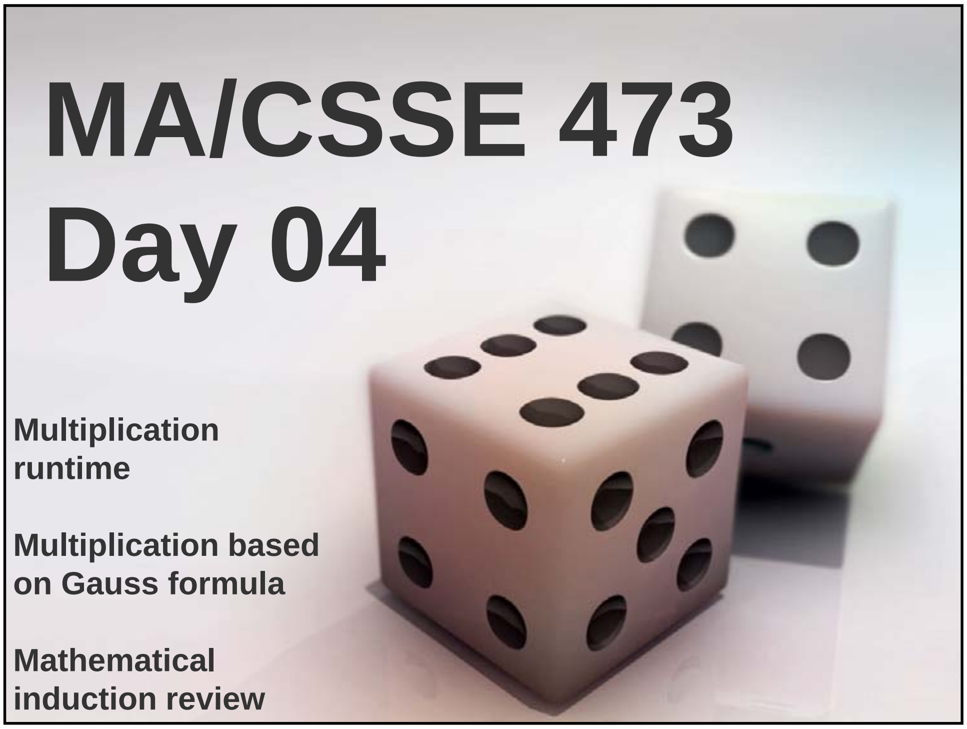
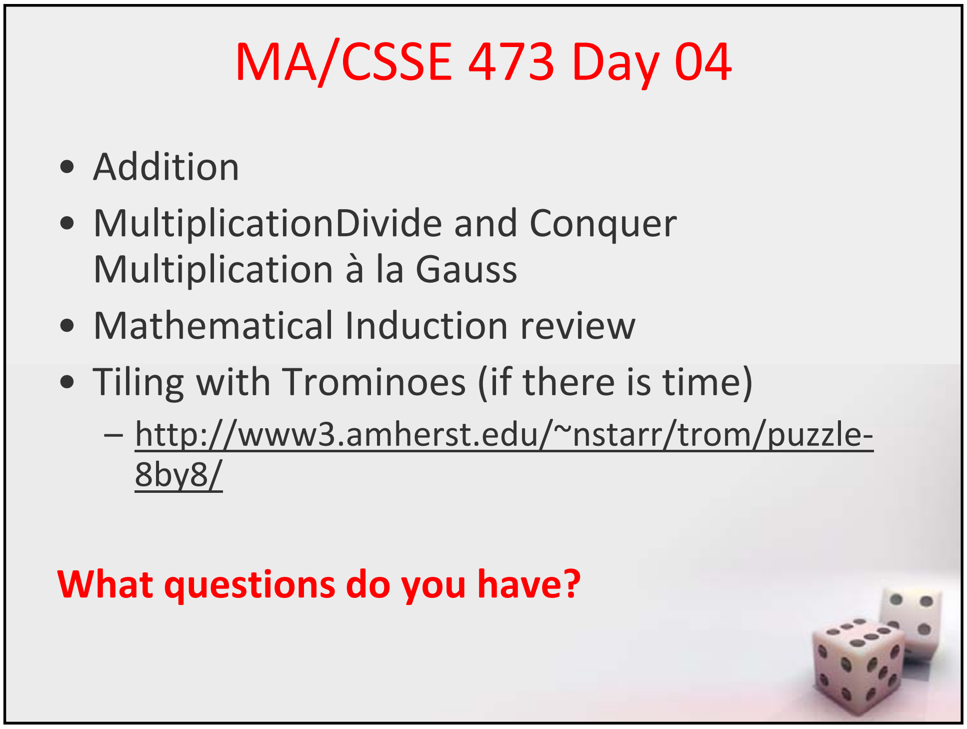


