

MATHEMATICAL INDUCTION

$$\textcircled{1} \quad \sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

Check for $n = 1, 2, 3, 4$ — TRUE

\textcircled{2} For p a prime number

$2^p - 1$ is prime (MERTSENNE PRIME NUMBER)

True for $p = 2, 3, 5, 7$.

\textcircled{3} All swans are white

True for all swans we've seen so far.

\textcircled{3} Is an example of empirical induction
— not good enough for math

\textcircled{2} fails for $p = 11$ (but works again for $p = 13, 17, 19$)

\textcircled{1} Is true because you've seen it in lecture notes

But how to prove it? 
 "Falling Dominoes principle"

INDUCTION PRINCIPLE

If $P(n)$ is a statement for integers such that

(a) $P(n_0)$ is true

BASE CASE

(b) If for $n \geq n_0$, it holds that $P(n)$ true,

then $P(n+1)$ is also true

INDUCTION STEP

Then $P(n)$ is true for all $n \geq n_0$

How to prove the induction principle?

We can't — we accept it as obviously true
An AXIOM

Follows from the LEAST NUMBER PRINCIPLE

and implies it — so they are equivalent

Prove ① by induction

Base case ($n = 1$)

$$1^2 = \frac{1 \cdot 2 \cdot 3}{6} \quad \text{OK}$$

Induction step

Suppose

$$\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=1}^{n+1} i^2 = \sum_{i=1}^n i^2 + (n+1)^2$$

$$= \frac{n(n+1)(2n+1)}{6} + (n+1)^2 \quad \begin{bmatrix} \text{induction} \\ \text{hypothesis} \end{bmatrix}$$

$$= \frac{n(n+1)(2n+1)}{6} + \frac{6(n+1)(n+1)}{6}$$

$$= \frac{(n+1)(n(2n+1) + 6(n+1))}{6}$$

$$= \frac{(n+1)(2n^2 + 7n + 6)}{6}$$

$$= \frac{(n+1)(n+2)(2n+3)}{6}$$

The equality follows
by the induction principle

Ex Show that $5 \mid (6^n - 5n + 4)$ $\forall n \in \mathbb{Z}^+$ XII

Base case ($n=1$)

$$6^1 - 5 \cdot 1 + 4 = 5 \quad \text{OK}$$

Induction step

Suppose $5 \mid (6^n - 5n + 4)$

$$6^{n+1} - 5(n+1) + 4 =$$

$$= 6 \cdot 6^n - 5n - 5 + 4$$

$$= (6^n - 5n + 4) + (5 \cdot 6^n - 5)$$

$5 \mid (6^n - 5n + 4)$ by induction hypothesis

$$5 \cdot 6^n - 5 = 5 \cdot (6^n - 1)$$

$$\text{so } 5 \mid (6^{n+1} - 5(n+1) - 4)$$

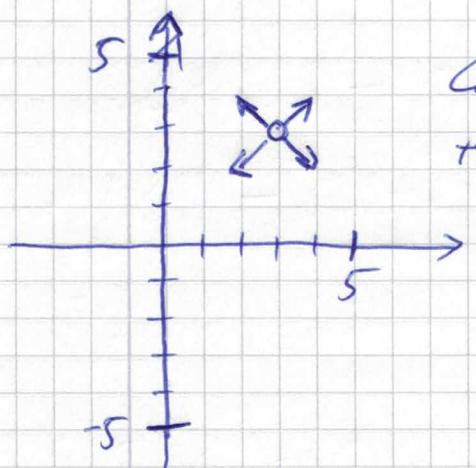
The claim follows by the induction principle

Properties preserved throughout program or procedure

Important tool to argue correctness
Especially in for- or while-loops

Ex DIAGONALLY MOVING ROBOT IN THE PLANE

Robot starts at $(0, 0)$



Can move diagonally from (x, y)

$$(x+1, y+1)$$

$$(x+1, y-1)$$

$$(x-1, y+1)$$

$$(x-1, y-1)$$

Can the robot ever be at $(1, 0)$?

Claim For any position (x, y) visited it holds that $x+y$ is an even number

Proof: Do the proof by induction

STRONG INDUCTION

If

- (a) $P(n_0)$ is true, and
 (b) for any $n \geq n_0$ it holds that
 if $P(n_0), P(n_0+1), \dots, P(n)$ are all true
 then $P(n+1)$ true

then

 $P(n)$ is true for all integers $n \geq n_0$

Actually same induction principle, but sometimes convenient to formulate it this way.

THEOREM Every positive integer ≥ 2 is a product of primes

Proof By strong induction

Base case ($n=2$)

2 is a prime

Induction step

If $n+1$ is a prime, then we are done.

Otherwise, by definition $\exists a, b \in \mathbb{Z}^+$
 such that $n+1 = a \cdot b$

By inductive hypothesis, a & b are both products of primes. Hence,
 so is $n+1$

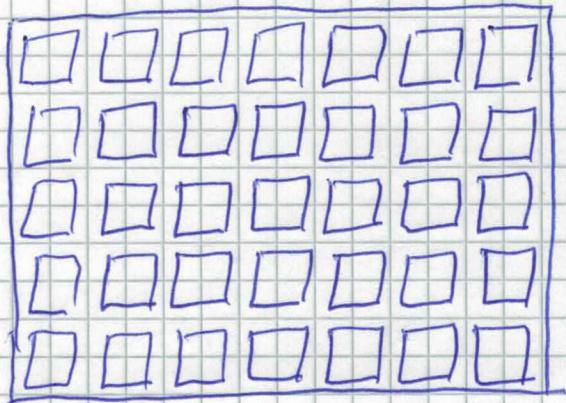
The theorem follows by the induction principle

Given $n \times m$ chocolate bar.

