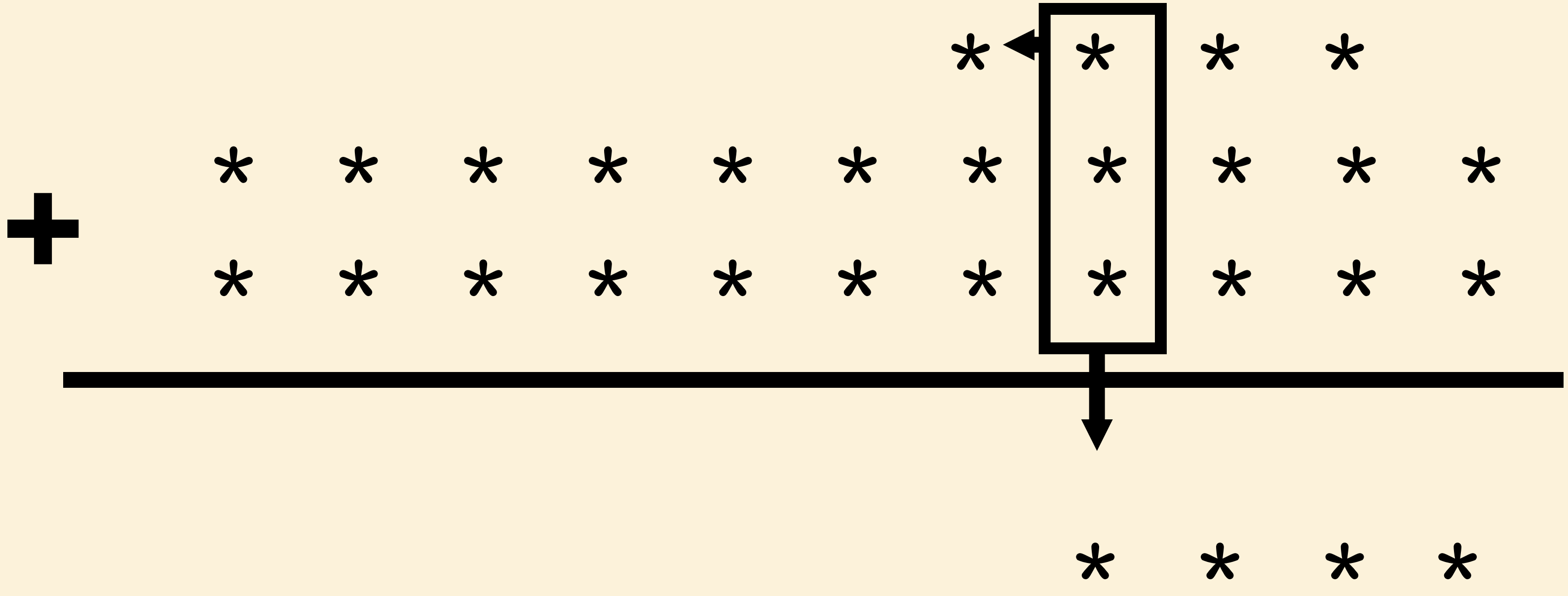
How to add 2 n-bit numbers.



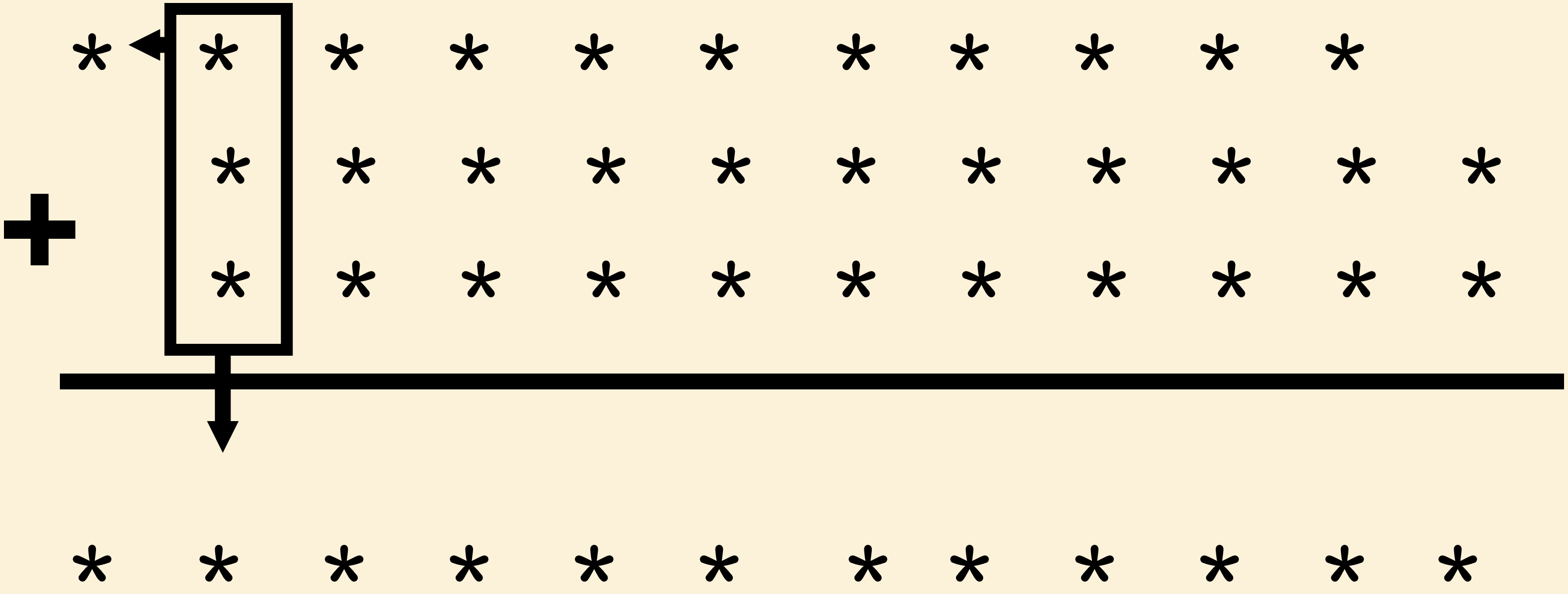
How to add 2 n-bit numbers.



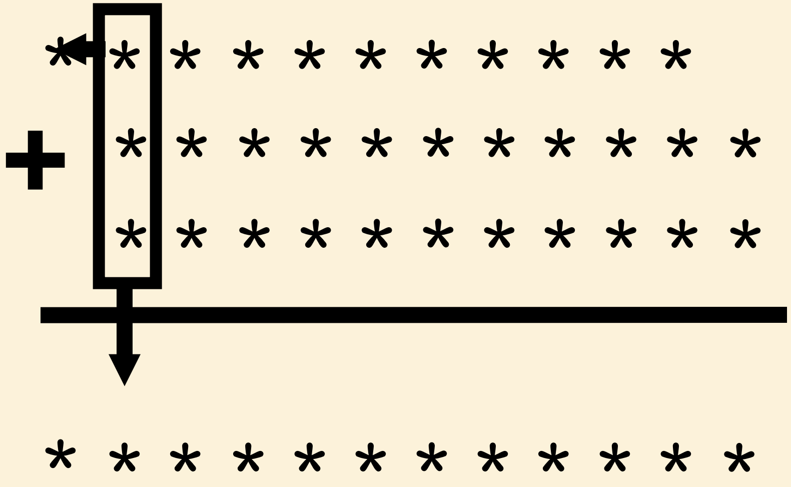
How to add 2 n-bit numbers.



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.

$$\rightarrow T(n) \in \mathrm{O}(n)$$

Is there a faster way to add?

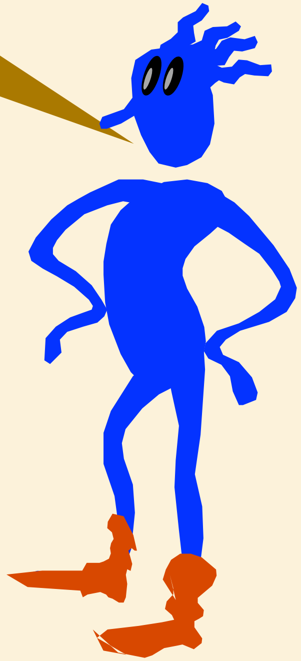
- **QUESTION:** Is there an algorithm to add two n-bit numbers whose time grows sub-linearly in n?

Any algorithm for addition must read all of the input bits

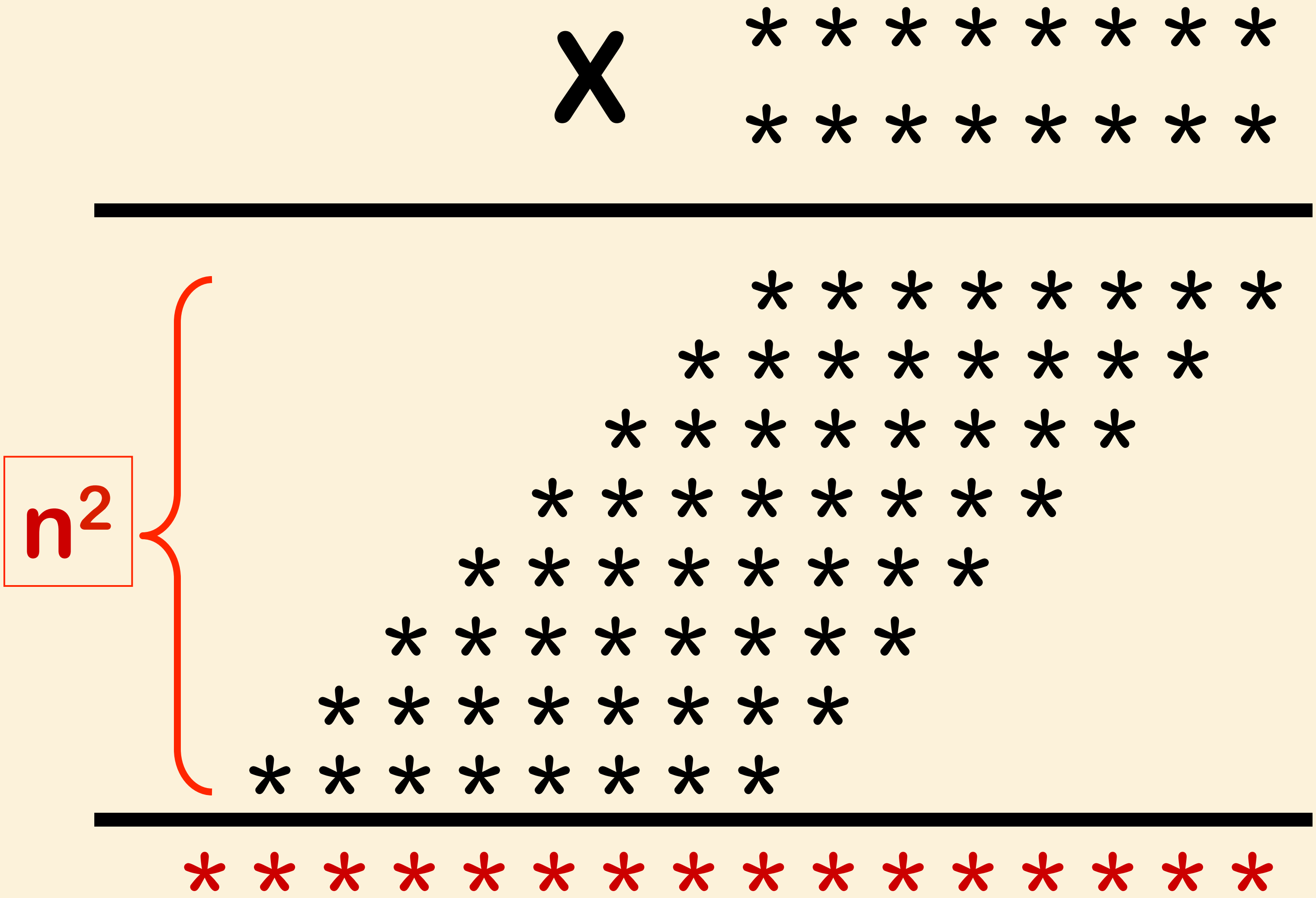
- Suppose there is a mystery algorithm that does not examine each bit
- Give the algorithm a pair of numbers. There must be some unexamined bit position *i* in one of the numbers
- If the algorithm returns the wrong answer, we have found a bug
- If the algorithm is correct, flip the bit at position i and give the algorithm this new input.
- The algorithm must return the same answer, which now is wrong.

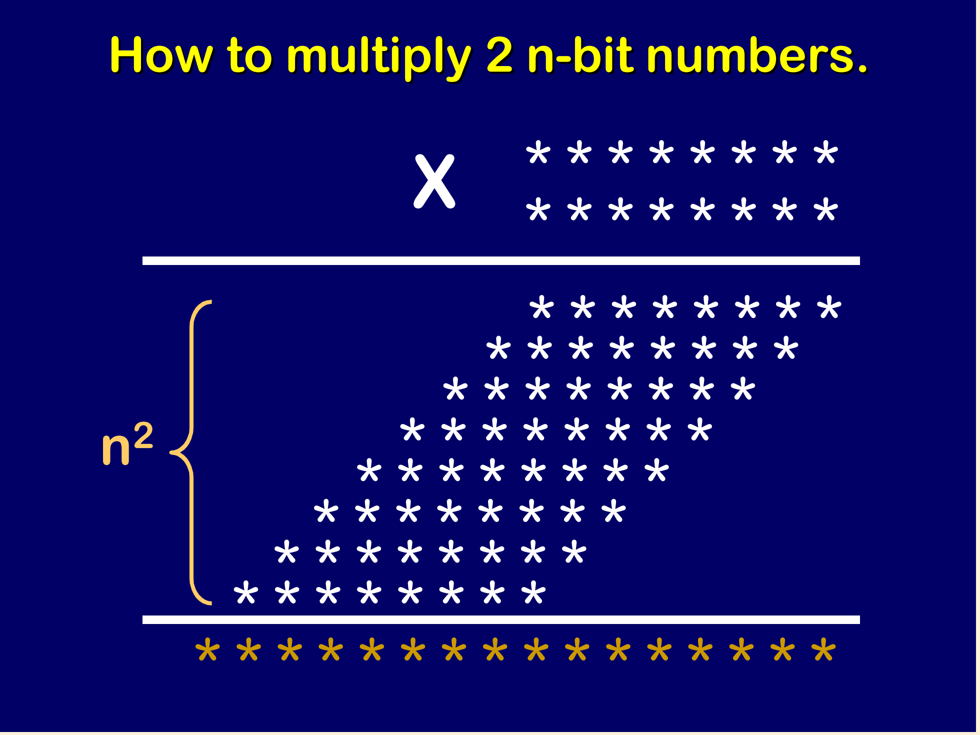
So any algorithm for addition must use time at least linear in the size of the numbers.

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



How to multiply 2 n-bit numbers.

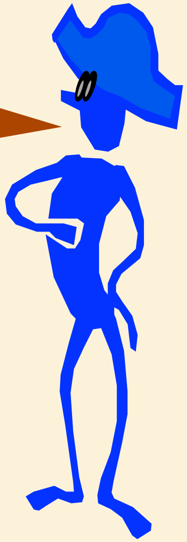




How to multiply 2 n-bit numbers.

```
                      * * * * * * * *
                X     * * * * * * * *
      _________________________________
                      * * * * * * * *
                    * * * * * * * *
                  * * * * * * * *
     n²         * * * * * * * *
              * * * * * * * *
            * * * * * * * *
          * * * * * * * *
        * * * * * * * *
      _________________________________
      * * * * * * * * * * * * * * * *
```

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



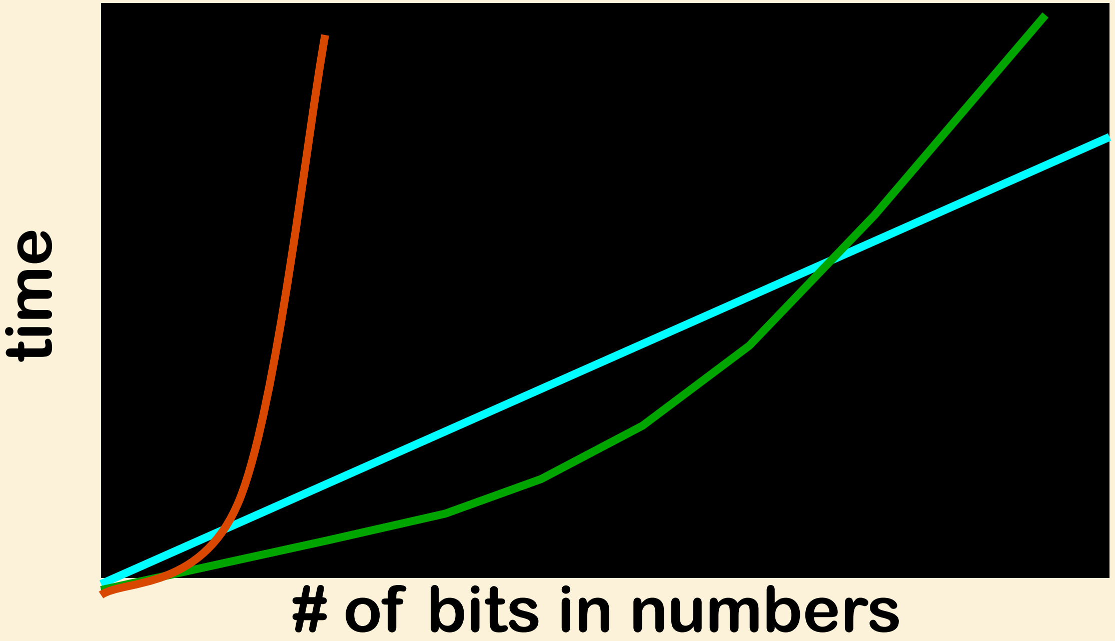
How to multiply 2 n-bit numbers: Kindergarten Algorithm

$$a \times b = \underbrace{a + a + a + \ldots + a}_{b}$$

$T(n) = \theta(bn) = \Theta(2^n n)$!

Fast?

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



End of Lecture 6

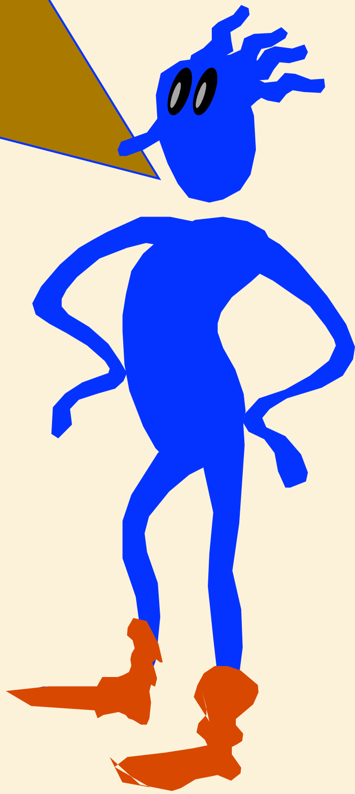
Neat! We have demonstrated that multiplication is a harder problem than addition.

Mathematical confirmation of our common sense.

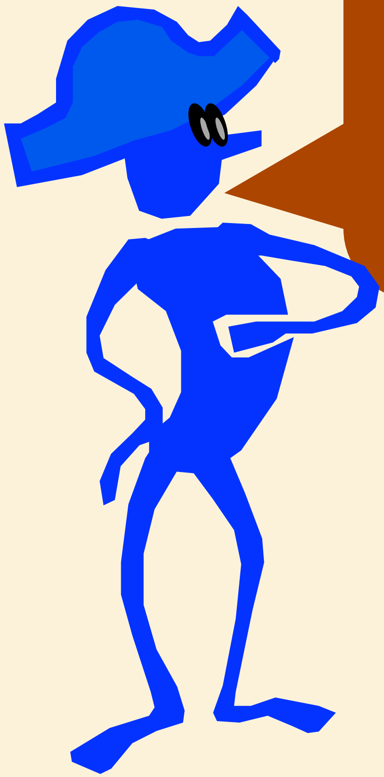
Don’t jump to conclusions!

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.

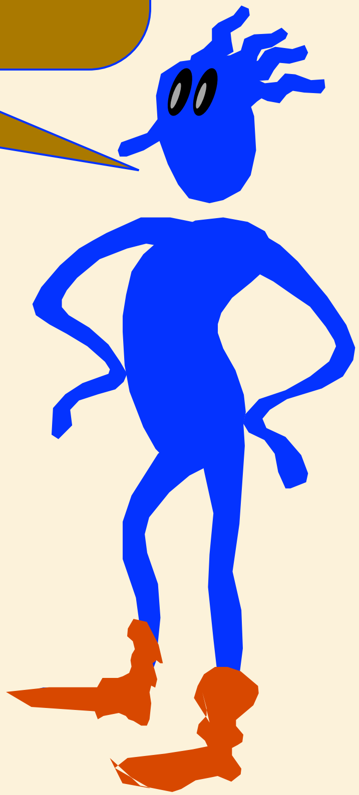


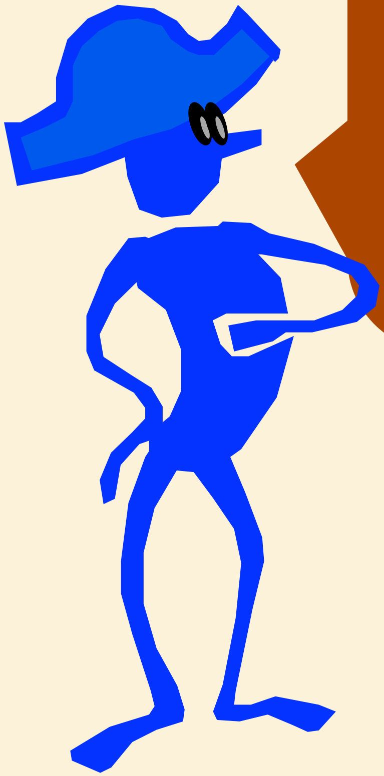
Grade School Addition: θ(n) time
Grade School Multiplication: θ(n^2) time



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

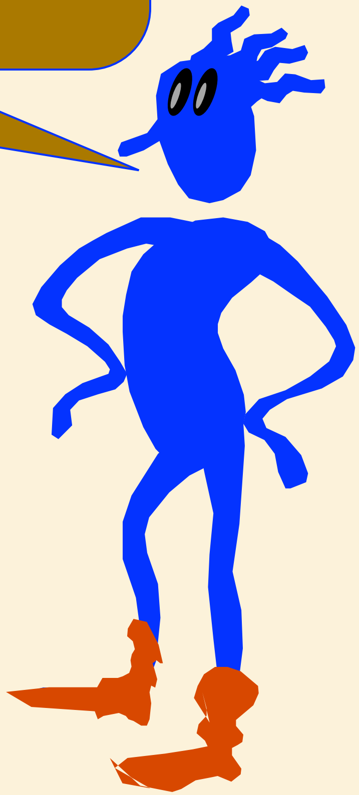
Despite years of research, no one knows!





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

Good question!



Recursive Divide And Conquer

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

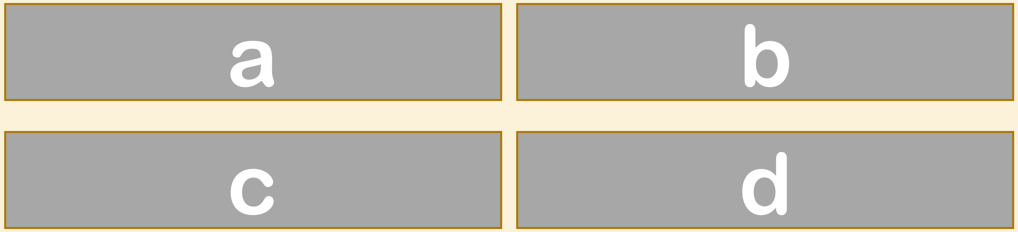
Multiplication of 2 n-bit numbers

• 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)?

Recurrence Relation

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

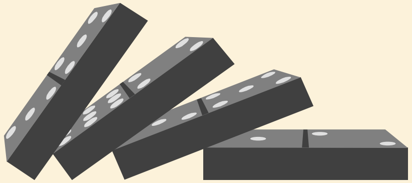
```
MULT(X,Y):
 If $|X| = |Y| = 1$ then RETURN XY
 Break X into a;b and Y into c;d
 RETURN
  MULT(a,c) $2^n$ + (MULT(a,d) + MULT(b,c)) $2^{n/2}$ + MULT(b,d)
```

For example

- $T(1) = 1$
- $T(n) = 4\,T(n/2) + n$

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

Technique 1: (Substitution)



- Recurrence: $T(1) = 1$

  $T(n) = 4T(n/2) + n,\ n = 2,4,8\mathrm{K}$

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

- Proof: $(*) \rightarrow T(1) = 2 - 1 = 1$

  Now suppose (*) is satisfied for $n/2$.

  $\rightarrow T(n/2) = 2\left(n/2\right)^2 - n/2 = n^2/2 - n/2$

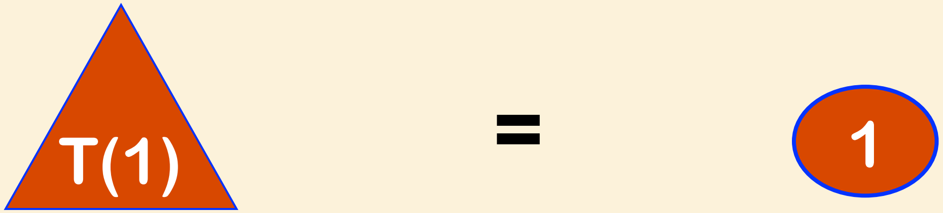
  Then by the recurrence relation,

  $T(n) = 4T(n/2) + n = 2n^2 - 2n + n = 2n^2 - n.$

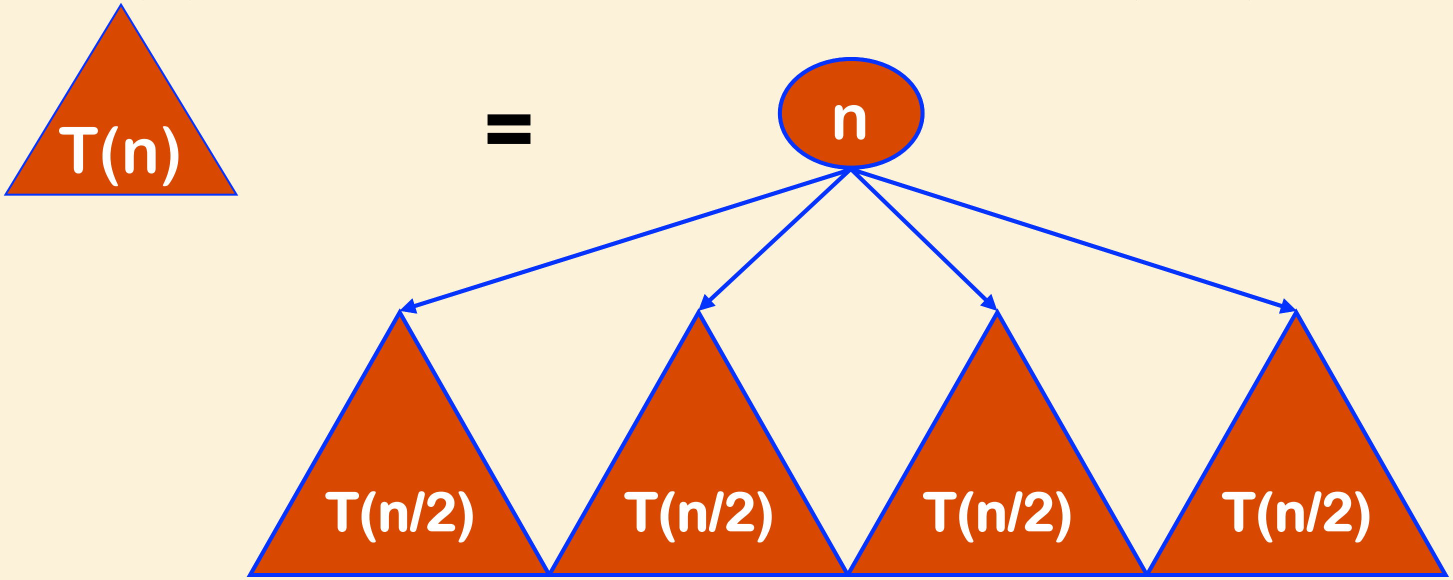
  Thus (*) is also satisfied for n

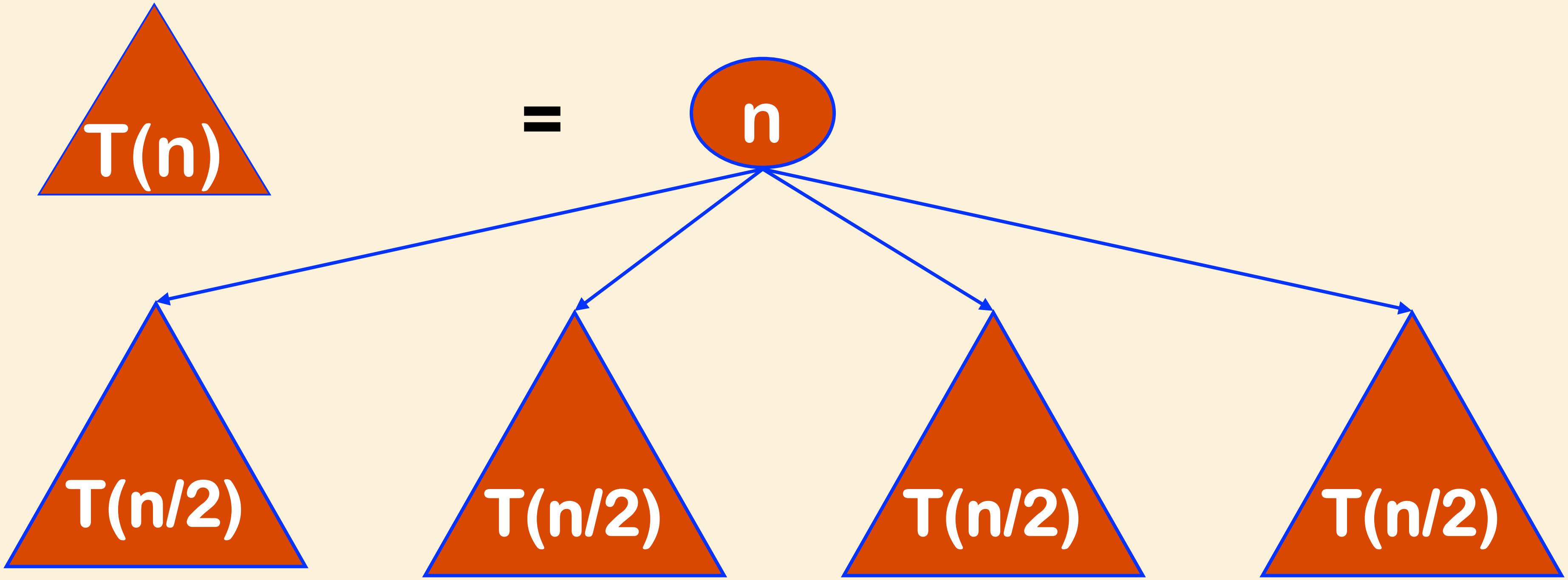
Technique 2: Recursion Tree

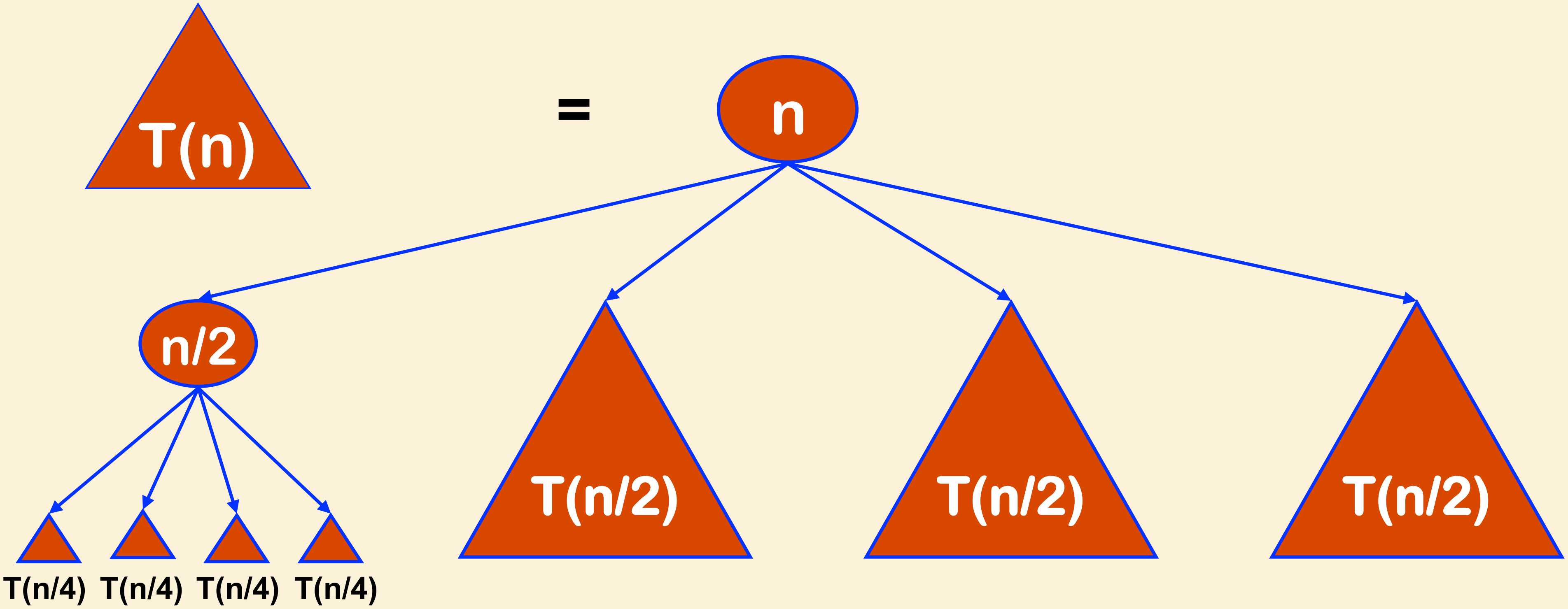
- $T(1) = 1$



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




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



T(n)