MA/CSSE 473
Day 04

Multiplication runtime

Multiplication based on Gauss formula

Mathematical induction review

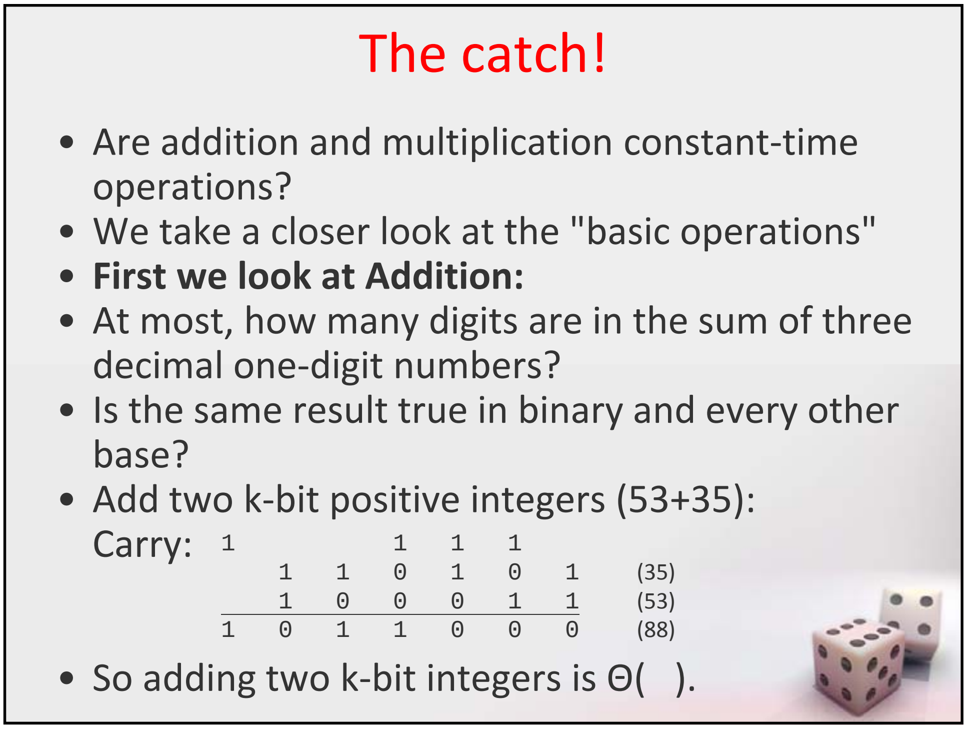


MA/CSSE 473 Day 04

- Addition
- MultiplicationDivide and Conquer
  Multiplication à la Gauss
- Mathematical Induction review
- Tiling with Trominoes (if there is time)
  - http://www3.amherst.edu/~nstarr/trom/puzzle-8by8/

What questions do you have?

EFFICIENT INTEGER ADDITION AND MULTIPLICATION



The catch!

- Are addition and multiplication constant-time operations?
- We take a closer look at the "basic operations"
- **First we look at Addition:**
- At most, how many digits are in the sum of three decimal one-digit numbers?
- Is the same result true in binary and every other base?
- Add two k-bit positive integers (53+35):

```
Carry:  1           1     1     1
              1     1     0     1     0     1       (35)
              1     0     0     0     1     1       (53)
        -----------------------------------------
        1     0     1     1     0     0     0       (88)
```

- So adding two k-bit integers is Θ(   ).

"Ordinary" Multiplication of two k-bit numbers

- Example: multiply 13 by 11

```
            1  1  0  1
         x  1  0  1  1
----------------------
            1  1  0  1    (1101 times 1)
         1  1  0  1       (1101 times 1, shifted once)
      0  0  0  0          (1101 times 0, shifted twice)
   1  1  0  1             (1101 times 1, shifted thrice)
----------------------
1  0  0  0  1  1  1  1    (binary 143)
```

- There are k shift operations, followed by addition of k rows of 2k bits each, so the whole multiplication is Θ( ) ?
- Can we do better?

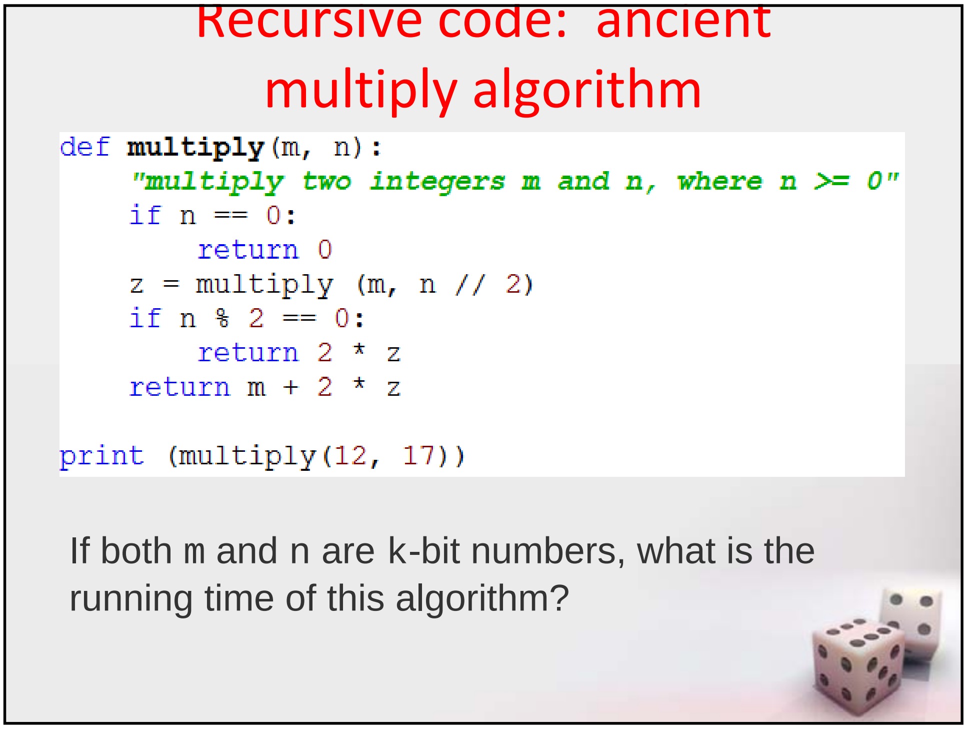
Multiplication by an Ancient Method

- This approach was known to Al Khwarizimi
- According to Dasgupta, *et al*, still used today in some European countries
- Repeat until 1st number is 1, keeping all results:
  - Divide 1st number by 2 (rounding down)
  - double 2nd number
- Example

11	13
5	26
~~2~~	~~52~~
1	104
	143

Then strike out any rows whose first number is even, and add up the remaining numbers in the second column.

