

New Views of the Idea of
Mathematical Induction

Herbert S. Wilf
University of Pennsylvania
Philadelphia, PA

Induction is:

bowling pins

Induction is:

dominoes

Induction is:

Prove that the sum of the first n positive integers is $n(n + 1)/2$.

It is all of that—
and a lot more

In addition to being a tool for proving theorems that have n 's in them, induction is also

- for constructing mathematical objects recursively
- for designing algorithms that do their tasks inductively
- for analyzing recursively designed structures or algorithms or whatever
- about recurrence relations and difference equations
- about computer languages in which recursive programs can be directly programmed.

The factorial of n

For nonnegative integers n , we can define

$$f(n) = \begin{cases} 1, & \text{if } n = 0; \\ nf(n-1), & \text{else.} \end{cases}$$

Theorem 1 *For all integers $n > 0$, $f(n)$ is the product of the positive integers $\leq n$.*

Proof. The proof is by (guess what?) on n .

□

A complete Mathematica program for computing $n!$ recursively might look like this:

```
In[1] := nfact[n_] := If[n==0,1,n nfact[n-1]]
```

and here's some output-

```
In[2] := nfact[10]
```

```
Out[2]= 3628800
```

The GCD

For integers $m > 0$ and $n \geq 0$ we can define

$$f(m, n) = \begin{cases} m, & \text{if } n = 0; \\ f(n, m \bmod n), & \text{else.} \end{cases}$$

Example:

$$\begin{aligned} f(26, 16) &= f(16, 10) = f(10, 6) = f(6, 4) \\ &= f(4, 2) = f(2, 0) = 2 \end{aligned}$$

Theorem 2 For $m > 0$, $n \geq 0$, $f(m, n)$ is the “largest” integer that divides both m and n .

Proof. By induction on $n \geq 0$, the inductive proposition $P(n)$ being that $f(m, n)$ is the GCD for all $m > 0$. True for $n = 0$. Then use the division (Euclidean) algorithm

$$m = qn + m \bmod n,$$

and the inductive hypothesis to prove the theorem (details omitted). \square

A complete Maple program for the gcd might look like this:

```
gcddiv:=proc(m,n);  
    if n=0  
        then RETURN(m)  
        else RETURN(gcddiv(n,m mod n)) fi;  
end:
```