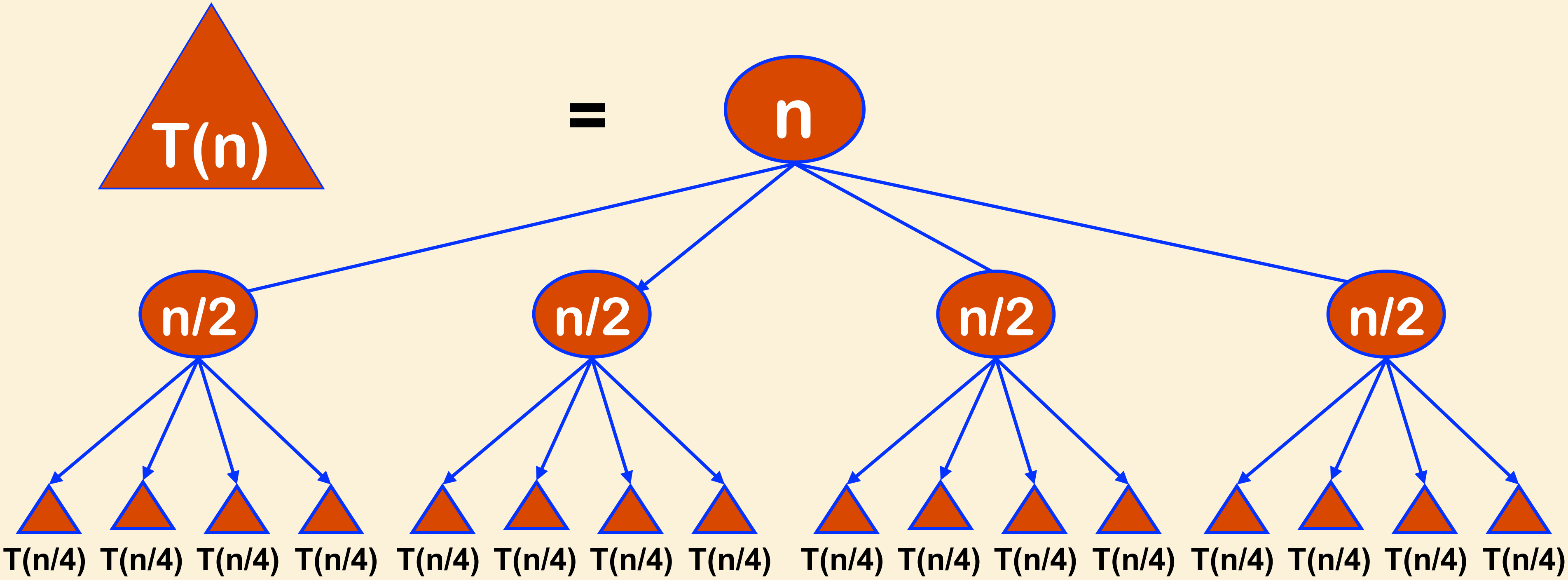
=

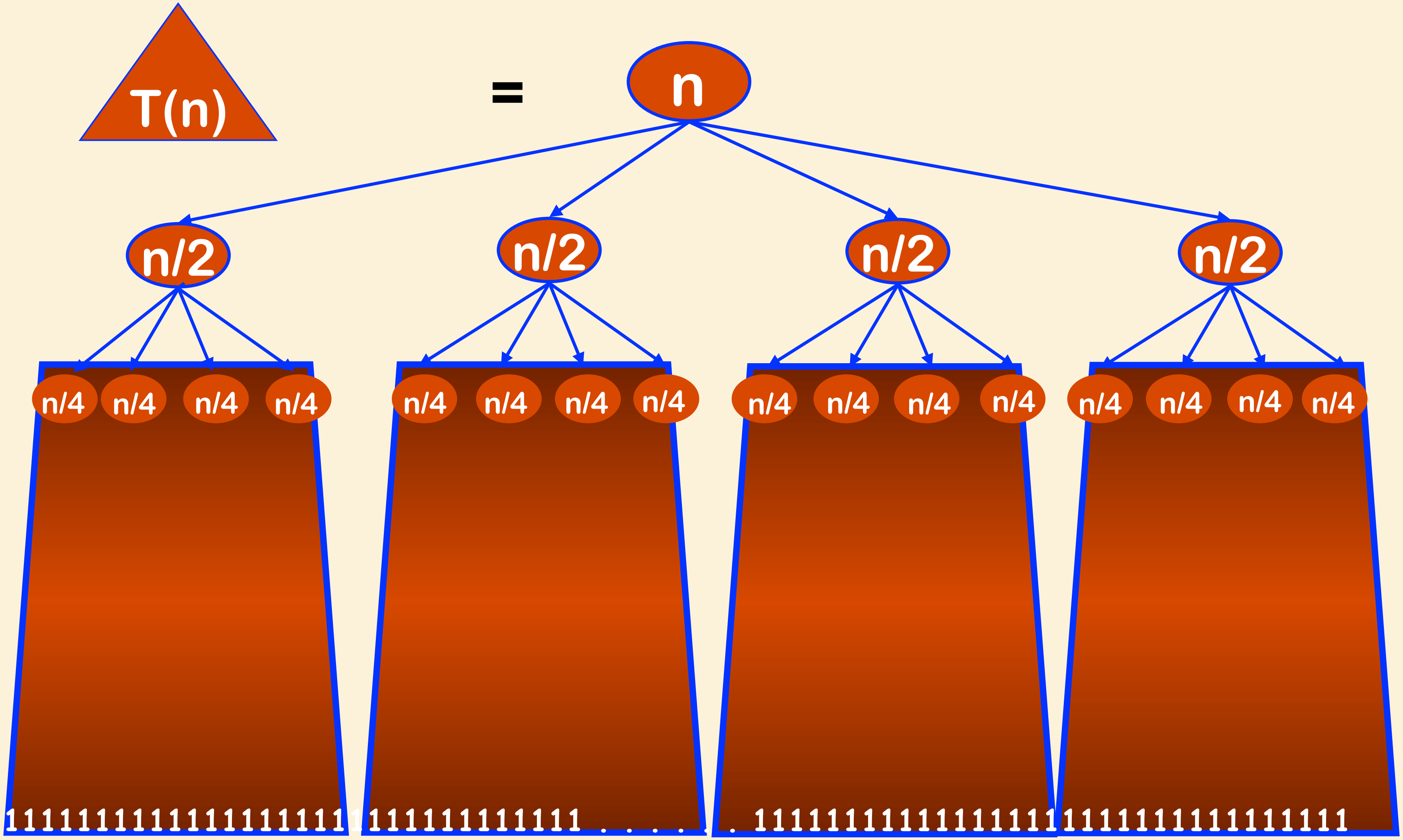
n

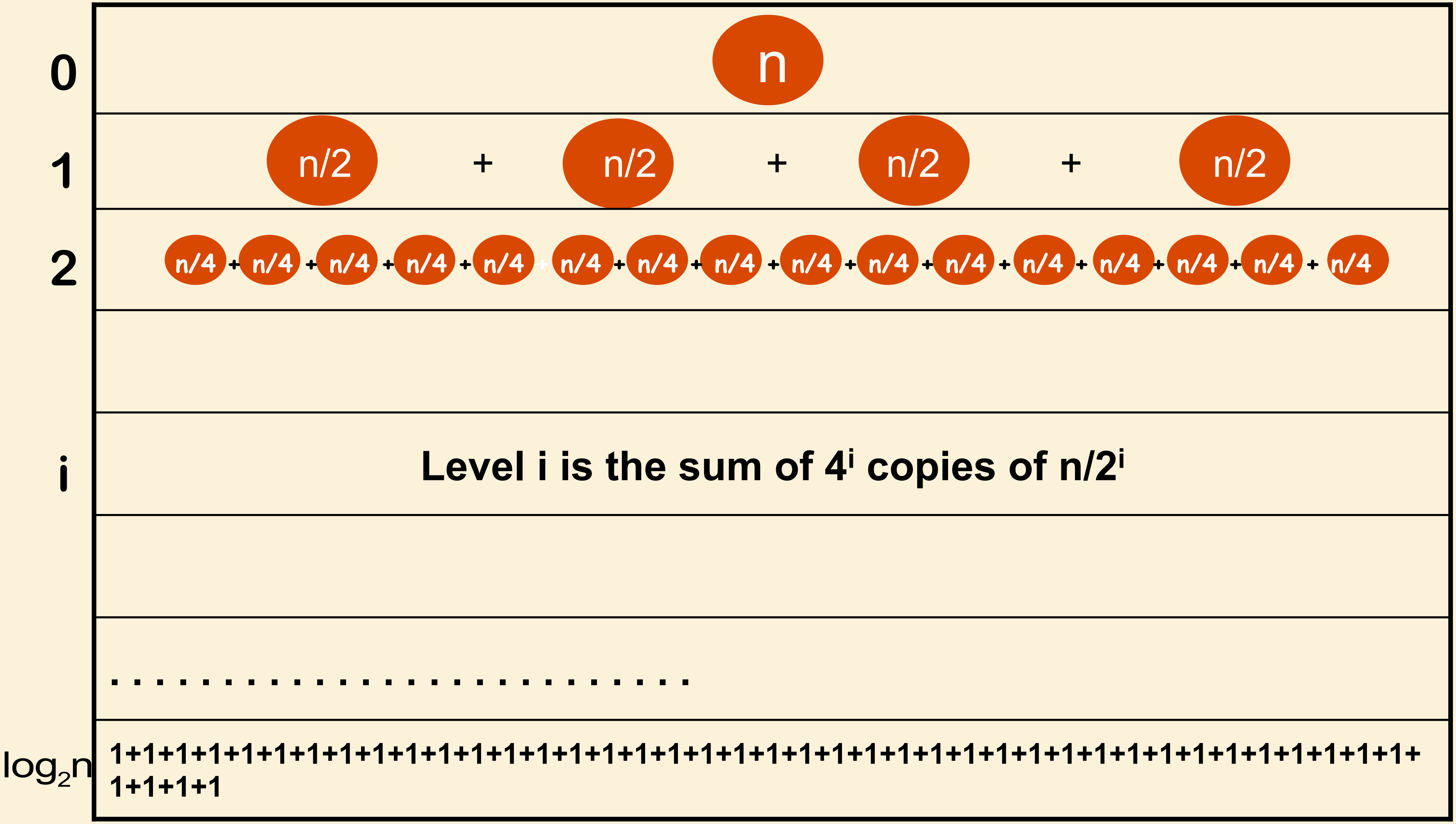
n/2

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

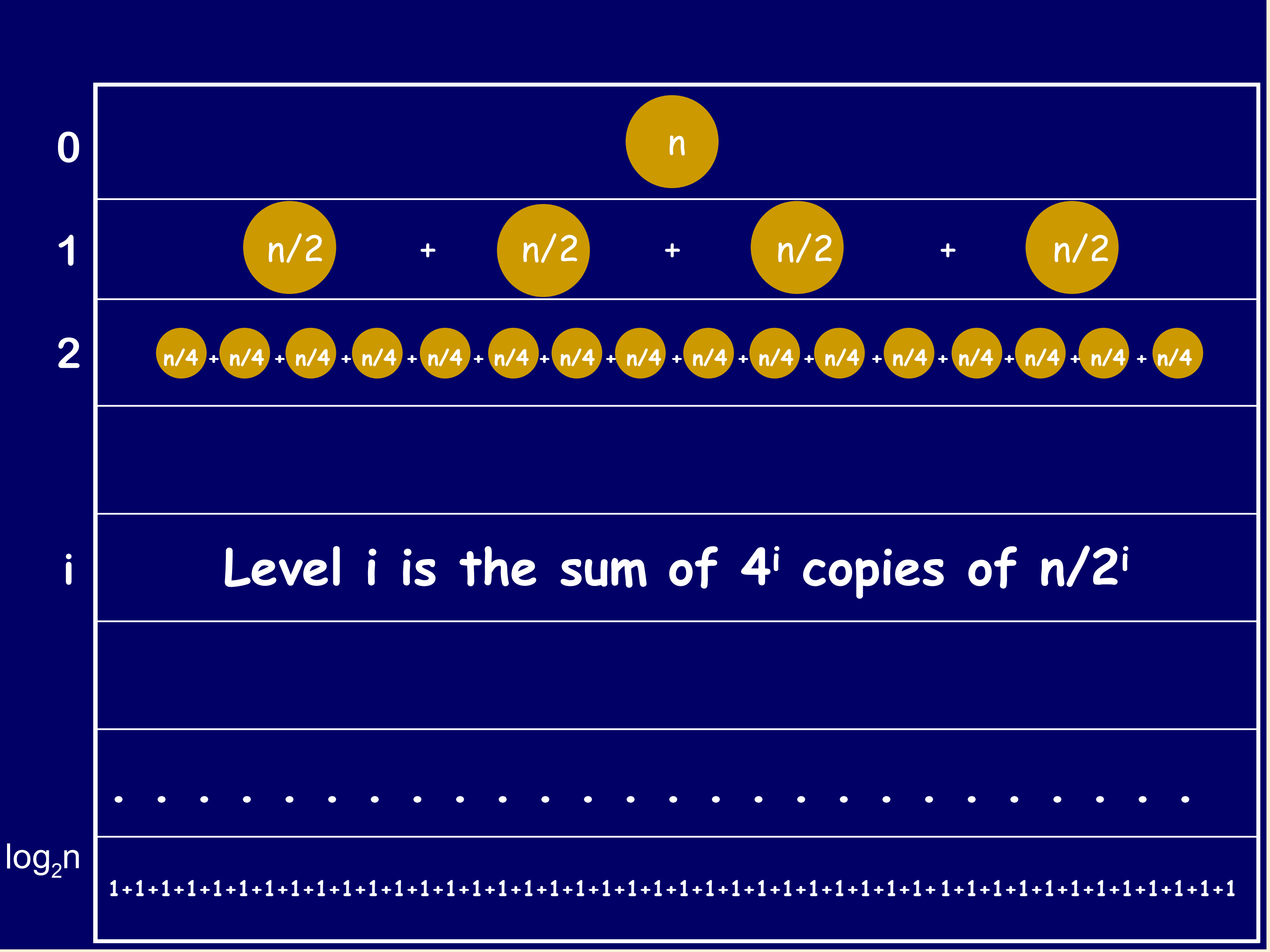
1 n/2 + n/2 + n/2 + n/2

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**



Level i is the sum of 4^i copies of $n/2^i$

$=1 \cdot n$

$= 4 \cdot n/2$

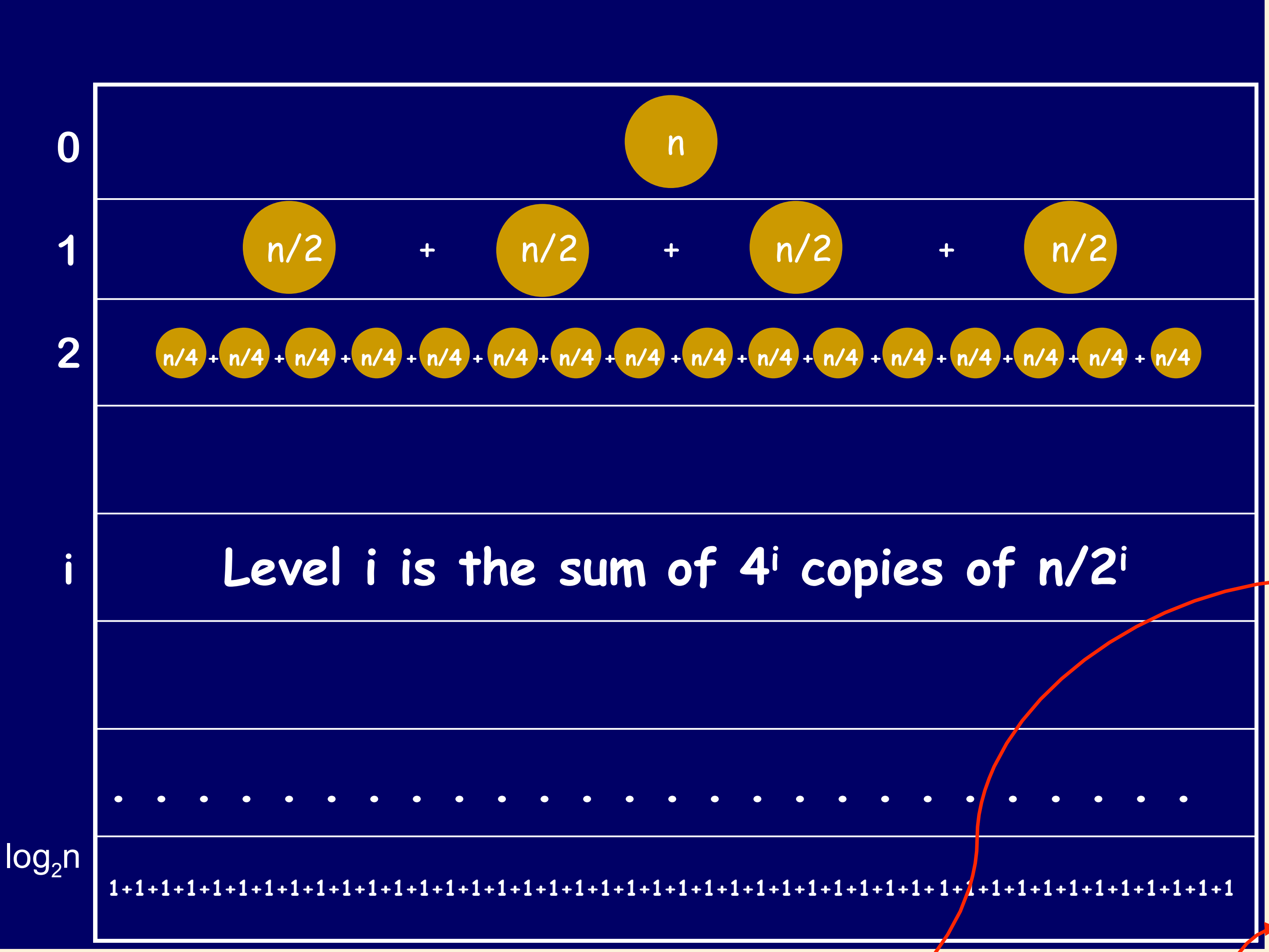
$= 16 \cdot n/4$

$= 4^i \cdot n/2^i$

$= 4^{\log n} \cdot n/2^{\log n}$

$=2^{2\log n} \cdot 1=n^2$

$$n\sum_{i=0}^{\log n} 2^i = n\left(2^{\log n+1}-1\right) = n\left(2n-1\right) = 2n^2 - n \in \Theta(n^2)$$



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

$=1 \cdot n$

$= 4 \cdot n/2$

$= 16 \cdot n/4$

$= 4^i \cdot n/2^i$

$= 4^{\log n} \cdot n/2^{\log n}$

$=2^{2\log n} \cdot 1 = n^2$

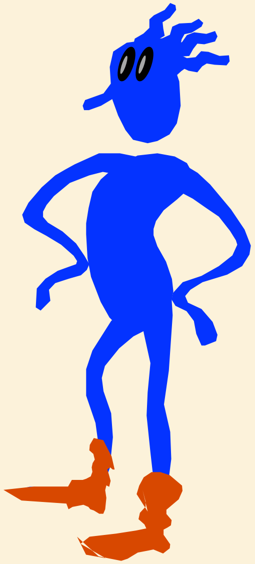
Geometric Increasing - dominated by last term: $\sum f(n) = \Theta(f(n)) = \Theta(n^2)$

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

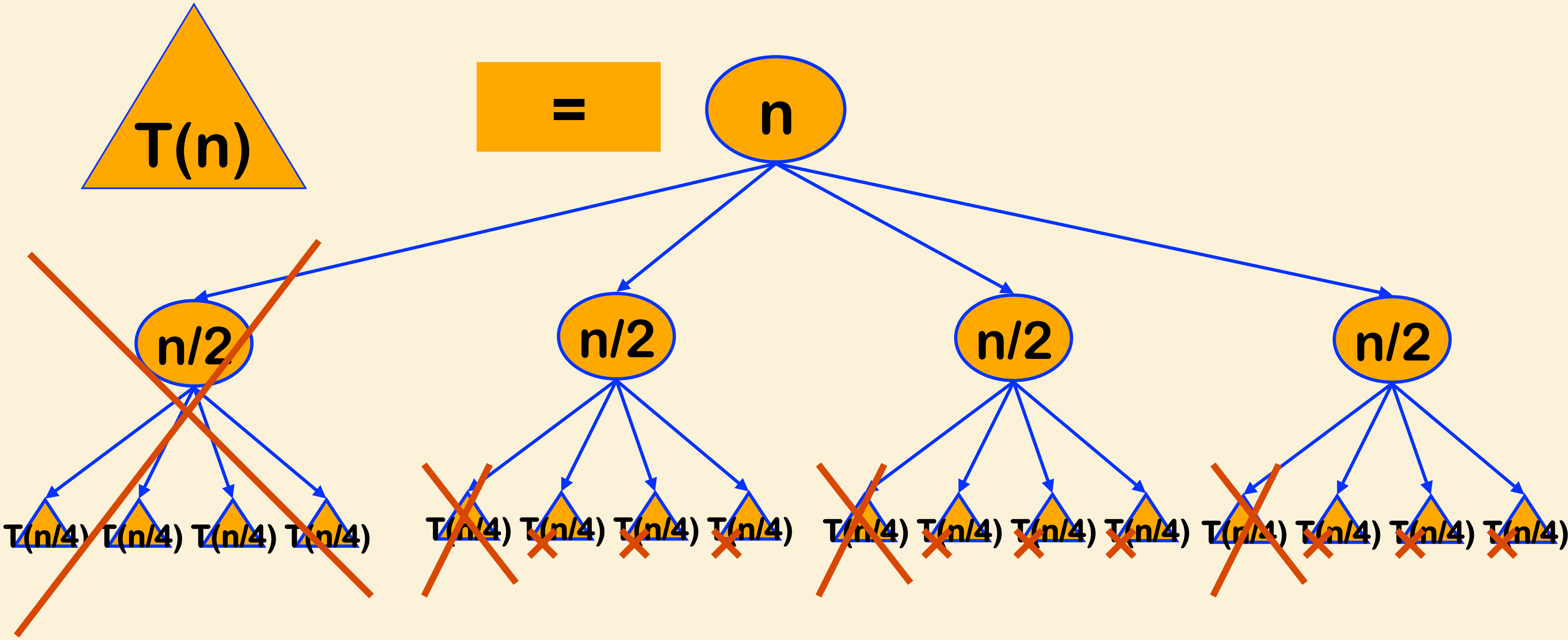
- $A_1 = ac$
- $A_3 = bd$
- $m_3 \quad = (a+b)(c+d) = \cancel{ac} + \boxed{ad + bc} + \cancel{bd}$
- $A_2 = m_3 – A_1 - A_3 = \quad ad + bc$

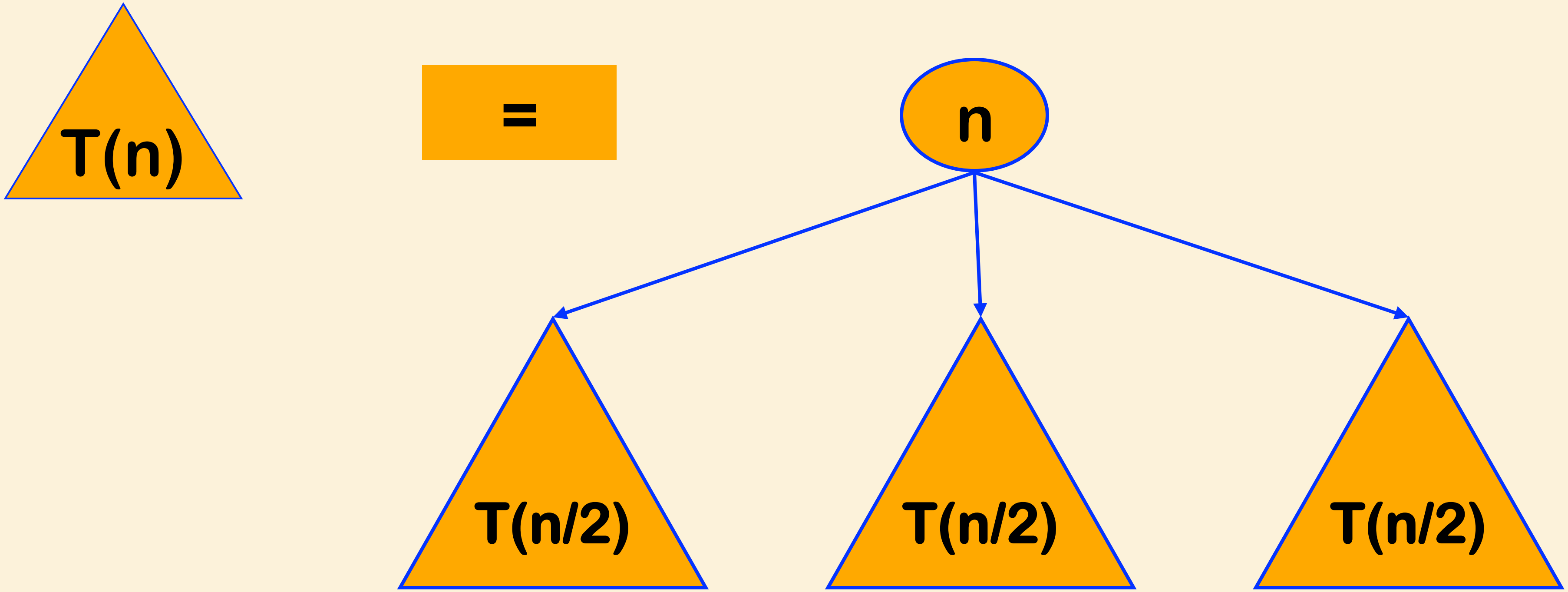
Gaussified MULT (Karatsuba 1962)

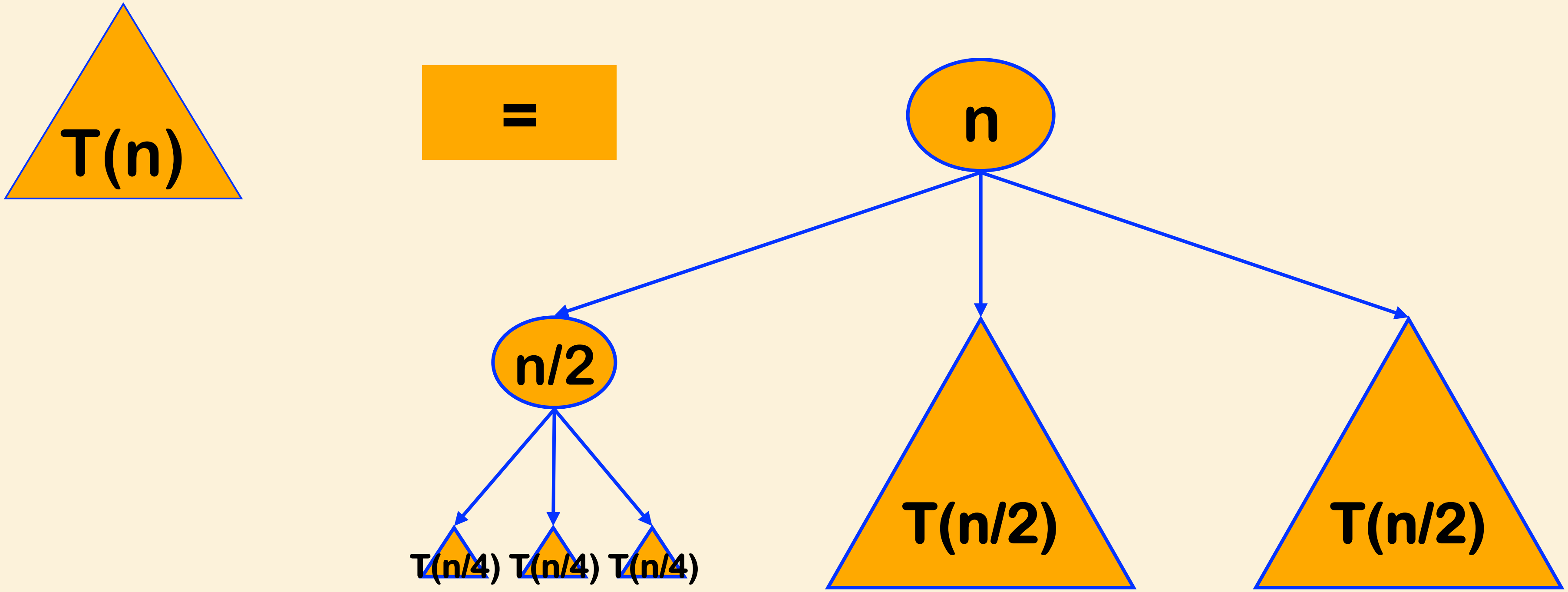
```
MULT(X,Y):
  If $|X| = |Y| = 1$ then RETURN XY
  Break X into a;b and Y into c;d
  e = MULT(a,c) and f =MULT(b,d)
  RETURN  $e2^n + (\text{MULT}(a+b, c+d) - e - f) 2^{n/2} + f$
```

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

(More precisely: $T(n) = 2\, T(n/2) + T(n/2 + 1) + kn$)


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)

=

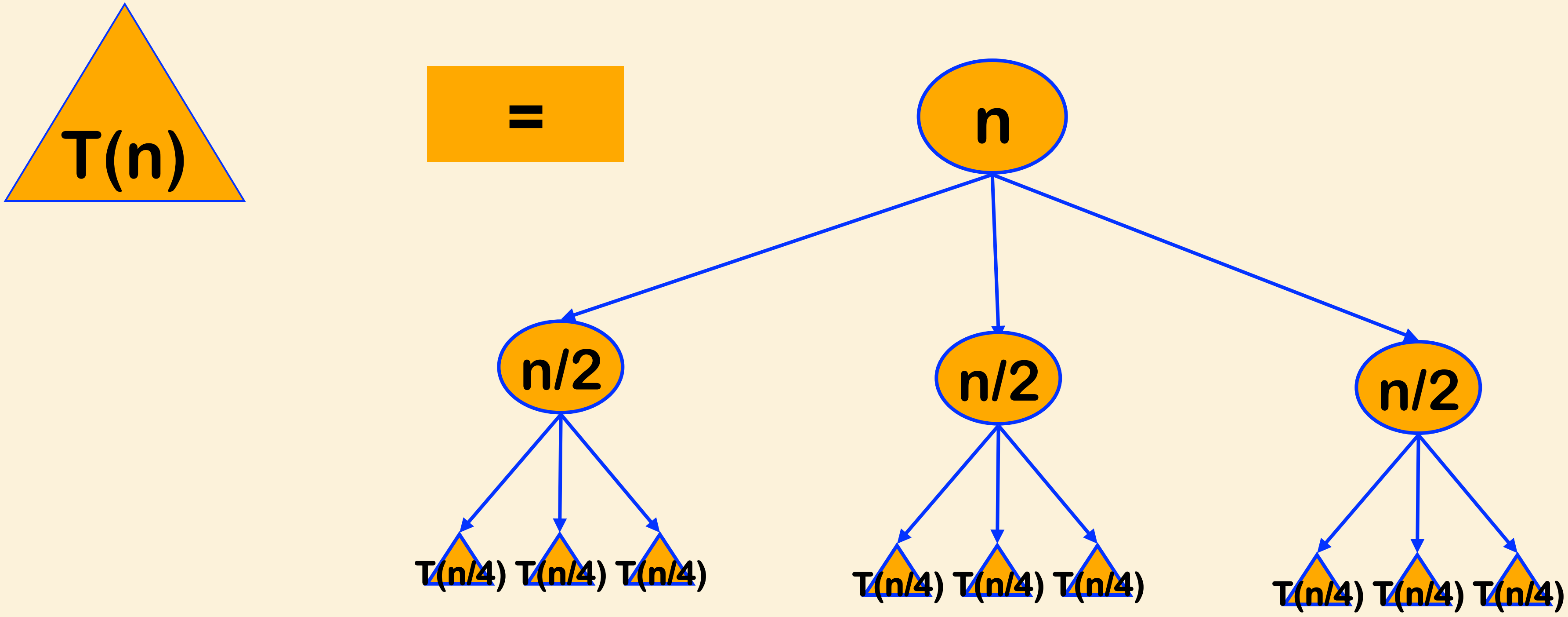
n

n/2

T(n/2)

T(n/2)

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



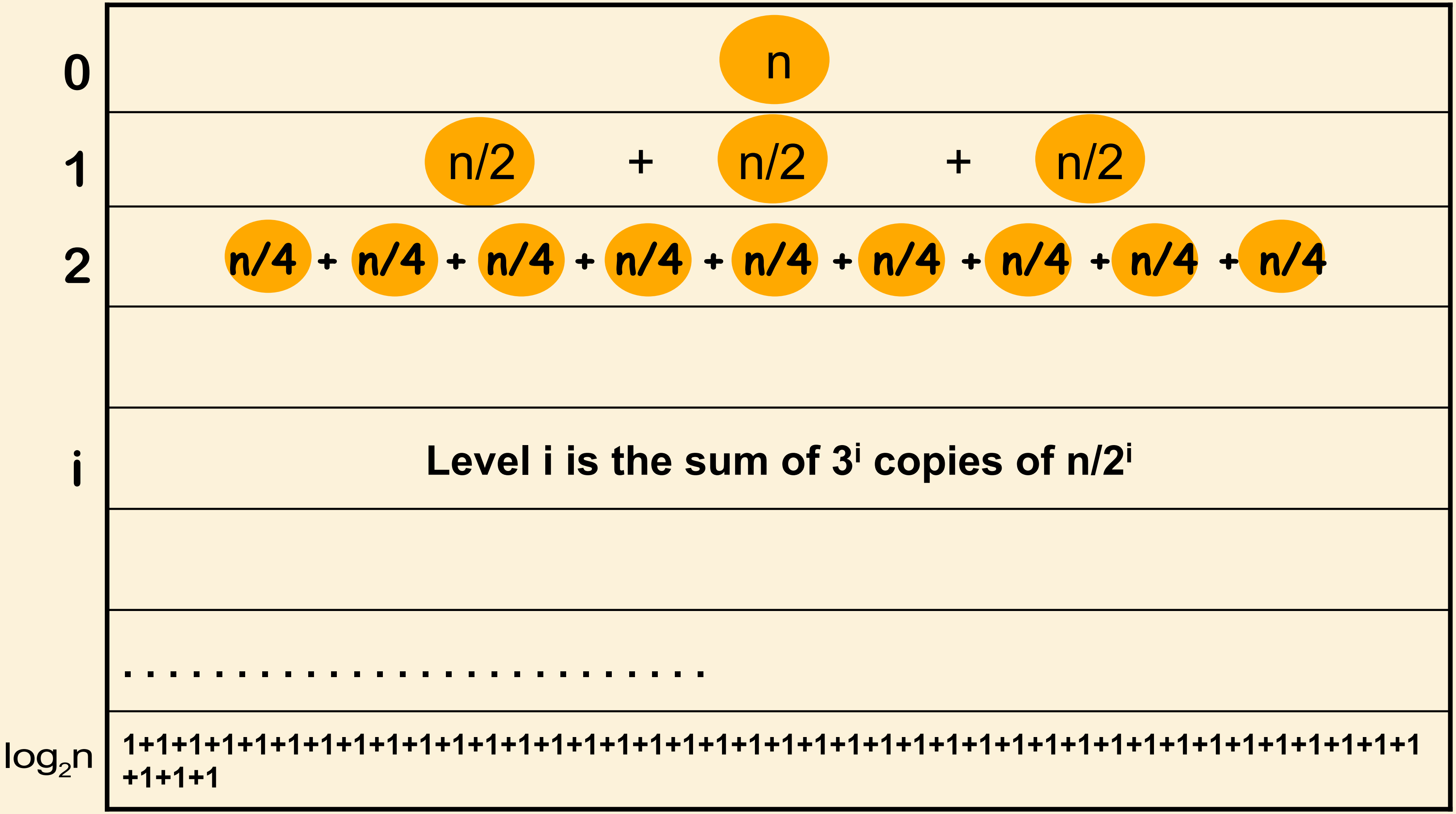
T(n)

=

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

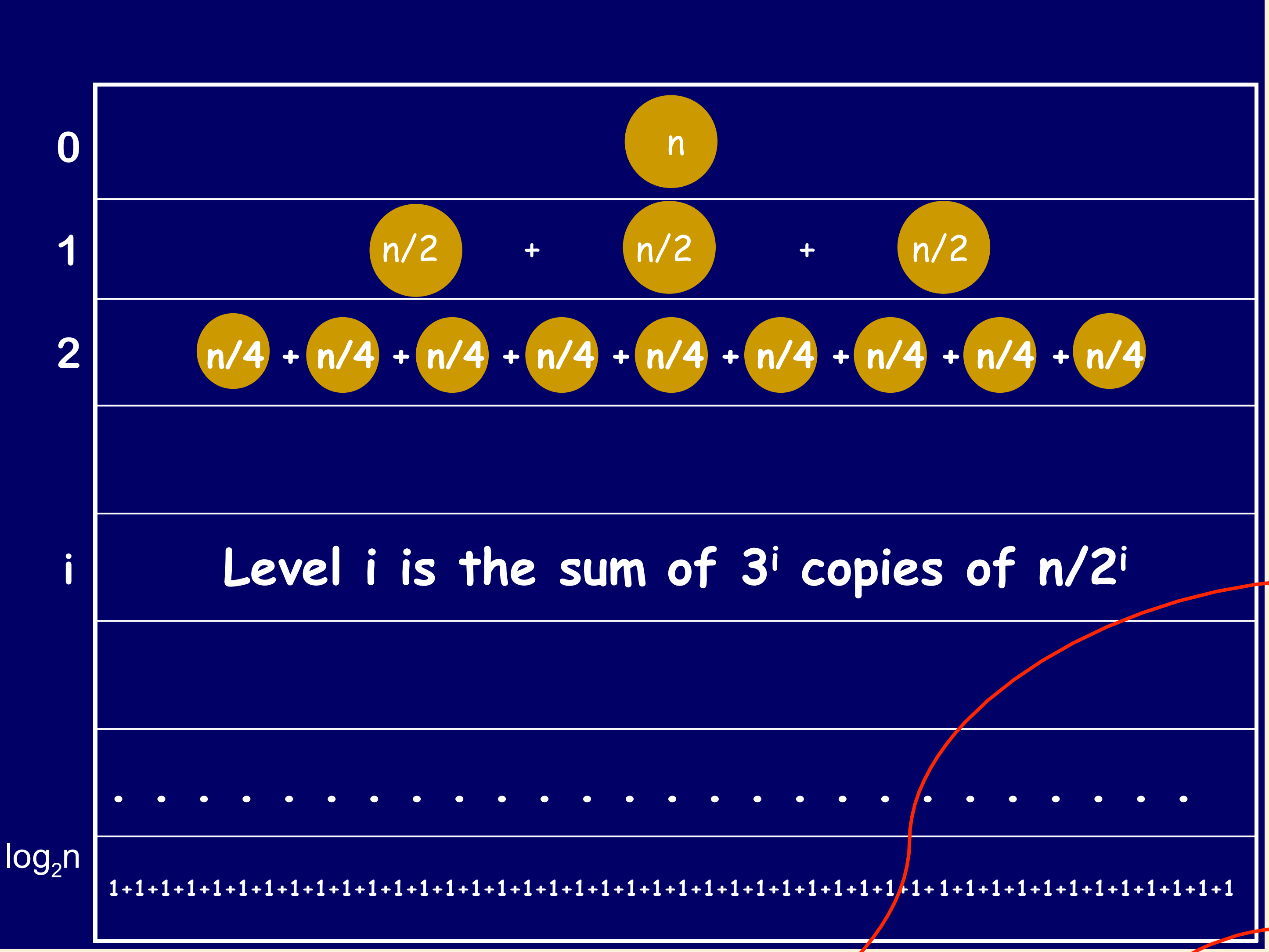
1 n/2 + n/2 + n/2

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



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

$=1 \cdot n$

$= 3 \cdot n/2$

$= 9 \cdot n/4$

$= 3^i \cdot n/2^i$

$= 3^{\log n} \cdot n/2^{\log n}$

$= n^{\log 3} \cdot 1$

Geometric Increasing - dominated by last term: $\sum f(n) = \Theta(f(n)) = \Theta(n^{\log 3})$; $\Theta(n^{1.58\ldots})$

Dramatic improvement for large n

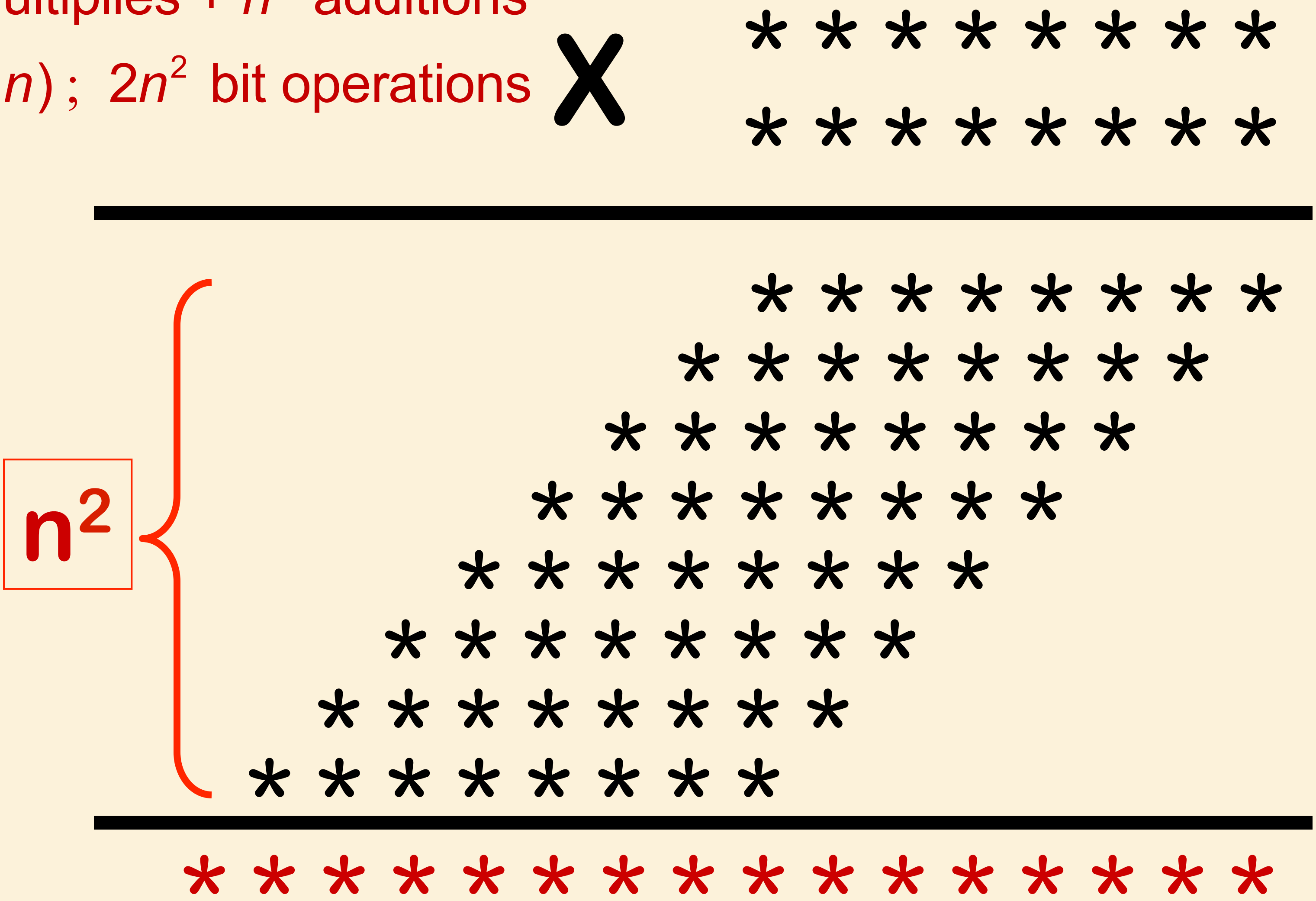
Not just a 25% savings!

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

Grade-School Multiplication

n^2 multiplies + n^2 additions

→ $T(n)$; $2n^2$ bit operations



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

↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑

$$T(n) \approx 3\, T(n/2) + kn$$

What is k? e.g., *k*=8

Dramatic improvement for large n

Not just a 25% savings!

$$\theta(2n^2) \text{ vs } \theta(8n^{1.58..})$$

Example:

A networking simulation requires 10 million multiplications of 16-bit integers.

Suppose that each bit operation takes 4 picosec on your machine (realistic).

Grade School Multiplication Time = 2 days 9 hours (do it over the weekend!)

Karatsuba Multiplication Time = 5.4 minutes (just enough time to grab a coffee!)

MATLAB takes 0.07 seconds on my machine (don't blink!)

Multiplication Algorithms

Kindergarten	$n2^n$
Grade School	n^2
Karatsuba	$n^{1.58\ldots}$
Fastest Known (**Schönhage-Strassen algorithm**, 1971)	n logn loglogn

What a difference a single recursive call makes!

- What are the underlying principles here?
- How can we systematically predict which recursive algorithms are going to save time, and which are not?

Recurrence Relations

$$T(1) = 1$$

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

Recurrence Relations ≈ Time of Recursive Program

```
procedure  Eg(int n)
    if(n≤1) then
        put “Hi”
     else
        loop i=1..f(n)
            put “Hi”
        loop i=1..a
            Eg(n/b)
```

- Recurrence relations arise from the timing of recursive programs.

- Let **T(n)** be the # of “Hi”s on an input of “size” **n**.

Recurrence Relations
≈ Time of Recursive Program

```
procedure  Eg(int n)
    if($n \le 1$) then
        put "Hi"
     else
        loop i=1..f(n)
          put "Hi"
        loop i=1..a
          Eg(n/b)
```

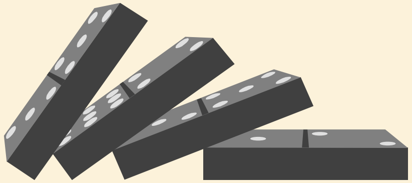
Given size 1, the program outputs $T(1)=1$ Hi's.

Given size n, the stackframe outputs **f(n)** Hi's.

Recursing on ***a*** instances of size **n/b** generates **aT(n/b)** "Hi"s.

For a total of **$T(1) = 1$**; **$T(n) = a \cdot T(n/b) + f(n)$** "Hi"s.

Technique 1: (Substitution)



- Recurrence: $T(1) = 1$

  $T(n) = 4T(n/2) + n, \; n = 2,4,8K$

- Guess: $(*)\; T(n) = 2n^2 - n$

- Proof: $(*) \rightarrow T(1) = 2 - 1 = 1$

  Now suppose (*) is satisfied for $n/2$.

  $\rightarrow T(n/2) = 2\left(n/2\right)^2 - n/2 = n^2/2 - n/2$

  Then by the recurrence relation,

  $T(n) = 4T(n/2) + n = 2n^2 - 2n + n = 2n^2 - n.$

  Thus (*) is also satisfied for n

More Generally, and Formally

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

Hypothesis: $T(n)=\Theta(n^2)$ i.e., $\exists c_1,c_2,n_0>0:\forall n\geq n_0, c_1n^2 \leq T(n) \leq c_2n^2$

Step 1. Lower Bound

Suppose that lower bound holds for $\lfloor i/2 \rfloor$, i.e., $c_1\lfloor i/2 \rfloor^2 \leq T(\lfloor i/2 \rfloor)$

Substituting,

$$\begin{aligned} T(i) = 4T\left(\lfloor i/2 \rfloor\right)+i &\geq 4c_1\lfloor i/2 \rfloor^2+i \\ &\geq 4c_1\left(\frac{i-1}{2}\right)^2+i \\ &= 4c_1\left(i^2/4-i/2+1/4\right)+i \\ &= c_1\left[i^2-2i+1\right]+i \end{aligned}$$

Suppose that $c_1=\frac{1}{2}$

Then $T(i)\geq\frac{1}{2}\left[i^2-2i+1\right]+i = \frac{1}{2}i^2+\frac{1}{2} \geq \frac{1}{2}i^2 = c_1i^2$ Thus lower bound holds for i!

To Summarize

If lower bound holds for $\lfloor i/2 \rfloor$, i.e., $c_1 \lfloor i/2 \rfloor^2 \le T(\lfloor i/2 \rfloor)$ with $c_1 = \frac{1}{2}$,

Then lower bound holds for i, i.e., $c_1 i^2 \le T(i)$

Base Case

Does lower bound hold for $i = 1$?

$$c_1 i^2 = \frac{1}{2}(1)^2 = \frac{1}{2} \le T(i) = 1$$ Yes!

By induction, must also hold for $i = 2,3,4,5,\ldots$

e.g.,

$i = 1 \rightarrow i = 2,3$

$i = 2 \rightarrow i = 4,5$

$i = 3 \rightarrow i = 6,7$

M

Follow similar process to prove upper bound.