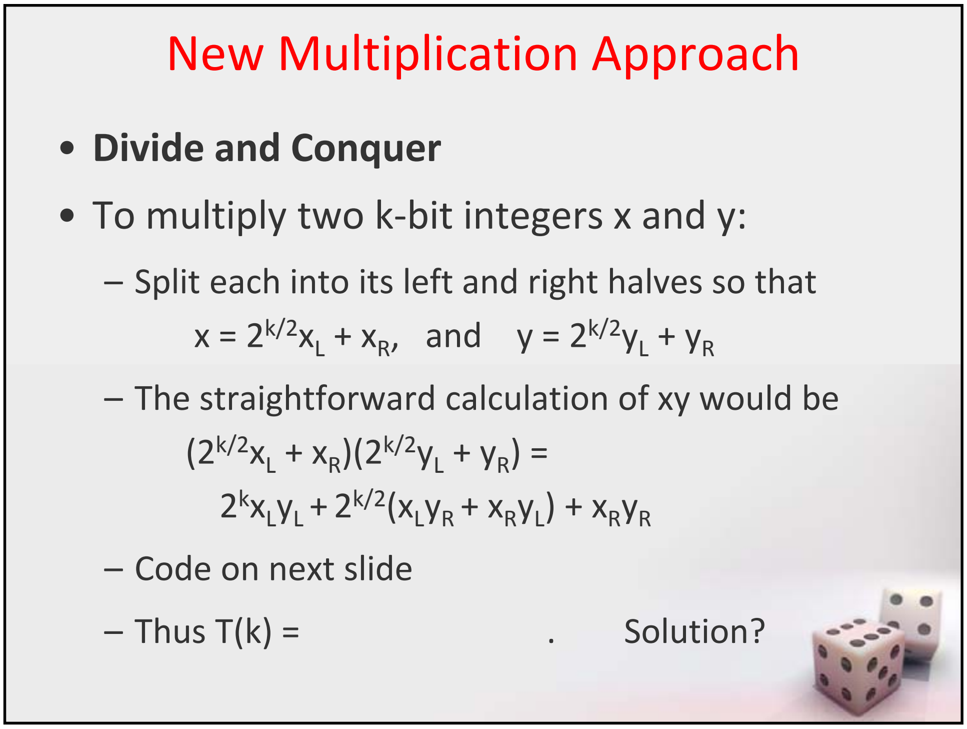
- Correct? Analysis



Recursive code: ancient multiply algorithm

```
def multiply(m, n):
    "multiply two integers m and n, where n >= 0"
    if n == 0:
        return 0
    z = multiply (m, n // 2)
    if n % 2 == 0:
        return 2 * z
    return m + 2 * z

print (multiply(12, 17))
```

If both m and n are k-bit numbers, what is the running time of this algorithm?



New Multiplication Approach

- **Divide and Conquer**
- To multiply two k-bit integers x and y:
  - Split each into its left and right halves so that
    $x = 2^{k/2}x_L + x_R$, and $y = 2^{k/2}y_L + y_R$
  - The straightforward calculation of xy would be
    $(2^{k/2}x_L + x_R)(2^{k/2}y_L + y_R) =$
    $2^k x_L y_L + 2^{k/2}(x_L y_R + x_R y_L) + x_R y_R$
  - Code on next slide
  - Thus T(k) = . Solution?

For reference: The Master Theorem

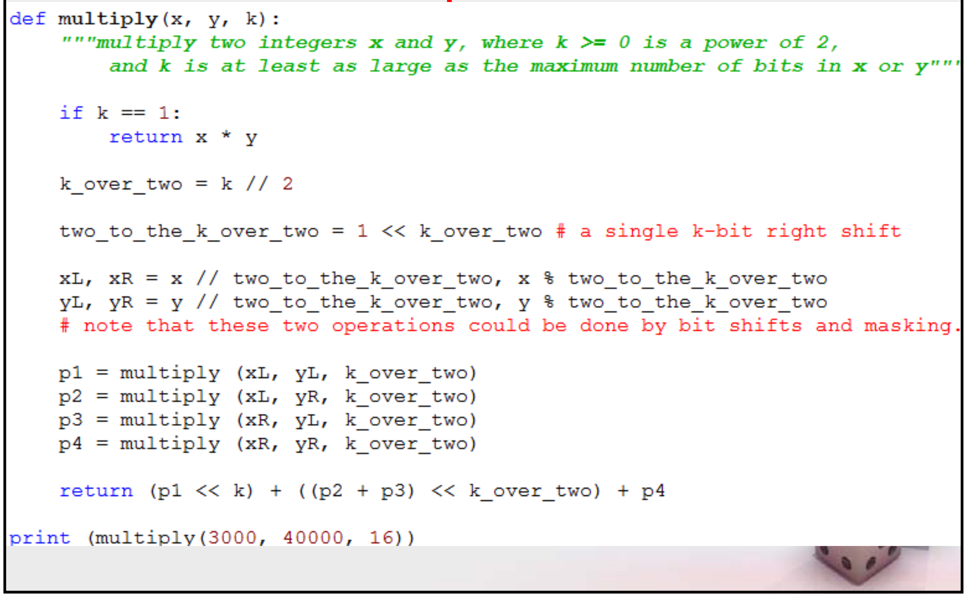
- The Master Theorem for Divide and Conquer recurrence relations:
- Consider the recurrence $T(n) = aT(n/b) + f(n)$, $T(1)=c$, where $f(n) = \Theta(n^k)$ and $k \geq 0$,
- The solution is
  - $\Theta(n^k)$ if $a < b^k$
  - $\Theta(n^k \log n)$ if $a = b^k$
  - $\Theta(n^{\log_b a})$ if $a > b^k$

For details, see Levitin pages 483-485 or Weiss section 7.5.3.

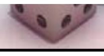
Grimaldi's Theorem 10.1 is a special case of the Master Theorem.

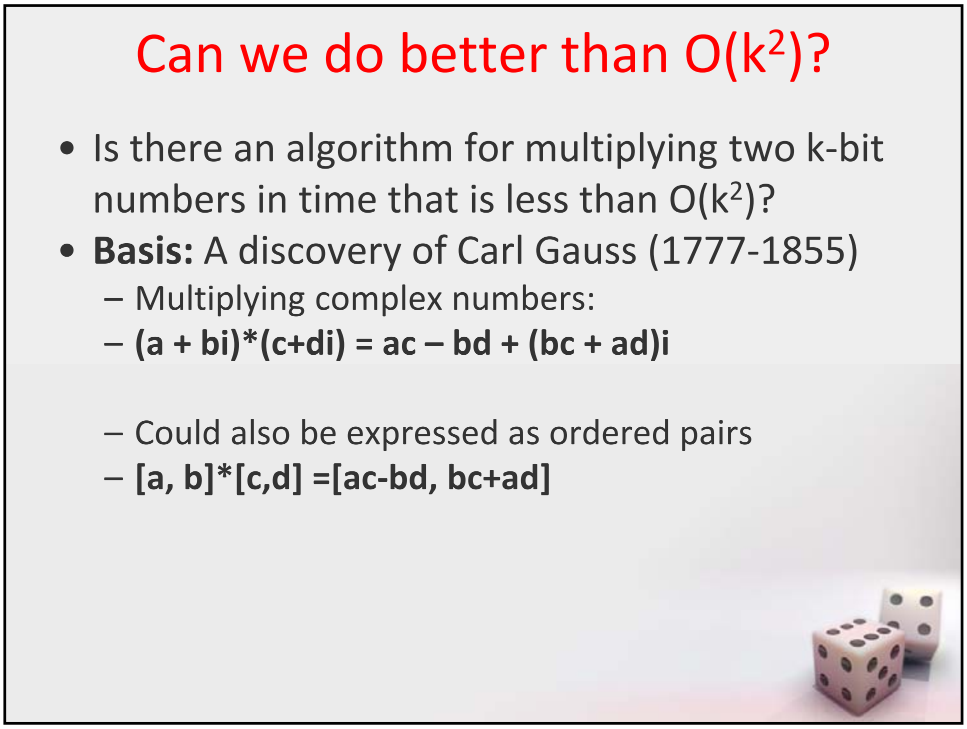
We will use this theorem often. You should review its proof soon (Weiss's proof is a bit easier than Levitin's).