and here's some output-

```
> gcddiv(80,120);
```

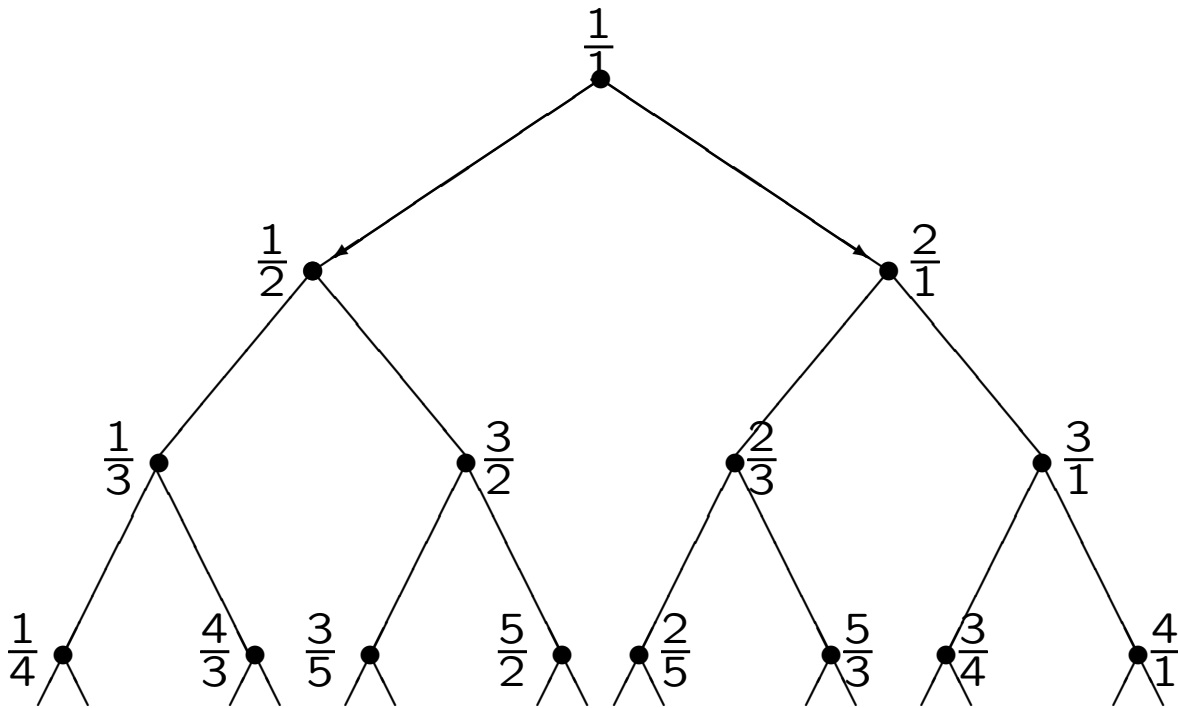
40

```
> gcddiv(1042668,4253226);
```

6

The Tree of All Fractions

Parent is $\frac{i}{j}$
Left child is $\frac{i}{i+j}$ Right child is $\frac{i+j}{j}$



Theorem 3 *Every positive fraction r/s occurs once and only once in this tree, and it occurs in lowest terms.*

Proof.

- By induction down the levels of the tree, everything occurs in lowest terms.
- Let $P(r, s) = \text{prop. that } r/s \text{ occurs at most once.}$ Surely $P(1, 1)$ is true (what would the parent of the 2nd occurrence be?). If some $P(r, s)$ is false, choose one with minimum s . If $r < s$ then $P(r, s - r)$, and otherwise $P(r - s, s)$ is false, #
- Suppose some fraction r/s does not occur. If $s > r$, choose such a missing fraction with smallest s . Then $r/(s - r)$ must occur somewhere. But its left child is r/s . # Similarly if $s < r$. \square

Remark: That proof actually finds a path from (1,1) to (r,s); e.g., where does (13,17) occur?

$$\frac{13}{17} \rightarrow \frac{13}{4} \rightarrow \frac{9}{4} \rightarrow \frac{5}{4} \rightarrow \frac{1}{4} \rightarrow \frac{1}{3} \rightarrow \frac{1}{2} \rightarrow \frac{1}{1}$$

But don't stop there! **In Maple:**

```
path:=proc(r,s);  
  if r=s then RETURN('')  
    elif r>s then RETURN(cat(path(r-s,s),R))  
    else RETURN(cat(path(r,s-r),L))  
  fi;  
end;
```

Some output:

```
> path(13,17);  
      LLLRRRL
```

Drunk with power, we try:

```
> path(31,10);  
  LLLLLLLLLRRR  
> path(314,100);  
  RRRRRLLLLLLLLRRR  
> path(3141,1000);  
  LRRRRRLRRRRRRRRRRLLLLLLLLRRR  
> path(31415,10000);  
  LRRRRRRRRRLRRRRRRRRRRRRRRLLLLLLLLRRR
```

For the cognoscenti: the Stern-Brocot tree is obtained from this one by sorting each level in L-R ascending order.

That was an example of

- A recursive construction of a tree, and
- an inductive proof of the properties of the tree, followed by
- a recursive program for finding paths in the tree, and
- a "drunk with power" reward, namely using the program to find things that would have been quite time-consuming to find by hand.

I think that these are all top-drawer experiences for our undergrads to have.

Some recursive computer languages:

**Maple, Mathematica, C++, Java, Pascal,
...**

Some nonrecursive computer languages:

Fortran, Basic

How to multiply 2×2 matrices

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

Here's one way:

$$c_{ij} = a_{i1} * b_{1j} + a_{i2} * b_{2j} \quad (i, j = 1, 2)$$

Costs:

- Additions/subtractions of numbers: 4
- Multiplications of numbers: 8

But there's another way to do

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

First do these seven things-

$$I = (a_{12} - a_{22}) * (b_{21} + b_{22})$$

$$II = (a_{11} + a_{22}) * (b_{11} + b_{22})$$

$$III = (a_{11} - a_{21}) * (b_{11} + b_{12})$$

$$IV = (a_{11} + a_{12}) * b_{22}$$

$$V = a_{11} * (b_{12} - b_{22})$$

$$VI = a_{22} * (b_{21} - b_{11})$$

$$VII = (a_{21} + a_{22}) * b_{11}$$

and then do these four things-

$$c_{11} = I + II - IV + VI$$

$$c_{12} = IV + V$$

$$c_{21} = VI + VII$$

$$c_{22} = II - III + V - VII.$$

Costs:

- Additions/subtractions of numbers: 18
(14 more than before)
- Multiplications of numbers: 7
(1 less than before)

Why should I care?

Because, not only can we multiply 2×2 matrices faster, but by induction, this idea, of Volker Strassen (1969) allows us to multiply $n \times n$ matrices faster too.

To see how that is done, I'll repeat the previous slide using capital letters.