Solving Technique 2
Guess Form and Calculate Coefficients

- **Recurrence Relation:**

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

- **Guess:** $T(n) = an^2 + bn + c$

- **Verify:**

Left Hand Side	Right Hand Side
$T(1)$ = $a+b+c$	**1**
$T(n)$	**$4T(n/2) + n$**
$= an^2 + bn + c$	$= 4\,[a\,(^n/_2)^2 + b\,(^n/_2) + c] + n$
	$= an^2 + (2b+1)n + 4c$

Solving Technique 2
Guess Form and Calculate Coefficients

- **Recurrence Relation:**

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

- **Guess:** $T(n) = an^2 + bn + c$

- **Verify:**

Left Hand Side	Right Hand Side
$T(1) = a+b+c$	1
$T(n)$	$4T(n/2) + n$
$= an^2 + bn + c$	$= 4\,[a\,(^n/_2)^2 + b\,(^n/_2) + c] + n$
$c = 4c$	$= an^2 + (2b+1)n + 4c$
$\rightarrow c = 0$	

Solving Technique 2
Guess Form and Calculate Coefficients

- **Recurrence Relation:**

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

- **Guess:** $T(n) = an^2 + bn + 0$

- **Verify:**

Left Hand Side	Right Hand Side
$T(1) = a+b+c$	1
$T(n)$	$4T(n/2) + n$
$= an^2 + bn + c$	$= 4\,[a\,(^n/_2)^2 + b\,(^n/_2) + c] + n$
$b = 2b+1$	$= an^2 + (2b+1)n + 4c$
→ $b = -1$	

Solving Technique 2
Guess Form and Calculate Coefficients

- **Recurrence Relation:**

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

- **Guess:** $T(n) = an^2 - 1n + 0$

- **Verify:**

Left Hand Side	Right Hand Side
$T(1) = a+b+c$	1
$T(n)$	$4T(n/2) + n$
$= an^2 + bn + c$	$= 4\,[a\,(^n/_2)^2 + b\,(^n/_2) + c] + n$
	$= an^2 + (2b+1)n + 4c$
$a=a$	

Solving Technique 2
Guess Form and Calculate Coefficients

- **Recurrence Relation:**

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

- **Guess:** $T(n) = an^2 - 1n + 0 \rightarrow T(n) = 2n^2 - n$

- **Verify:**

$a+b+c=1$

$a-1+0=1$

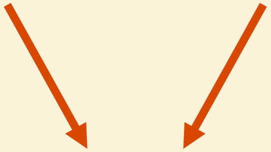
$\rightarrow a=2$

Left Hand Side	Right Hand Side
$T(1) = a+b+c$	1
$T(n)$	$4T(n/2) + n$
$= an^2 + bn + c$	$= 4\,[a\,(^n/_2)^2 + b\,(^n/_2) + c] + n$
	$= an^2 + (2b+1)n + 4c$

Solving Technique 3
Approximate Form and Calculate Exponent

- **Recurrence Relation:**

$$T(1) = 1 \ \& \ T(n) = aT(n/b) + f(n)$$



which is bigger?

Guess

Solving Technique 3
Calculate Exponent

- **Recurrence Relation:**

  $T(1) = 1$ & $T(n) = aT(n/b) + f(n)$

- **Guess:** $aT(n/b) \ll f(n)$
- **Simplify:** $T(n) \approx f(n)$

In this case, the answer is easy.

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

Solving Technique 3
Calculate Exponent

- **Recurrence Relation:**

  $T(1) = 1$ & $T(n) = aT(n/b) + f(n)$

- **Guess:** **$aT(n/b) >> f(n)$**
- **Simplify:** **$T(n) \approx aT(n/b)$**

In this case, the answer is harder.

Solving Technique 3
Calculate Exponent

- **Recurrence Relation:**

$$T(1) = 1 \;\&\; T(n) = aT(n/b)$$

- **Guess:** $T(n) = cn^{\alpha} = cn^{({}^{\log a}/_{\log b})}$

- **Verify:**

Left Hand Side	Right Hand Side
T(n) $= \cancel{c}\cancel{n}^{\alpha}$ $1 = a\,b^{-\alpha}$ $b^{\alpha} = a$ $\alpha \log b = \log a$ $\alpha = {}^{\log a}/_{\log b}$	**aT(n/b)** $= a\,[c\,({}^{n}/_{b})^{\alpha}]$ $= \cancel{c}\,a\,b^{-\alpha}\,\cancel{n}^{\alpha}$

Solving Technique 3
Calculate Exponent

- **Recurrence Relation:**

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

- **Guess:** $T(n) = cn^{\alpha} = cn^{(\log a/\log b)} = cn^{\log 4/\log 2} = cn^{2}$

- **Verify:**

Left Hand Side	Right Hand Side
$T(n)$ $= \cancel{c}\cancel{n}^{\alpha}$ $1 = a\, b^{-\alpha}$ $b^{\alpha} = a$ $\alpha \log b = \log a$ $\alpha = \log a / \log b$	$aT(n/b)$ $= a\,[c\,(n/b)^{\alpha}]$ $= \cancel{c}\, a\, b^{-\alpha}\, \cancel{n}^{\alpha}$

Solving Technique 3
Calculate Exponent

- **Recurrence Relation:**

$$T(1) = 1 \;\&\; T(n) = aT(n/b) + f(n)$$

If bigger then

$$T(n) = \Theta(n^{(\log a/\log b)})$$

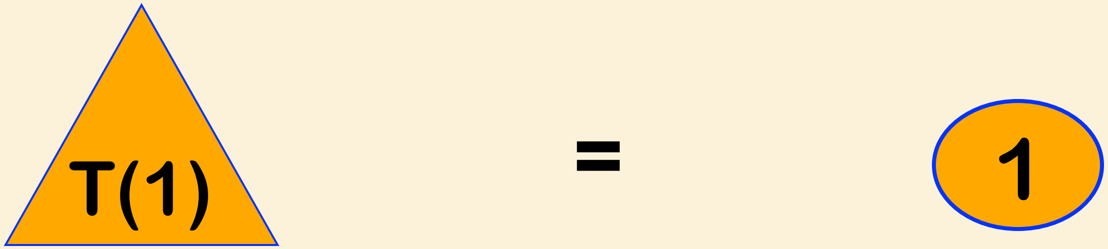
If bigger then

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

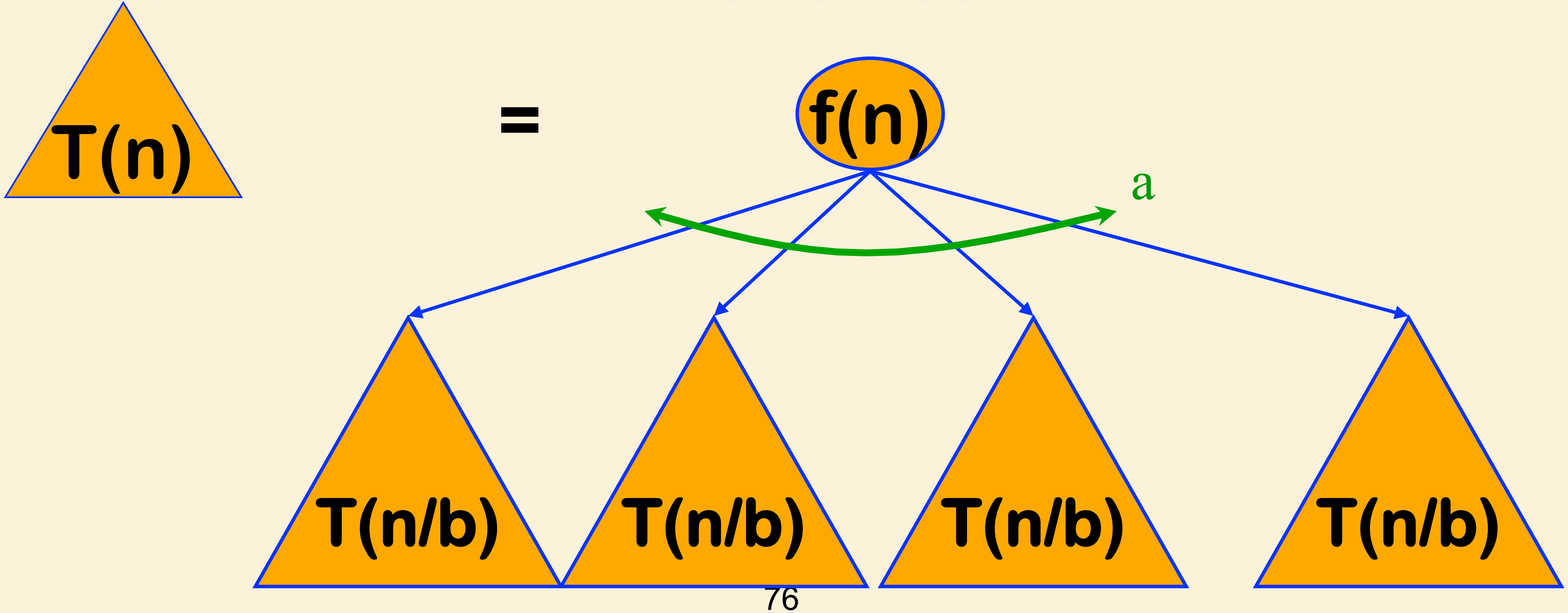
And if $\mathbf{aT(n/b) \approx f(n)}$
what is T(n) then?

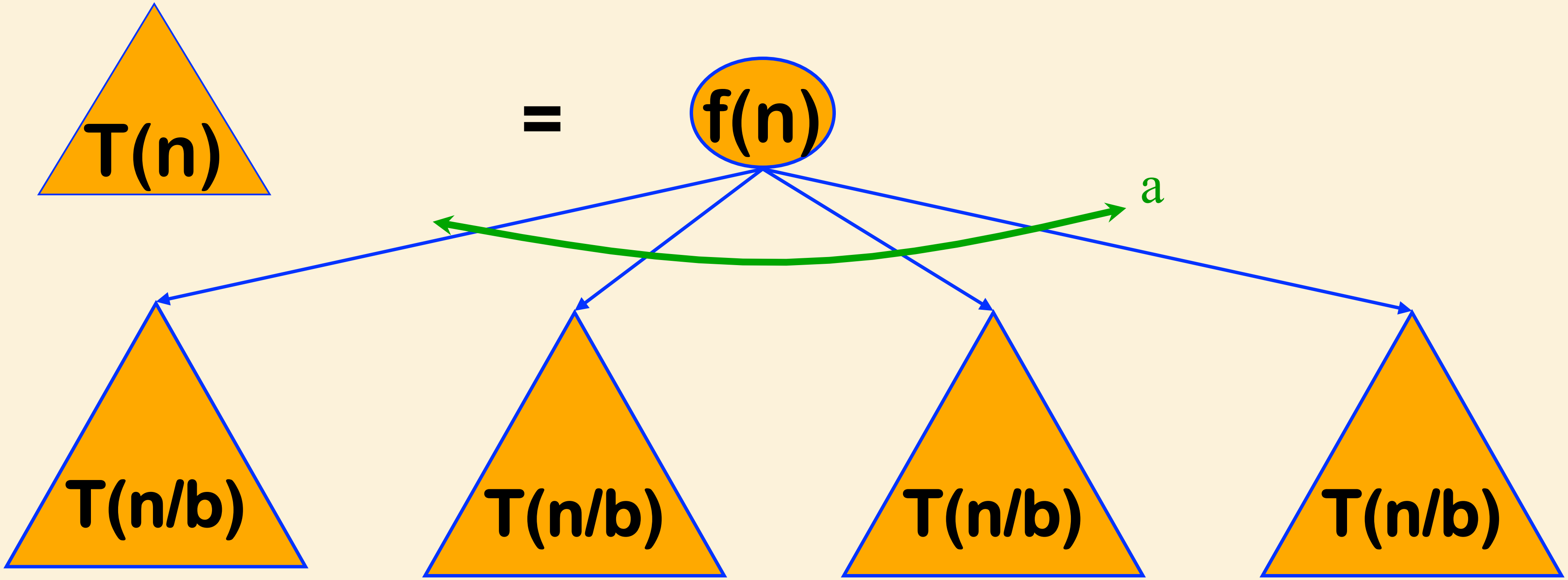
Technique 4: Recursion Tree Method

- T(1) = 1



- T(n) = a T(n/b) + f(n)

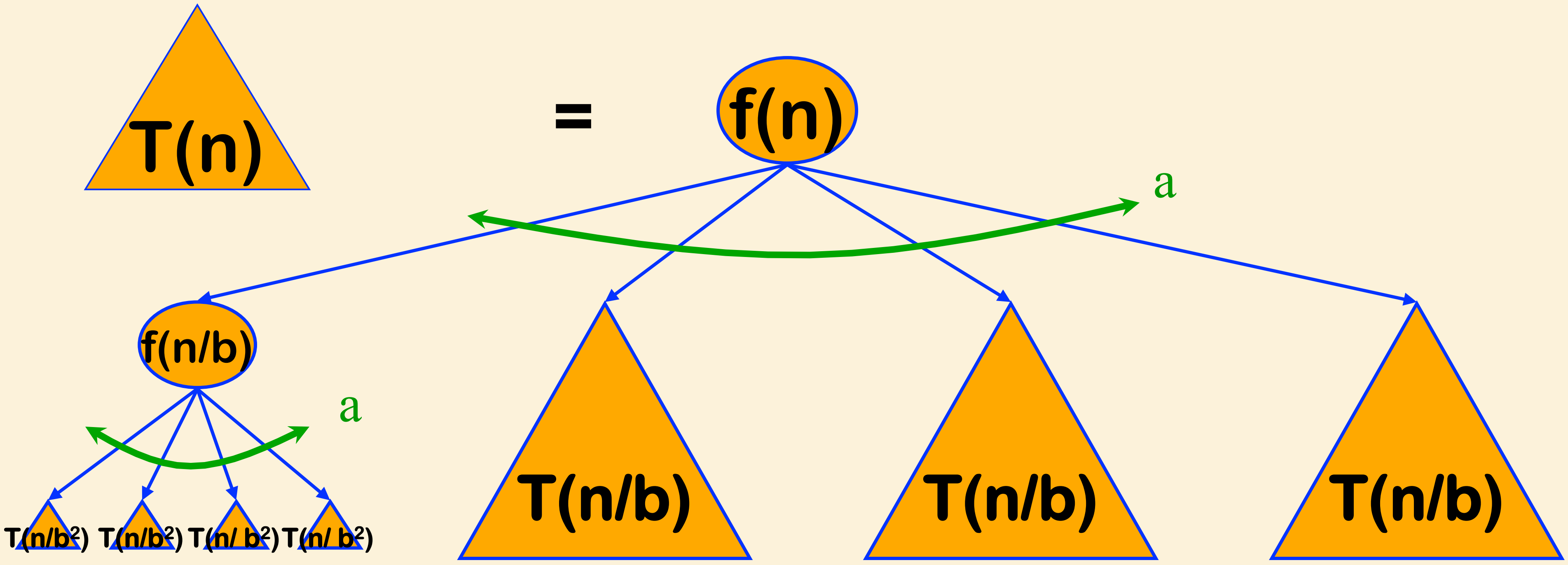


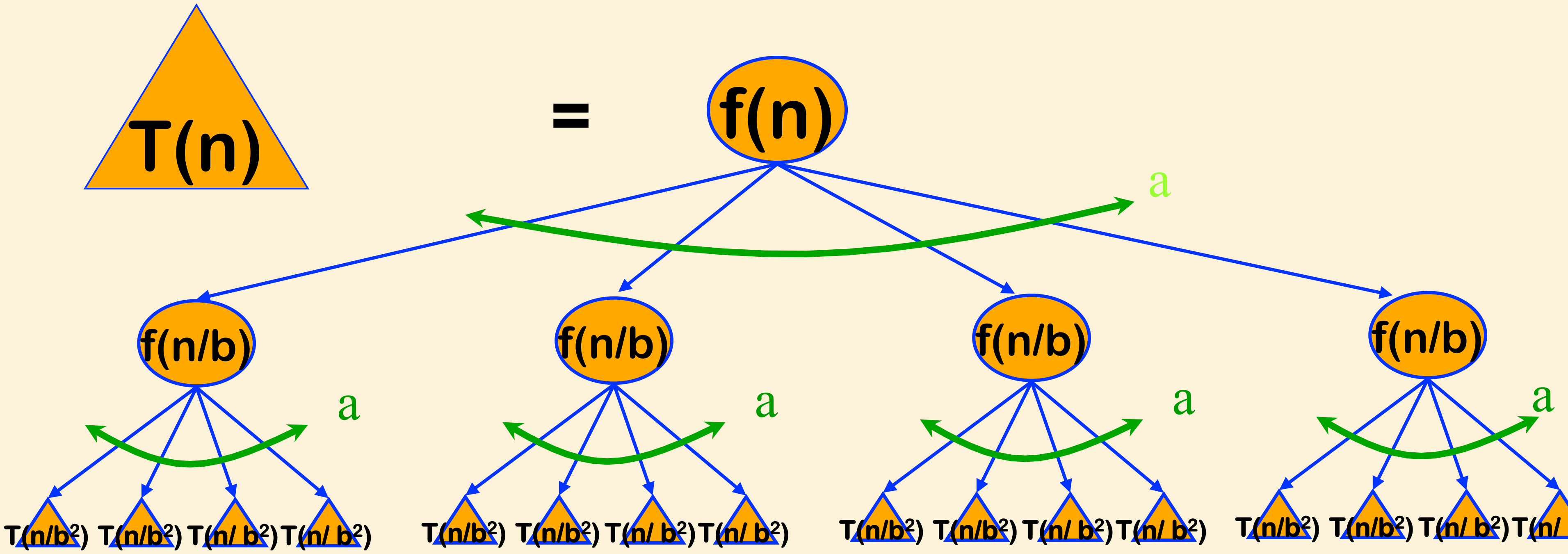


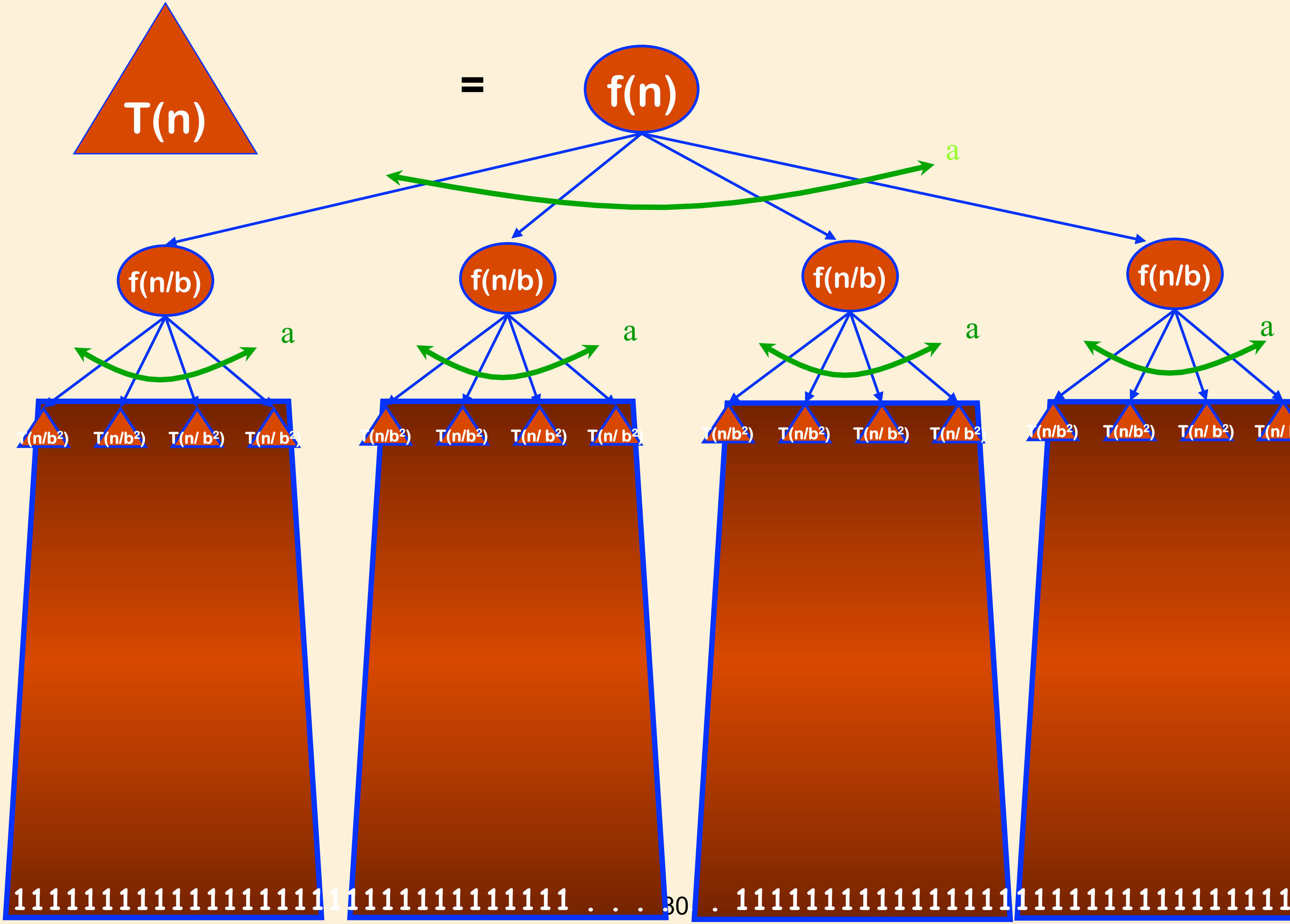
T(n) = f(n)

a

T(n/b) T(n/b) T(n/b) T(n/b)

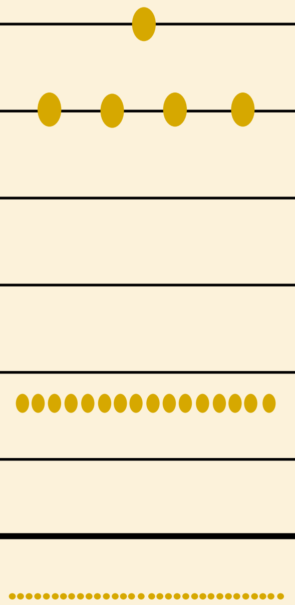

T(n)
=
f(n)
a
f(n/b)
a
T(n/b²)
T(n/b²)
T(n/ b²)
T(n/ b²)
T(n/b)
T(n/b)
T(n/b)


T(n)
=
f(n)
a
f(n/b)
f(n/b)
f(n/b)
f(n/b)
a
a
a
a
T(n/b2) T(n/b2) T(n/ b2) T(n/ b2)
T(n/b2) T(n/b2) T(n/ b2) T(n/ b2)
T(n/b2) T(n/b2) T(n/ b2) T(n/ b2)
T(n/b2) T(n/b2) T(n/ b2) T(n/


T(n)
=
f(n)
a
f(n/b)
f(n/b)
f(n/b)
f(n/b)
a
a
a
a
T(n/b²)
T(n/b²)
T(n/ b²)
T(n/ b²)
T(n/b²)
T(n/b²)
T(n/ b²)
T(n/ b²)
T(n/b²)
T(n/b²)
T(n/ b²)
T(n/ b²)
T(n/b²)
T(n/b²)
T(n/ b²)
T(n/
11111111111111111111111111111111 111111111111111111111111111111111

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

Level	# Instances				
0					
1					
2					
i					
h					



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

Level	# Instances	Instance size			
0	●				
1	● ● ● ●				
2					
i					
	●●●●●●●●●●●●●●●●				
h					

Evaluating: T(n) = aT(n/b)+f(n)

Level	# Instances	Instance size			
0	●	n			
1	● ● ● ●				
2					
i					
	●●●●●●●●●●●●●●●●				
h					

●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●

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

Level	# Instances	Instance size			
0	●	n			
1	● ● ● ●	**n/b**			
2					
i					
	●●●●●●●●●●●●●●●●				
h					

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

Level	# Instances	Instance size			
0	●	n			
1	● ● ● ●	**n/b**			
2		**n/b^2**			
i					
	●●●●●●●●●●●●●●●●				
h					

●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●

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

Level	# Instances	Instance size			
0	●	n			
1	● ● ● ●	**n/b**			
2		**n/b^2**			
i		**n/b^i**			
	●●●●●●●●●●●●●●●●				
h		**n/b^h**			

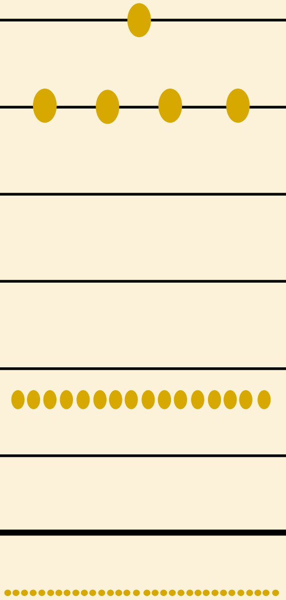
●●●●●●●●●●●●●●●● ●●●●●●●●●●●●●●●●

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

Level	# Instances	Instance size			
0		n			
1		**n/b**			
2		**n/b^2**			
i		**n/b^i**			
h		**n/b^h**			

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

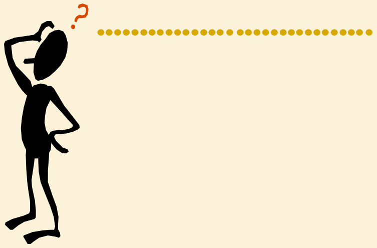
Level	# Instances	Instance size			
0		n			
1		n/b			
2		n/b^2			
i		n/b^i			
h		$n/b^h = 1$			



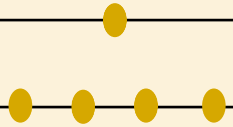
base case

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

Level	# Instances	Instance size			
0	●	n			
1	● ● ● ●	**n/b**			
2		**n/b^2**			
i		**n/b^i**			
	●●●●●●●●●●●●●●●●				
h		**n/b^h = 1**			

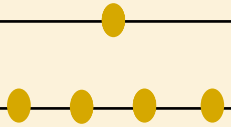


Evaluating: $T(n) = aT(n/b)+f(n)$

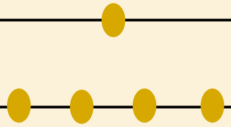
Level	# Instances	Instance size			
0		n			
1		n/b			
2		n/b^2			
i		n/b^i			
$h = {}^{\log n}/_{\log b}$		$n/b^h = 1$			

$$b^h = n$$
$$h \log b = \log n$$
$$h = {}^{\log n}/_{\log b}$$

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

Level	# Instances	Instance size	Work in stack frame		
0		n			
1		**n/b**			
2		**n/b^2**			
i		**n/b^i**			
$h = {}^{\log n}/_{\log b}$		**1**			

Evaluating: T(n) = aT(n/b)+f(n)

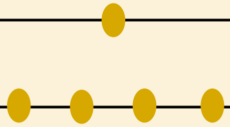
Level	# Instances	Instance size	Work in stack frame		
0		n	**f(n)**		
1		**n/b**	**f(n/b)**		
2		**n/b^2**	**$f(n/b^2)$**		
i		**n/b^i**	**$f(n/b^i)$**		
$h = {}^{\log n}/_{\log b}$		**1**	**T(1)**		

Evaluating: T(n) = aT(n/b)+f(n)

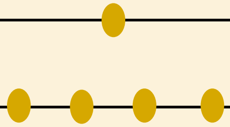
Level	# Instances	Instance size	Work in stack frame	# stack frames	
0	●	n	**f(n)**		
1	● ● ● ●	**n/b**	**f(n/b)**		
2		**n/b^2**	**f(n/b^2)**		
i		**n/b^i**	**f(n/b^i)**		
	●●●●●●●●●●●●●●●●				
$h = {}^{\log n}/_{\log b}$		**n/b^h**	**T(1)**		

●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●

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

Level	# Instances	Instance size	Work in stack frame	# stack frames	
0		n	$f(n)$	1	
1		n/b	$f(n/b)$	a	
2		n/b^2	$f(n/b^2)$	a^2	
i		n/b^i	$f(n/b^i)$	a^i	
$h = {}^{\log n}/_{\log b}$		n/b^h	$T(1)$	a^h	

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

Level	# Instances	Instance size	Work in stack frame	# stack frames	
0		n	$f(n)$	1	
1		n/b	$f(n/b)$	a	
2		n/b^2	$f(n/b^2)$	a^2	
i		n/b^i	$f(n/b^i)$	a^i	
$h = {}^{\log n}/_{\log b}$		n/b^h	$T(1)$	a^h	

Evaluating: T(n) = aT(n/b)+f(n)

Level	# Instances	Instance size	Work in stack frame	# stack frames	
0	●	n	**f(n)**	1	
1	● ● ● ●	**n/b**	**f(n/b)**	a	
2		**n/b²**	**f(n/b²)**	**a²**	
i		**n/bⁱ**	**f(n/bⁱ)**	**aⁱ**	
	● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ●				
h = $^{\log n}/_{\log b}$		**n/bʰ**	**T(1)**	**aʰ**	

$$a^h = a^{\log n/\log b} = n^{\log a/\log b}$$

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

Level	# Instances	Instance size	Work in stack frame	# stack frames	Work in Level
0		n	**f(n)**	1	
1	●	**n/b**	**f(n/b)**	a	
2	● ● ● ●	**n/b^2**	**$f(n/b^2)$**	**a^2**	
i		**n/b^i**	**$f(n/b^i)$**	**a^i**	
	●●●●●●●●●●●●●●●●				
$h = {}^{\log n}/_{\log b}$		**n/b^h**	**T(1)**		

●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●

$n^{\log a / \log b}$

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

Level	# Instances	Instance size	Work in stack frame	# stack frames	Work in Level
0	●	n	$f(n)$	1	$1 \cdot f(n)$
1	●●●●	n/b	$f(n/b)$	a	$a \cdot f(n/b)$
2		n/b^2	$f(n/b^2)$	a^2	$a^2 \cdot f(n/b^2)$
i		n/b^i	$f(n/b^i)$	a^i	$a^i \cdot f(n/b^i)$
	●●●●●●●●●●●●●●●●				
$h = {}^{\log n}/_{\log b}$		n/b^h	$T(1)$		

$n^{\log a/\log b}$ $\quad$ $n^{\log a/\log b} \cdot T(1)$

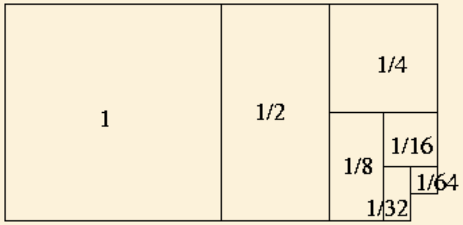
Total Work $T(n) = \sum_{i=0..h} a^i \cdot f(n/b^i)$

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

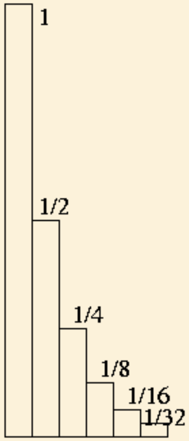
$$= \sum_{i=0..h} a^i \cdot f(n/b^i)$$

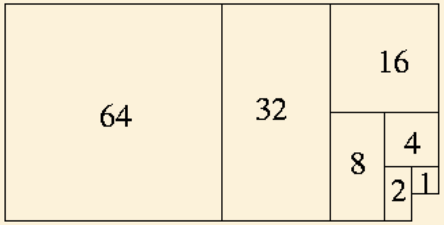
If a Geometric Sum

$$\sum_{i=0..n} x^i = \theta(\max(\text{first term, last term}))$$

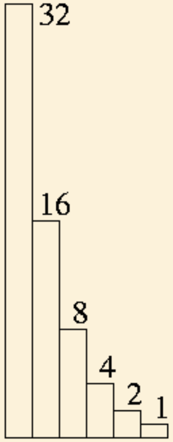


1+1/2+1/4+1/8+1/16+1/32+1/64+... = 2





1+2+4+8+16+32+64 = 2*64 - 1



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

Level		Instance size	Work in stack frame	# stack frames	Work in Level
0		n	**f(n)**	1	**$1 \cdot f(n)$**
1	●	**n/b**	**f(n/b)**	a	**$a \cdot f(n/b)$**
2	● ● ● ●	**n/b^2**	**$f(n/b^2)$**	**a^2**	**$a^2 \cdot f(n/b^2)$**
i		**n/b^i**	**$f(n/b^i)$**	**a^i**	**$a^i \cdot f(n/b^i)$**
	●●●●●●●●●●●●●●●●				
$h = {}^{\log n}/_{\log b}$		**n/b^h**	**T(1)**		

$n^{\log a/\log b}$ $n^{\log a/\log b} \cdot$ **T(1)**

Dominated by Top Level or Base Cases

End of Lecture 7

Master Theorem: Intuition

Suppose $T(n) = aT(n/b) + f(n),\ a \geq 1,\ b > 1$

Work at top level $= f(n)$

Work at bottom level = number of base cases = $n^{\log_b a} = n^{\log a / \log b}$

Running time = max(work at top, work at bottom) = $\max(f(n), n^{\log_b a})$

If they are equal, then all levels are important:

Running time = sum of work over all levels = $n^{\log_b a} \log n$

Theorem 4.1 (Master Theorem)

Suppose $T(n) = aT(n/b) + f(n),\ a \geq 1,\ b > 1$

1. IF $\exists \varepsilon > 0$ such that $f(n) \in O(n^{\log_b a - \varepsilon})$

   THEN $T(n) \in \theta(n^{\log_b a})$

   Dominated by base cases

2. IF $f(n) \in \theta(n^{\log_b a})$

   THEN $T(n) \in \theta(n^{\log_b a} \log n)$

   Work at each level is comparable:
   Sum work over all levels

3. IF $\exists \varepsilon > 0$ such that $f(n) \in \Omega(n^{\log_b a + \varepsilon})$

   Dominated by top level work

   AND

   $\exists c < 1, n_0 > 0$ such that $af(n/b) \leq cf(n)\ \forall n \geq n_0$

   Additional regularity condition

   THEN $T(n) \in \theta(f(n))$

Theorem 4.1 (Master Theorem)

Suppose $T(n) = aT(n/b) + f(n),\ a \geq 1,\ b > 1$

e.g., $T(n) = 4T(n/2) + f(n) \rightarrow \log_b a = \log_2 4 = 2$

1. IF $\exists \varepsilon > 0$ such that $f(n) \in O(n^{\log_b a - \varepsilon})$

   THEN $T(n) \in \theta(n^{\log_b a})$

$f(n) = n^2$? ✗

$f(n) = n^{1.97}$? ✓

e.g., $\varepsilon = 0.01$

2. IF $f(n) \in \theta(n^{\log_b a})$

   THEN $T(n) \in \theta(n^{\log_b a} \log n)$

3. IF $\exists \varepsilon > 0$ such that $f(n) \in \Omega(n^{\log_b a + \varepsilon})$

   AND

   $\exists c < 1, n_0 > 0$ such that $af(n/b) \leq cf(n)\ \forall n \geq n_0$

   THEN $T(n) \in \theta(f(n))$

Example 2: $T(n) = 4T(n/2) + 2^n$

$$\left.\begin{aligned} a &= 4 \\ b &= 2 \end{aligned}\right\} n^{\log_b a} = n^2$$

$f(n) = 2^n$

Thus $f(n) \in \Omega(n^{\log_b a+\varepsilon})$ (Case 3: dominated by top level)

Theorem 4.1 (Master Theorem)

Suppose $T(n) = aT(n/b) + f(n),\ a \ge 1,\ b > 1$

1. IF $\exists \varepsilon > 0$ such that $f(n) \in O(n^{\log_b a - \varepsilon})$

   THEN $T(n) \in \theta(n^{\log_b a})$

2. IF $f(n) \in \theta(n^{\log_b a})$

   THEN $T(n) \in \theta(n^{\log_b a} \log n)$

$$\left.\begin{array}{l} a = 4 \\ b = 2 \end{array}\right\} n^{\log_b a} = n^2$$

$f(n) = 2^n$

3. IF $\exists \varepsilon > 0$ such that $f(n) \in \Omega(n^{\log_b a + \varepsilon})$ ← Dominated by top level work

   AND

   $\exists c < 1, n_0 > 0$ such that $af(n/b) \le cf(n)\ \forall n \ge n_0$ But what about this?

   THEN $T(n) \in \theta(f(n))$

Example 2: $T(n) = 4T(n/2) + 2^n$

$$\left.\begin{array}{l} a = 4 \\ b = 2 \end{array}\right\} n^{\log_b a} = n^2$$

$f(n) = 2^n$

Thus $f(n) \in \Omega(n^{\log_b a+\varepsilon})$ (Case 3: dominated by top level)

Additional regularity condition:

$\exists c < 1, n_0 > 0$ such that $af(n/b) \le cf(n)\ \forall n \ge n_0$

Thus we require that $4 \cdot 2^{n/2} \le c2^n$

$\leftrightarrow c \ge 4 \cdot 2^{-n/2}$

Let $n_0 = 6 \rightarrow c \ge \dfrac{1}{2}$

$\rightarrow$ regularity condition holds for $n_0 = 6,\ c = 0.5$

Thus $T(n) = \theta(f(n)) = \theta(2^n)$

Example 3: $T(n) = 4T(n/2) + n\log_5 n$

$$\left.\begin{aligned} a &= 4 \\ b &= 2 \end{aligned}\right\} \; n^{\log_b a} = n^2$$

$f(n) = n\log_5 n$

Thus $f(n) \in O(n^{\log_b a - \varepsilon})$ (Case 1: dominated by base cases)

Thus T(n)=$\theta(n^{\log_b a})$ =θ(n^2)

Theorem 4.1 (Master Theorem)

Suppose $T(n) = aT(n/b) + f(n),\ a \geq 1,\ b > 1$

$$\left.\begin{aligned} a &= 4 \\ b &= 2 \end{aligned}\right\} \quad n^{\log_b a} = n^2$$

$$f(n) = n \log_5 n$$

1. IF $\exists \varepsilon > 0$ such that $f(n) \in O(n^{\log_b a - \varepsilon})$

   THEN $T(n) \in \theta(n^{\log_b a})$

Dominated by base cases

2. IF $f(n) \in \theta(n^{\log_b a})$

   THEN $T(n) \in \theta(n^{\log_b a} \log n)$

3. IF $\exists \varepsilon > 0$ such that $f(n) \in \Omega(n^{\log_b a + \varepsilon})$

   AND

   $\exists c < 1, n_0 > 0$ such that $af(n/b) \leq cf(n)\ \forall n \geq n_0$

   THEN $T(n) \in \theta(f(n))$

Example 4: $T(n) = 4T(n/2) + n^2$

$$\left.\begin{array}{l} a = 4 \\ b = 2 \end{array}\right\} n^{\log_b a} = n^2$$

$f(n) = n^2$

Thus $f(n) \in \theta(n^{\log_b a})$ (Case 2: all levels significant)

Thus $T(n) = \theta(n^{\log_b a} \log n) = \theta(\mathrm{n}^2 \log n)$

Theorem 4.1 (Master Theorem)

Suppose $T(n) = aT(n/b) + f(n),\ a \geq 1,\ b > 1$

1. IF $\exists \varepsilon > 0$ such that $f(n) \in O(n^{\log_b a - \varepsilon})$

   THEN $T(n) \in \theta(n^{\log_b a})$

2. IF $f(n) \in \theta(n^{\log_b a})$

   THEN $T(n) \in \theta(n^{\log_b a} \log n)$

3. IF $\exists \varepsilon > 0$ such that $f(n) \in \Omega(n^{\log_b a + \varepsilon})$

   AND

   $\exists c < 1, n_0 > 0$ such that $af(n/b) \leq cf(n)\ \forall n \geq n_0$

   THEN $T(n) \in \theta(f(n))$

$$\left.\begin{array}{l} a = 4 \\ b = 2 \end{array}\right\} n^{\log_b a} = n^2$$

$f(n) = n^2$

Work at each level is comparable:

Sum work over all levels

Master Theorem Case 3: When the Regularity Condition Fails

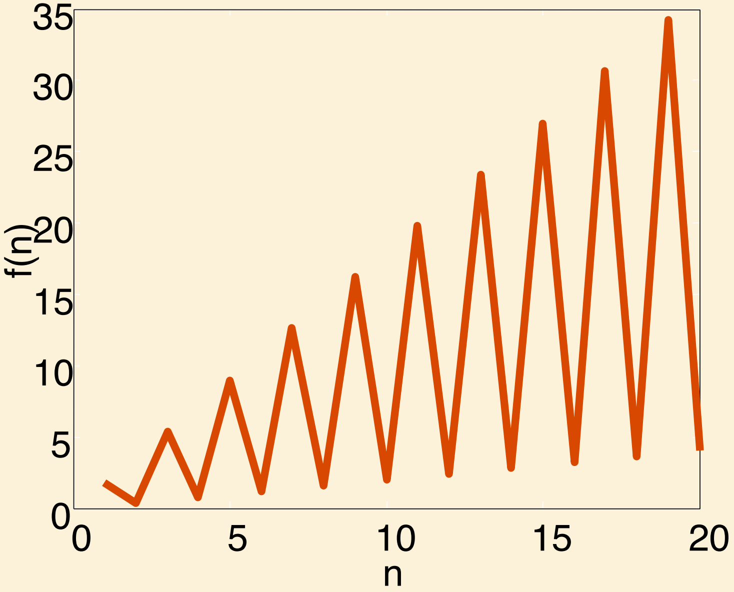
e.g. $T(n) = T(n/2) + n(1 - .8\cos \pi n)$

Here $\log_b a = \log_2 1 = 0 \rightarrow n^{\log_b a} = n^0 = 1$

and $f(n) = n(1 - .8\cos \pi n) \geq .2n \in \Omega(n)$

Thus $f(n) \in \Omega\left(n^{\log_b a + \varepsilon}\right)$, suggesting that Case 3 applies.

But does the regularity condition hold?



Master Theorem Case 3: When the Regularity Condition Fails

e.g. $T(n) = T(n/2) + n(1 - .8\cos \pi n)$ Does the regularity condition hold?

We require that $af(n/b) \le cf(n)$ for some constant $c < 1, \forall n \ge n_0$.

$\leftrightarrow f(n/2) \le cf(n)$

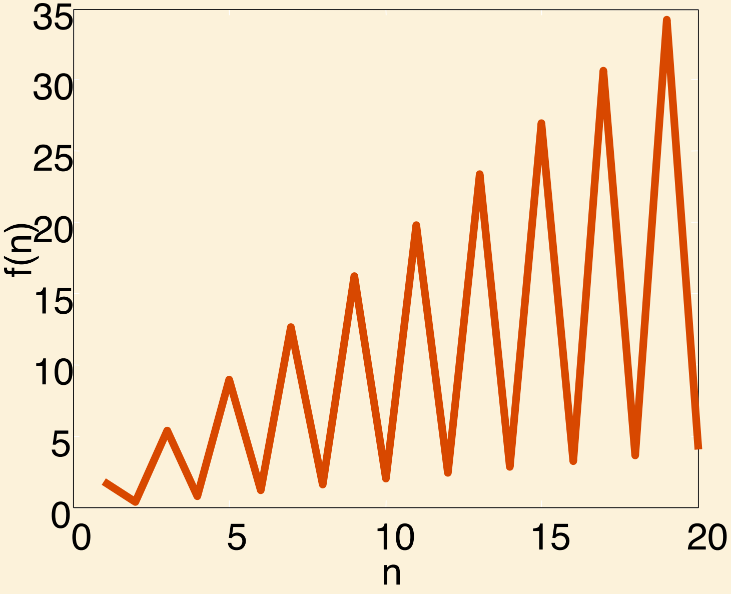
$\leftrightarrow (n/2)(1 - .8\cos(\pi n/2)) \le cn(1 - .8\cos \pi n)$

$\leftrightarrow (1/2)(1 - .8\cos(\pi n/2)) \le c(1 - .8\cos \pi n)$

Given arbitrary n_0, select an $n \ge n_0$ such that n is even and $n/2$ is odd

Then we require that $(1/2)(1 + .8) \le c(1 - .8)$

$\leftrightarrow .9 \le .2c \leftrightarrow c \ge 4.5$



Thus the regularity condition does not hold.