Code for divide-and-conquer multiplication



```
def multiply(x, y, k):
    """multiply two integers x and y, where k >= 0 is a power of 2,
        and k is at least as large as the maximum number of bits in x or y"""

    if k == 1:
        return x * y

    k_over_two = k // 2

    two_to_the_k_over_two = 1 << k_over_two # a single k-bit right shift

    xL, xR = x // two_to_the_k_over_two, x % two_to_the_k_over_two
    yL, yR = y // two_to_the_k_over_two, y % two_to_the_k_over_two
    # note that these two operations could be done by bit shifts and masking.

    p1 = multiply (xL, yL, k_over_two)
    p2 = multiply (xL, yR, k_over_two)
    p3 = multiply (xR, yL, k_over_two)
    p4 = multiply (xR, yR, k_over_two)

    return (p1 << k) + ((p2 + p3) << k_over_two) + p4

print (multiply(3000, 40000, 16))
```

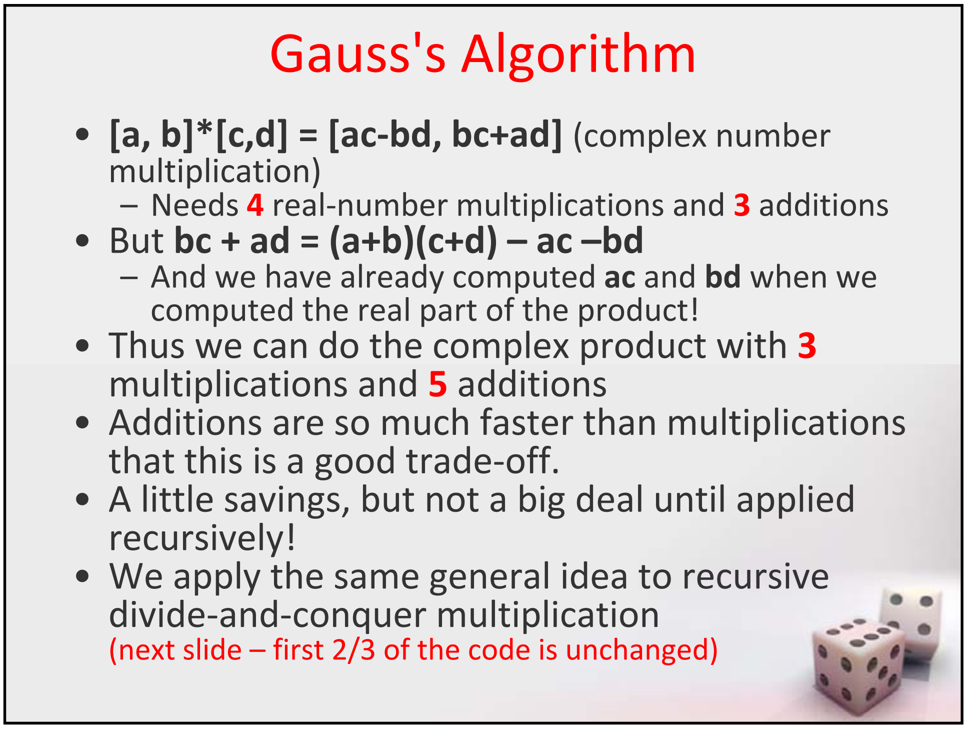




Can we do better than O(k^2)?

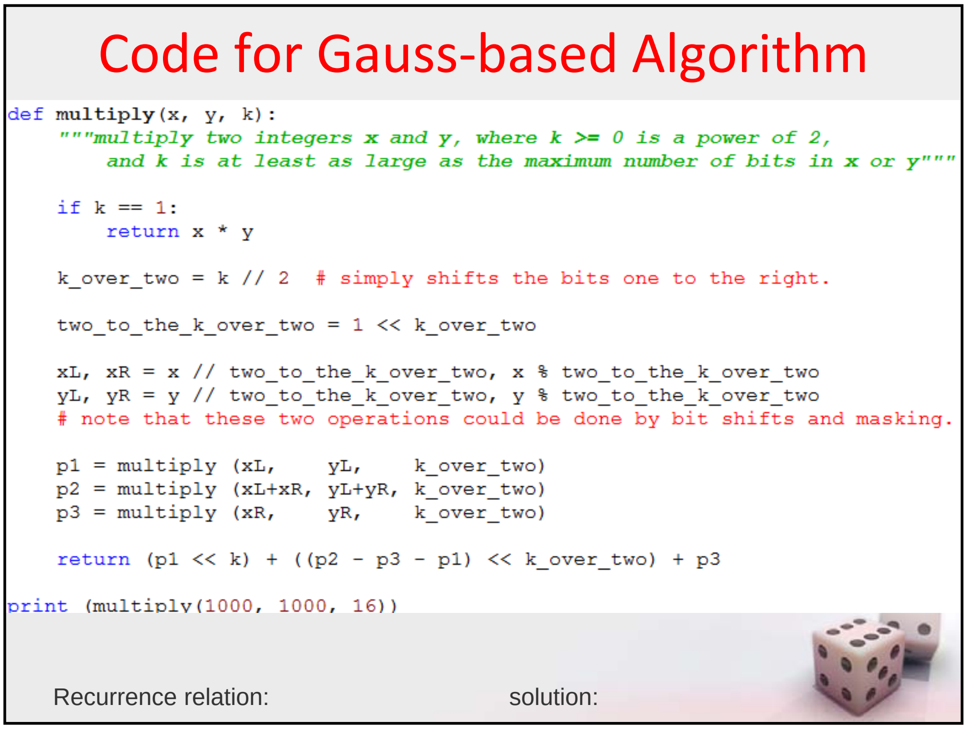
- Is there an algorithm for multiplying two k-bit numbers in time that is less than O(k^2)?
- **Basis:** A discovery of Carl Gauss (1777-1855)
  - Multiplying complex numbers:
  - **(a + bi)*(c+di) = ac – bd + (bc + ad)i**