There's another way to do

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

First do these seven things-

$$\begin{aligned} I &= (A_{12} - A_{22}) * (B_{21} + B_{22}) \\ II &= (A_{11} + A_{22}) * (B_{11} + B_{22}) \\ III &= (A_{11} - A_{21}) * (B_{11} + B_{12}) \\ IV &= (A_{11} + A_{12}) * B_{22} \\ V &= A_{11} * (B_{12} - B_{22}) \\ VI &= A_{22} * (B_{21} - B_{11}) \\ VII &= (A_{21} + A_{22}) * B_{11} \end{aligned}$$

and then do these four things-

$$\begin{aligned} C_{11} &= I + II - IV + VI \\ C_{12} &= IV + V \\ C_{21} &= VI + VII \\ C_{22} &= II - III + V - VII. \end{aligned}$$

How to multiply big matrices A and B :

1. Pad the matrices A and B by new rows and columns of 0's until they are both $2^n \times 2^n$, for some n .
2. (*The base case*) If $n = 0$ then return the product AB of 1×1 matrices.
3. Otherwise, partition each of A, B into four submatrices of size $2^{n-1} \times 2^{n-1}$.
4. (*The recursive call*) Do the seven matrix multiplications shown on the previous slide by recursive calls to this fast-multiply method, with n replaced by $n - 1$.
5. Return the C matrix, as shown on the previous slide. \square

How fast is it?

Let $f(n)$ be the number of multiplications of numbers that are needed to multiply two $2^n \times 2^n$ matrices, by this method. Then

$$f(n) = 7f(n - 1), \quad (f(0) = 1),$$

which gives $f(n) = 7^n$.

So 7^n multiplications of numbers are needed to multiply two $2^n \times 2^n$ matrices. That is:

$N^{\log_2 7} = N^{2.81\dots}$ multiplications of numbers are needed to multiply two $N \times N$ matrices.

Likewise, it turns out that at most $KN^{2.81\dots}$ additions are needed.

An improvement!

That was an example of

- a recursively defined method of multiplying matrices, followed by
- an inductive analysis of its speed and
- the solution of a simple recurrence relation.

The program, if written in a recursive computer language, would look just like the inductive description of the method. No "*and so on*" 's or "*continuing in this way we see that*" 's.

Binary Search

Given an alphabetized list of the 5,000,000 policyholders of insurance company XX, find out if N. Bourbaki is on the list.

To do that,

1. Look at the name "YYY" at the middle of the list
2. If YYY is N. Bourbaki, then exit triumphantly.
3. If YYY lexicographically follows N. Bourbaki, then repeat the search on the first half of the list
4. Otherwise, repeat the search on the second half of the list.

More formally, given an ordered list X of items $x_1 < x_2 < x_3 < \dots < x_N$, and a new item x . To answer the question "Is x in the list X ?",

```
FindIt(x,X)=
  If X is empty then return "No",
    else
  If X has just one item
    then
      if that item is x, then return "Yes"
        else return "No"
    else
  Let y be the element in the "middle" of X.
  If x=y then return "Yes",
    else
  If x<y then do FindIt(x,top half of X)
    else do FindIt(x,lower half of X)
End.
```

How fast is that?

Each time FindIt calls itself on either the upper or lower half of the list, the list is at most half the size the it used to be. So after j calls to itself the length of the list will be at most $N/2^j$. If this is < 1 , the program will have halted. That is, if $N/2^j < 1$, we're done. So j never gets bigger than $\log_2 N$.

Thus we can find out if a list of 5,000,000 policyholders contains N. Bourbaki after at most

$$\log_2 (5000000) = 22.253 \dots$$

interrogations of list members.

Great homework problem to program! Lots of snares, traps and pitfalls! Fun!

Induction begets recurrence relations

1. Differential equation: $y'' = y' + y$
2. Try: $y(x) = e^{rx}$
3. Substitute and find: $r^2 = r + 1$
4. General solution: $y = c_1 e^{\frac{1+\sqrt{5}}{2}x} + c_2 e^{\frac{1-\sqrt{5}}{2}x}$.

- 1' Difference equation: $y_{n+2} = y_{n+1} + y_n$
- 2' Try: $y_n = r^n$
- 3' Substitute and find: $r^2 = r + 1$
- 4' General solution: $y_n = c_1 \left(\frac{1+\sqrt{5}}{2}\right)^n + c_2 \left(\frac{1-\sqrt{5}}{2}\right)^n$

For more examples of the constructive uses of induction, including the FFT, Quicksorting, finding maximum independent sets in graphs, etc., you can download Chapter 2 of *Algorithms and Complexity* from

`<http://www.cis.upenn.edu/~wilf>`