Master Theorem Case 3: When the Regularity Condition Fails

So what is the solution?

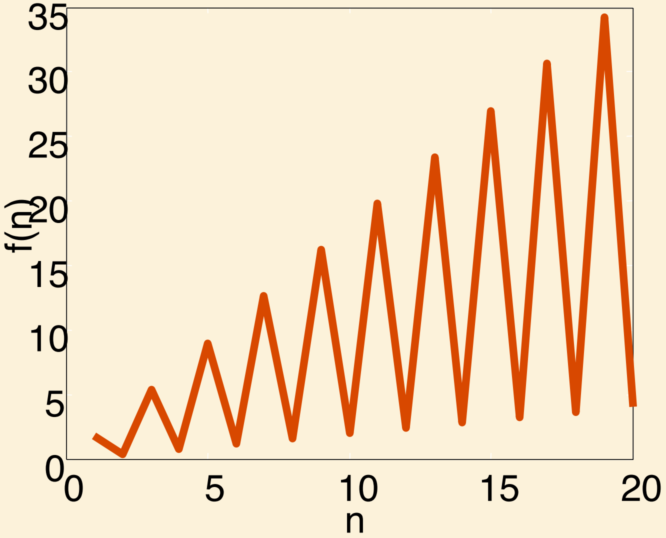
$$T(n) = T(n/2) + n(1 - .8\cos \pi n)$$

Note that $f(n) = n(1 - .8\cos \pi n) \in \Theta(n)$

So in this case,

$T(n) \in \Theta(f(n)) = \Theta(n)$, despite failure of the reg. condition.

Question: Are there failures of the reg. condition that result in $T(n) \notin \Theta(f(n))$?



Master Theorem Case 3: When the Regularity Condition Fails

Question : Are there failures of the reg. condition that result in $T(n) \notin \Theta(f(n))$?

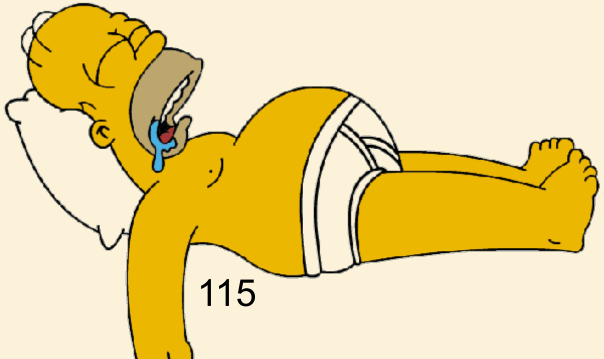
Consider $T(n) = 2T(n/2) + f(n)$

where $f(n) = \begin{cases} n^3 \text{ when } \lceil \log_2 n \rceil \text{ is even} \\ n^2 \text{ when } \lceil \log_2 n \rceil \text{ is odd} \end{cases}$

Think about this puzzle and ask yourself:

1. Is the first condition of Case 3 satisfied?
2. Is the second (regularity) condition of Case 3 satisfied?
3. Is $T(n) \in \Theta(f(n))$?

Let's sleep on it.



Central Algorithmic Techniques

Recursion

Different Representations of Recursive Algorithms

Views	Pros
Code	- Implement on Computer
Stack of Stack Frames	- Run on Computer
Tree of Stack Frames	- View entire computation
Friends & Strong Induction	- Worry about one step at a time.

Code
Representation of an Algorithm

```
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
  e $10^n$ + (MULT(a+b, c+d) – e - f) $10^{n/2}$ + f
```

Pros and Cons?

Code
Representation of an Algorithm

Pros:

- Runs on computers
- Precise and succinct

Cons:

- I am not a computer
- I need a higher level of intuition.
- Prone to bugs
- Language dependent

Different Representations of Recursive Algorithms

Views	Pros
Code	- Implement on Computer
Stack of Stack Frames	- Run on Computer
Tree of Stack Frames	- View entire computation
Friends & Strong Induction	- Worry about one step at a time.

Stack of Stack Frames

```
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
  e $10^n$ + (MULT(a+b, c+d) – e - f) $10^{n/2}$ + f
```

X = 2133
Y = 2312
ac =
bd =
(a+b)(c+d) =
XY =

Stack Frame: A particular execution of one routine on one particular input instance.

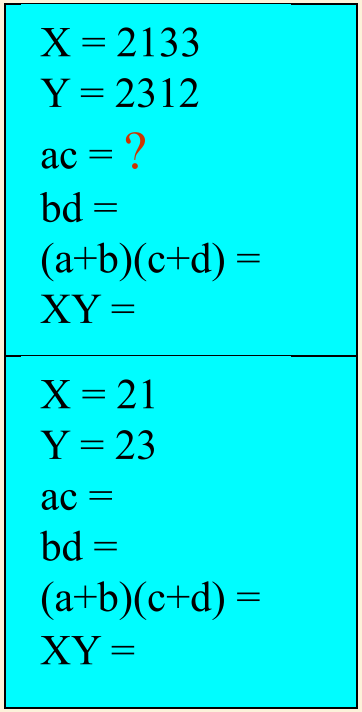
Stack of Stack Frames

```
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
  e $10^n$ + (MULT(a+b, c+d) – e - f) $10^{n/2}$ + f
```

X = 2133
Y = 2312
ac = ?
bd =
(a+b)(c+d) =
XY =

Stack of Stack Frames

```
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
  e $10^n$ + (MULT(a+b, c+d) – e - f) $10^{n/2}$ + f
```

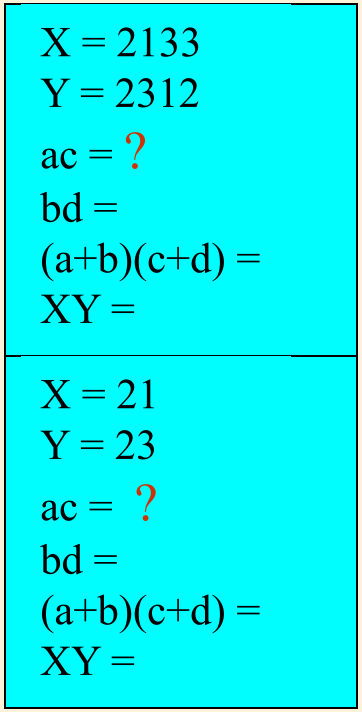


X = 2133
Y = 2312
ac = ?
bd =
(a+b)(c+d) =
XY =

X = 21
Y = 23
ac =
bd =
(a+b)(c+d) =
XY =

Stack of Stack Frames

```
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
  e $10^n$ + (MULT(a+b, c+d) – e - f) $10^{n/2}$ + f
```

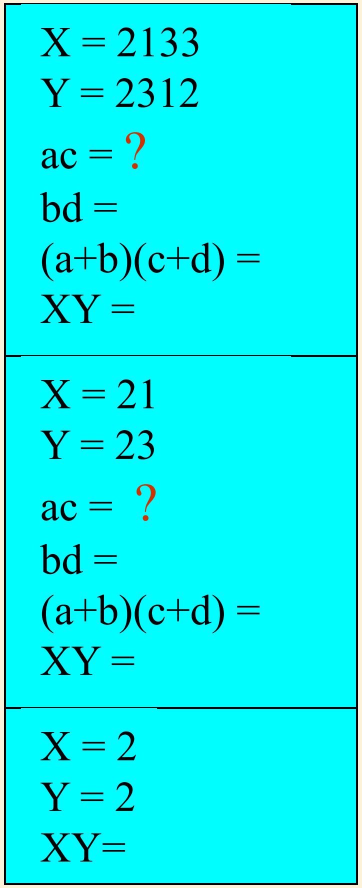


X = 2133
Y = 2312
ac = ?
bd =
(a+b)(c+d) =
XY =

X = 21
Y = 23
ac = ?
bd =
(a+b)(c+d) =
XY =

Stack of Stack Frames

```
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
  e $10^n$ + (MULT(a+b, c+d) – e - f) $10^{n/2}$ + f
```



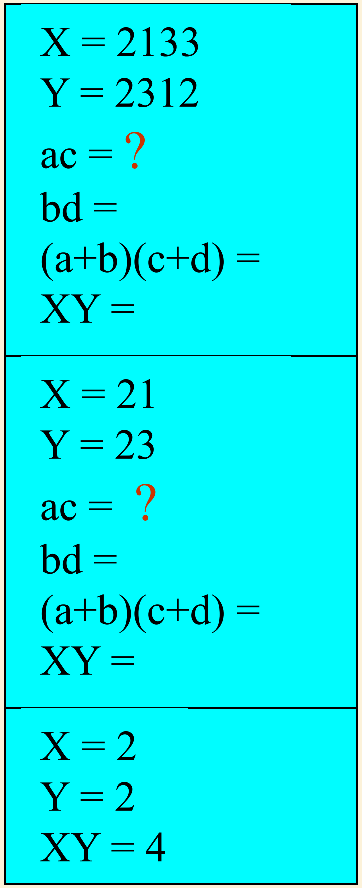
X = 2133
Y = 2312
ac = ?
bd =
(a+b)(c+d) =
XY =

X = 21
Y = 23
ac = ?
bd =
(a+b)(c+d) =
XY =

X = 2
Y = 2
XY=

Stack of Stack Frames

```
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
  $e\ 10^{n} + (\mathrm{MULT}(a+b,\ c+d) - e - f)\ 10^{n/2} + f$
```



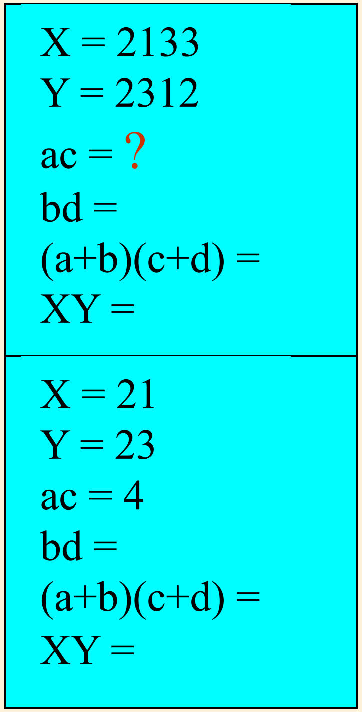
X = 2133
Y = 2312
ac = ?
bd =
(a+b)(c+d) =
XY =

X = 21
Y = 23
ac = ?
bd =
(a+b)(c+d) =
XY =

X = 2
Y = 2
XY = 4

Stack of Stack Frames

```
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
  e 10^n + (MULT(a+b, c+d) – e - f) 10^{n/2} + f
```

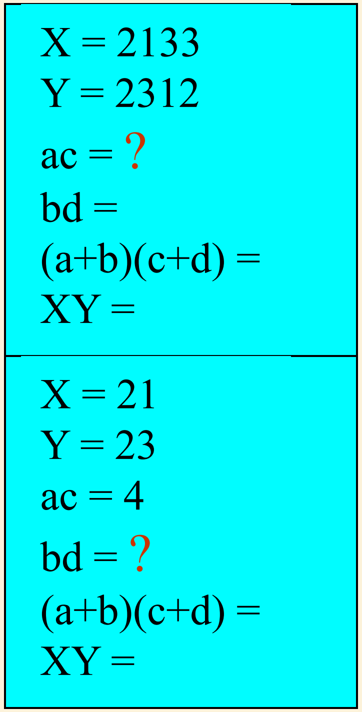


X = 2133
Y = 2312
ac = ?
bd =
(a+b)(c+d) =
XY =

X = 21
Y = 23
ac = 4
bd =
(a+b)(c+d) =
XY =

Stack of Stack Frames

```
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
  $e\ 10^n + (\text{MULT}(a+b, c+d) - e - f)\ 10^{n/2} + f$
```

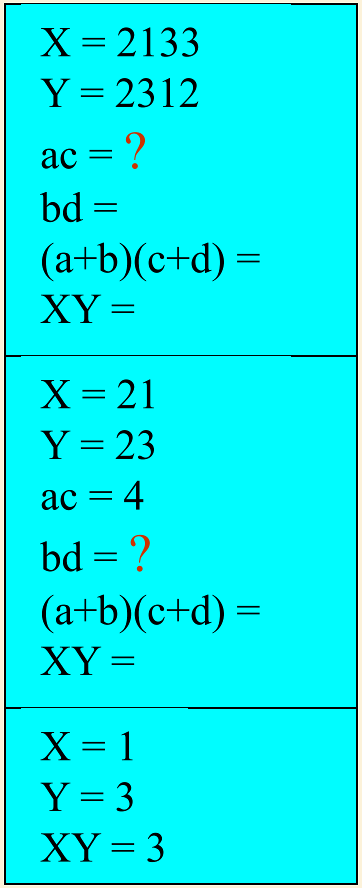


X = 2133
Y = 2312
ac = ?
bd =
(a+b)(c+d) =
XY =

X = 21
Y = 23
ac = 4
bd = ?
(a+b)(c+d) =
XY =

Stack of Stack Frames

```
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
  e $10^n$ + (MULT(a+b, c+d) – e - f) $10^{n/2}$ + f
```



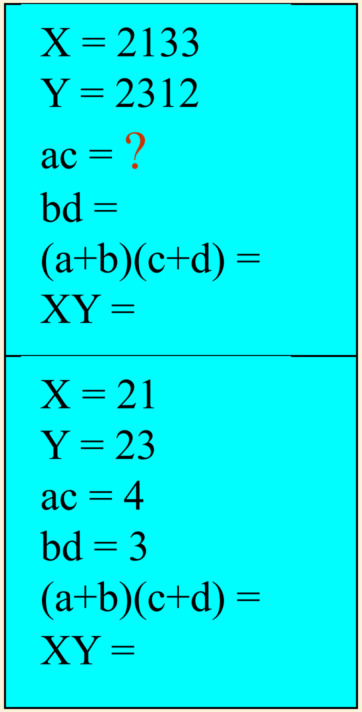
X = 2133
Y = 2312
ac = ?
bd =
(a+b)(c+d) =
XY =

X = 21
Y = 23
ac = 4
bd = ?
(a+b)(c+d) =
XY =

X = 1
Y = 3
XY = 3

Stack of Stack Frames

```
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
  $e\ 10^n + (MULT(a+b, c+d) – e - f)\ 10^{n/2} + f$
```

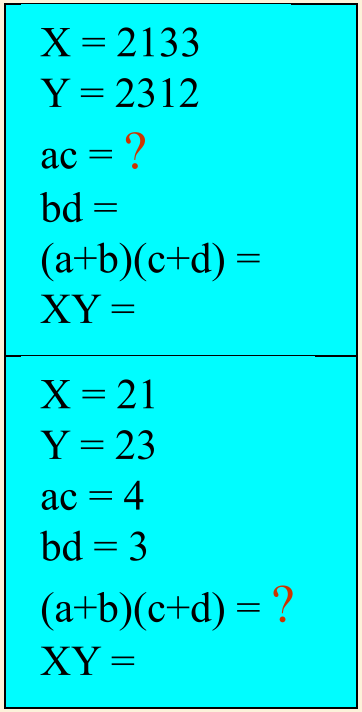


X = 2133
Y = 2312
ac = ?
bd =
(a+b)(c+d) =
XY =

X = 21
Y = 23
ac = 4
bd = 3
(a+b)(c+d) =
XY =

Stack of Stack Frames

```
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
  e $10^n$ + (MULT(a+b, c+d) – e - f) $10^{n/2}$ + f
```

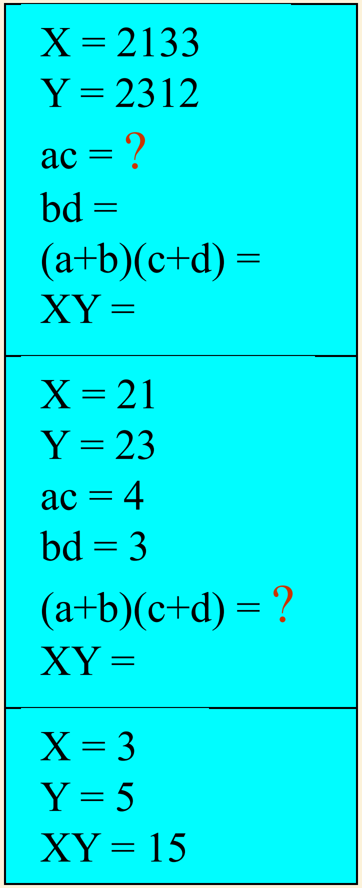


X = 2133
Y = 2312
ac = ?
bd =
(a+b)(c+d) =
XY =

X = 21
Y = 23
ac = 4
bd = 3
(a+b)(c+d) = ?
XY =

Stack of Stack Frames

```
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
  $e\ 10^n + (\text{MULT}(a+b, c+d) - e - f)\ 10^{n/2} + f$
```



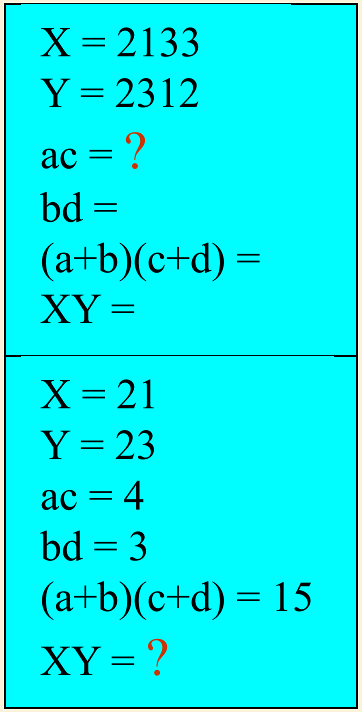
X = 2133
Y = 2312
ac = ?
bd =
(a+b)(c+d) =
XY =

X = 21
Y = 23
ac = 4
bd = 3
(a+b)(c+d) = ?
XY =

X = 3
Y = 5
XY = 15

Stack of Stack Frames

```
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
  e $10^n$ + (MULT(a+b, c+d) – e - f) $10^{n/2}$ + f
```

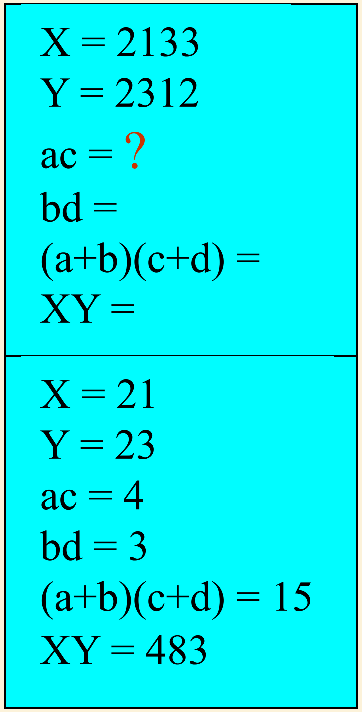


X = 2133
Y = 2312
ac = ?
bd =
(a+b)(c+d) =
XY =

X = 21
Y = 23
ac = 4
bd = 3
(a+b)(c+d) = 15
XY = ?

Stack of Stack Frames

```
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
  e $10^n$ + (MULT(a+b, c+d) – e - f) $10^{n/2}$ + f
```



X = 2133
Y = 2312
ac = ?
bd =
(a+b)(c+d) =
XY =

X = 21
Y = 23
ac = 4
bd = 3
(a+b)(c+d) = 15
XY = 483

Stack of Stack Frames

```
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
  $e\ 10^n + (\text{MULT}(a+b,\ c+d) - e - f)\ 10^{n/2} + f$
```

X = 2133
Y = 2312
ac = 483
bd =
(a+b)(c+d) =
XY =

Stack of Stack Frames

```
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
  $e\ 10^n + (\text{MULT}(a+b, c+d) - e - f)\ 10^{n/2} + f$
```

X = 2133

Y = 2312

ac = 483

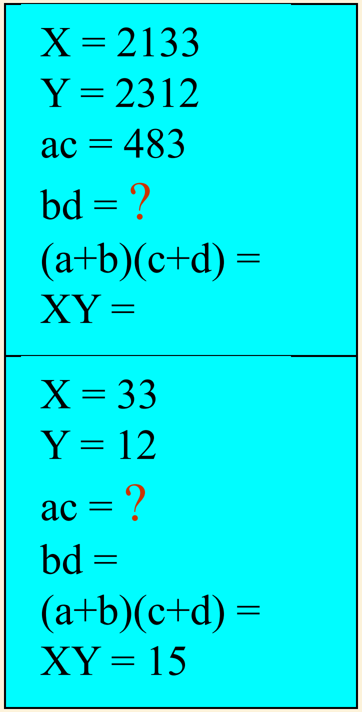
bd = ?

(a+b)(c+d) =

XY =

Stack of Stack Frames

```
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
  e 10^n + (MULT(a+b, c+d) – e - f) 10^{n/2} + f
```

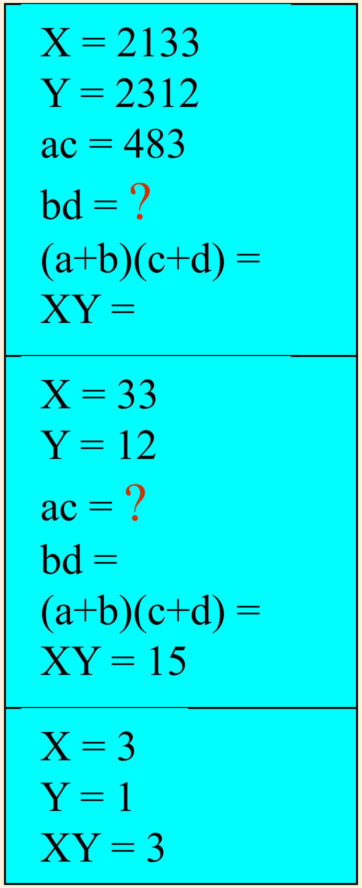


X = 2133
Y = 2312
ac = 483
bd = ?
(a+b)(c+d) =
XY =

X = 33
Y = 12
ac = ?
bd =
(a+b)(c+d) =
XY = 15

Stack of Stack Frames

```
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
  $e\ 10^n + (\text{MULT}(a+b, c+d) - e - f)\ 10^{n/2} + f$
```



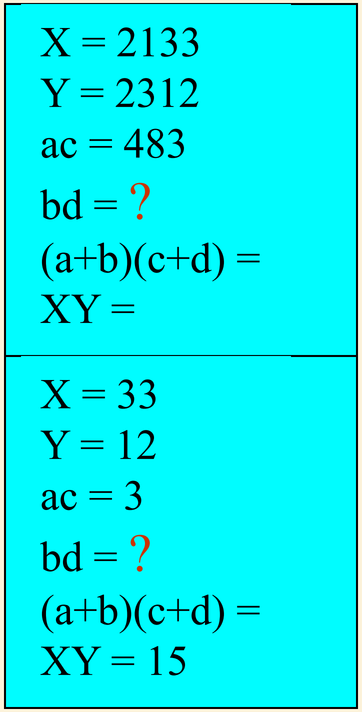
X = 2133
Y = 2312
ac = 483
bd = ?
(a+b)(c+d) =
XY =

X = 33
Y = 12
ac = ?
bd =
(a+b)(c+d) =
XY = 15

X = 3
Y = 1
XY = 3

Stack of Stack Frames

```
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
  $e\ 10^n + (\text{MULT}(a+b,\ c+d) - e - f)\ 10^{n/2} + f$
```

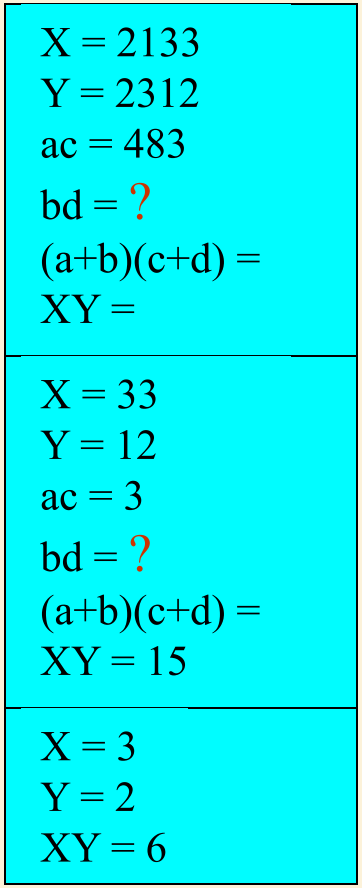


X = 2133
Y = 2312
ac = 483
bd = ?
(a+b)(c+d) =
XY =

X = 33
Y = 12
ac = 3
bd = ?
(a+b)(c+d) =
XY = 15

Stack of Stack Frames

```
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
  $e\ 10^n + (\text{MULT}(a+b, c+d) - e - f)\ 10^{n/2} + f$
```



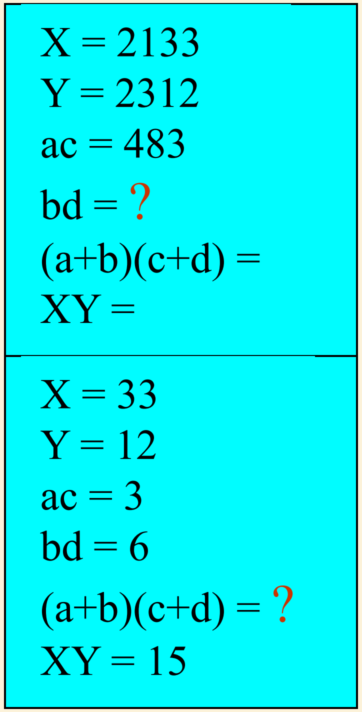
X = 2133
Y = 2312
ac = 483
bd = ?
(a+b)(c+d) =
XY =

X = 33
Y = 12
ac = 3
bd = ?
(a+b)(c+d) =
XY = 15

X = 3
Y = 2
XY = 6

Stack of Stack Frames

```
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
  e $10^n$ + (MULT(a+b, c+d) – e - f) $10^{n/2}$ + f
```

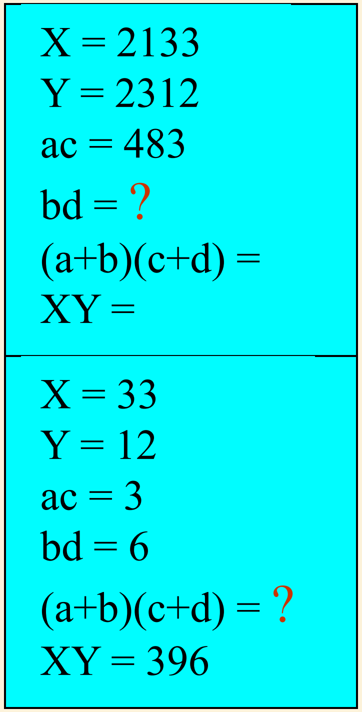


X = 2133
Y = 2312
ac = 483
bd = ?
(a+b)(c+d) =
XY =

X = 33
Y = 12
ac = 3
bd = 6
(a+b)(c+d) = ?
XY = 15

Stack of Stack Frames

```
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
  e $10^n$ + (MULT(a+b, c+d) – e - f) $10^{n/2}$ + f
```



X = 2133
Y = 2312
ac = 483
bd = ?
(a+b)(c+d) =
XY =

X = 33
Y = 12
ac = 3
bd = 6
(a+b)(c+d) = ?
XY = 396

An so on ….

Stack of Stack Frames Representation of an Algorithm

Pros:

- Traces what actually occurs in the computer
- Concrete.

Cons:

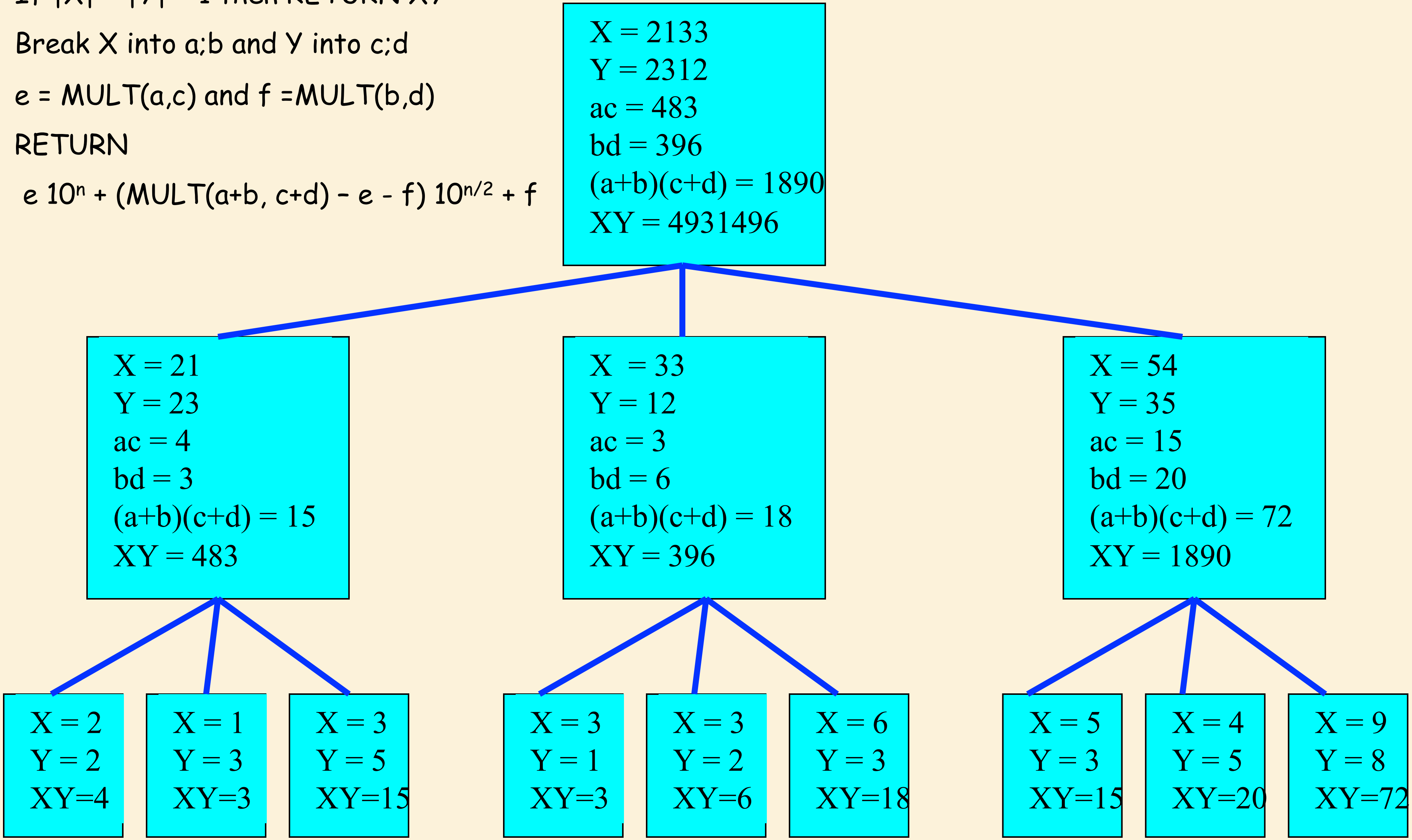
- Described in words it is impossible to follow
- Does not explain why it works.
- Demonstrates for only one of many inputs.

Different Representations of Recursive Algorithms

Views	Pros
Code	- Implement on Computer
Stack of Stack Frames	- Run on Computer
Tree of Stack Frames	- View entire computation
Friends & Strong Induction	- Worry about one step at a time.

Tree of Stack Frames

```
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
  e 10^n + (MULT(a+b, c+d) – e - f) 10^{n/2} + f
```



X = 2133
Y = 2312
ac = 483
bd = 396
(a+b)(c+d) = 1890
XY = 4931496

X = 21
Y = 23
ac = 4
bd = 3
(a+b)(c+d) = 15
XY = 483

X = 33
Y = 12
ac = 3
bd = 6
(a+b)(c+d) = 18
XY = 396

X = 54
Y = 35
ac = 15
bd = 20
(a+b)(c+d) = 72
XY = 1890

X = 2
Y = 2
XY=4

X = 1
Y = 3
XY=3

X = 3
Y = 5
XY=15

X = 3
Y = 1
XY=3

X = 3
Y = 2
XY=6

X = 6
Y = 3
XY=18

X = 5
Y = 3
XY=15

X = 4
Y = 5
XY=20

X = 9
Y = 8
XY=72

Stack of Stack Frames Representation of an Algorithm

Pros:

- View the entire computation.
- Good for computing the running time.

Cons:

- Must describe entire tree.
  - For each stack frame
    - input instance
    - computation
    - solution returned

Different Representations of Recursive Algorithms

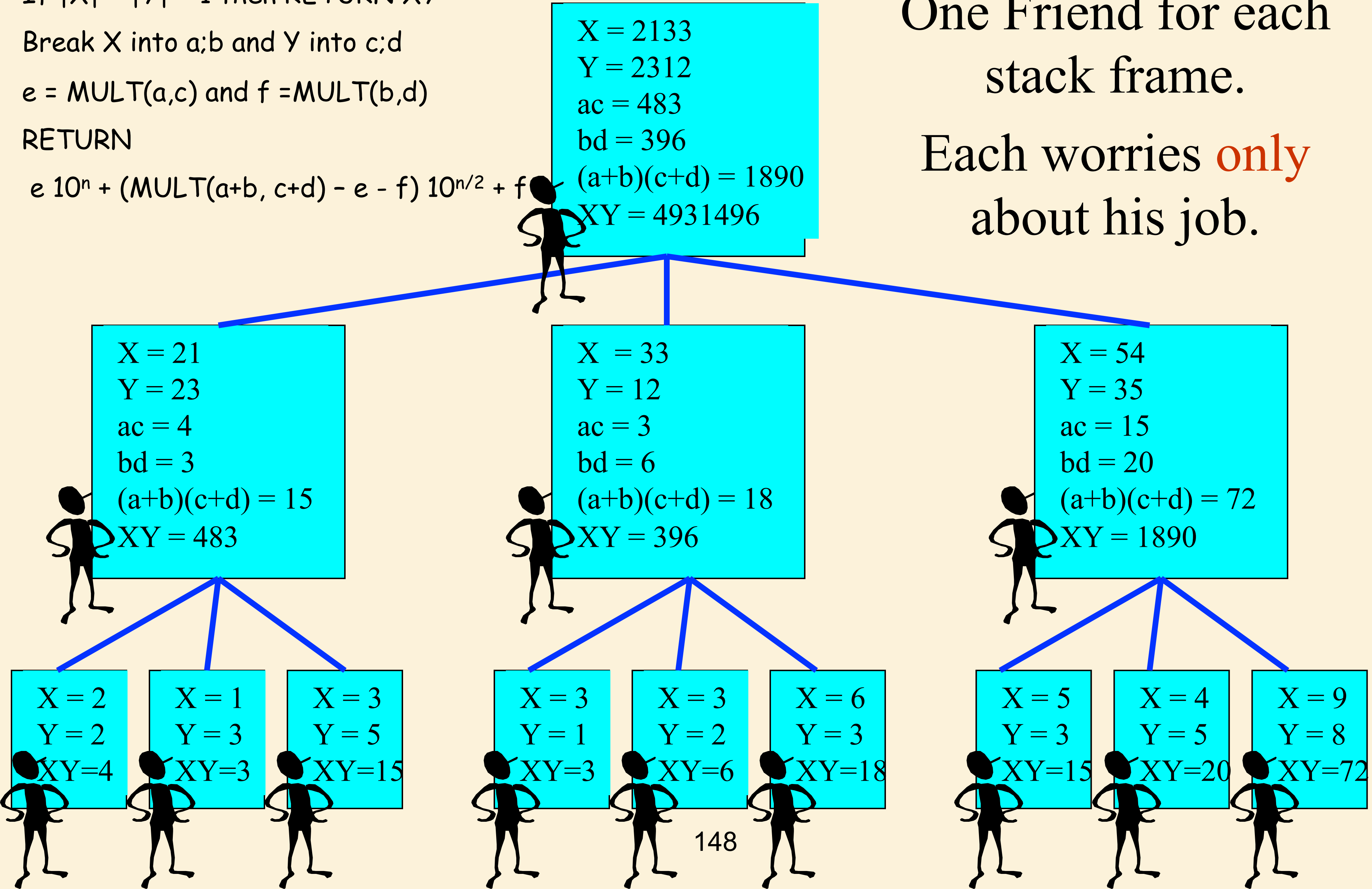
Views	Pros
Code	- Implement on Computer
Stack of Stack Frames	- Run on Computer
Tree of Stack Frames	- View entire computation
Friends & Strong Induction	- Worry about one step at a time.

Friends & Strong Induction

```
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
  $e\ 10^n + (\text{MULT}(a+b,\ c+d) - e - f)\ 10^{n/2} + f$
```

One Friend for each stack frame.

Each worries only about his job.



X = 2133
Y = 2312
ac = 483
bd = 396
(a+b)(c+d) = 1890
XY = 4931496

X = 21
Y = 23
ac = 4
bd = 3
(a+b)(c+d) = 15
XY = 483

X = 33
Y = 12
ac = 3
bd = 6
(a+b)(c+d) = 18
XY = 396

X = 54
Y = 35
ac = 15
bd = 20
(a+b)(c+d) = 72
XY = 1890

X = 2
Y = 2
XY=4

X = 1
Y = 3
XY=3

X = 3
Y = 5
XY=15

X = 3
Y = 1
XY=3

X = 3
Y = 2
XY=6

X = 6
Y = 3
XY=18

X = 5
Y = 3
XY=15

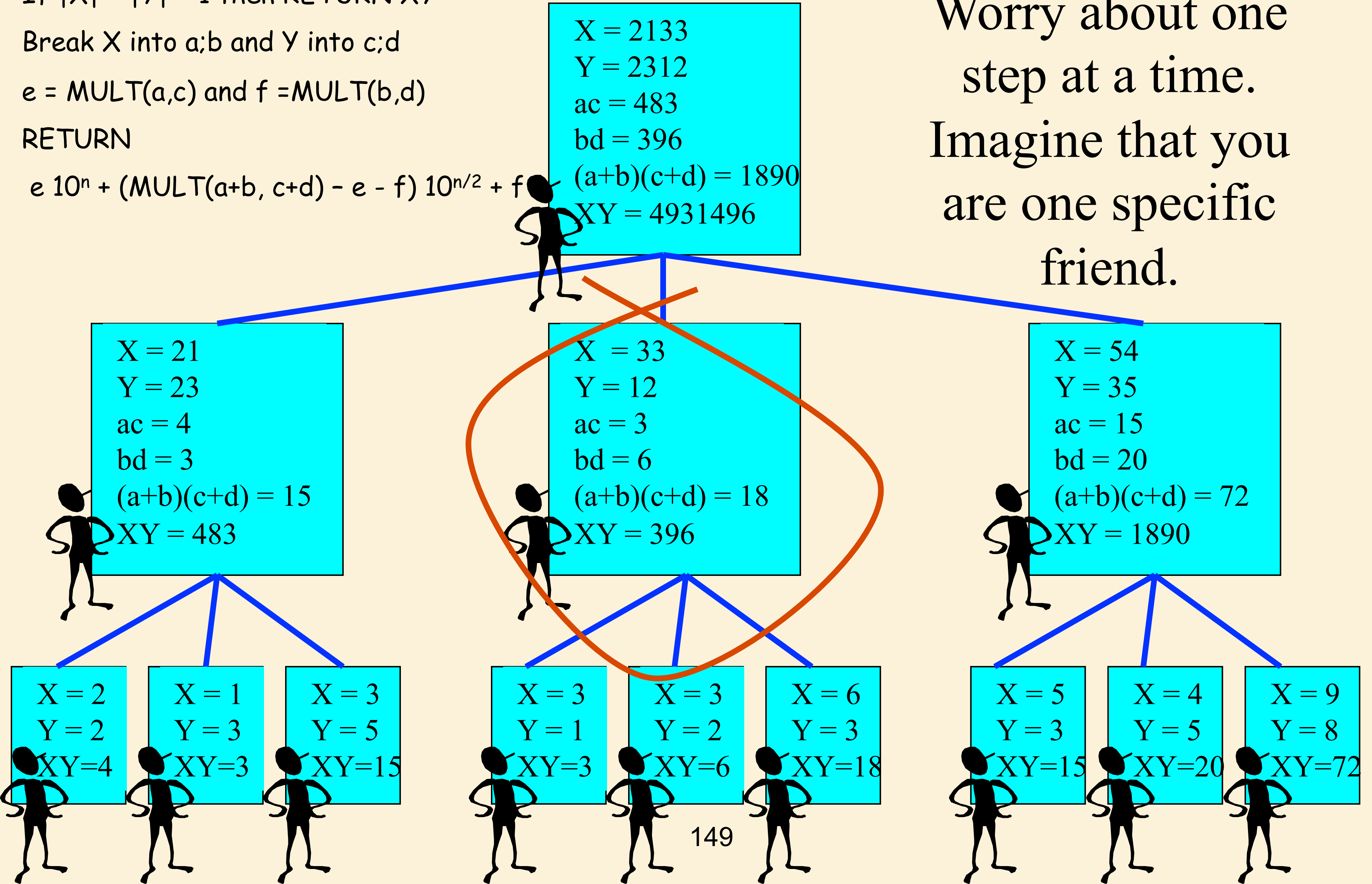
X = 4
Y = 5
XY=20

X = 9
Y = 8
XY=72

Friends & Strong Induction

```
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
  e $10^n$ + (MULT(a+b, c+d) – e - f) $10^{n/2}$ + f
```

Worry about one step at a time. Imagine that you are one specific friend.