  - Could also be expressed as ordered pairs
  - **[a, b]*[c,d] =[ac-bd, bc+ad]**



Gauss's Algorithm

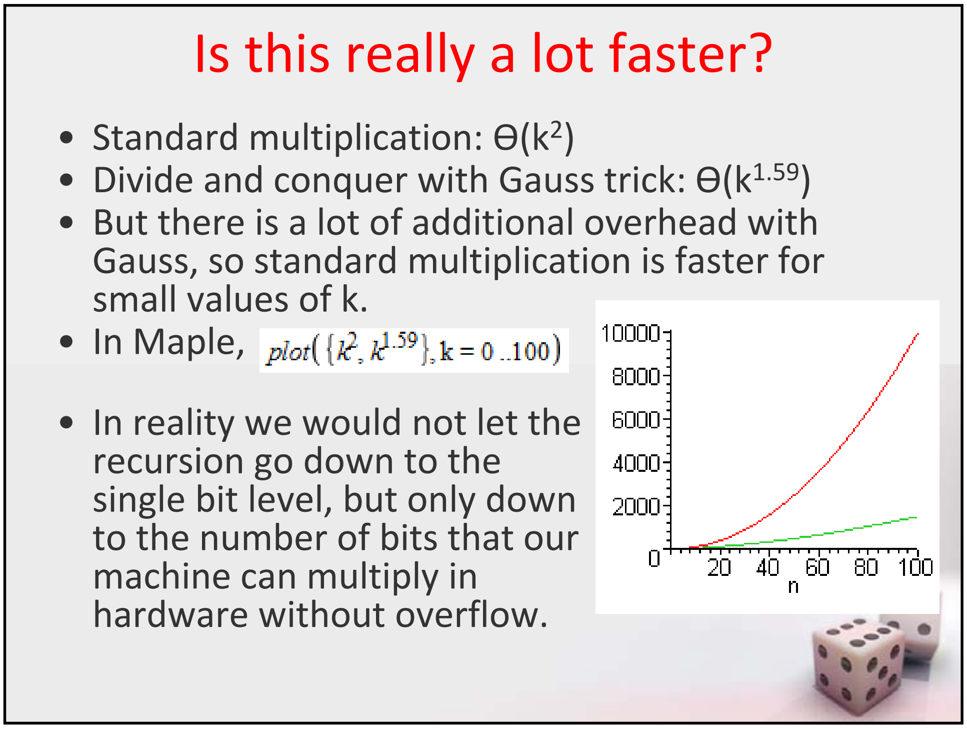
- **[a, b]*[c,d] = [ac-bd, bc+ad]** (complex number multiplication)
  - Needs **4** real-number multiplications and **3** additions
- But **bc + ad = (a+b)(c+d) – ac –bd**
  - And we have already computed **ac** and **bd** when we computed the real part of the product!
- Thus we can do the complex product with **3** multiplications and **5** additions
- Additions are so much faster than multiplications that this is a good trade-off.
- A little savings, but not a big deal until applied recursively!
- We apply the same general idea to recursive divide-and-conquer multiplication
(next slide – first 2/3 of the code is unchanged)



Code for Gauss-based Algorithm

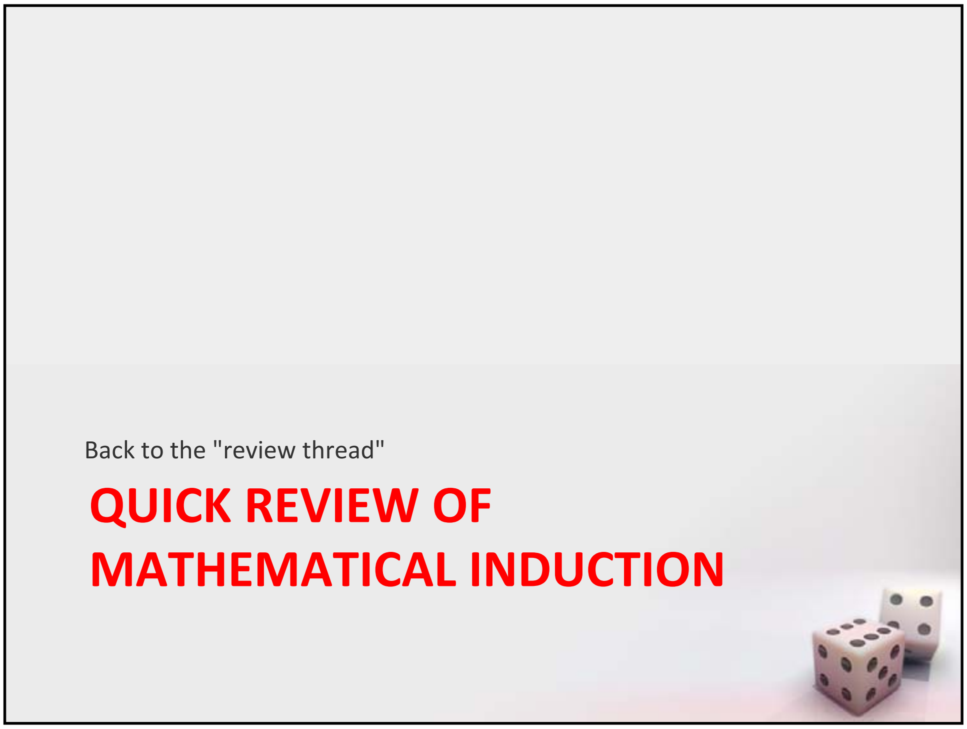
```
def multiply(x, y, k):
    """multiply two integers x and y, where k >= 0 is a power of 2,
       and k is at least as large as the maximum number of bits in x or y"""

    if k == 1:
        return x * y

    k_over_two = k // 2  # simply shifts the bits one to the right.

    two_to_the_k_over_two = 1 << k_over_two

    xL, xR = x // two_to_the_k_over_two, x % two_to_the_k_over_two
    yL, yR = y // two_to_the_k_over_two, y % two_to_the_k_over_two
    # note that these two operations could be done by bit shifts and masking.

    p1 = multiply (xL,    yL,    k_over_two)
    p2 = multiply (xL+xR, yL+yR, k_over_two)
    p3 = multiply (xR,    yR,    k_over_two)

    return (p1 << k) + ((p2 - p3 - p1) << k_over_two) + p3

print (multiply(1000, 1000, 16))
```

Recurrence relation: solution:



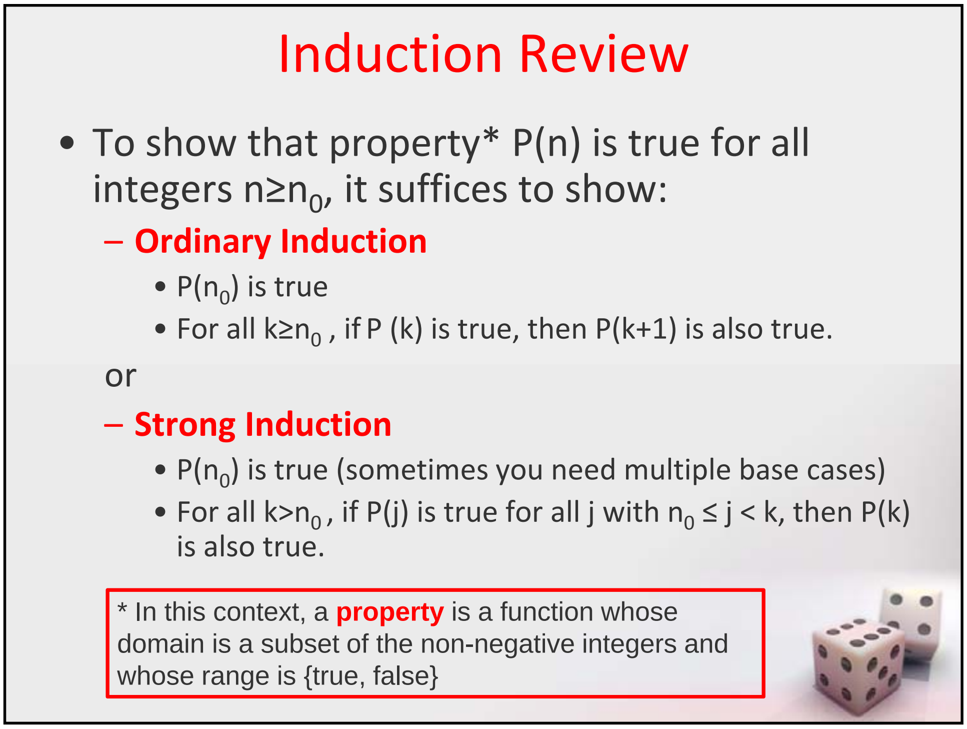
Is this really a lot faster?

- Standard multiplication: Θ(k^2)
- Divide and conquer with Gauss trick: Θ($k^{1.59}$)
- But there is a lot of additional overhead with Gauss, so standard multiplication is faster for small values of k.
- In Maple, $plot(\{k^2, k^{1.59}\}, k = 0..100)$
- In reality we would not let the recursion go down to the single bit level, but only down to the number of bits that our machine can multiply in hardware without overflow.



Back to the "review thread"

QUICK REVIEW OF MATHEMATICAL INDUCTION



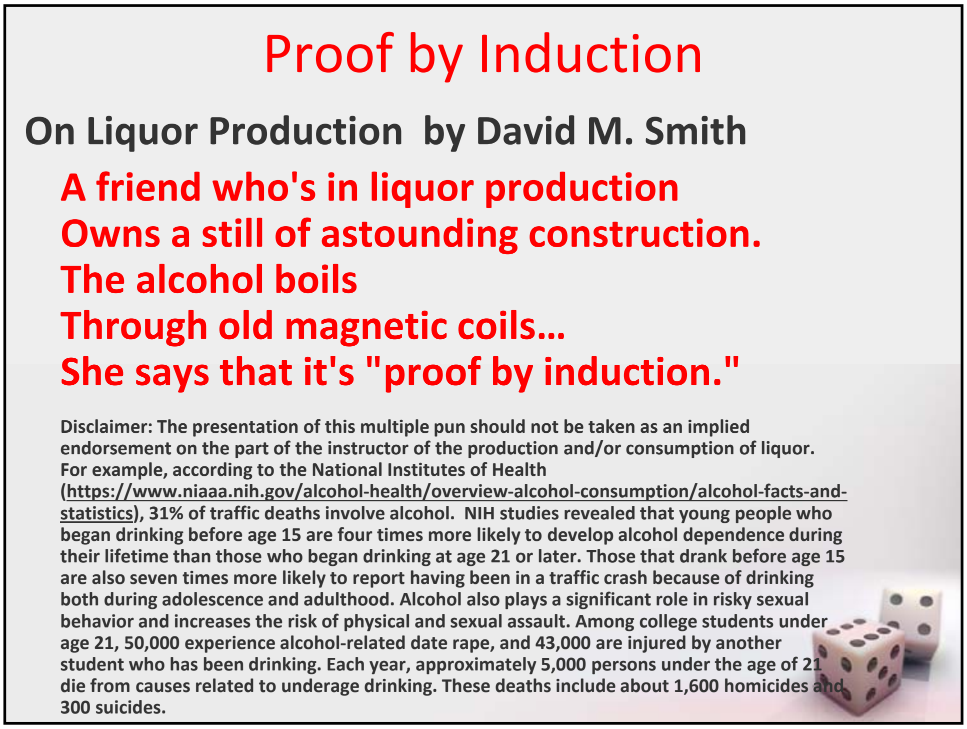
Induction Review

- To show that property* P(n) is true for all integers $n \geq n_0$, it suffices to show:
  - **Ordinary Induction**
    - $P(n_0)$ is true
    - For all $k \geq n_0$, if P (k) is true, then P(k+1) is also true.

  or

  - **Strong Induction**
    - $P(n_0)$ is true (sometimes you need multiple base cases)
    - For all $k > n_0$, if P(j) is true for all j with $n_0 \leq j < k$, then P(k) is also true.