X = 2133
Y = 2312
ac = 483
bd = 396
(a+b)(c+d) = 1890
XY = 4931496

X = 21
Y = 23
ac = 4
bd = 3
(a+b)(c+d) = 15
XY = 483

X = 33
Y = 12
ac = 3
bd = 6
(a+b)(c+d) = 18
XY = 396

X = 54
Y = 35
ac = 15
bd = 20
(a+b)(c+d) = 72
XY = 1890

X = 2
Y = 2
XY=4

X = 1
Y = 3
XY=3

X = 3
Y = 5
XY=15

X = 3
Y = 1
XY=3

X = 3
Y = 2
XY=6

X = 6
Y = 3
XY=18

X = 5
Y = 3
XY=15

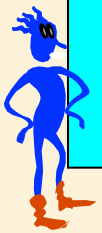
X = 4
Y = 5
XY=20

X = 9
Y = 8
XY=72

Friends & Strong Induction

```
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
  $e\ 10^n$ + (MULT(a+b, c+d) – $e$ - $f$) $10^{n/2}$ + $f$
```

- Consider your input instance



X = 33

Y = 12

ac =

bd =

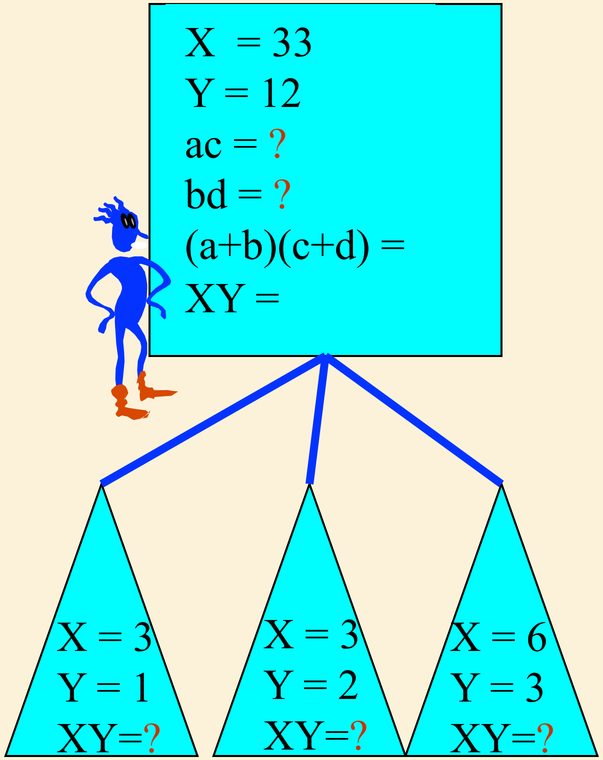
(a+b)(c+d) =

XY =

Friends & Strong Induction

```
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
  e 10^n + (MULT(a+b, c+d) – e - f) 10^{n/2} + f
```

- Consider your input instance
- Allocate work
  - Construct one or more subinstances



X = 33
Y = 12
ac = ?
bd = ?
(a+b)(c+d) =
XY =

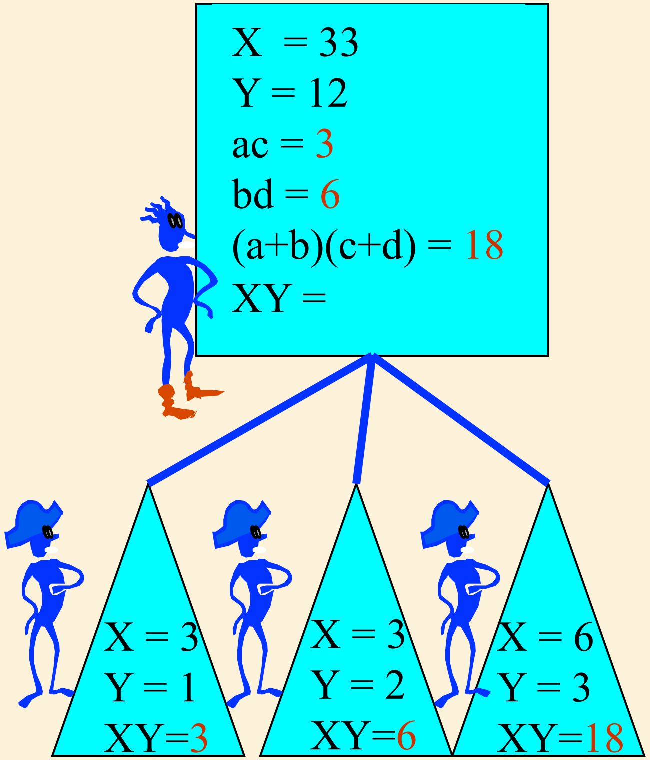
X = 3
Y = 1
XY=?

X = 3
Y = 2
XY=?

X = 6
Y = 3
XY=?

Friends & Strong Induction

```
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
  e 10^n + (MULT(a+b, c+d) – e - f) 10^{n/2} + f
```

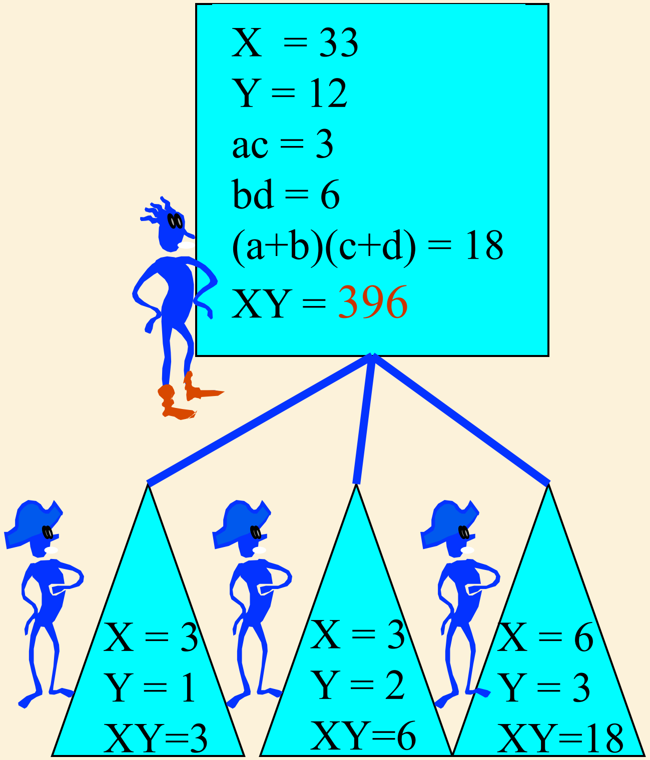


- Consider your input instance
- Allocate work
  - Construct one or more subinstances
  - Assume by magic your friends give you the answer for these.

Friends & Strong Induction

```
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
  $e\ 10^{n} + (MULT(a+b, c+d) – e - f)\ 10^{n/2} + f$
```

- Consider your input instance
- Allocate work
  - Construct one or more subinstances
  - Assume by magic your friends give you the answer for these.
- Use this help to solve your own instance.



X = 33
Y = 12
ac = 3
bd = 6
(a+b)(c+d) = 18
XY = 396

X = 3
Y = 1
XY=3

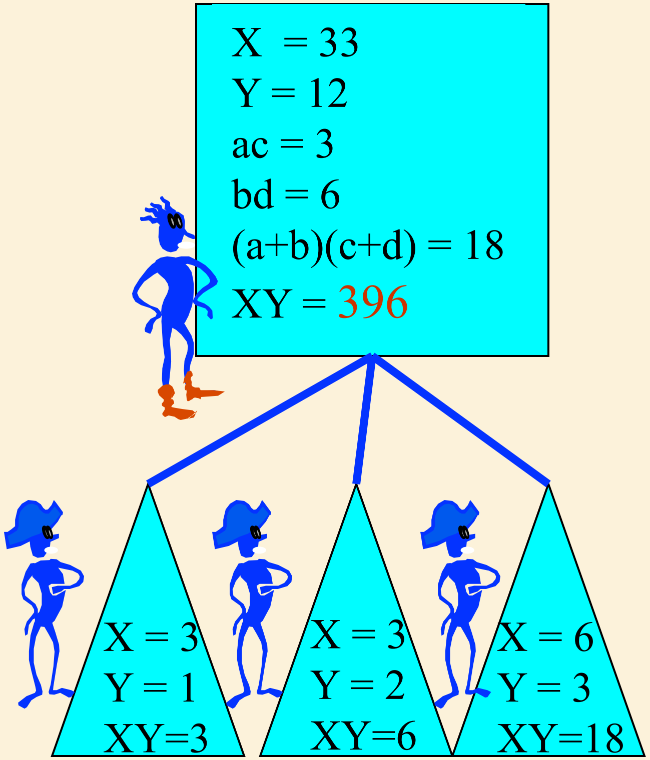
X = 3
Y = 2
XY=6

X = 6
Y = 3
XY=18

Friends & Strong Induction

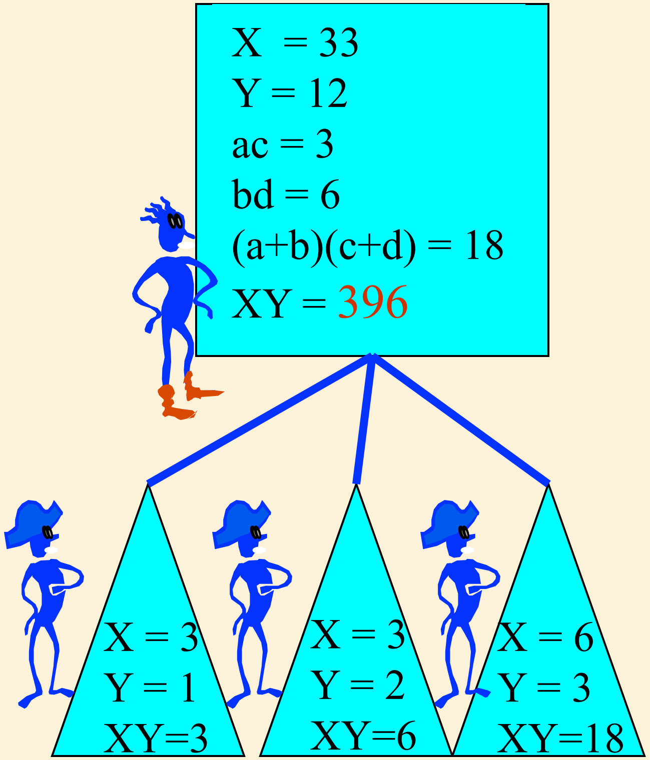
```
MULT(X,Y):
 If |X| = |Y| = 1 then RETURN XY
 Break X into a;b and Y into c;d
 e = MULT(a,c) and f =MULT(b,d)
 RETURN
  e 10^n + (MULT(a+b, c+d) – e - f) 10^{n/2} + f
```

- Consider your input instance
- Allocate work
    - Construct one or more subinstances
    - Assume by magic your friends give you the answer for these.
- Use this help to solve your own instance.
- Do not worry about anything else, e.g.,
    - Who your boss is.
    - How your friends solve their instance.



Friends & Strong Induction

```
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
  e $10^n$ + (MULT(a+b, c+d) – e - f) $10^{n/2}$ + f
```



X = 33
Y = 12
ac = 3
bd = 6
(a+b)(c+d) = 18
XY = 396

X = 3
Y = 1
XY=3

X = 3
Y = 2
XY=6

X = 6
Y = 3
XY=18

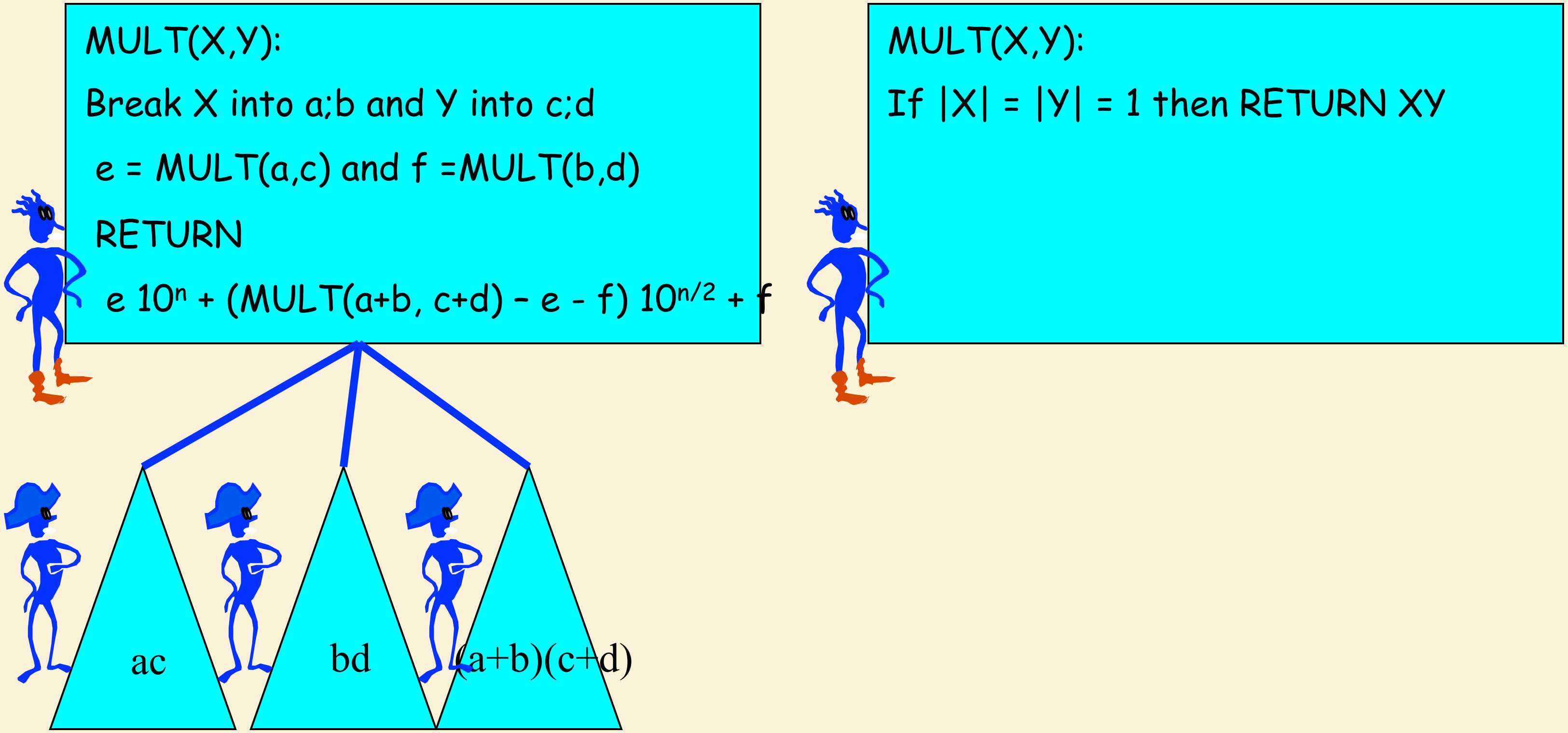
This technique is often referred to as

Divide and Conquer

Friends & Strong Induction

```
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
   e 10^n + (MULT(a+b, c+d) – e - f) 10^{n/2} + f
```

Consider generic instances.



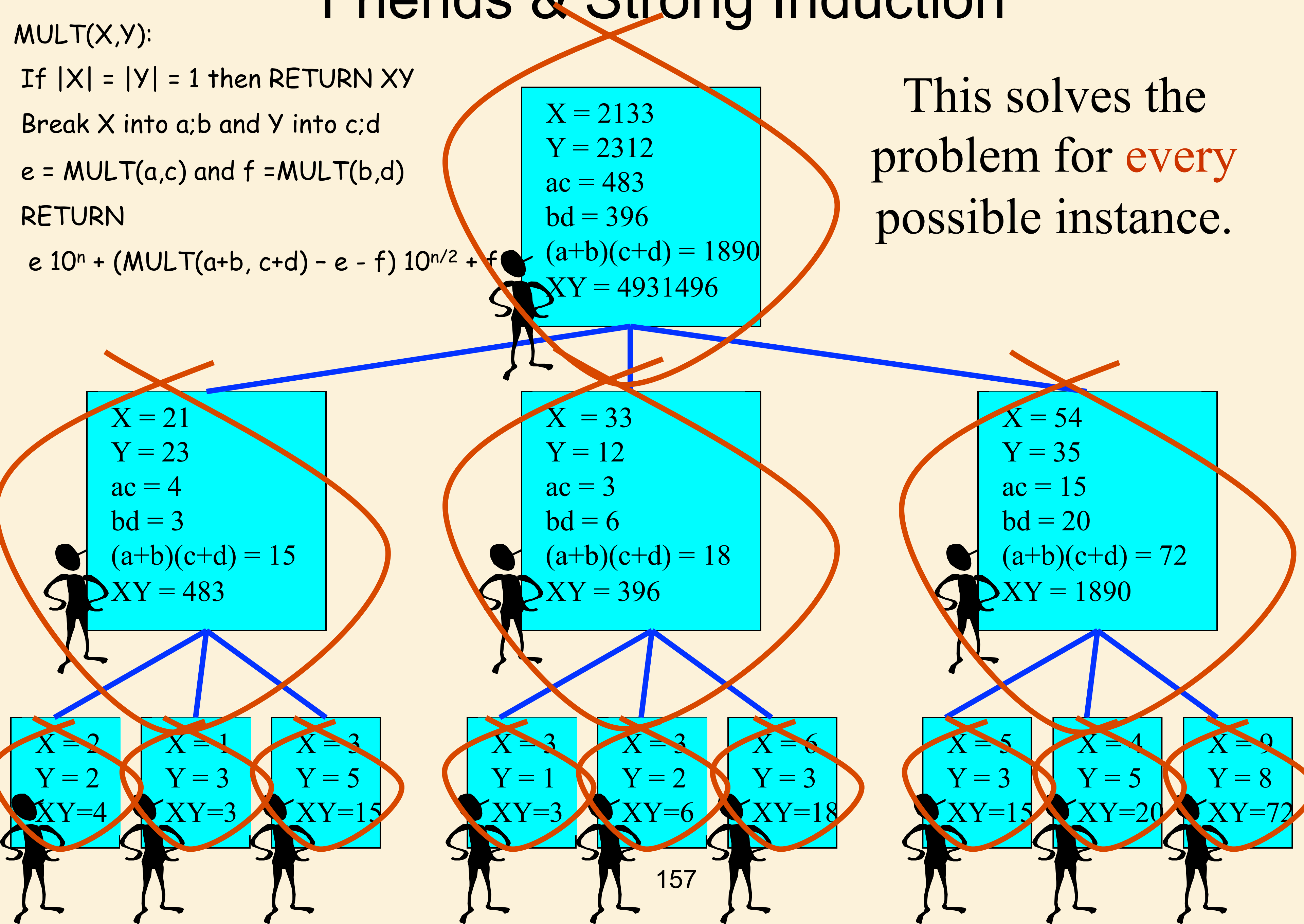
```
MULT(X,Y):
Break X into a;b and Y into c;d
 e = MULT(a,c) and f =MULT(b,d)
 RETURN
  e 10^n + (MULT(a+b, c+d) – e - f) 10^{n/2} + f
```

```
MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
```

Friends & Strong Induction

```
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
  e $10^n$ + (MULT(a+b, c+d) – e - f) $10^{n/2}$ + f
```

This solves the problem for every possible instance.



X = 2133
Y = 2312
ac = 483
bd = 396
(a+b)(c+d) = 1890
XY = 4931496

X = 21
Y = 23
ac = 4
bd = 3
(a+b)(c+d) = 15
XY = 483

X = 33
Y = 12
ac = 3
bd = 6
(a+b)(c+d) = 18
XY = 396

X = 54
Y = 35
ac = 15
bd = 20
(a+b)(c+d) = 72
XY = 1890

X = 2
Y = 2
XY=4

X = 1
Y = 3
XY=3

X = 3
Y = 5
XY=15

X = 3
Y = 1
XY=3

X = 3
Y = 2
XY=6

X = 6
Y = 3
XY=18

X = 5
Y = 3
XY=15

X = 4
Y = 5
XY=20

X = 9
Y = 8
XY=72

Friends & Strong Induction

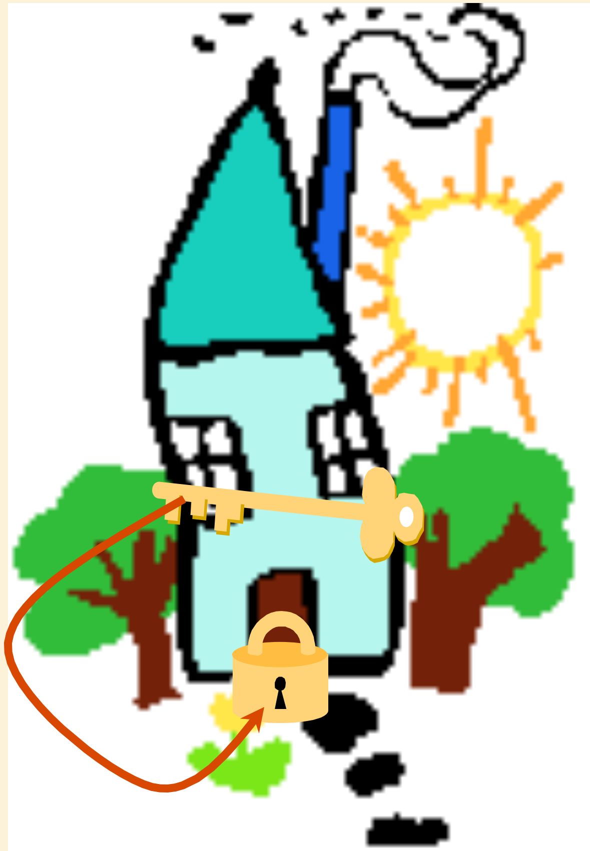
Recursive Algorithm:

- Assume you have an algorithm that works.
- Use it to write an algorithm that works.

Friends & Strong Induction

Recursive Algorithm:

- Assume you have an algorithm that works.
- Use it to write an algorithm that works.



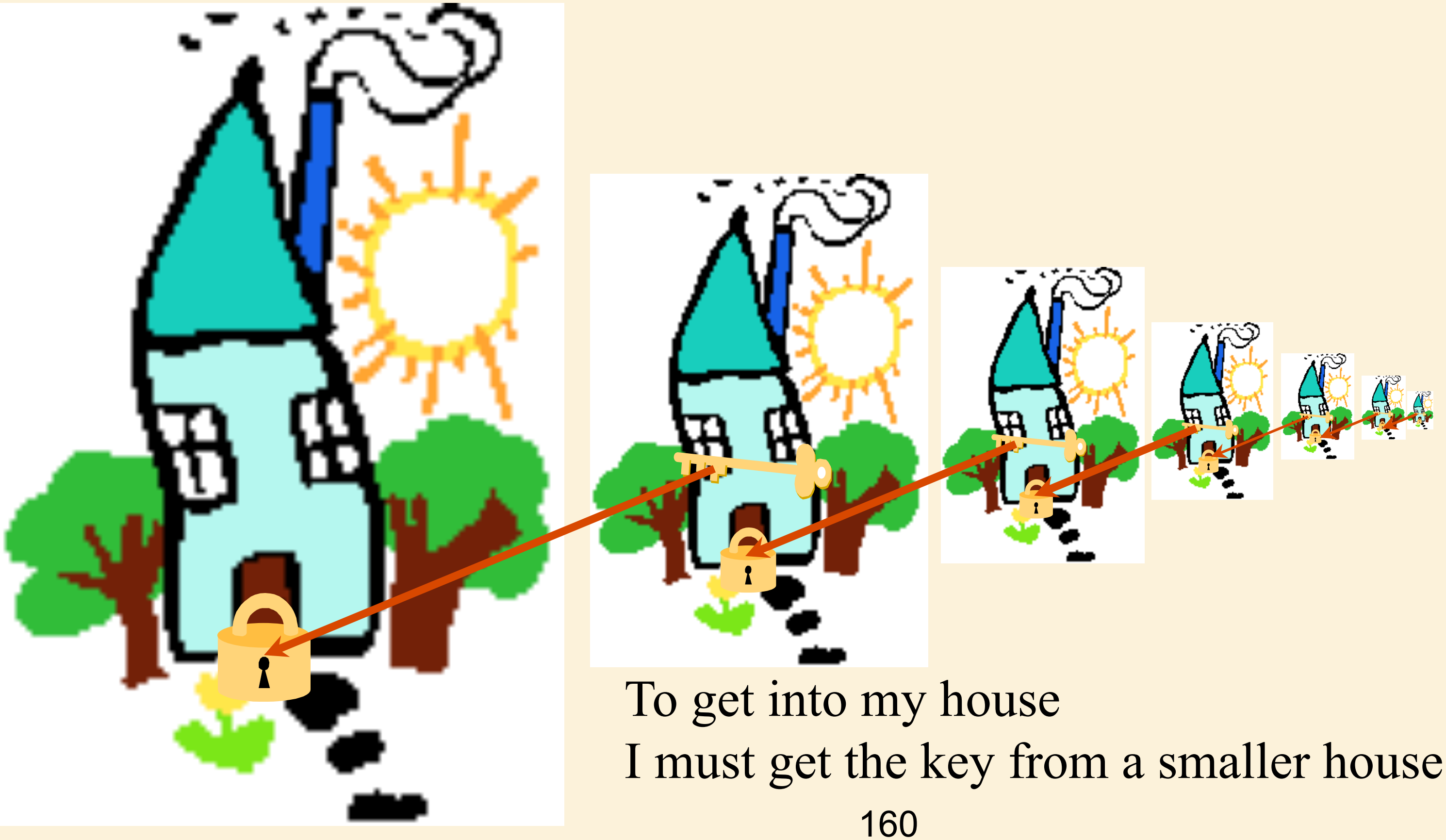
If I could get in,
I could get the key.
Then I could unlock the door
so that I can get in.

Circular Argument!

Friends & Strong Induction

Recursive Algorithm:

- Assume you have an algorithm that works.
- Use it to write an algorithm that works.



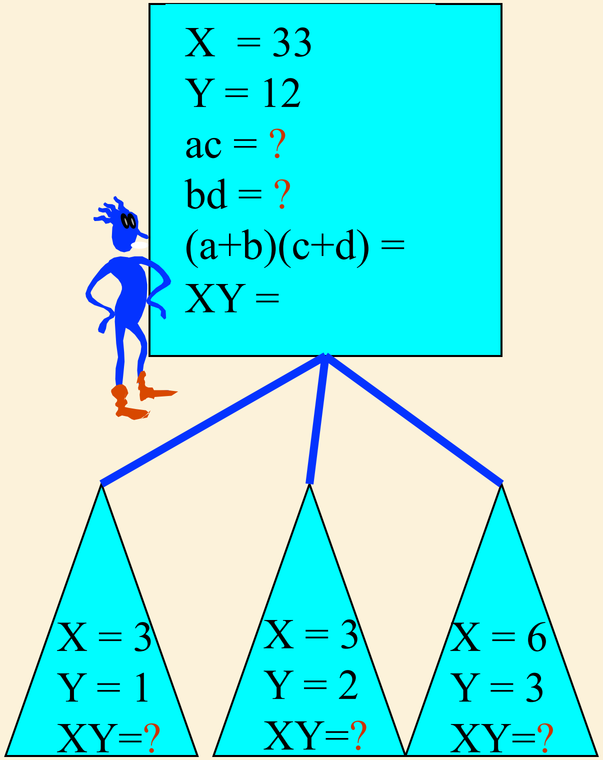
To get into my house
I must get the key from a smaller house

Friends & Strong Induction

```
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
  $e\ 10^n + (\text{MULT}(a+b,\ c+d) – e - f)\ 10^{n/2} + f$
```

- Allocate work
  - Construct one or more subinstances

Each subinstance must be
a smaller instance
to the same problem.



X = 33
Y = 12
ac = ?
bd = ?
(a+b)(c+d) =
XY =

X = 3
Y = 1
XY=?

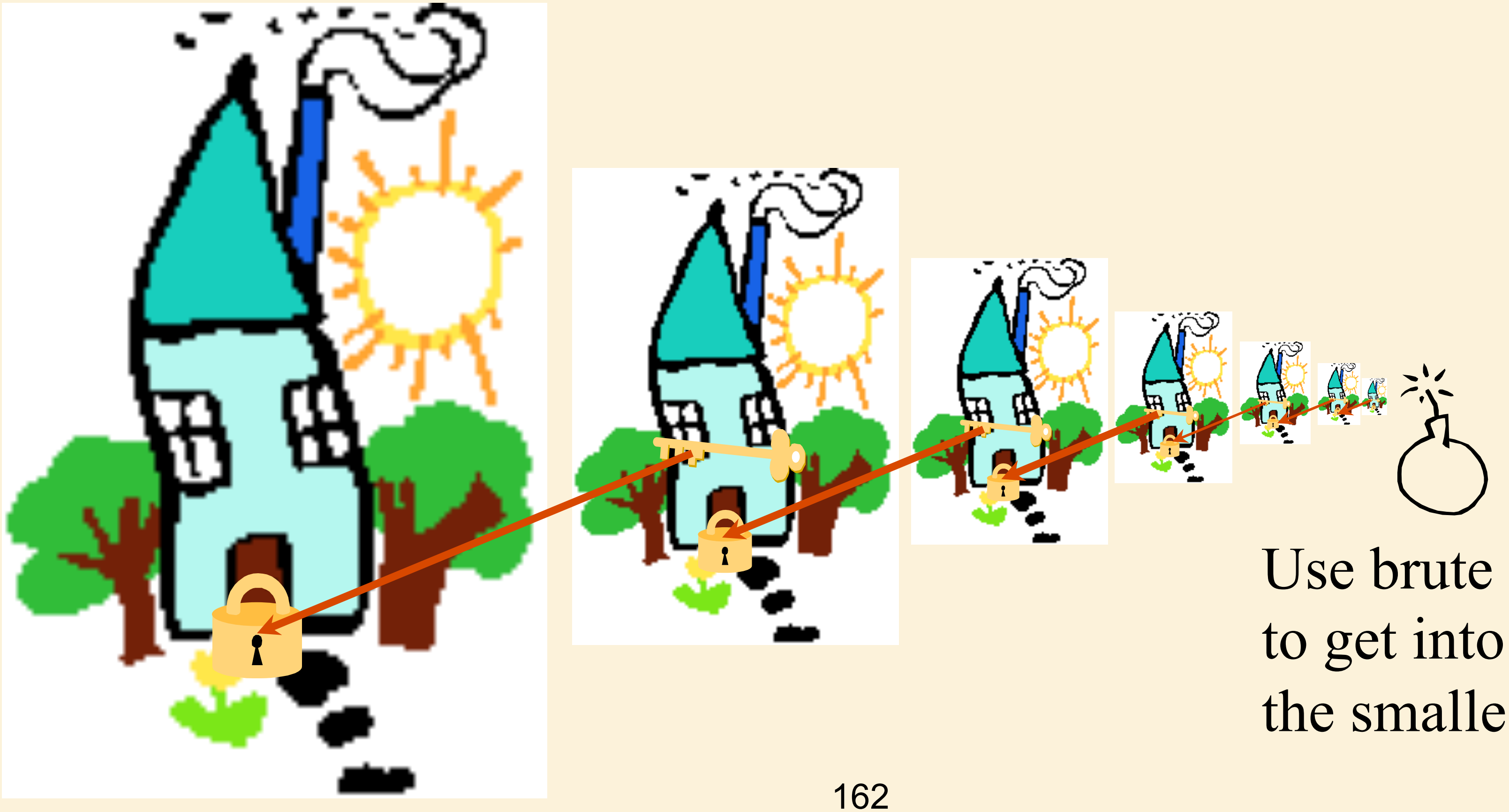
X = 3
Y = 2
XY=?

X = 6
Y = 3
XY=?

Friends & Strong Induction

Recursive Algorithm:

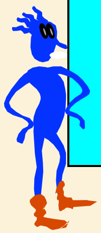
- Assume you have an algorithm that works.
- Use it to write an algorithm that works.



Use brute force
to get into
the smallest house.

Friends & Strong Induction

```
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
  $e\ 10^n + (\text{MULT}(a+b, c+d) - e - f)\ 10^{n/2} + f$
```



```
MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
```

Use brute force
to solve the base case
instances.

Friends & Strong Induction

Carefully write the specifications for the problem.

Preconditions: Set of legal instances (inputs) Why?

Postconditions: Required output

Friends & Strong Induction

Carefully write the specifications for the problem.

Preconditions:	Set of legal instances (inputs)	•To be sure that we solve the problem for every legal instance. •So that we know –what we can give to a friend.
Postconditions:	Required output	•So that we know –what is expected of us. –what we can expect from our friend.

Related to Loop Invariants

Applications of Recursion

Another Numerical Computation Example

The Greatest Common Divisor (GCD) Problem

- Given two integers, what is their greatest common divisor?
- e.g., gcd(56,24) = 8

Notation:

Given $d,a \in ¢$:

$d \mid a \leftrightarrow d$ divides $a \leftrightarrow \exists k \in ¢ : a = kd$

Note: All integers divide 0: $d \mid 0 \forall d \in ¢$

Important Property:

$d \mid a$ and $d \mid b \rightarrow d \mid \left(ax + by\right) \forall x, y \in ¢$

Euclid’s Trick

Euclid of Alexandria,
"The Father of Geometry"
c. 300 BC

Important Property:

$d \mid a$ and $d \mid b \rightarrow d \mid (ax+by)\, \forall x,y \in ¢$

Idea: Use this property to make the GCD problem easier!

Consequence: *e.g.*,

$\gcd(a,b)=\gcd(a-b,b) \longrightarrow \gcd(56,24)=\gcd(56-24,24)=\gcd(32,24)$ **Good!**

$\gcd(a,b)=\gcd(a-2b,b) \longrightarrow \gcd(56,24)=\gcd(56-2\times 24,24)=\gcd(8,24)$ **Better!**

$\gcd(a,b)=\gcd(a-3b,b) \longrightarrow \gcd(56,24)=\gcd(56-3\times 24,24)=\gcd(-16,24)$ **Too Far!**

N

What is the optimal choice?

$\gcd(a,b)=\gcd(a \bmod b,b) \rightarrow \gcd(56,24)=\gcd(56 \bmod 24,24)=\gcd(8,24)$

Euclid’s Algorithm (*circa* 300 BC)

```
Euclid(a,b)
<Precondition: $a$ and $b$ are positive integers>
<Postcondition: returns gcd($a,b$)>
  if $b = 0$ then
    return($a$)
  else
    return(Euclid($b, a \bmod b$))
```

Precondition met, since $a \bmod b \in$ ¢

Postcondition met, since

1. $b = 0 \rightarrow \gcd(a,b) = \gcd(a,0) = a$
2. Otherwise, $\gcd(a,b) = \gcd(b, a \bmod b)$
3. Algorithm halts, since $0 \leq a \bmod b < b$

End of Lecture 8

Time Complexity

```
Euclid(a,b)
  if $b = 0$ then
    return($a$)
  else
    return(Euclid($b, a \bmod b$))
```

Claim: 2nd argument drops by factor of at least 2 every 2 iterations.

Proof:

Iteration	Arg 1	Arg 2
i	a	b
$i+1$	b	$a \bmod b$
$i+2$	$a \bmod b$	$b \bmod (a \bmod b)$

Case 1: $a \bmod b \le b/2$. Then $b \bmod (a \bmod b) < a \bmod b \le b/2$ ✓

Case 2: $b > a \bmod b > b/2$. Then $b \bmod (a \bmod b) < b/2$ ✓

Time Complexity

```
Euclid(a,b)
  if $b = 0$ then
    return($a$)
  else
    return(Euclid($b, a \bmod b$))
```

Let k = total number of recursive calls to Euclid.

Let n = input size ; number of bits used to represent a and b.

Then $2^{k/2}$; b ; $2^{n/2} \rightarrow k$; n.

Each stackframe must compute $a \bmod b$, which takes more than constant time.

It can be shown that the resulting time complexity is $T(n) \in O(n^2)$.

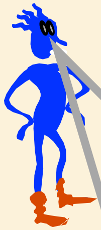
Applications of Recursion

Data Organization

A Simple Example:

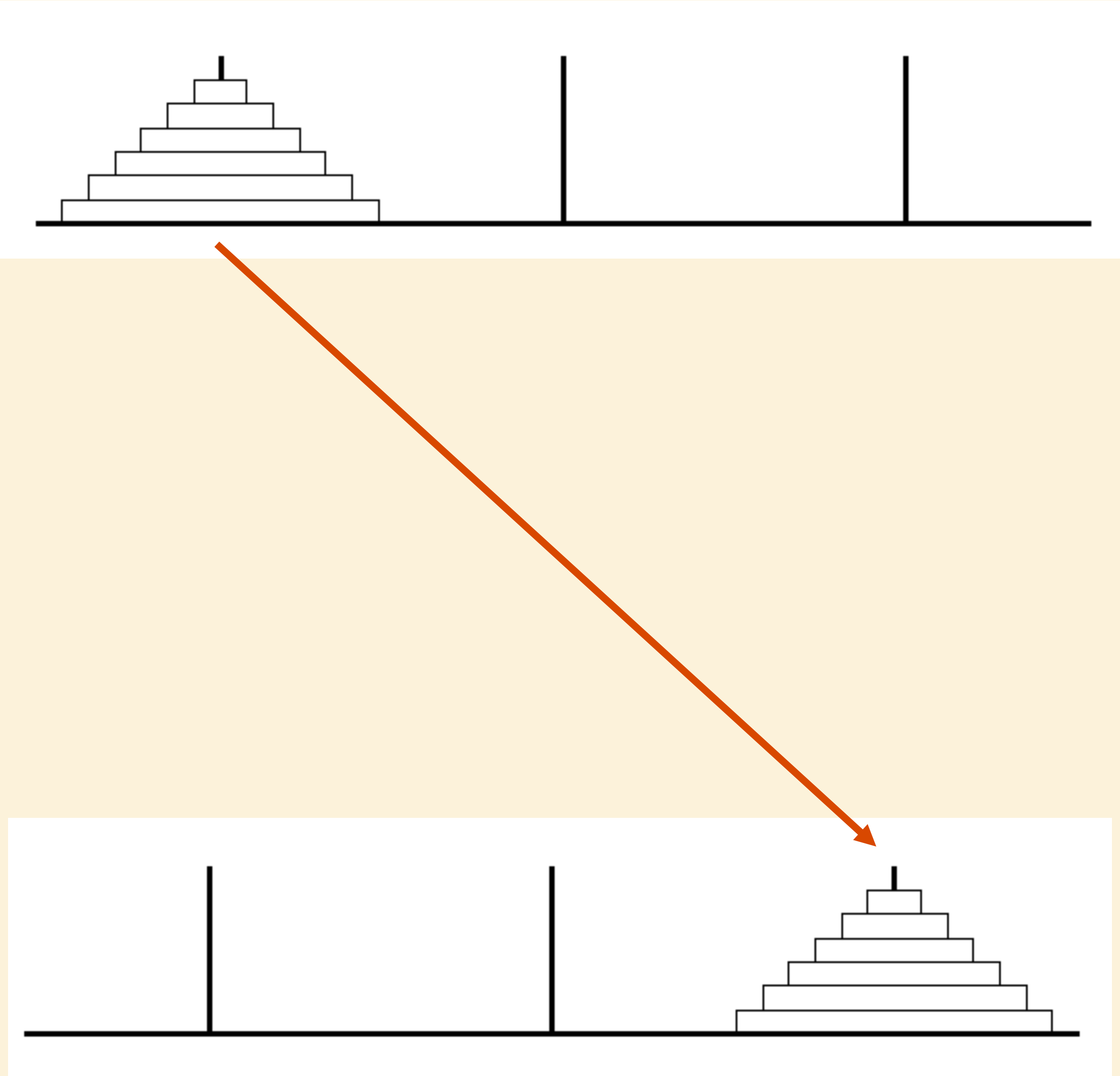
The Tower of Hanoi

Tower of Hanoi

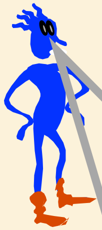


This job of mine is a bit daunting. Where do I start?

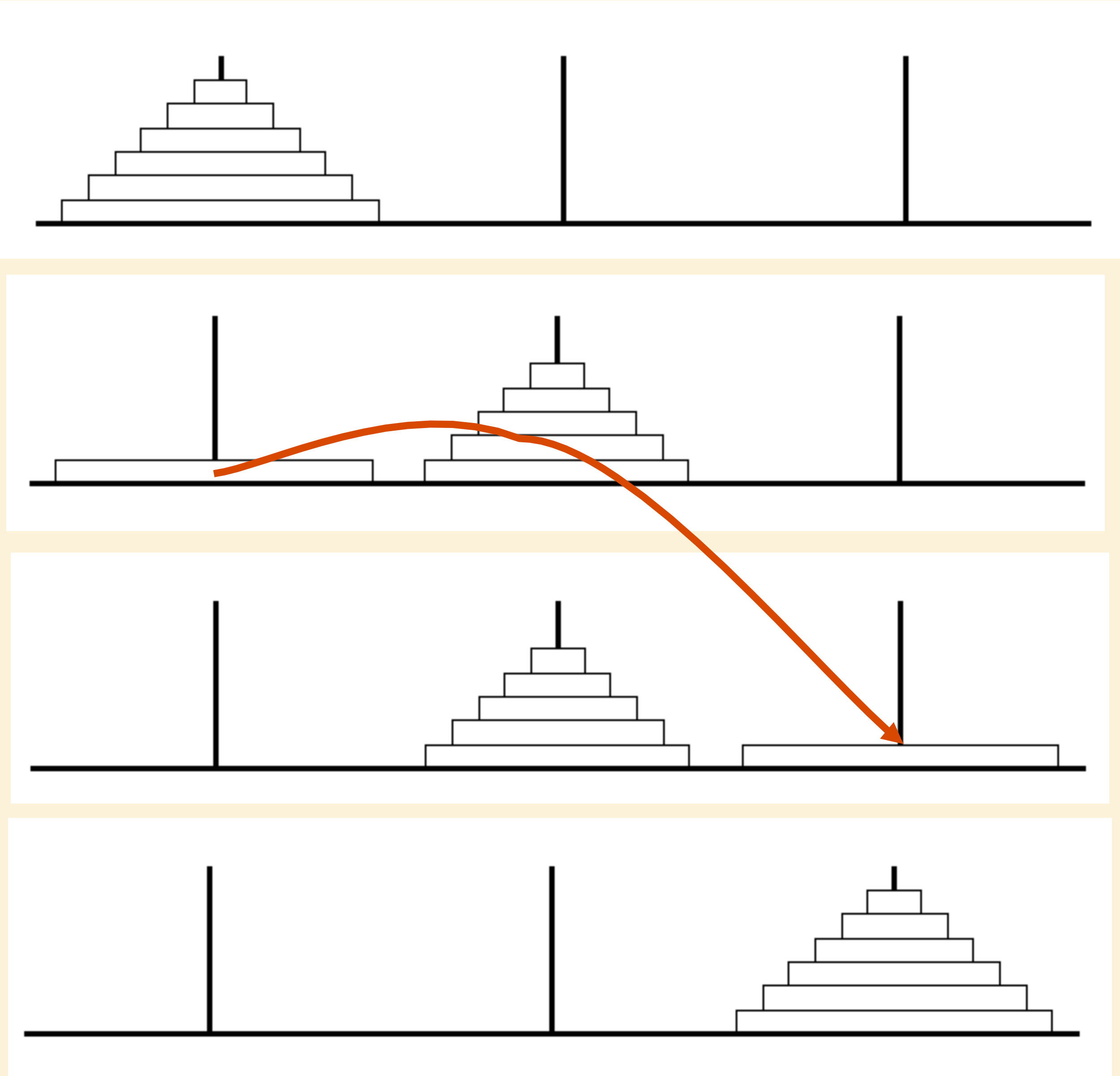
And I am lazy.