* In this context, a **property** is a function whose domain is a subset of the non-negative integers and whose range is {true, false}

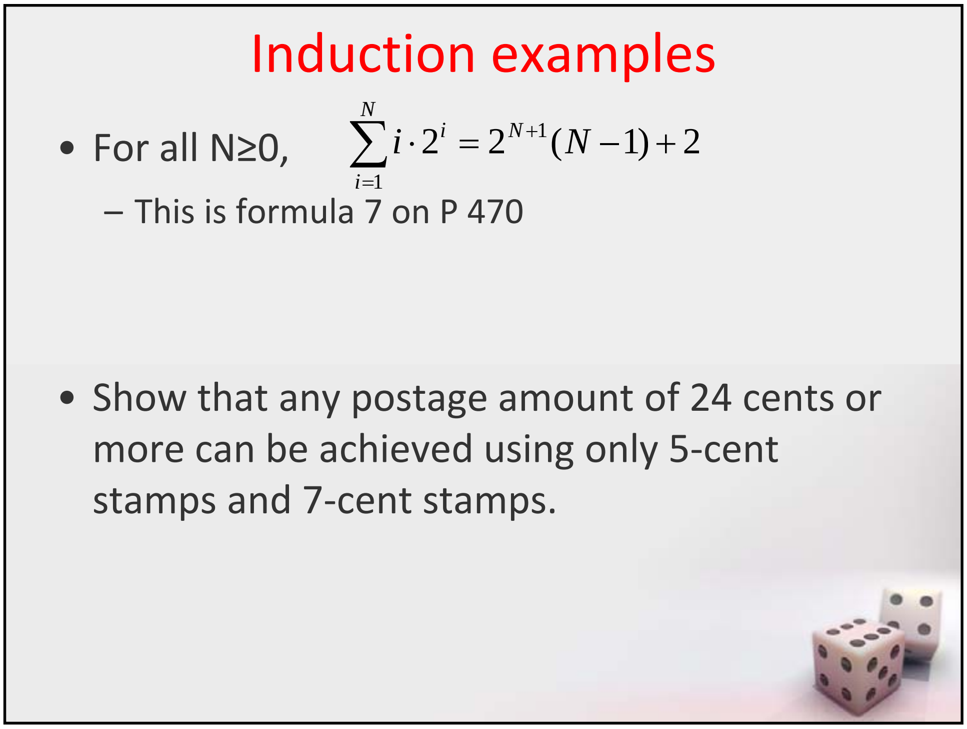


Proof by Induction

On Liquor Production by David M. Smith

A friend who's in liquor production
Owns a still of astounding construction.
The alcohol boils
Through old magnetic coils…
She says that it's "proof by induction."

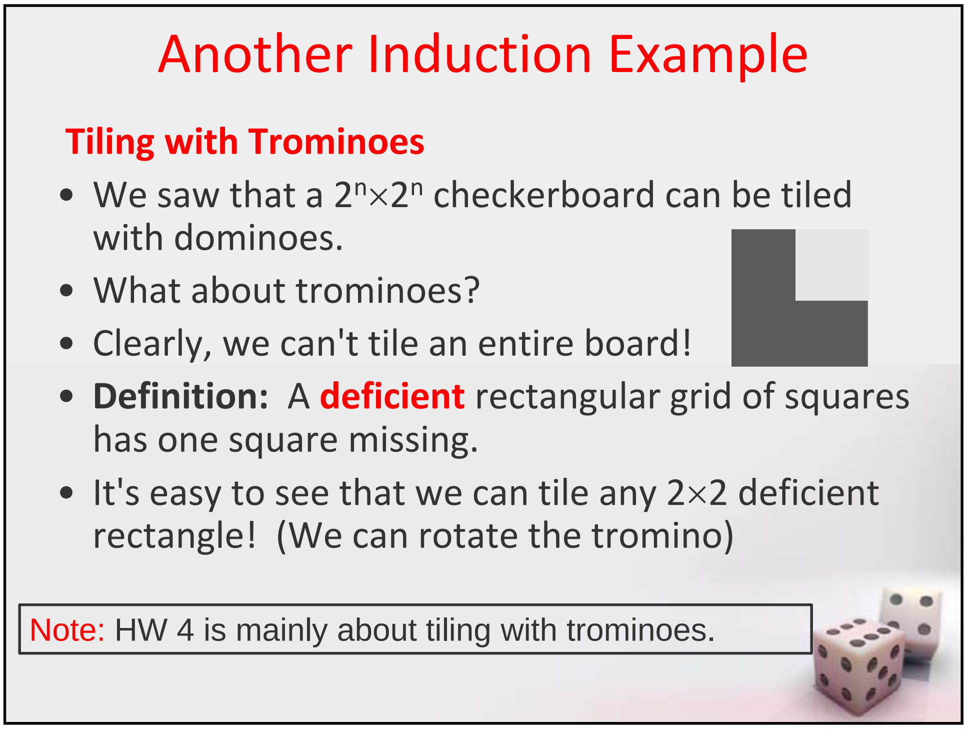
Disclaimer: The presentation of this multiple pun should not be taken as an implied endorsement on the part of the instructor of the production and/or consumption of liquor. For example, according to the National Institutes of Health (https://www.niaaa.nih.gov/alcohol-health/overview-alcohol-consumption/alcohol-facts-and-statistics), 31% of traffic deaths involve alcohol. NIH studies revealed that young people who began drinking before age 15 are four times more likely to develop alcohol dependence during their lifetime than those who began drinking at age 21 or later. Those that drank before age 15 are also seven times more likely to report having been in a traffic crash because of drinking both during adolescence and adulthood. Alcohol also plays a significant role in risky sexual behavior and increases the risk of physical and sexual assault. Among college students under age 21, 50,000 experience alcohol-related date rape, and 43,000 are injured by another student who has been drinking. Each year, approximately 5,000 persons under the age of 21 die from causes related to underage drinking. These deaths include about 1,600 homicides and 300 suicides.



Induction examples

- For all N≥0, $\sum_{i=1}^{N} i\cdot 2^i = 2^{N+1}(N-1)+2$
  - This is formula 7 on P 470

- Show that any postage amount of 24 cents or more can be achieved using only 5-cent stamps and 7-cent stamps.