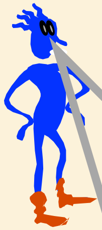
Tower of Hanoi



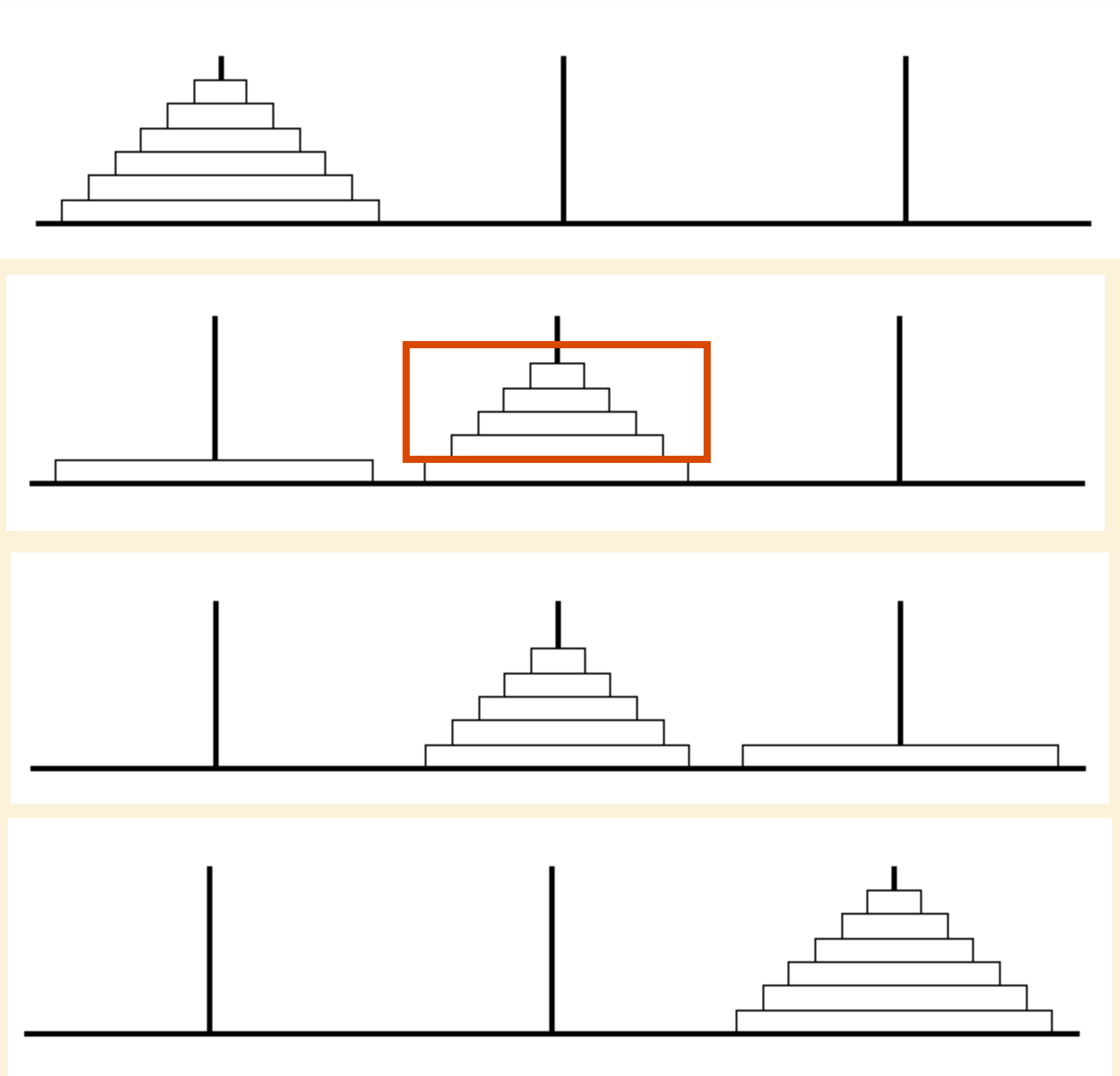
At some point, the biggest disk moves.
I will do that job.



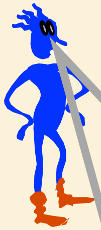
Tower of Hanoi



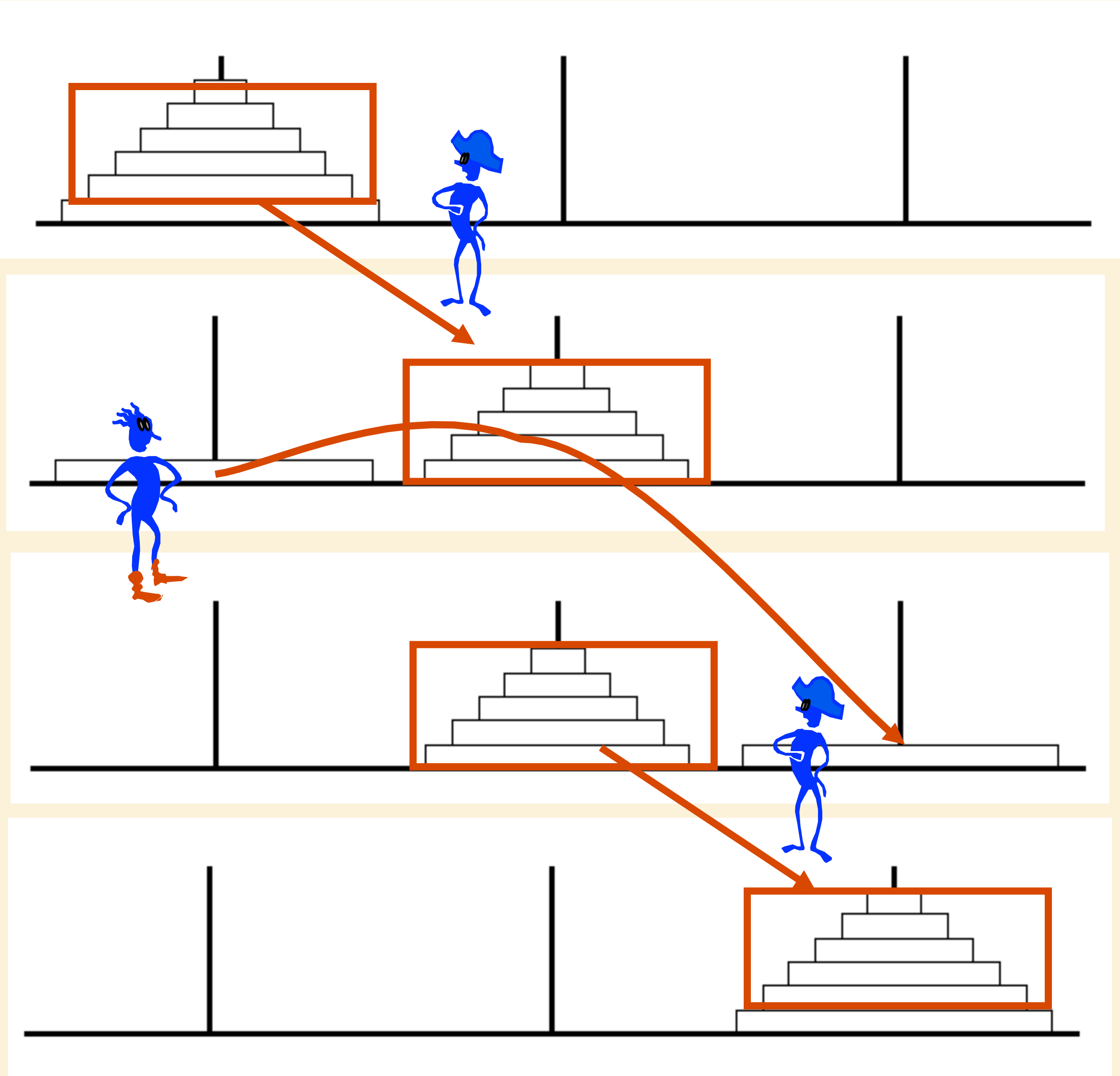
To do this,
the other disks
must be in the
middle.



Tower of Hanoi



How will these move?
I will get a friend to do it.
And another to move these.
I only move the big disk.



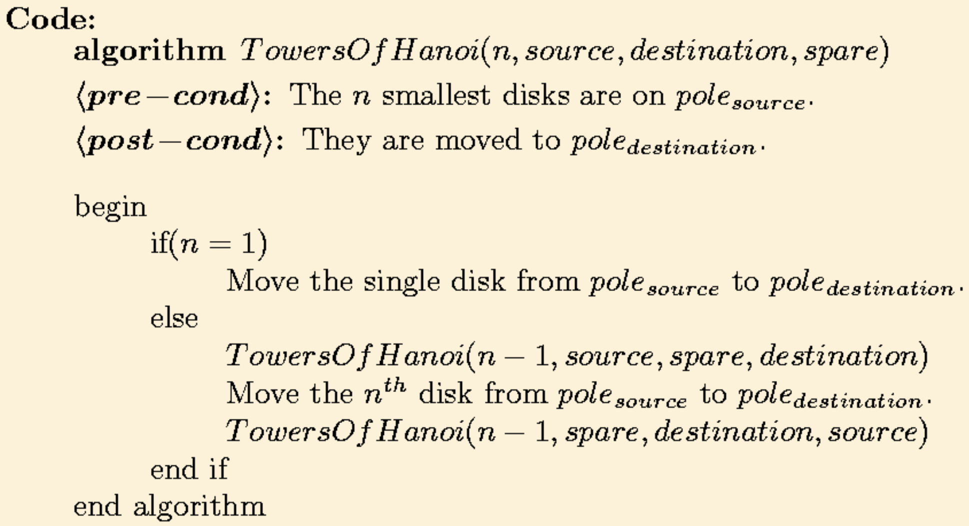
Tower of Hanoi

Code:

```
algorithm $TowersOfHanoi(n, source, destination, spare)$
$\langle pre-cond \rangle$: The $n$ smallest disks are on $pole_{source}$.
$\langle post-cond \rangle$: They are moved to $pole_{destination}$.

begin
    if($n = 1$)
        Move the single disk from $pole_{source}$ to $pole_{destination}$.
    else
        $TowersOfHanoi(n-1, source, spare, destination)$
        Move the $n^{th}$ disk from $pole_{source}$ to $pole_{destination}$.
        $TowersOfHanoi(n-1, spare, destination, source)$
    end if
end algorithm
```

Tower of Hanoi



Code:

```
algorithm $TowersOfHanoi(n, source, destination, spare)$
⟨pre−cond⟩: The $n$ smallest disks are on $pole_{source}$.
⟨post−cond⟩: They are moved to $pole_{destination}$.

begin
    if($n = 1$)
        Move the single disk from $pole_{source}$ to $pole_{destination}$.
    else
        $TowersOfHanoi(n-1, source, spare, destination)$
        Move the $n^{th}$ disk from $pole_{source}$ to $pole_{destination}$.
        $TowersOfHanoi(n-1, spare, destination, source)$
    end if
end algorithm
```

Time:

$T(1) = 1,$

$T(n) = 1 + 2T(n\text{-}1) \quad \approx 2T(n\text{-}1)$

$\approx 2(2T(n\text{-}2)) \quad \approx 4T(n\text{-}2)$

$\approx 4(2T(n\text{-}3)) \quad \approx 8T(n\text{-}3)$

$\approx 2^i\, T(n\text{-}i)$

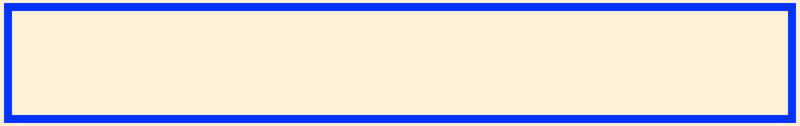
$\approx 2^n$

More Data Organization Examples

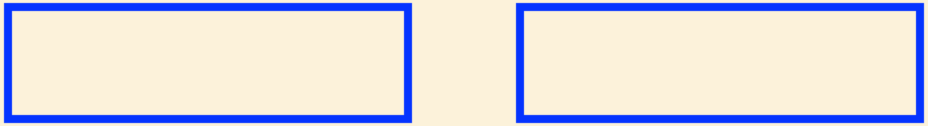
Sorting

Recursive Sorts

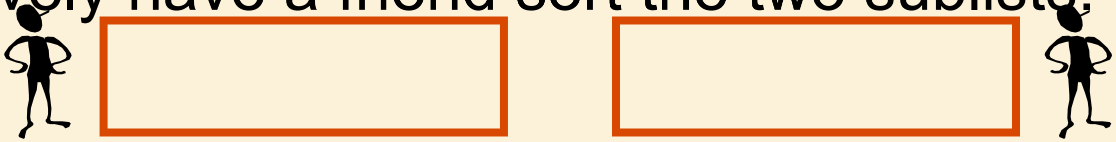
- Given list of objects to be sorted



- Split the list into two sublists.



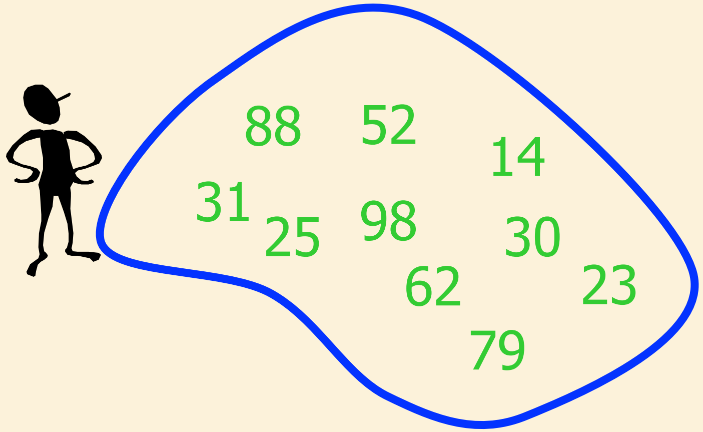
- Recursively have a friend sort the two sublists.



- Combine the two sorted sublists into one entirely sorted list.

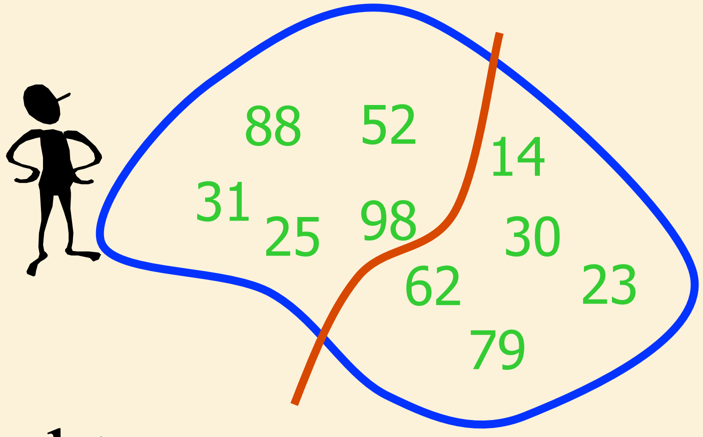
Example: Merge Sort

Merge Sort



Divide and Conquer

Merge Sort



Split Set into Two
(no real work)

Get one friend to sort the first half.

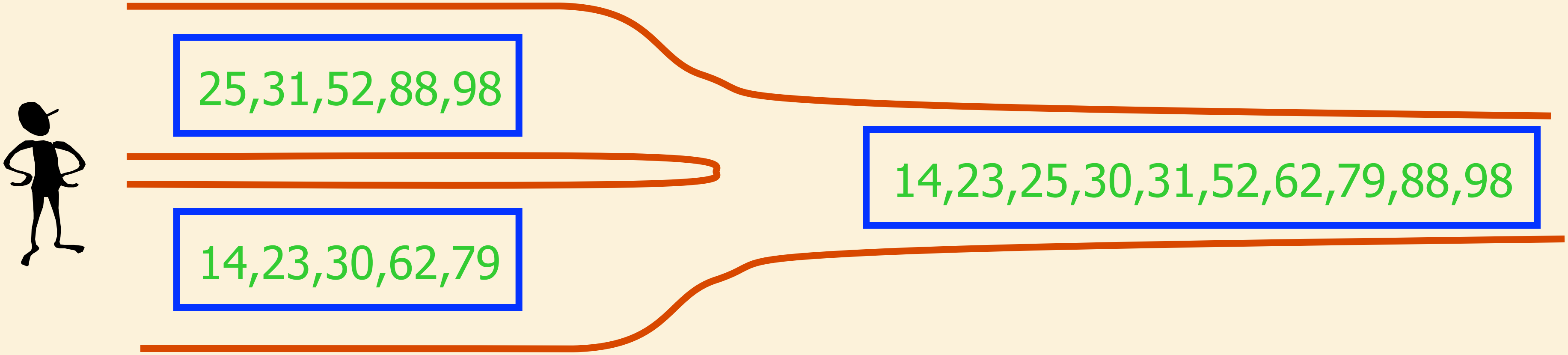
Get one friend to sort the second half.

25,31,52,88,98

14,23,30,62,79

Merge Sort

Merge two sorted lists into one



25,31,52,88,98

14,23,30,62,79

14,23,25,30,31,52,62,79,88,98

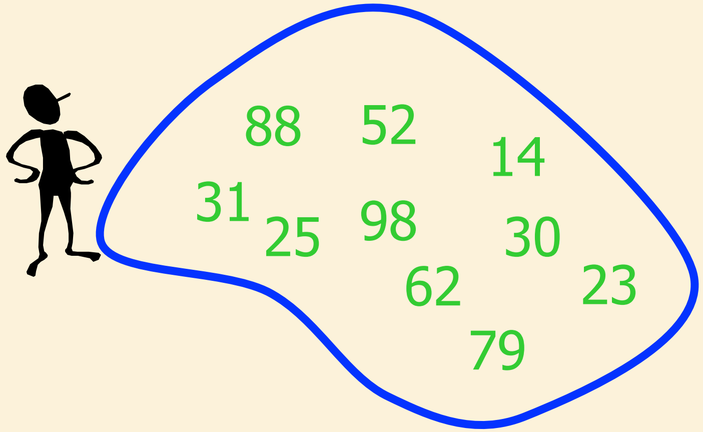
Merge Sort

Time: $T(n) = 2T(n/2) + \Theta(n)$

$$= \Theta(n \log(n))$$

Example: Quick Sort

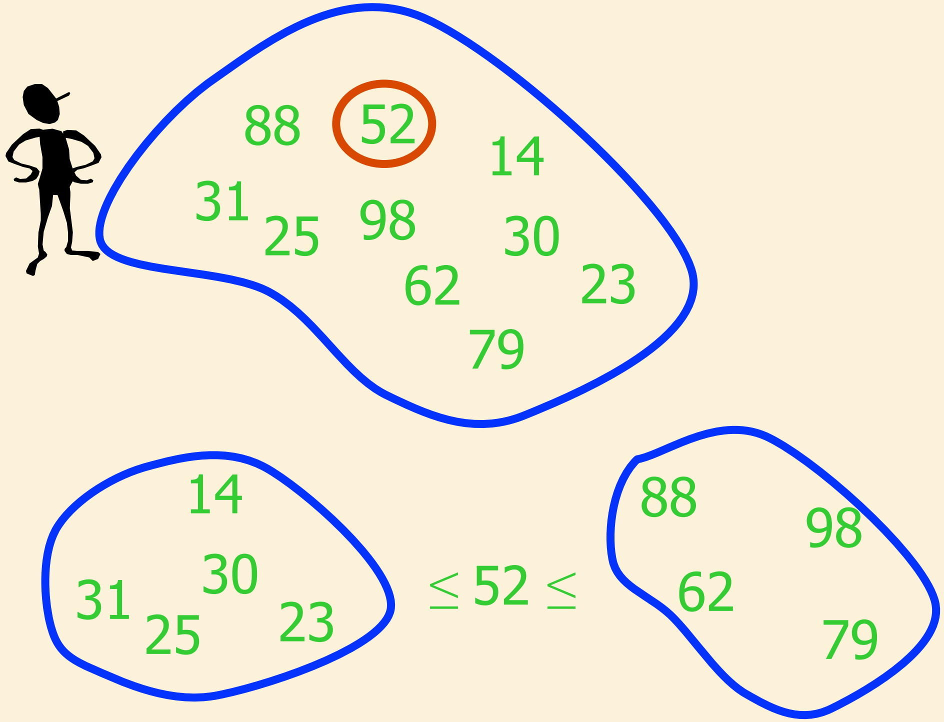
Quick Sort



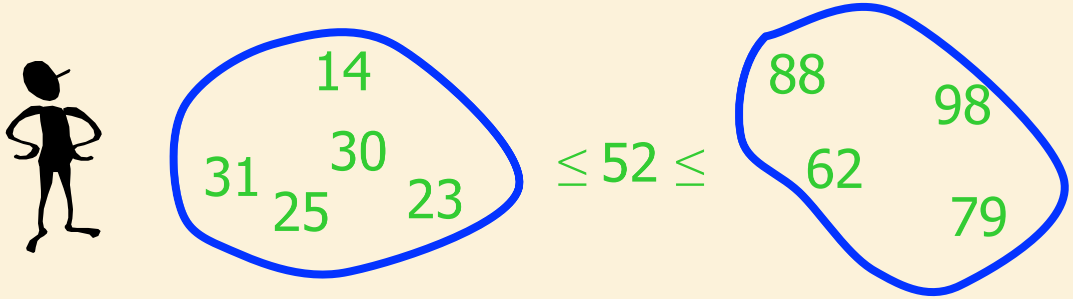
Divide and Conquer

Quick Sort

Partition set into two using randomly chosen pivot



Quick Sort



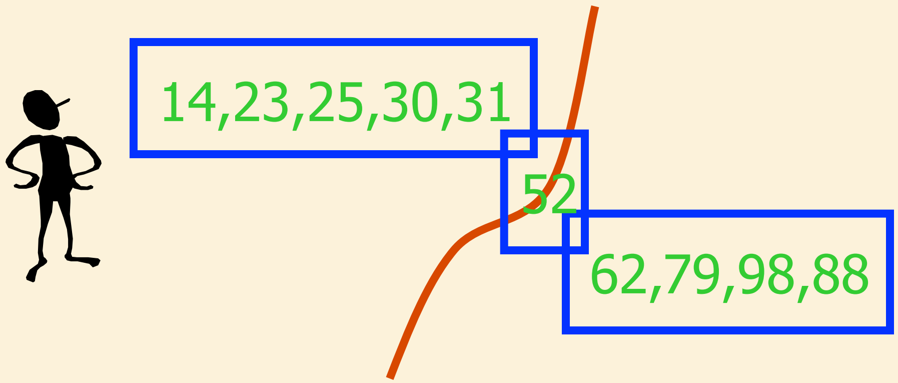
Get one friend to sort the first half.

14,23,25,30,31

Get one friend to sort the second half.

62,79,98,88

Quick Sort



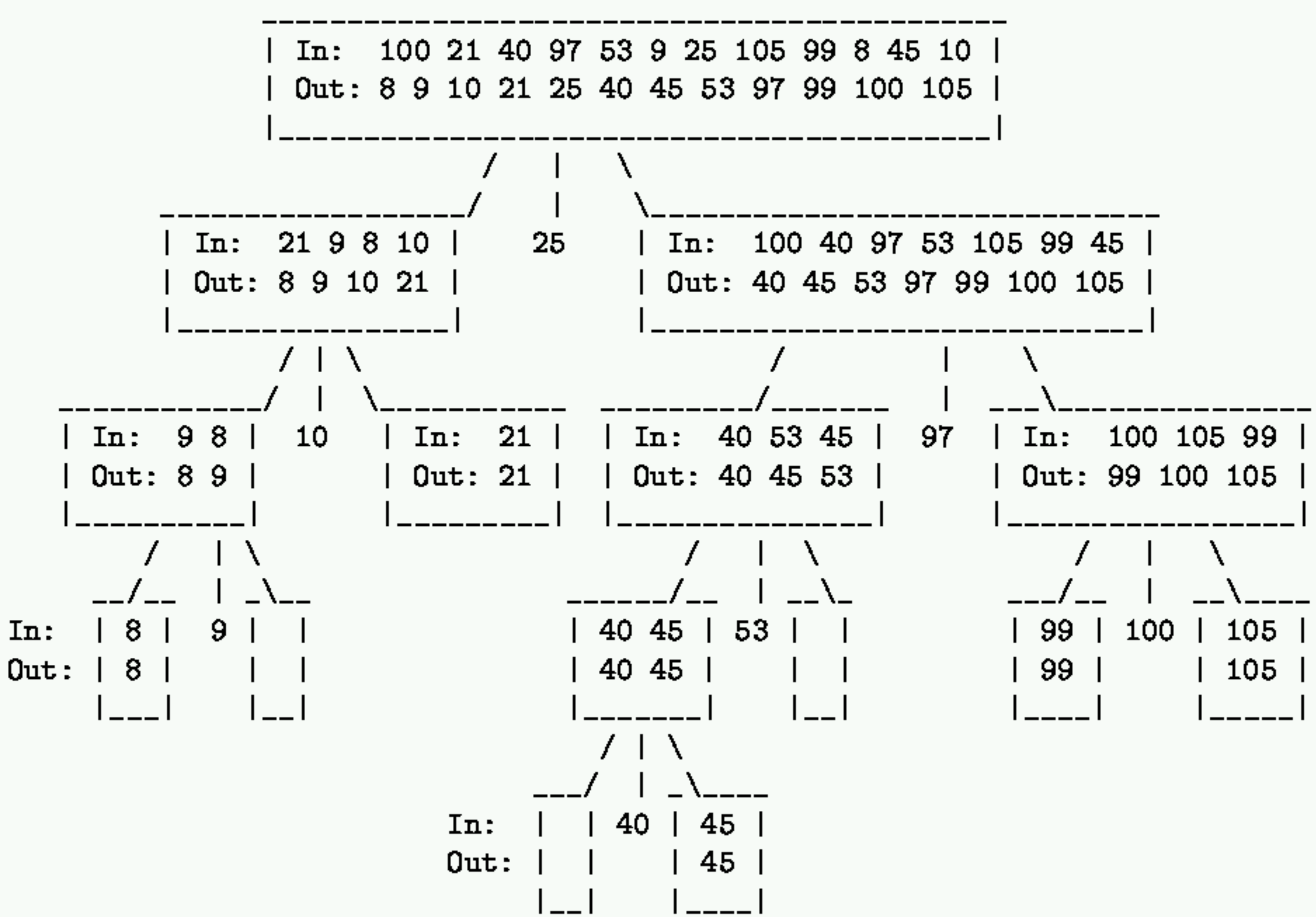
14,23,25,30,31

52

62,79,98,88

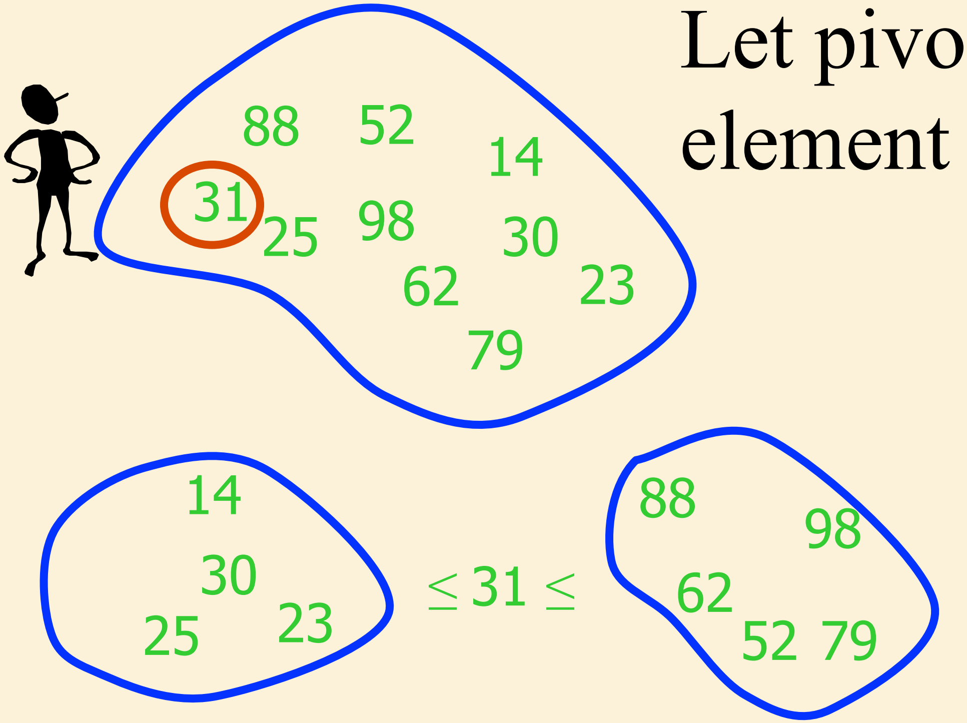
Glue pieces together.
(No real work)

14,23,25,30,31,52,62,79,88,98



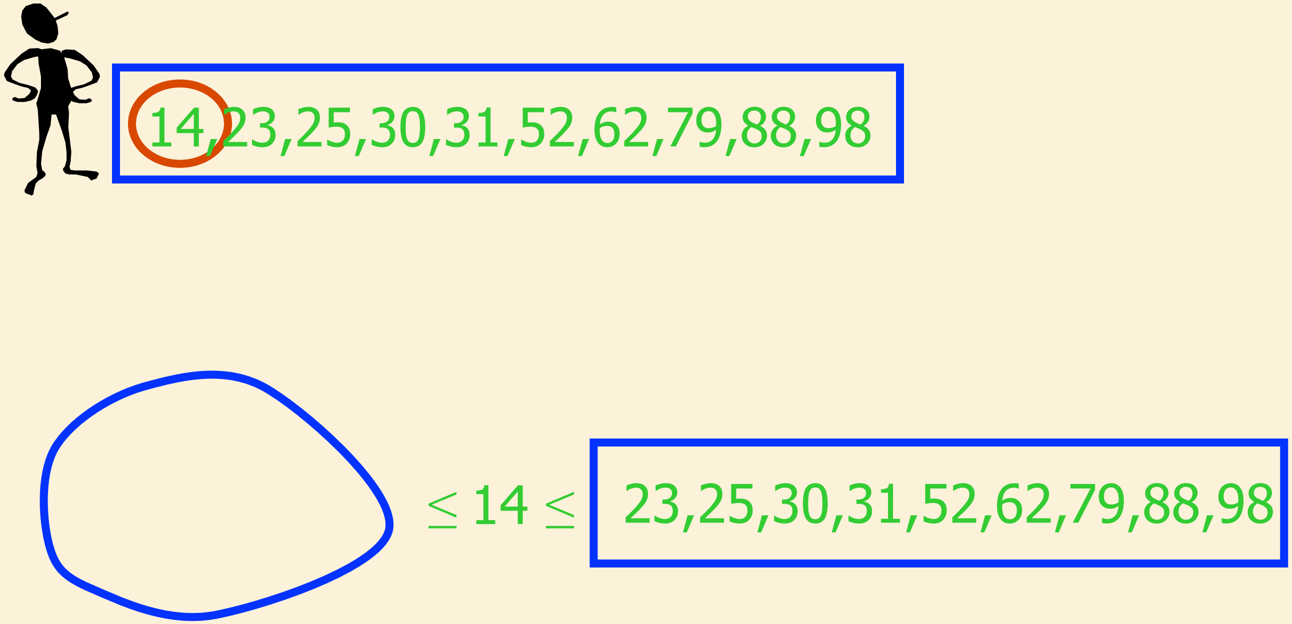
```
                ________________________________________________
               | In:  100 21 40 97 53 9 25 105 99 8 45 10 |
               | Out: 8 9 10 21 25 40 45 53 97 99 100 105 |
               |__________________________________________|
                            /   |   \
          _________________/    |    \______________________________
         | In:  21 9 8 10 |    25    | In:  100 40 97 53 105 99 45 |
         | Out: 8 9 10 21 |          | Out: 40 45 53 97 99 100 105 |
         |________________|          |_____________________________|
                / | \                        /         |    \
    ___________/  |  \___________   ________/______    |   ___\_____________
   | In:  9 8 |  10    | In:  21 |  | In:  40 53 45 |  97  | In:  100 105 99 |
   | Out: 8 9 |        | Out: 21 |  | Out: 40 45 53 |      | Out: 99 100 105 |
   |__________|        |_________|  |_______________|      |_________________|
        /   | \                         /   |  \               /   |   \
     __/__  | _\__               ______/__  | __\_         ___/__  |  __\____
In:  | 8 |  9 |  |               | 40 45 | 53 |  |         | 99 | 100 | 105 |
Out: | 8 |    |  |               | 40 45 |    |  |         | 99 |     | 105 |
     |___|    |__|               |_______|    |__|         |____|     |_____|
                                   / | \
                               ___/  | _\____
                          In:  |  | 40 | 45 |
                          Out: |  |    | 45 |
                               |__|    |____|
```

Quick Sort



Let pivot be the first element in the list?

Quick Sort

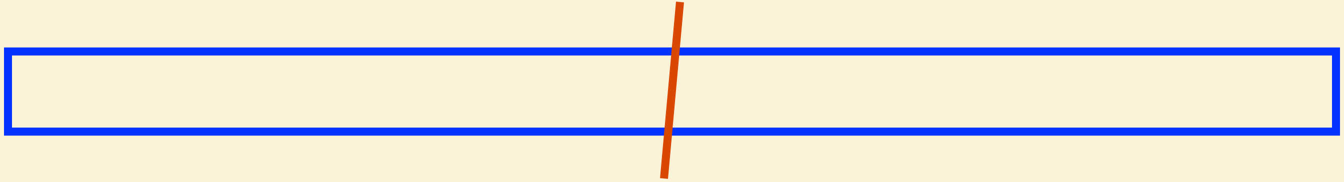


14,23,25,30,31,52,62,79,88,98

≤ 14 ≤ 23,25,30,31,52,62,79,88,98

If the list is already sorted,
then the list is worst case unbalanced.

Quick Sort



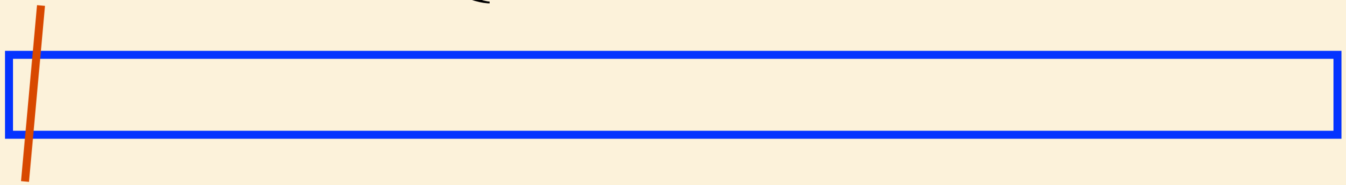
Best Time: $T(n) = 2T(n/2) + \Theta(n)$

$= \Theta(n \log(n))$

Worst Time:

Expected Time:

Quick Sort



Best Time: $T(n) = 2T(n/2) + \Theta(n)$

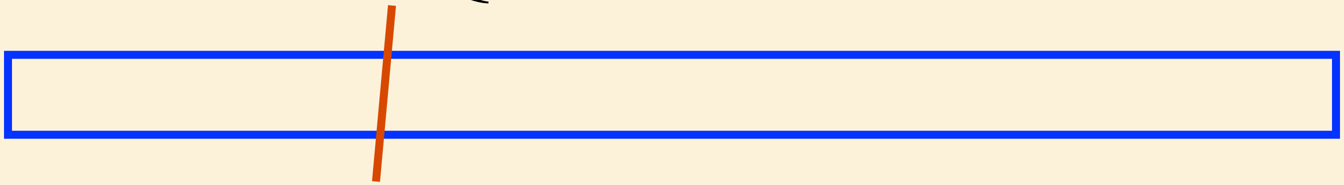
$= \Theta(n \log(n))$

Worst Time: $T(n) = T(1) + T(n-1) + \Theta(n)$

$= \Theta(n^2)$

Expected Time:

Quick Sort



Best Time: $T(n) = 2T(n/2) + \Theta(n)$

$= \Theta(n \log(n))$

Worst Time: $T(n) = T(0) + T(n-1) + \Theta(n)$

$= \Theta(n^2)$

Expected Time: $T(n) = \Theta(n\log(n))$

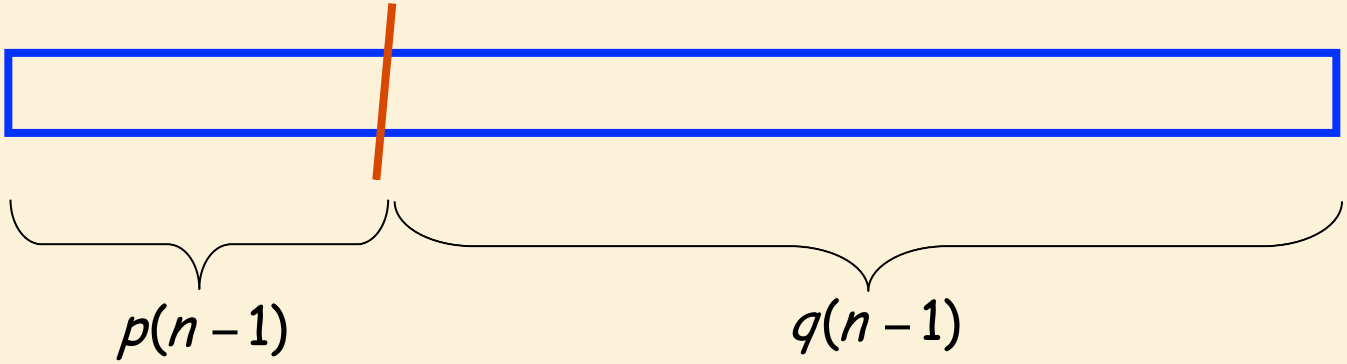
(The proof is not difficult, but it's a little long)

Expected Time Complexity for Quick Sort

Q: Why is it reasonable to expect $\Theta(n\log n)$ time complexity?

A: Because on average, the partition is not too unbalanced.

Example: Imagine a deterministic partition,
in which the 2 subsets are always in fixed proportion, i.e.,
$p(n-1)$ & $q(n-1)$, where p,q are constants, $p,q \in [0...1], p+q=1$.



Expected Time Complexity for Quick Sort

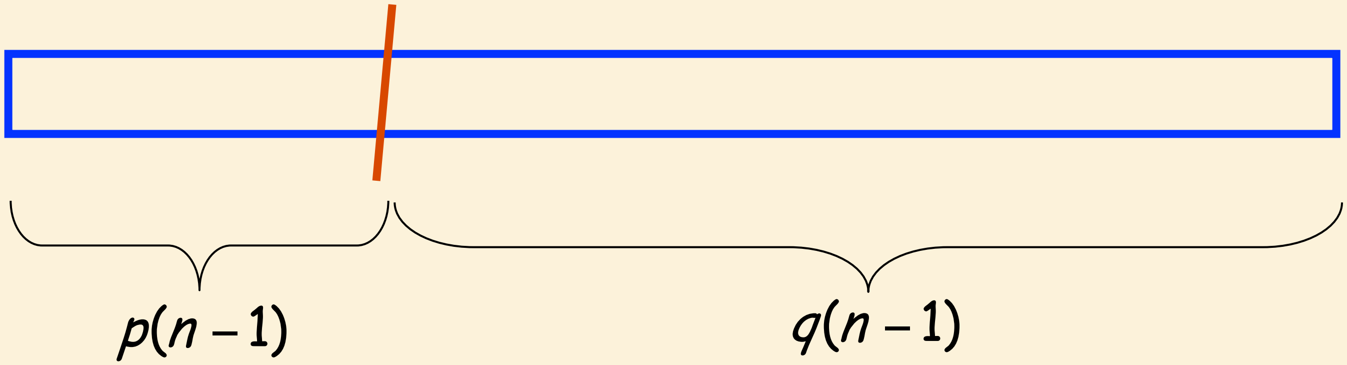
Then $T(n) = T(p(n-1)) + T(q(n-1)) + \Theta(n)$

wlog, suppose that $q > p$.

Then recursion tree has depth $\mathrm{k} \in \Theta(\log n)$:

$q^k n = 1 \rightarrow k = \log n / \log(1/q)$

$\Theta(n)$ work done per level $\rightarrow T(n) = \Theta(n \log n)$.