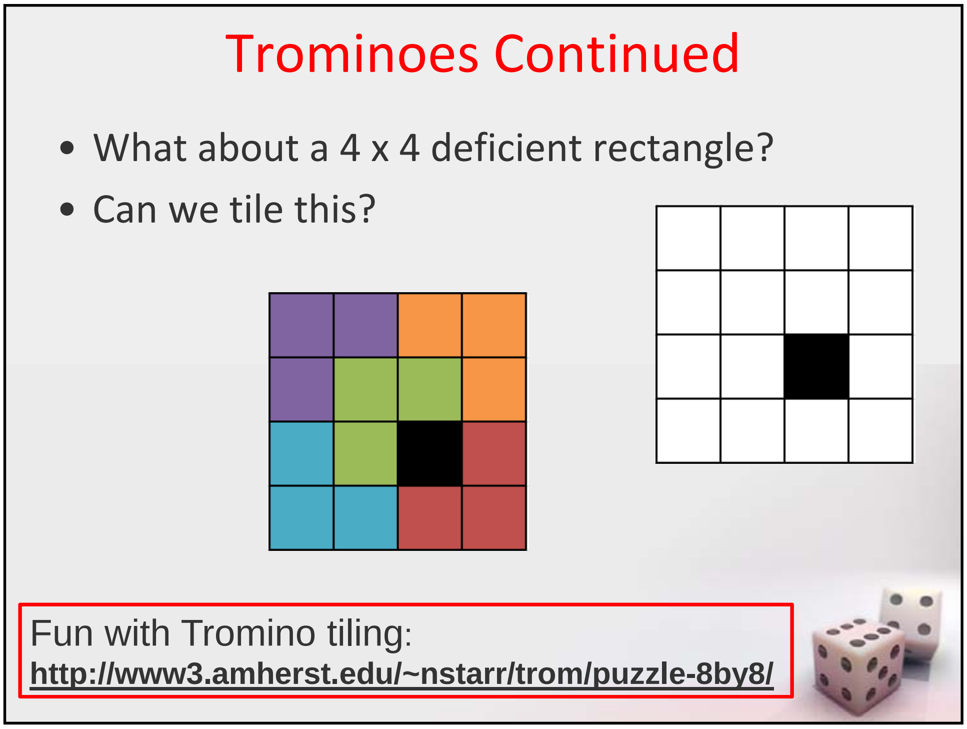


Another Induction Example

Tiling with Trominoes

- We saw that a $2^n \times 2^n$ checkerboard can be tiled with dominoes.
- What about trominoes?
- Clearly, we can't tile an entire board!
- **Definition:** A **deficient** rectangular grid of squares has one square missing.
- It's easy to see that we can tile any 2×2 deficient rectangle! (We can rotate the tromino)

Note: HW 4 is mainly about tiling with trominoes.



Trominoes Continued

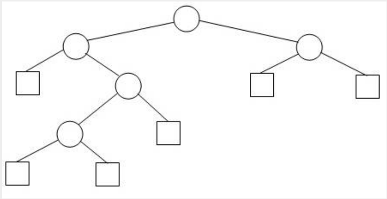
- What about a 4 x 4 deficient rectangle?
- Can we tile this?

Fun with Tromino tiling:
http://www3.amherst.edu/~nstarr/trom/puzzle-8by8/

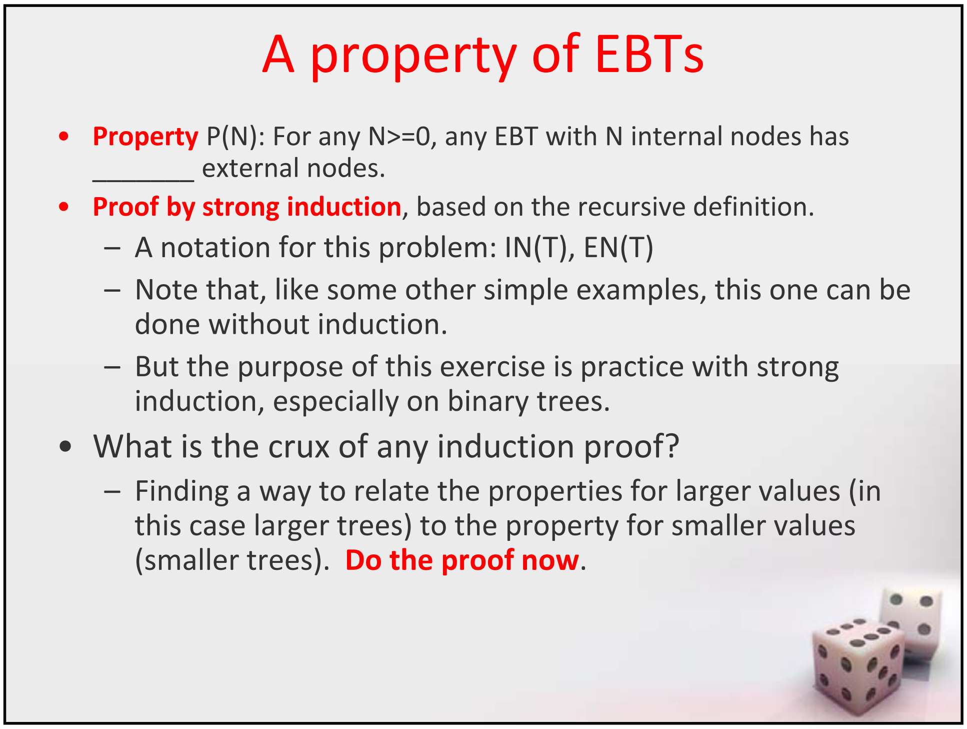
Trominoes Continued

- Prove by induction that we can tile any $2^n \times 2^n$ deficient rectangle with trominoes
- Base case: n=1 Done
- Assume that we can do it for n=k
- Show that we can do it for n=k+1
- Assume WLOG that the missing square is in the lower right quadrant of the rectangle
  - If it is somewhere else, we could simply rotate the board.
  - Can we place one tromino in a way that allows us to use the induction assumption?
  - Draw the picture

Another Induction Example
Extended Binary Tree (EBT)



- An Extended Binary tree is either
  - an ***external node***, or
  - an (**internal**) root node and two EBTs T_L and T_R.
- We draw internal nodes as circles and external nodes as squares.
  - Generic picture and detailed picture.
- This is simply an alternative way of viewing binary trees, in which we view the null pointers as “places” where a search can end or an element can be inserted.



A property of EBTs

- **Property** P(N): For any N>=0, any EBT with N internal nodes has ________ external nodes.
- **Proof by strong induction**, based on the recursive definition.
  - A notation for this problem: IN(T), EN(T)
  - Note that, like some other simple examples, this one can be done without induction.
  - But the purpose of this exercise is practice with strong induction, especially on binary trees.
- What is the crux of any induction proof?
  - Finding a way to relate the properties for larger values (in this case larger trees) to the property for smaller values (smaller trees). **Do the proof now**.