Properties of QuickSort

- In-place? ✓
- Stable? ✓
- Fast?
  - Depends.
  - Worst Case: $\Theta(n^2)$
  - Expected Case: $\Theta(n \log n)$, with small constants

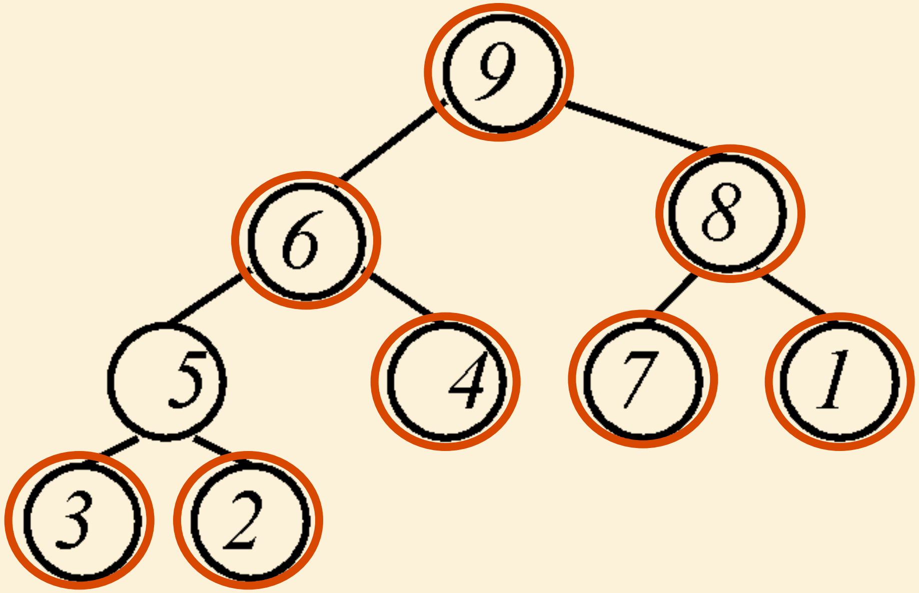
Heaps, Heap Sort, & Priority Queues

Heapsort

- O(*nlogn*) worst case – like merge sort
- Sorts in place – like insertion sort
- Combines the best of both algorithms

Heap Definition (MaxHeap)

- Balanced binary tree
- The value of each node ≥ each of the node's children.
- Left or right child could be next largest.



Where can 9 go?

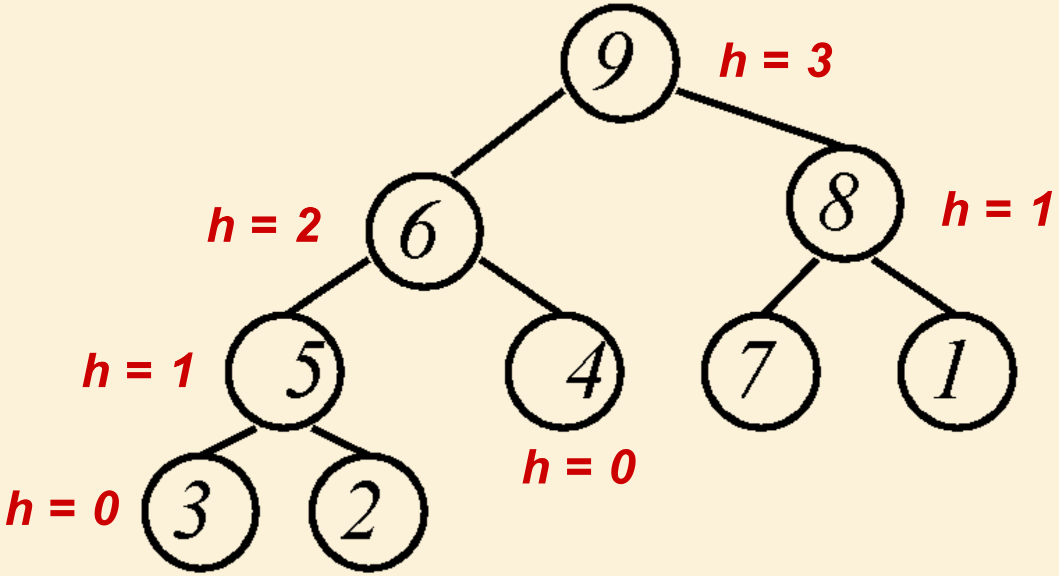
Where can 1 go?

Where can 8 go?

Maximum is at root.

Some Additional Properties of Heaps

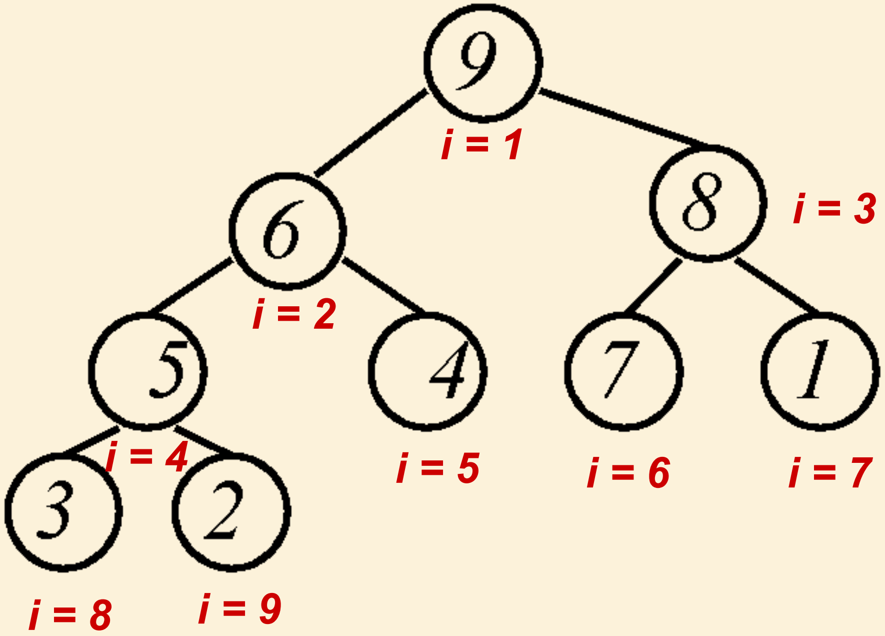
The height *h*(*i*) of a node *i* of the heap is the number of edges on the longest simple downard path from the node to a leaf.



The height *H* of a heap is the height of the root.

Some Additional Properties of Heaps

An n-element heap has height $H = \lfloor \log_2 n \rfloor$

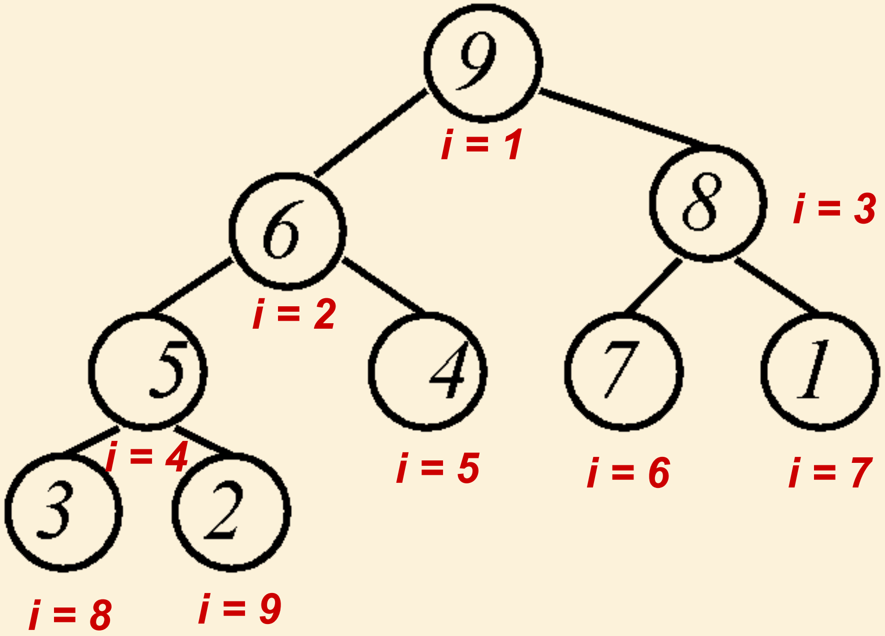


Some Additional Properties of Heaps

A heap of height H has at least $n = 2^{H}$ nodes.

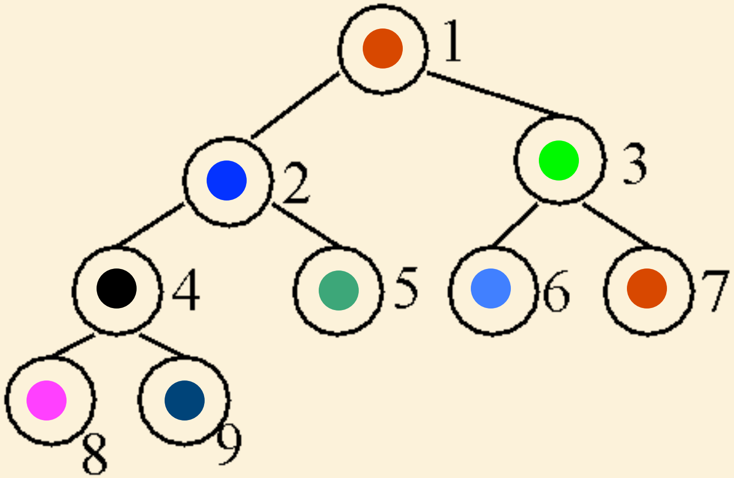
A heap of height H has at most $n = 2^{H+1}$-1 nodes.

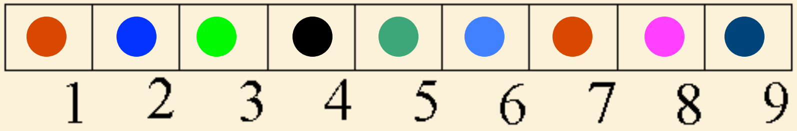
$$2^{H} \le n \le 2^{H+1} - 1$$



Heap Data Structure

Balanced Binary Tree Implemented by an Array





- The root is stored in $A[1]$
- The parent of $A[i]$ is $A[\lfloor \frac{i}{2} \rfloor]$.
- The left child of $A[i]$ is $A[2 \cdot i]$.
- The right child of $A[i]$ is $A[2 \cdot i + 1]$.
- The node in the far right of the bottom level is stored in $A[n]$.
- If $2i + 1 > n$, then the node does not have a right child.

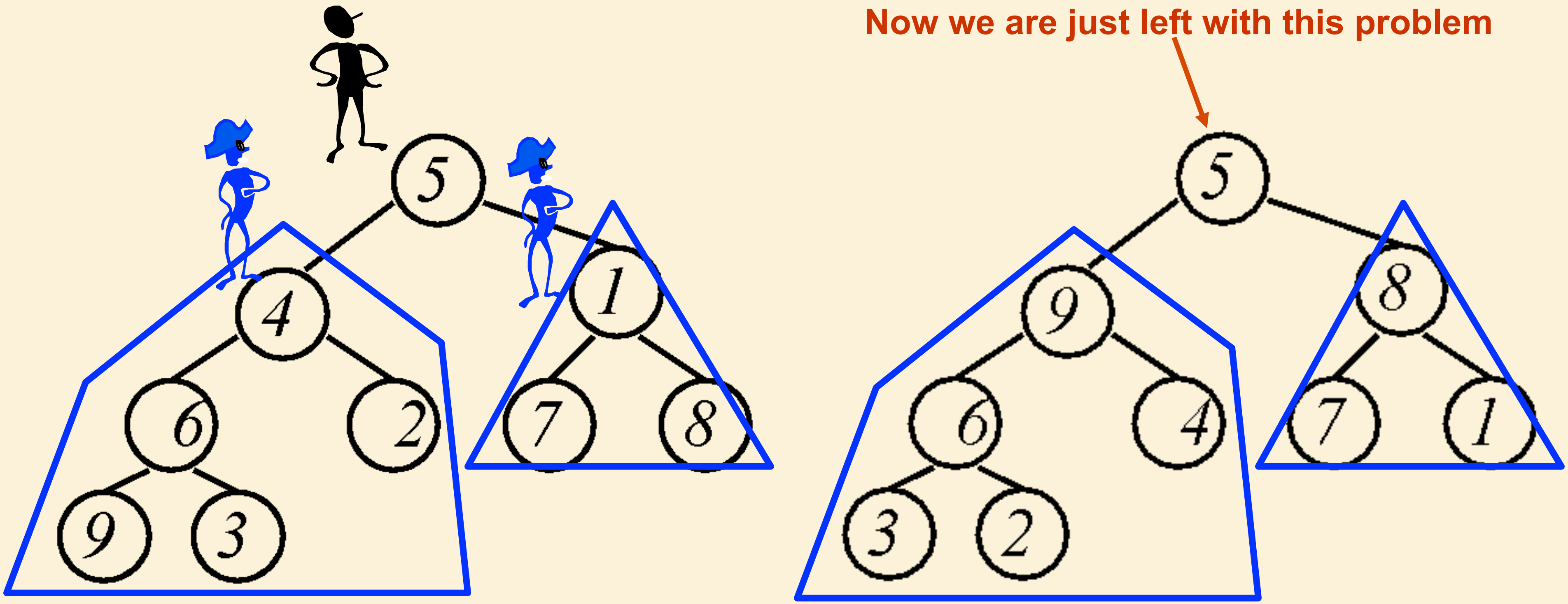
Make Heap

algorithm *MakeHeap*()

⟨*pre–cond*⟩: The input is an array of numbers, which can be viewed as a balanced binary tree of numbers.

⟨*post–cond*⟩: Its values are rearranged in place to make it heap.

Get help from friends



Now we are just left with this problem

Heapify

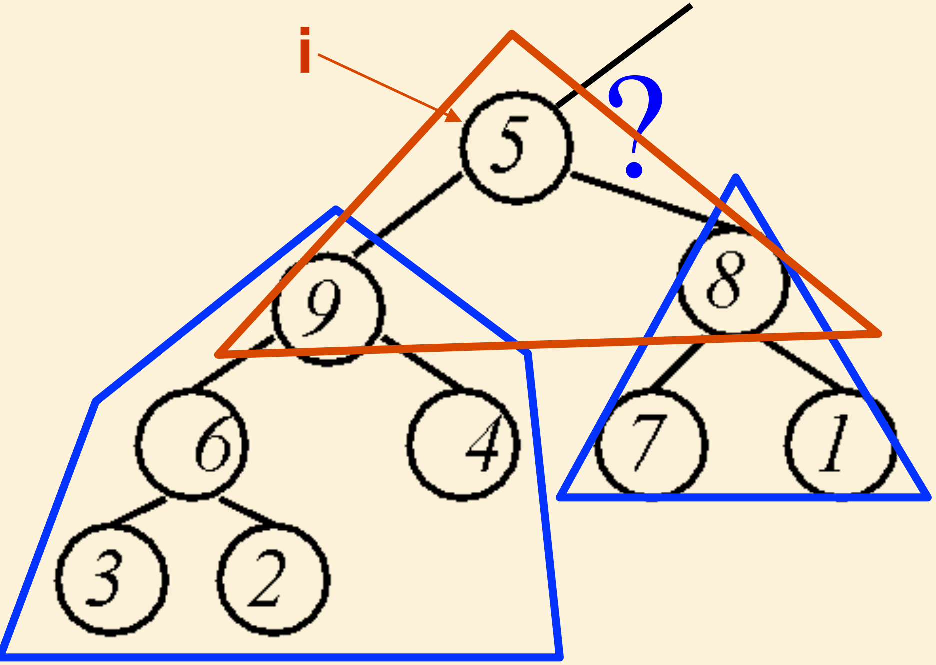
Max-Heapify(*A, i, n*)

<pre-cond>: Left and right subtrees of A[i] are max heaps.

<post-cond>: Subtree rooted at i is a heap.

Where should the maximum be?

Maximum must be at root.



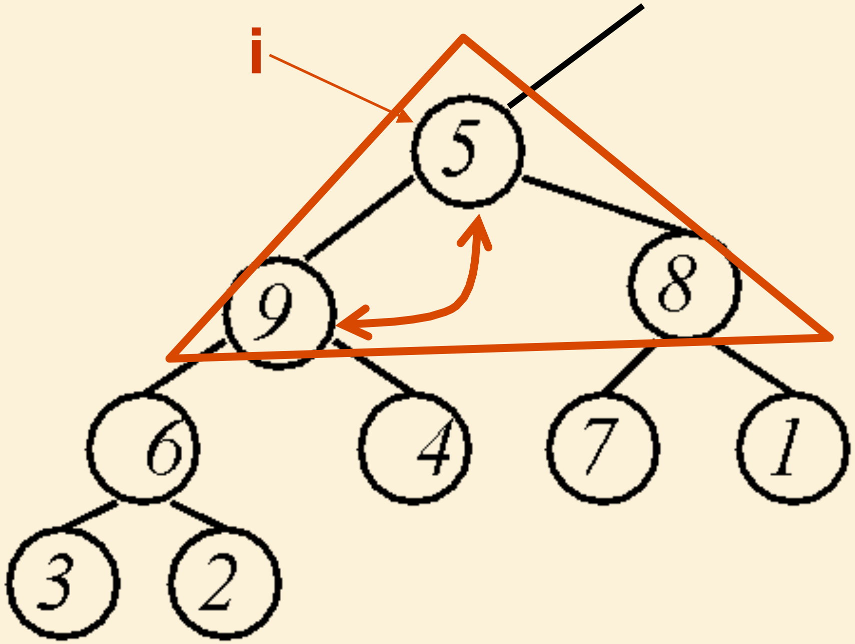
Heapify

Max-Heapify(*A, i, n*)

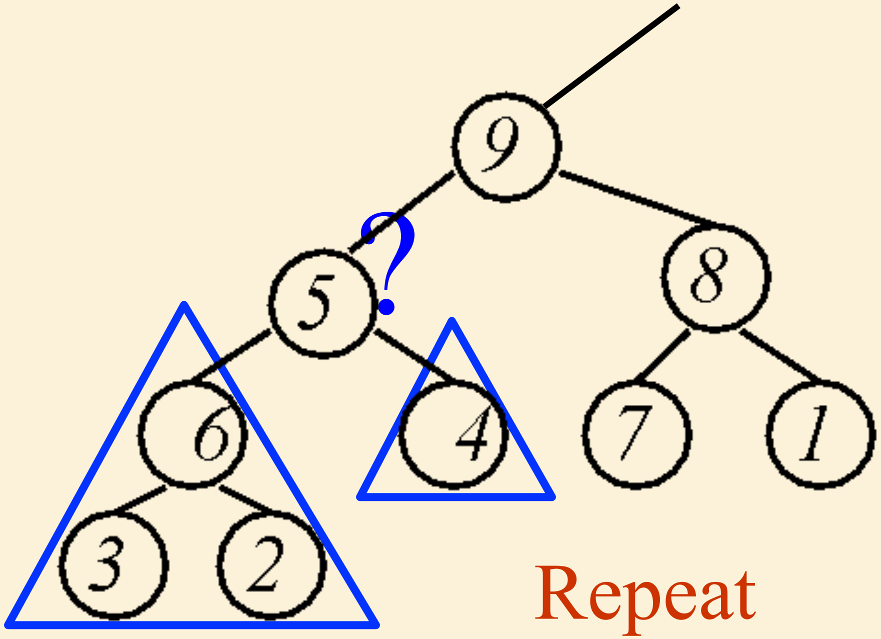
<pre-cond>: Left and right subtrees of A[i] are max heaps.

<post-cond>: Subtree rooted at i is a heap.

Find the maximum.



Put it in place



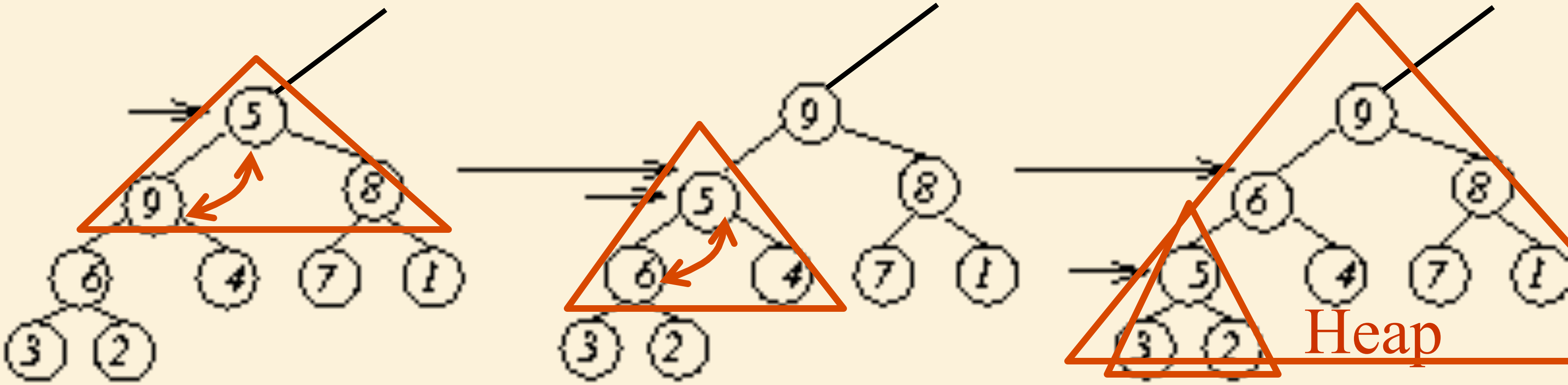
Repeat

Heapify

Max-Heapify(*A, i, n*)

<pre-cond>: Left and right subtrees of A[i] are max heaps.

<post-cond>: Subtree rooted at i is a heap.



Running Time: $T(n) = \Theta(\text{the height of tree}) = \Theta(\log n)$.

$T(n) = 1 \cdot T(n/2) + \Theta(1) = \Theta(\log n)$.

Max-Heapify(*A, i, n*)

<pre-cond>: Left and right subtrees of A[i] are max heaps.

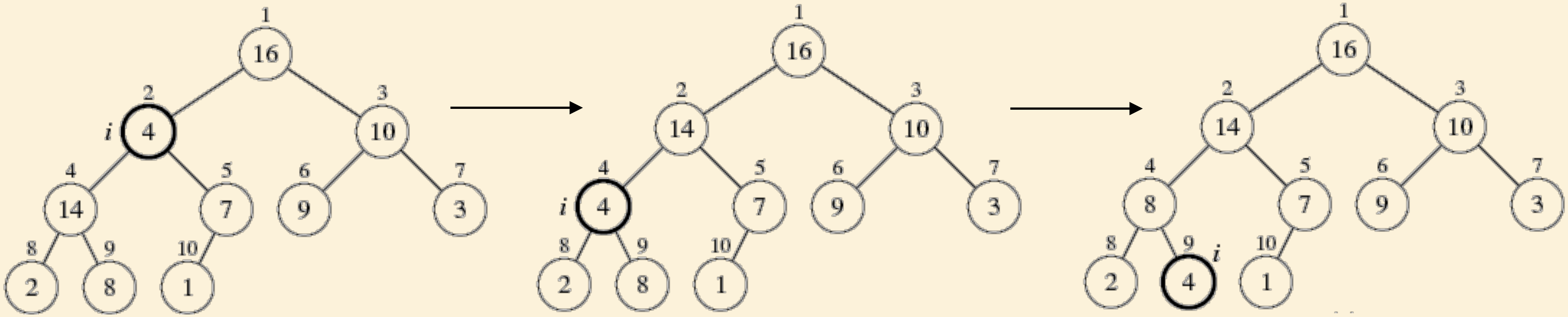
<post-cond>: Subtree rooted at i is a heap.

```
$l \leftarrow \text{LEFT}(i)$
$r \leftarrow \text{RIGHT}(i)$
if $l \le n$ and $A[l] > A[i]$
    then $largest \leftarrow l$
    else $largest \leftarrow i$
if $r \le n$ and $A[r] > A[largest]$
    then $largest \leftarrow r$
if $largest \ne i$
    then exchange $A[i] \leftrightarrow A[largest]$
              MAX-HEAPIFY$(A, largest, n)$
```

e.g., Max-Heapify (A,2,10)

→Max-Heapify (A,4,10)

→Max-Heapify (A,9,10)



End of Lecture 9

MakeHeap

- MakeHeap uses Max-Heapify to reorganize the tree from bottom to top to make it a heap.
- MakeHeap can be written concisely in either recursive or iterative form.

Recursive MakeHeap

Invoke as MakeHeap (*A*, *1*, *n*)

```
MakeHeap($A$,$i$,$n$)
<pre-cond>:A[iK n] is a balanced binary tree
<post-cond>:The subtree rooted at $i$ is a heap
if $i \le \lfloor n/4 \rfloor$ then
    MakeHeap($A$,$LEFT(i)$,$n$)
    MakeHeap($A$,$RIGHT(i)$,$n$)
Max-Heapify($A$,$i$,$n$)
```

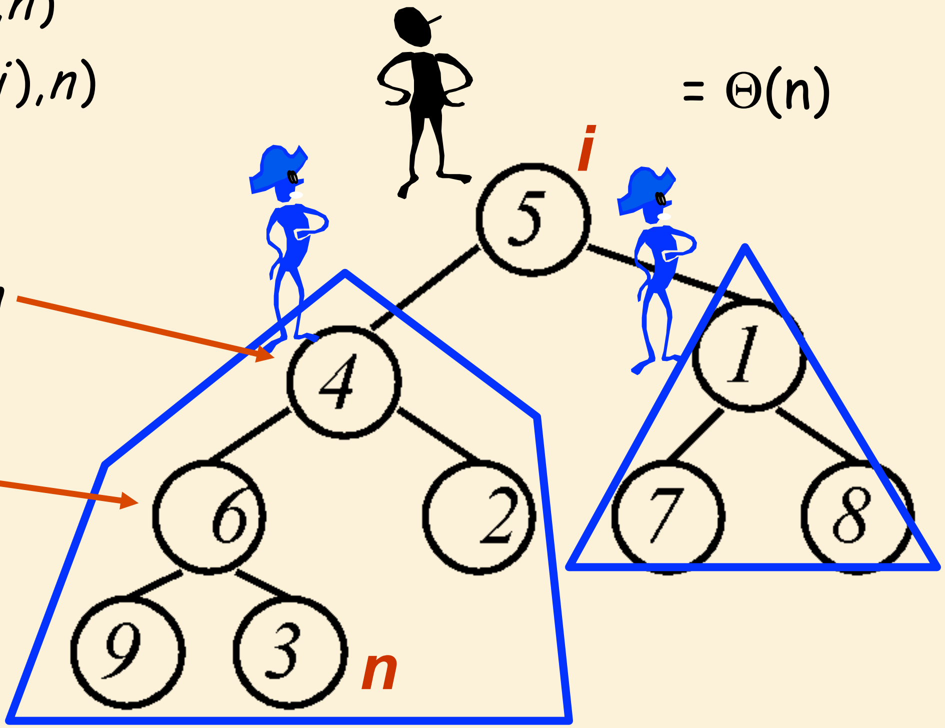
Running time:

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

$$= \Theta(n)$$

$\lfloor n/4 \rfloor$ is grandparent of n

$\lfloor n/2 \rfloor$ is parent of n

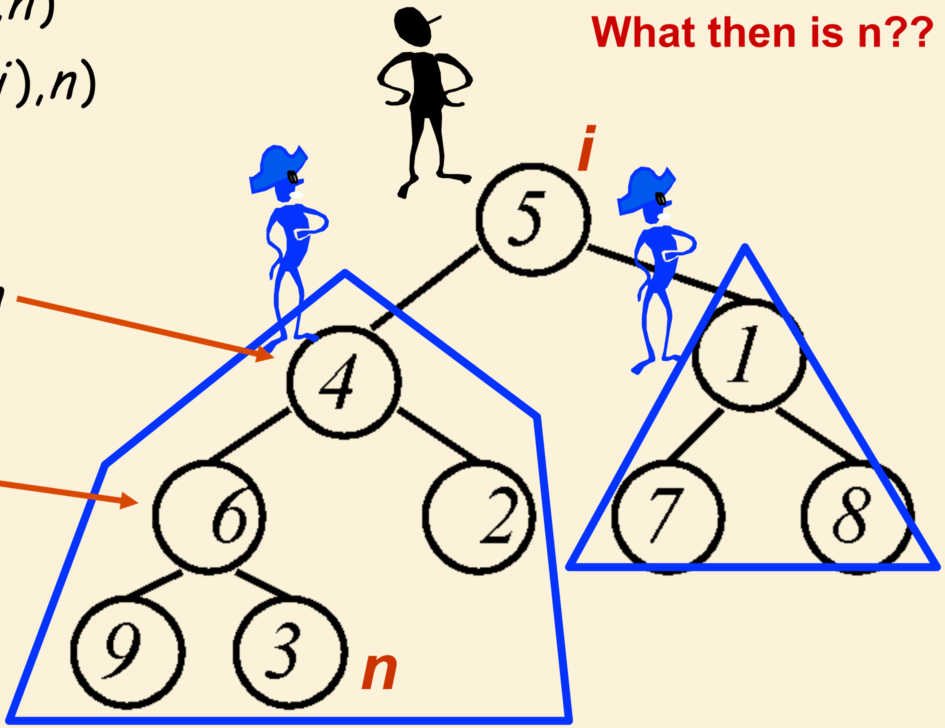


Recursive MakeHeap

```
MakeHeap($A$,$i$,$n$)
<pre-cond>:A[iK n] is a balanced binary tree
<post-cond>:The subtree rooted at $i$ is a heap
if $i \le \lfloor n/4 \rfloor$ then
    MakeHeap($A$,$LEFT(i)$,$n$)
    MakeHeap($A$,$RIGHT(i)$,$n$)
Max-Heapify($A$,$i$,$n$)
```

Question from last class: what if *i* = *4*??

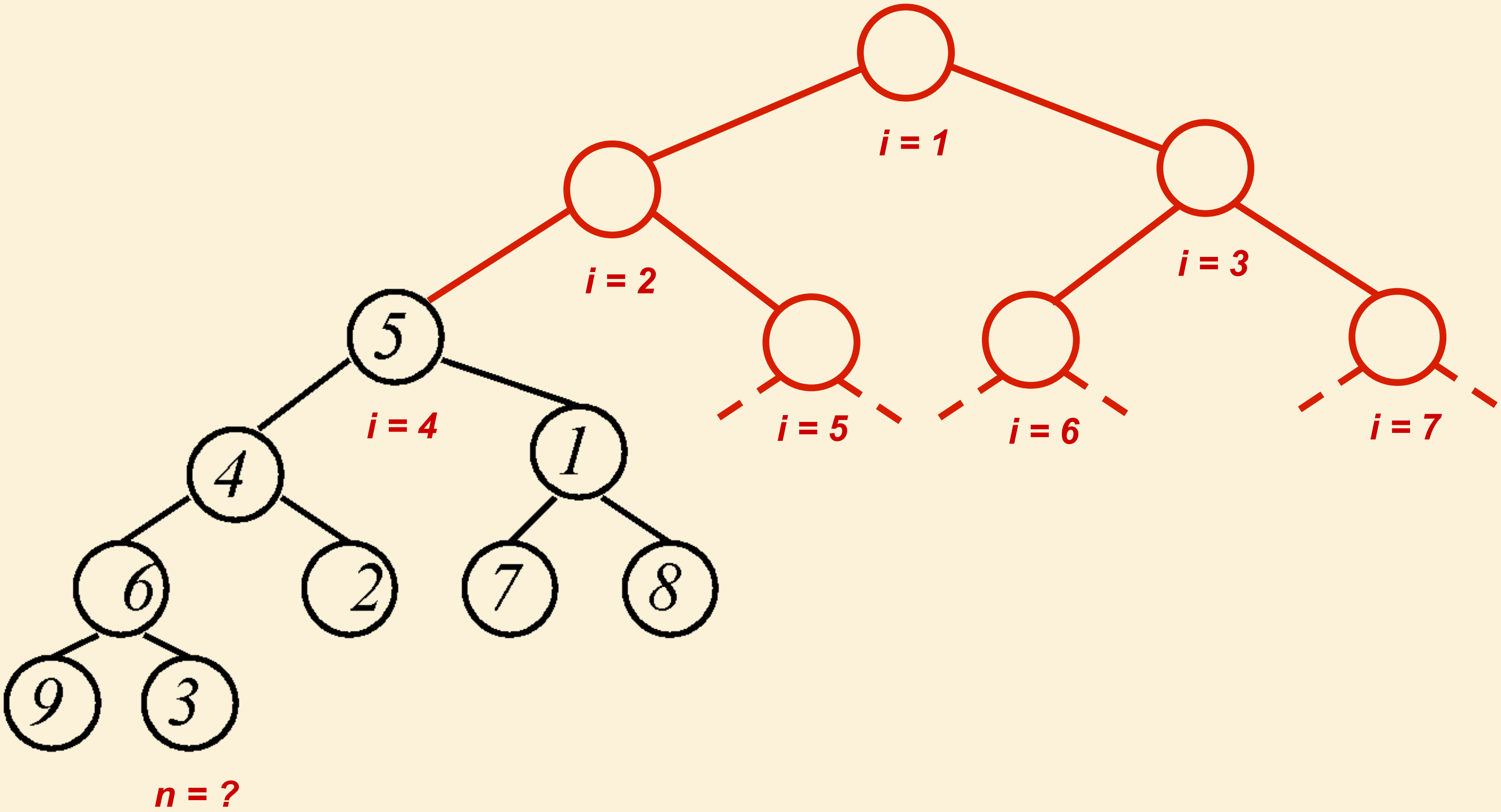
What then is n??



$\lfloor n/4 \rfloor$ is grandparent of n

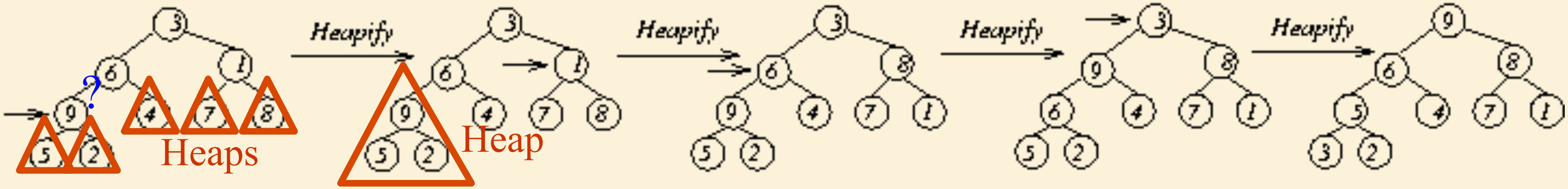
$\lfloor n/2 \rfloor$ is parent of n

Recursive MakeHeap



Iterative MakeHeap

```
MakeHeap(A,n)
<pre-cond>:A[1K n] is a balanced binary tree
<post-cond>:A[1K n] is a heap
for $i \leftarrow \lfloor n/2 \rfloor$ downto 1
    $< LI >$: All subtrees rooted at $i + 1K\ n$ are heaps
    Max-Heapify(A,i,n)
```



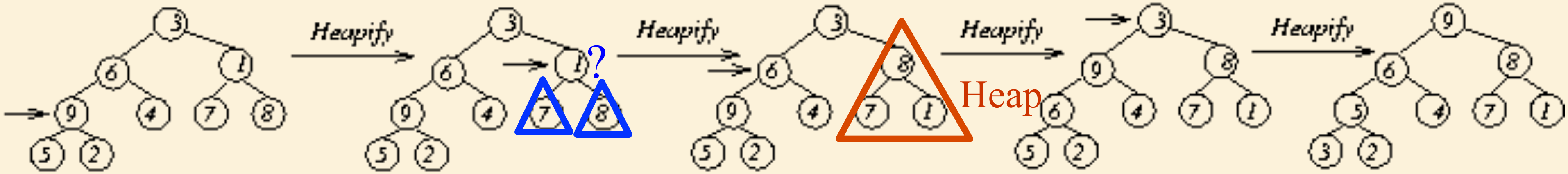
Max-Heapify(*A*, *i*, *n*)

<pre-cond>: Left and right subtrees of A[i] are max heaps.

<post-cond>: Subtree rooted at i is a heap.

Iterative MakeHeap

```
MakeHeap(A,n)
<pre-cond>:A[1K n] is a balanced binary tree
<post-cond>:A[1K n] is a heap
for $i \leftarrow \lfloor n/2 \rfloor$ downto 1
    $< LI >$: All subtrees rooted at $i+1$K $n$ are heaps
    Max-Heapify($A,i,n$)
```



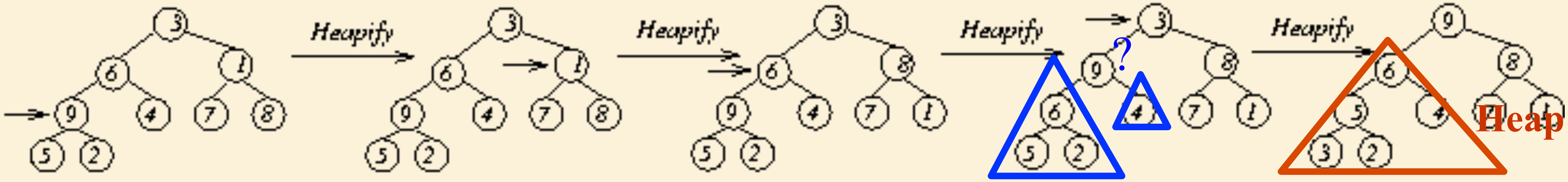
Max-Heapify(*A*, *i*, *n*)

<pre-cond>: Left and right subtrees of A[i] are max heaps.

<post-cond>: Subtree rooted at i is a heap.

Iterative MakeHeap

```
MakeHeap($A$,$n$)
<pre-cond>:A[1K n] is a balanced binary tree
<post-cond>:A[1K n] is a heap
for $i \leftarrow \lfloor n/2 \rfloor$ downto 1
    $< LI >$: All subtrees rooted at $i + 1\mathrm{K}\ n$ are heaps
    Max-Heapify($A$,$i$,$n$)
```



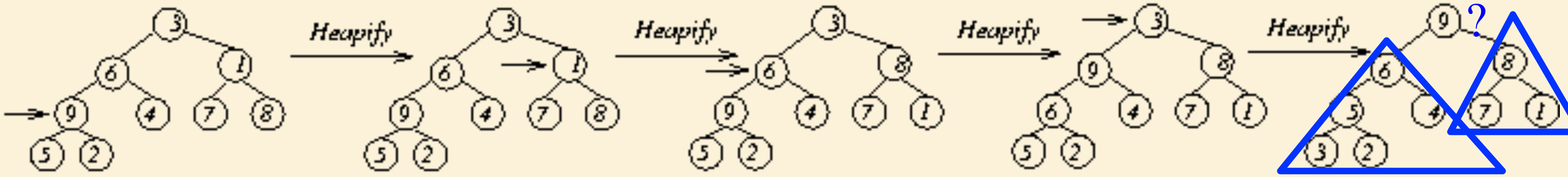
Max-Heapify(*A, i, n*)

<pre-cond>: Left and right subtrees of A[i] are max heaps.

<post-cond>: Subtree rooted at i is a heap.

Iterative MakeHeap

```
MakeHeap(A,n)
<pre-cond>:A[1K n] is a balanced binary tree
<post-cond>:A[1K n] is a heap
for $i \leftarrow \lfloor n/2 \rfloor$ downto 1
    $< LI >$: All subtrees rooted at $i + 1$K $n$ are heaps
    Max-Heapify(A,i,n)
```



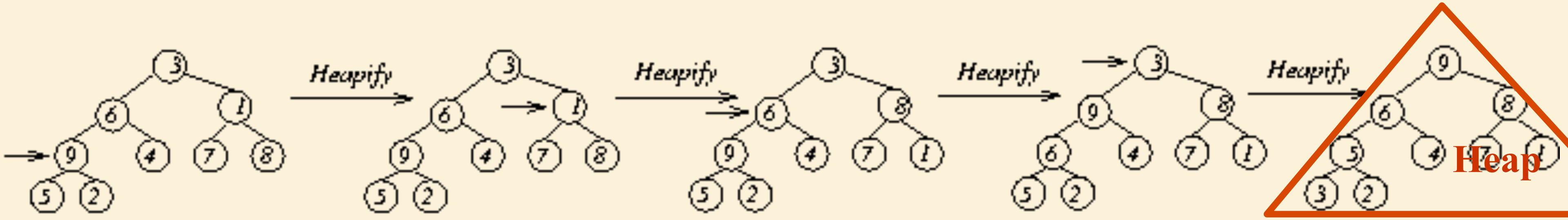
Max-Heapify(*A, i, n*)

<pre-cond>: Left and right subtrees of A[i] are max heaps.

<post-cond>: Subtree rooted at i is a heap.

Iterative MakeHeap

```
MakeHeap(A,n)
<pre-cond>:A[1K n] is a balanced binary tree
<post-cond>:A[1K n] is a heap
for $i \leftarrow \lfloor n/2 \rfloor$ downto 1
    $< LI >$: All subtrees rooted at $i + 1K\ n$ are heaps
    Max-Heapify(A,i,n)
```



Max-Heapify(*A, i, n*)

<pre-cond>: Left and right subtrees of A[i] are max heaps.

<post-cond>: Subtree rooted at i is a heap.

Iterative MakeHeap

```
MakeHeap($A,n$)
<pre-cond>:A[$1\ldots n$] is a balanced binary tree
<post-cond>:A[$1\ldots n$] is a heap
for $i \leftarrow \lfloor n/2 \rfloor$ downto 1
    $< LI >$: All subtrees rooted at $i+1\ldots n$ are heaps
    Max-Heapify($A,i,n$)
```

Runtime:

Height of heap $= \lfloor \log_2 n \rfloor$

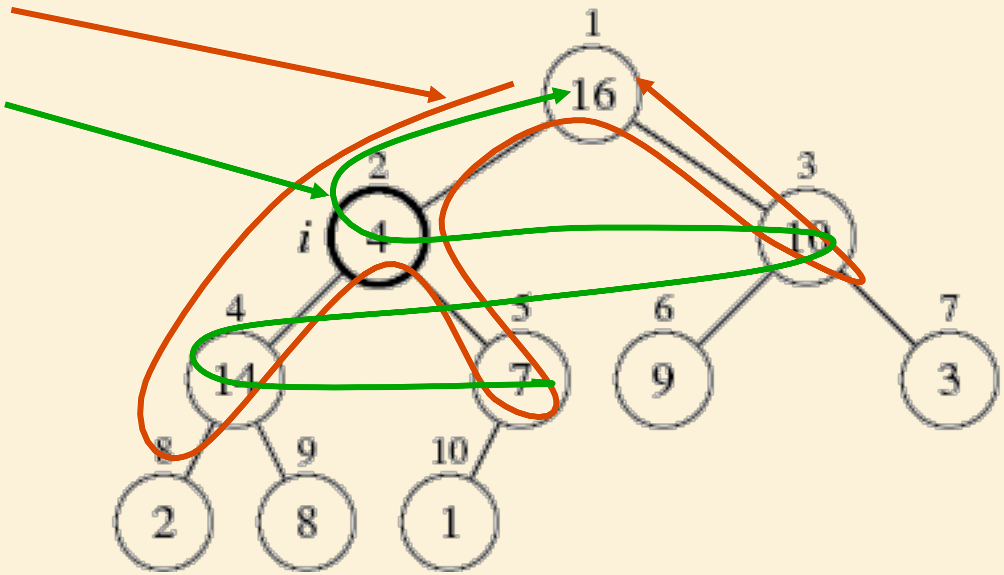
It can be shown that the number of nodes at height h $\leq \left\lceil \frac{n}{2^{h+1}} \right\rceil$

Time to heapify from node at height h$\in O(h)$

$$\rightarrow T(n) = \sum_{h=0}^{\lfloor \log_2 n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) \quad = O\left(n \sum_{h=0}^{\lfloor \log_2 n \rfloor} \frac{h}{2^h} \right) = O(n)$$

Iterative MakeHeap

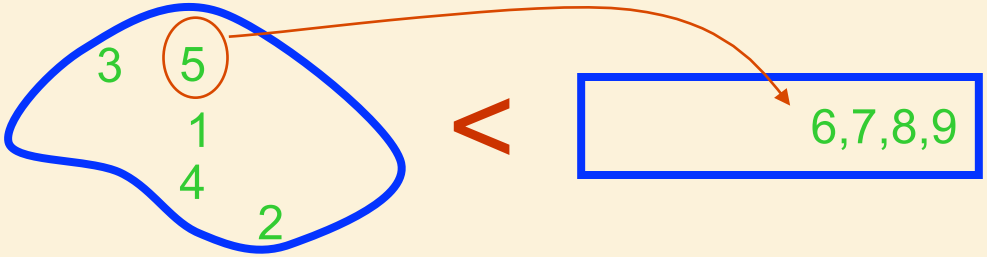
- Recursive and Iterative MakeHeap do essentially the same thing: Heapify from bottom to top.
- Difference:
  - Recursive is "depth-first"
  - Iterative is "breadth-first"



Using Heaps for Sorting

Selection Sort

Largest i values are sorted on the right.
Remaining values are off to the left.



Max is easier to find if a heap.

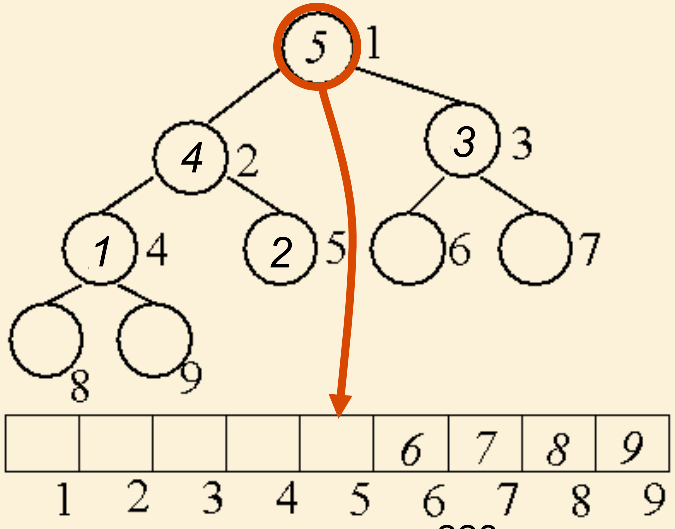
Heap Sort

HeapSort(*A*,*n*)

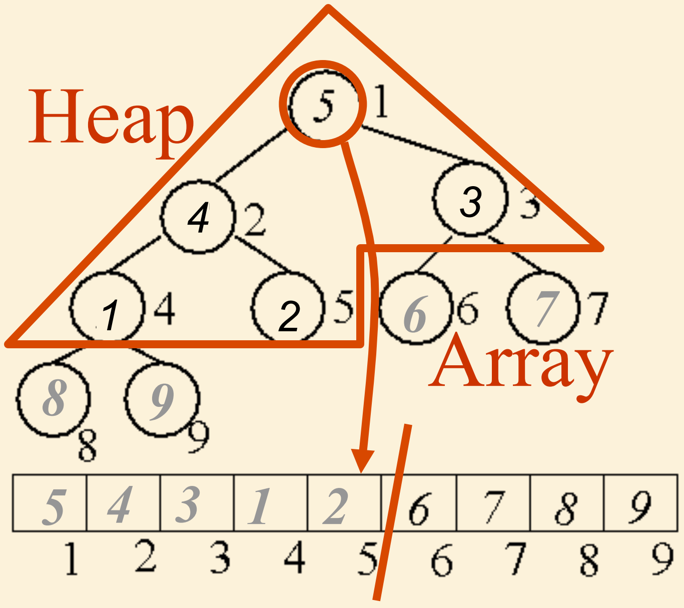
<pre-cond>:A[1...n] is a list of keys

<post-cond>:A[1...n] is sorted in non-decreasing order

Largest i values are sorted on side.
Remaining values are in a heap.



Heap Data Structure

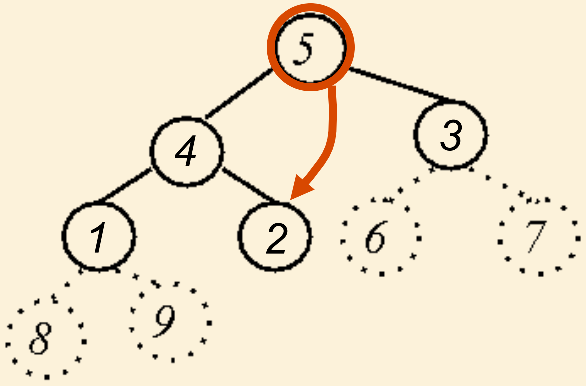


Heap

Array

5	4	3	1	2	6	7	8	9
1	2	3	4	5	6	7	8	9

Heap Array

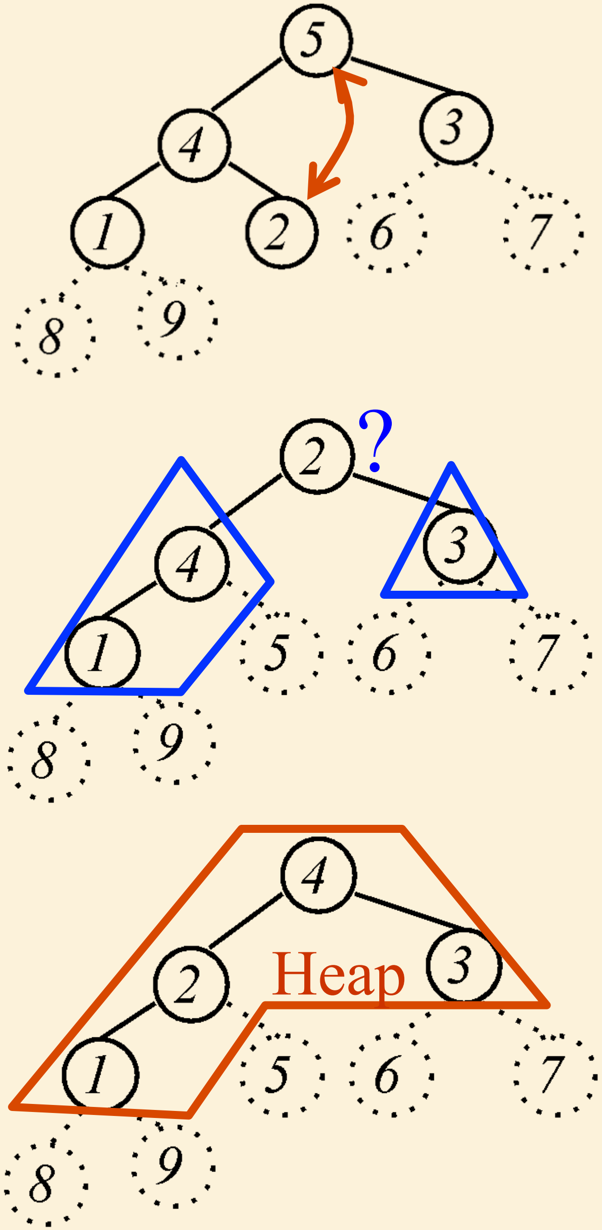


Heap Sort

Largest i values
are sorted on side.

Remaining values are
in a heap

Put next value
where it belongs.

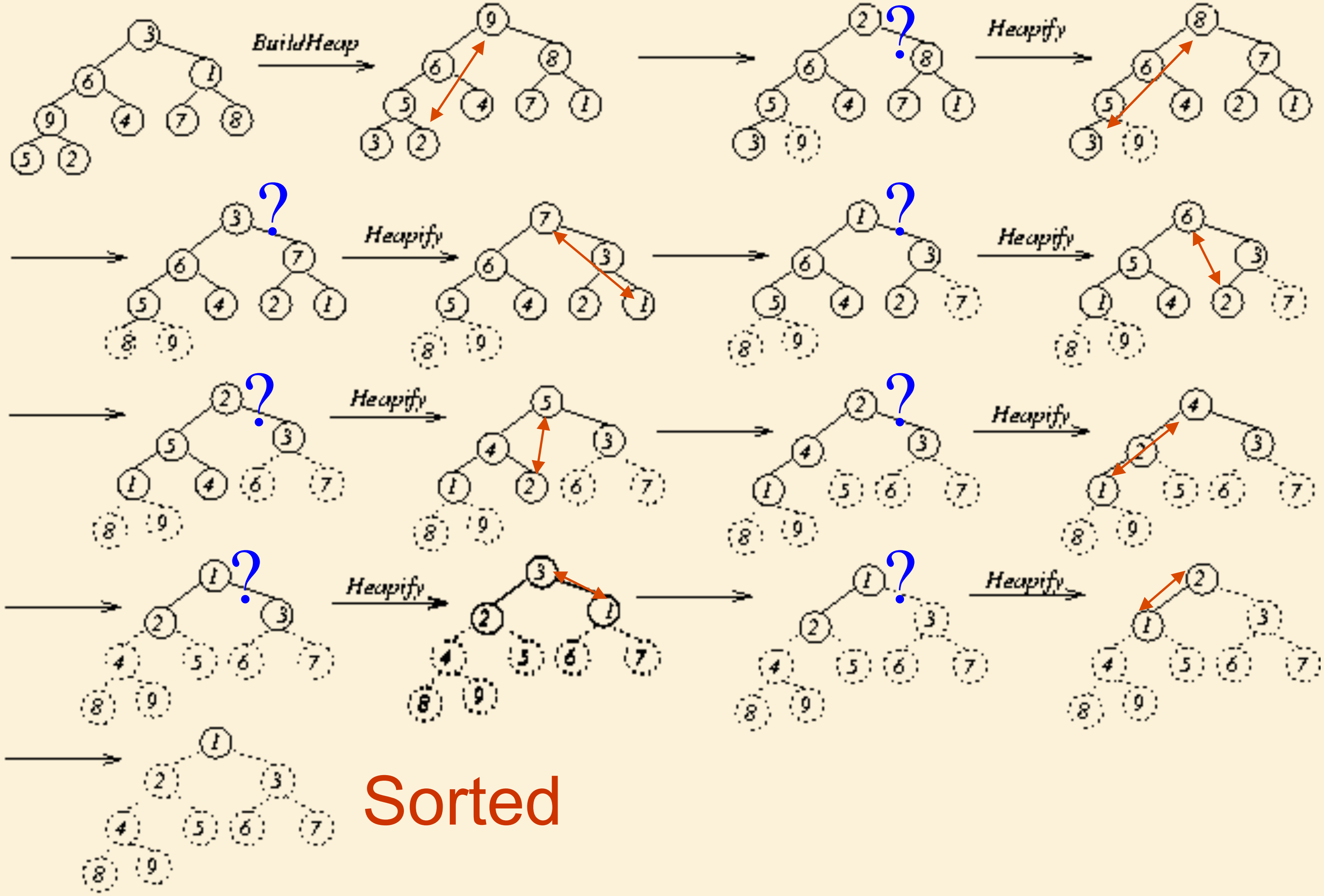


Max-Heapify(*A, i, n*)

<pre-cond>: Left and right subtrees of A[i] are max heaps.

<post-cond>: Subtree rooted at i is a heap.

Heap Sort


BuildHeap
Heapify
Heapify
Heapify
Heapify
Heapify
Heapify
Heapify
?
?
?
?
?
?
?
?
Sorted

Heap Sort

```
HeapSort($A$,$n$)
<pre-cond>:A[1...n] is a list of keys
<post-cond>:A[1...n] is sorted in non-decreasing order
MakeHeap(A,n)
for $i \leftarrow n$ downto 2
    $< LI >$: $A[1K\ i]$ is a heap
              $A[i+1K\ n]$ contains the largest keys in non-decreasing order
    exchange $A[1] \leftrightarrow A[i]$
    Max-Heapify($A$,1,$i-1$)
```

Running Time:

MakeHeap takes $\Theta(n)$ time.

Heapifing a tree of size i takes $\log(i)$.

$T(n) = \Theta(n) + \sum_{i=n}^{1} \log i.$ This sum is arithmetic.

$T(n)$ =n × maximum value $= \Theta(n \log n)$.

Other Applications of Heaps

Priority Queue

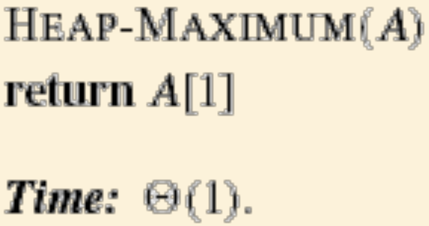
- Maintains dynamic set, A, of n elements, each with a key.
- Max-priority queue supports:

1. MAXIMUM(A)
2. EXTRACT-MAX(A, n)
3. INCREASE-KEY(A, i, key)
4. INSERT(A, key, n)

- Example Application: Schedule jobs on a shared computer.

Priority Queues cont'd...

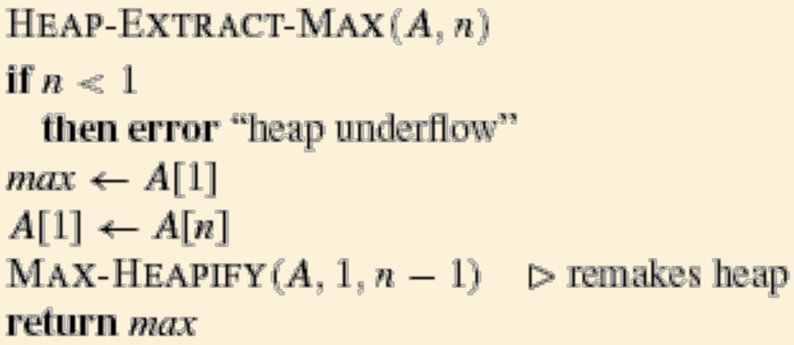
- MAXIMUM(A):



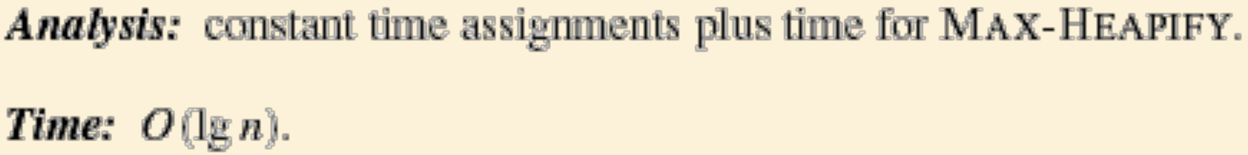
```
HEAP-MAXIMUM(A)
return A[1]
```

Time: $\Theta(1)$.

- EXTRACT-MAX(A,n):



```
HEAP-EXTRACT-MAX(A, n)
if $n < 1$
  then error "heap underflow"
$max \leftarrow A[1]$
$A[1] \leftarrow A[n]$
MAX-HEAPIFY(A, 1, $n - 1$)    ▷ remakes heap
return $max$
```

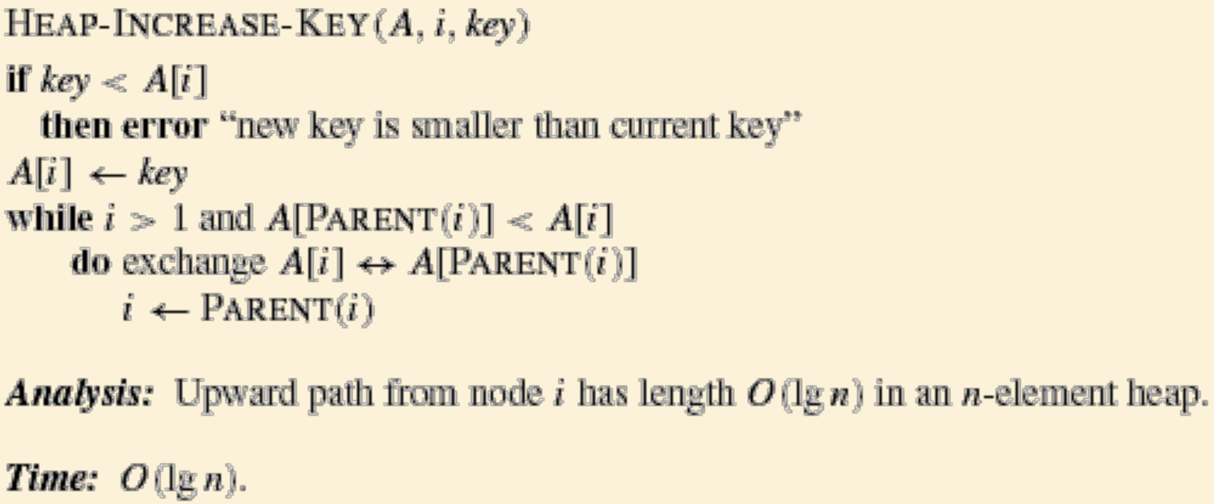


Analysis: constant time assignments plus time for MAX-HEAPIFY.

Time: $O(\lg n)$.

Priority Queue cont'd...

- INCREASE-KEY(A, i, key):

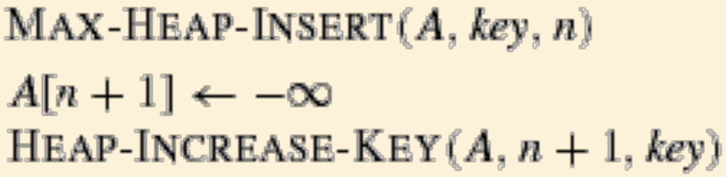


```
HEAP-INCREASE-KEY(A, i, key)
if key < A[i]
   then error "new key is smaller than current key"
A[i] ← key
while i > 1 and A[PARENT(i)] < A[i]
     do exchange A[i] ↔ A[PARENT(i)]
        i ← PARENT(i)
```

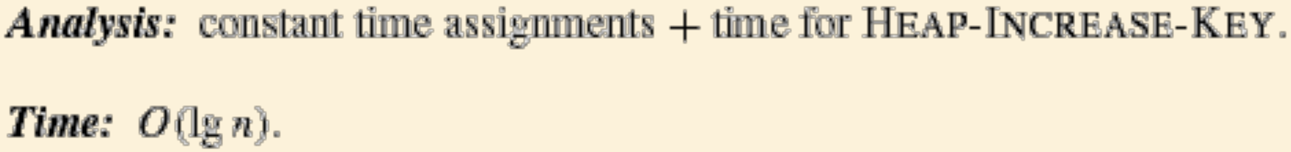
Analysis: Upward path from node i has length $O(\lg n)$ in an n-element heap.

Time: $O(\lg n)$.

- INSERT(A, key, n):



```
MAX-HEAP-INSERT(A, key, n)
A[n + 1] ← −∞
HEAP-INCREASE-KEY(A, n + 1, key)
```



Analysis: constant time assignments + time for HEAP-INCREASE-KEY.

Time: $O(\lg n)$.

Binary Search Trees

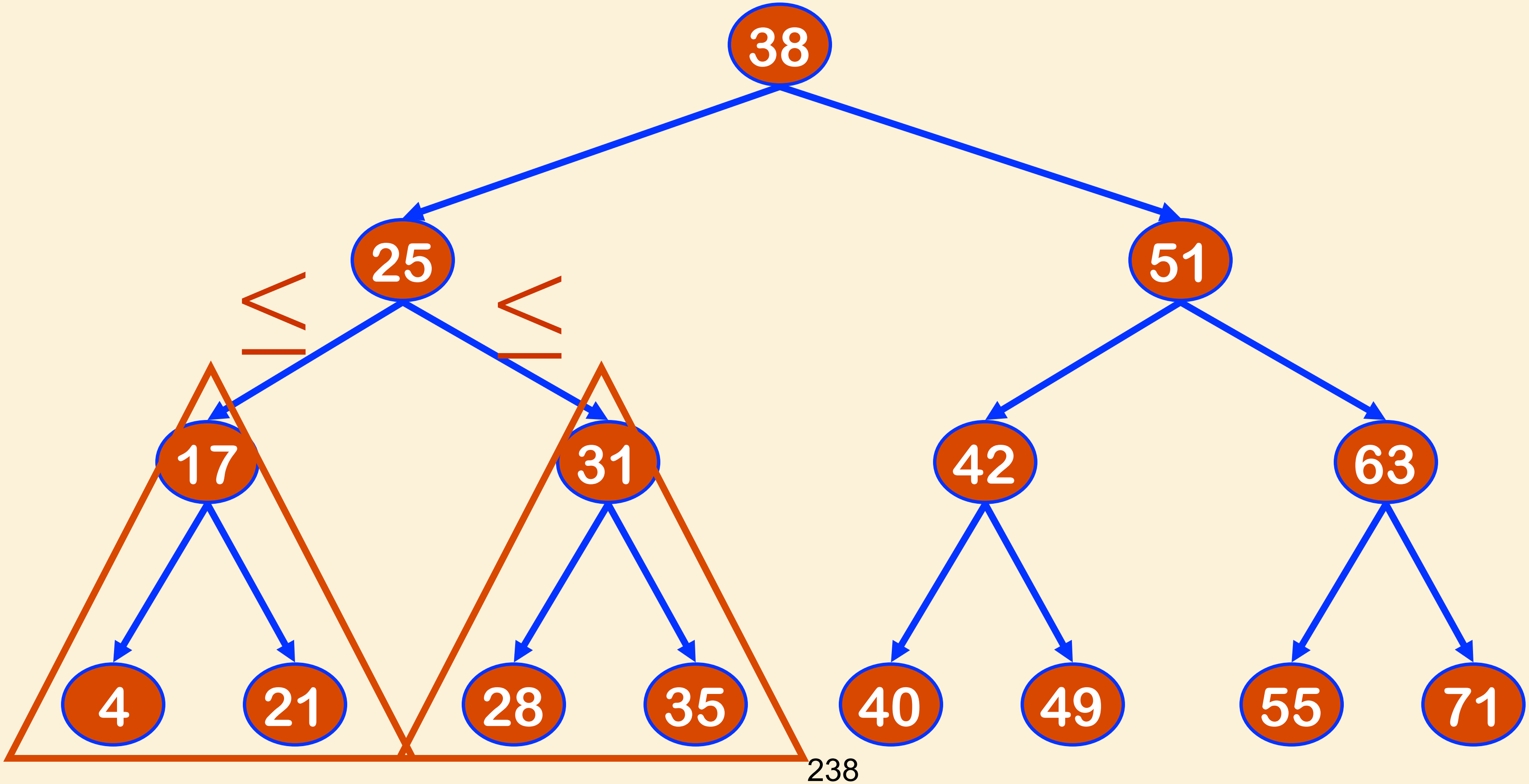
- Support many dynamic-set operations
- Basic operations take time proportional to height ***h*** of tree.

$\Theta(\log n)$ for balanced tree

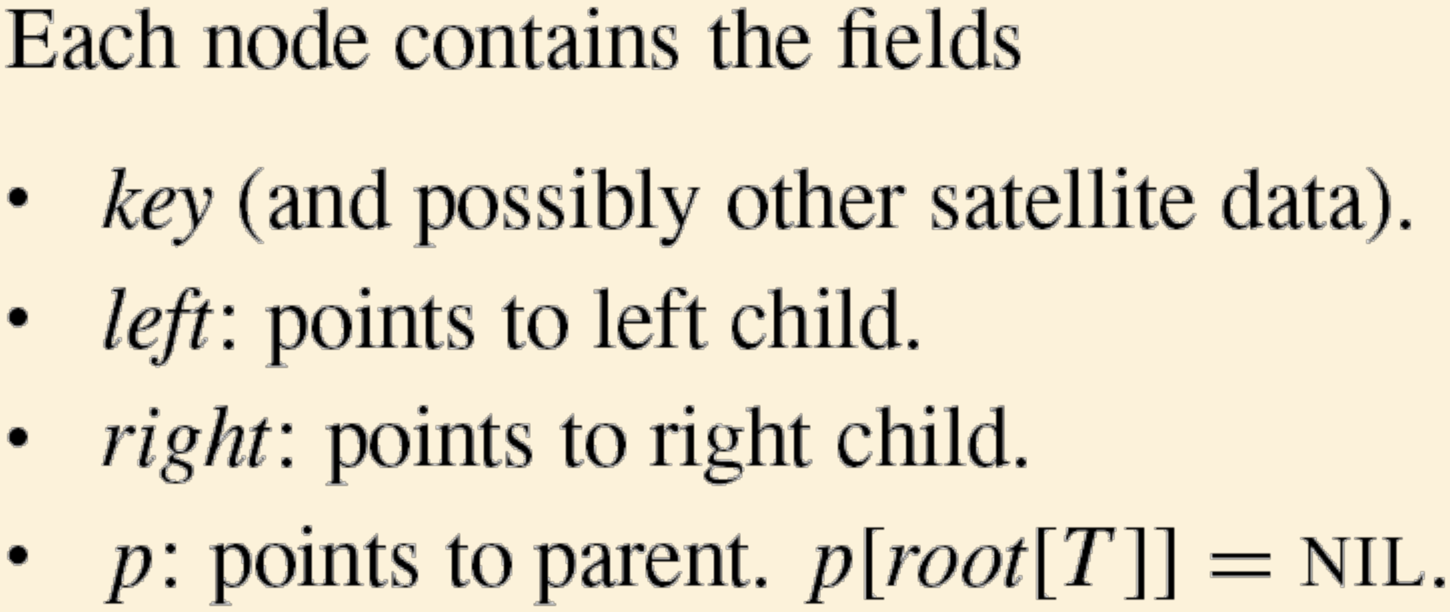
$\Theta(n)$ for worst-cased unbalanced tree

Binary Search Tree

Left children $\leq$ Node $\leq$ Right children



BST Data Structure



Each node contains the fields

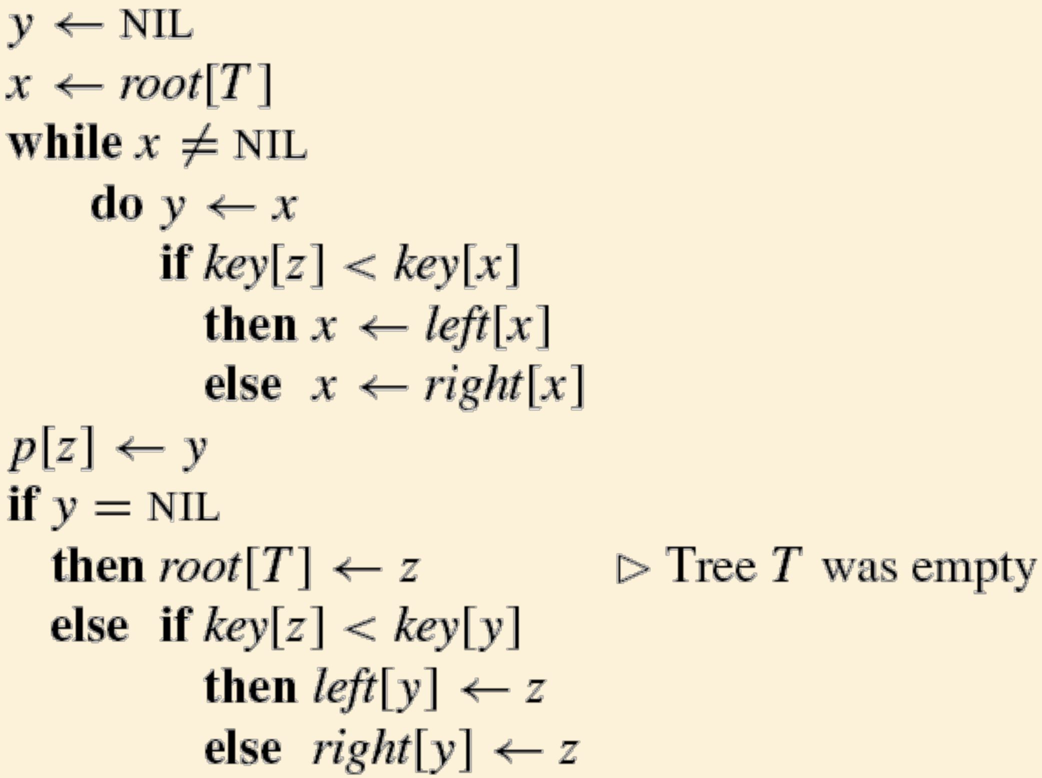
- *key* (and possibly other satellite data).
- *left*: points to left child.
- *right*: points to right child.
- *p*: points to parent. $p[root[T]] = \text{NIL}$.

Insertion

Tree-Insert(T,z)

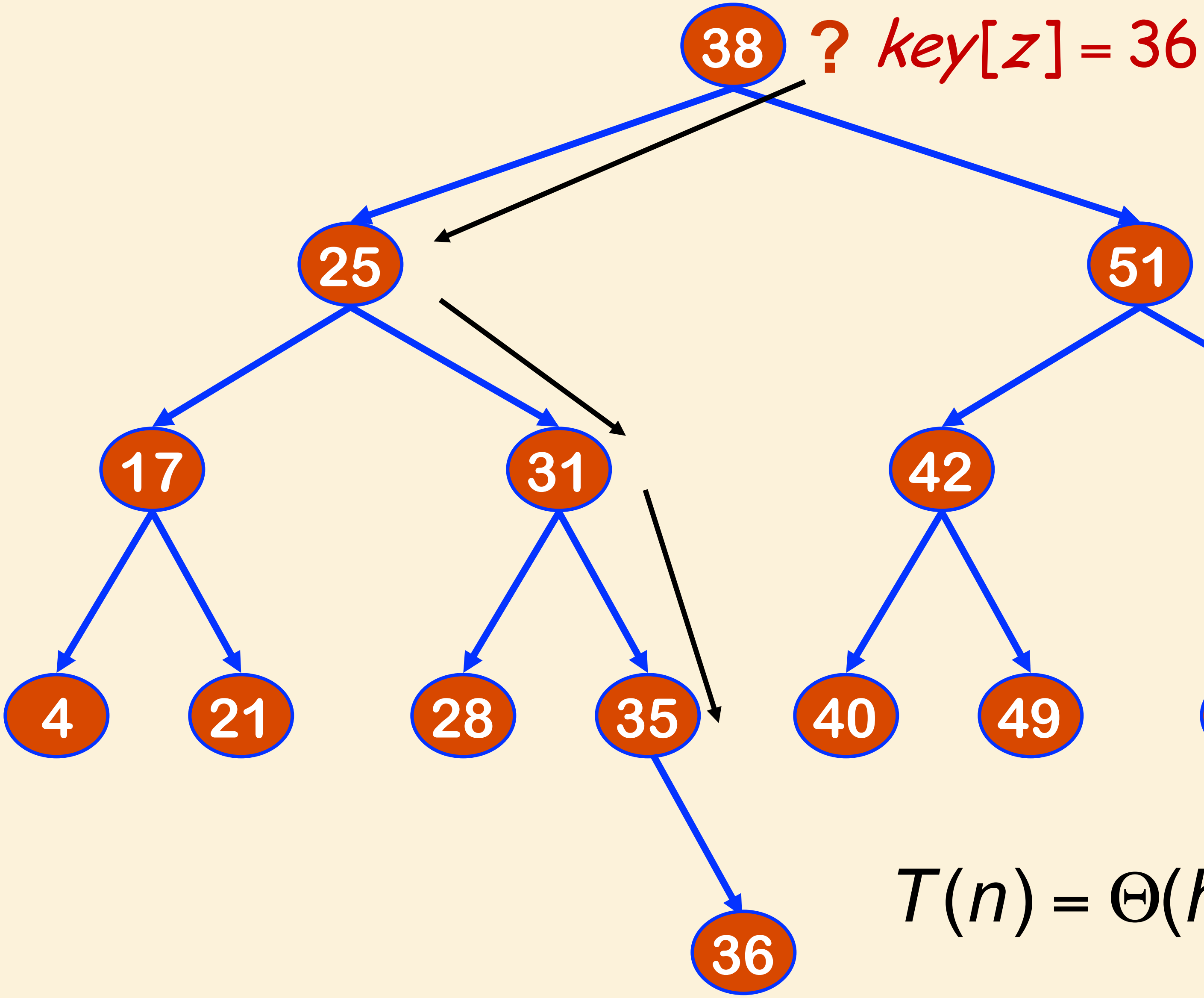
<pre-cond>: T is a BST, z a node to be inserted

<post-cond>: T is a BST with z inserted



```
y ← NIL
x ← root[T]
while x ≠ NIL
      do y ← x
         if key[z] < key[x]
           then x ← left[x]
           else  x ← right[x]
p[z] ← y
if y = NIL
   then root[T] ← z                    ▷ Tree T was empty
   else  if key[z] < key[y]
           then left[y] ← z
           else  right[y] ← z
```

Insertion



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

Building a Tree

```
Build-BST(Z)
<pre-cond>: $Z$ is a set of nodes
<post-cond>: Returns a BST consisting of the nodes in $Z$
$T \leftarrow$ NIL
for $z$ in $Z$
    Tree-Insert($T$,$z$)
```

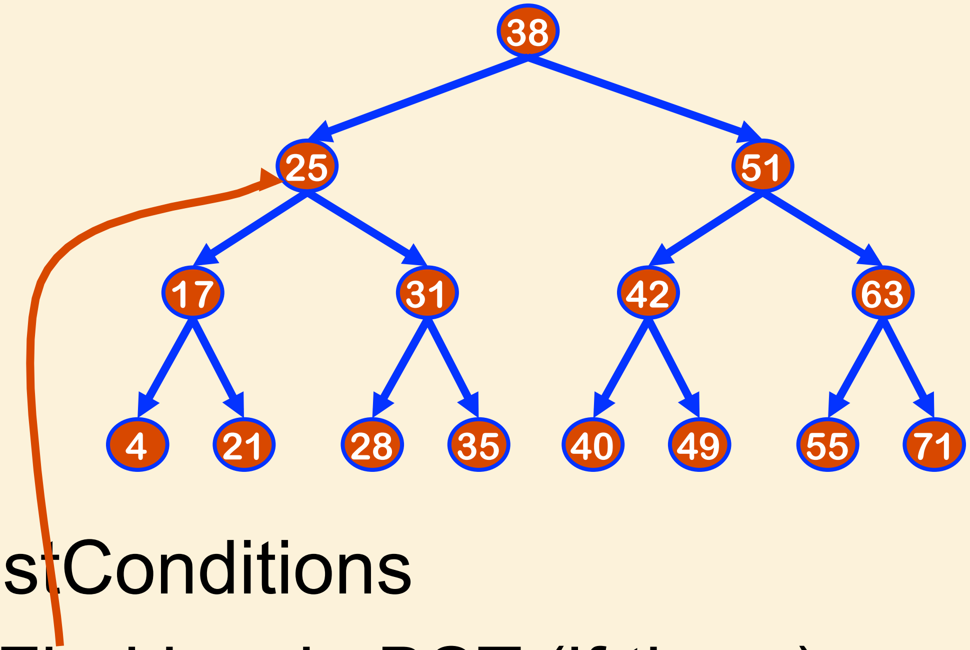
Time for each insertion $= \Theta(h)$

For balanced tree, number of nodes inserted into tree of height h is 2^h

Thus $T(n) = \sum_{h=0}^{\lfloor \log_2 n \rfloor} 2^h \Theta(h) = \Theta(n \log n)$

Searching the Tree

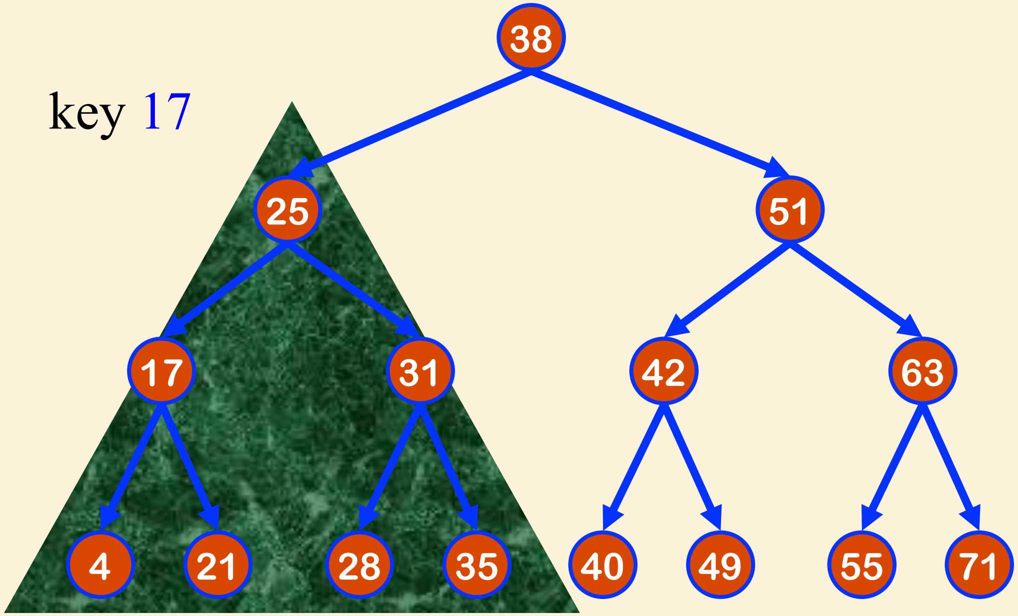
- PreConditions
  - Key 25
  - A binary search tree.



- PostConditions
  - Find key in BST (if there).

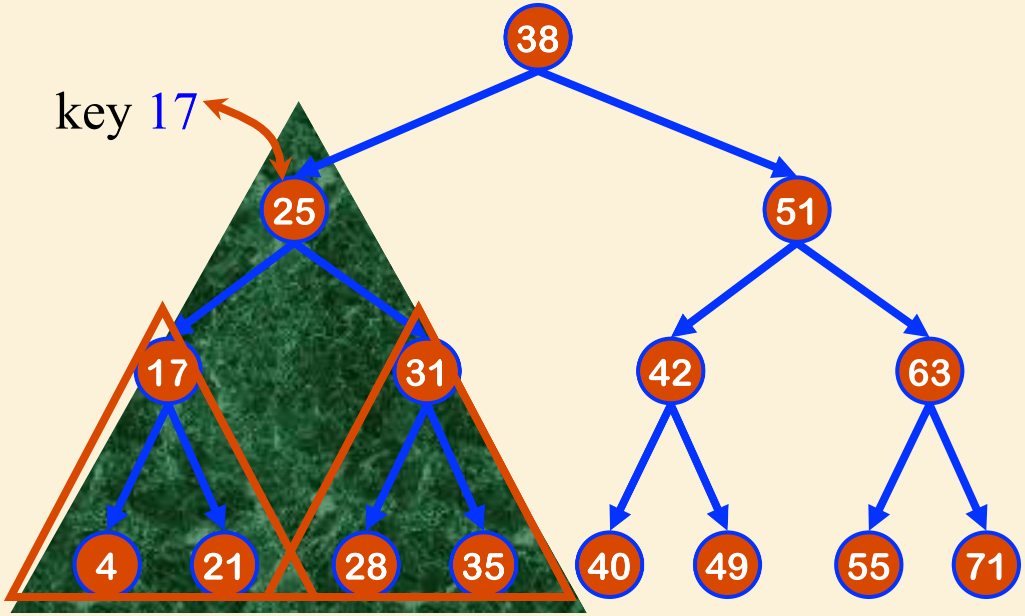
Searching the Tree

- Maintain a sub-tree.
- If the key is contained in the original tree, then the key is contained in the sub-tree.



Define Step

- Cut sub-tree in half.
- Determine which half the key would be in.
- Keep that half.



If key < root, then key is in left half.

If key = root, then key is found

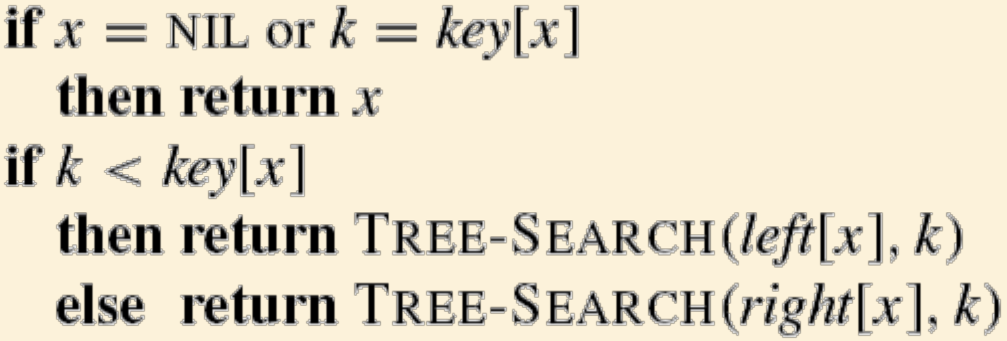
If key > root, then key is in right half.

Searching the Tree

Tree-Search(x,k)

<pre-cond>:x is a BST, k is a key to search for

<post-cond>: returns the node matching k if it exists or NIL otherwise



```
if $x = \text{NIL}$ or $k = key[x]$
    then return $x$
if $k < key[x]$
    then return TREE-SEARCH($left[x], k$)
    else return TREE-SEARCH($right[x], k$)
```

Runtime $= \Theta(h)$

Why use (balanced) binary search trees?

- What is the advantage over a sorted linear array?
  - Search time is the same
  - However, maintaining (inserting, deleting, modifying) is
    - $\Theta(\log n)$ for balanced BSTs
    - $\Theta(n)$ for arrays