How many breaks

(horizontal or vertical,
on a contiguous cake)

are needed to split the
bar into 1×1 pieces?



Use strong induction over $A = n \times m$
to prove that $A - 1$ breaks are needed

Base case (1×1 cake) : 0 breaks — OK

Induction step:

Consider first break splitting the cake
into 2 pieces. Use strong induction
on each piece.

Mathematical induction can also be used for complexity analysis

Consider binary search again

BINARY-SEARCH (A, l_0, h_0, key)

if ($l_0 > h_0$)

$O(1)$ return -1

"BASE CASE"

else

$O(1)$ } mid := $\lfloor (l_0 + h_0) / 2 \rfloor$
 if ($A[\text{mid}] == \text{key}$)
 return mid

else if ($A[\text{mid}] < \text{key}$)

Recursive call return BINARY-SEARCH ($A, \text{mid}+1, h_0, \text{key}$)

One of these { else // $A[\text{mid}] > \text{key}$

Recursive call returns BINARY-SEARCH ($A, l_0, \text{mid}-1, \text{key}$)

Let T_n = worst-case running time for (sorted) array of size n

T_1 : Comparison

If fail, recursive call with $l_0 > h_0$ terminates right away

Time: some constant K_1

T_n : Some constant K_2

plus recursive call

$$T_n \leq K_2 + T_{\lfloor n/2 \rfloor}$$

Last week, we argued that we should have

$$T_n = O(\log n)$$

Let us prove it using induction

BASE CASES

$$\begin{aligned} T_1 &\leq K_1 & \text{Set } K^* = K_1 + K_2 \\ T_2 &\leq K_2 + K_1 & \text{Ignore since } \log 1 = 0 \\ && \text{Enough to bound } T_2 \text{ & } T_3 \\ T_3 &\leq K_2 + K_1 & \leq K^* \log 2 \\ && \leq K^* \log 3 \end{aligned}$$

INDUCTION

Suppose for all $m < n$ we have shown that

$$T_m \leq K \log m$$

for some K (which should be a constant, but we can choose it later when we see what is needed). We get

$$\begin{aligned} T_n &\leq K_2 + T_{\lfloor n/2 \rfloor} \leq [IH] \\ &\leq K_2 + K \log(n/2) \\ &\leq K_2 + K \log(n/2) \\ &\leq K_2 + K(\log n - \log 2) \\ &= K_2 + \cancel{K \log n} - K \\ &\leq K \log n \quad \text{if } K \geq K_2 \end{aligned}$$

Base cases work if $K \geq K^* = K_1 + K_2$

Set $K = K^* \Rightarrow$ induction step goes through 

Formally, we have shown:

Let T_n worst-case running time
for binary search on list of size n

Then there exists a constant K^*
and $n_0 = 2$ so that for $n \geq n_0$

$$T_n \leq K^* \cdot \log n$$

Hence, $T_n = O(\log n)$.

Analysis of merge sort

$\text{MERGE-SORT}(A)$

if ($\text{length}(A) > 1$)

$\text{mid} := \lfloor (1 + \text{length}(A)) / 2 \rfloor$

$L := \text{new array of size } \text{mid}$

$R := \text{new array of size } \text{length}(A) - \text{mid}$

for ($i := 1$ upto mid)

$L[i] := A[i]$

for ($i := \text{mid} + 1$ upto $\text{length}(A)$)

$R[i - \text{mid}] := A[i]$

$\text{MERGE-SORT}(L)$

$\text{MERGE-SORT}(R)$

$\text{MERGE}(L, R, A)$

$O(n)$

Recursive call

Recursive call

Let

T_n = worst-case time complexity
for array of length n

Base case ($n = 1$) is $O(1)$, so $\leq K_1$
for some K_2

In general, linear time plus 2 recursive calls for lists of half the size $n/2$,
so

$$\leq K_2 n + 2 \cdot T_{n/2}$$

Actually, should be $T_{n/2}$ if we are really careful, but we will gloss over this (which is standard)

$$T_n = \begin{cases} K_1 & \text{if } n=1 \\ K_2 \cdot n + 2 \cdot T_{n/2} & \text{if } n>1 \end{cases}$$

25 XX

If we can prove upper bound on T_n , then get upper bound on time complexity of merge sort

$$T_1$$

$$K_1$$

$$T_2$$

$$2K_2 + 2K_1$$

$$\text{Set } K_3 = K_2 + K_1$$

Then

$$T_2 \leq K_3 \cdot 2 \log 2$$

This is the **BASE CASE** in our inductive argument

For the **INDUCTIVE STEP**, assume we know for all m , $2 \leq m \leq n$, that

$$T_m \leq K \cdot m \log m$$

INDUCTION HYPOTHESIS

(for some K that we will pin down shortly)

Then

$$\begin{aligned} T_n &= K_2 n + 2 \cdot T_{n/2} \\ &\leq K_2 n + 2 \cdot K \cdot n/2 \log(n/2) \\ &= K_2 n + K n (\log n - \log 2) \\ &= K_2 n + \boxed{K n \log n} - K n \\ &\leq K n \log n \quad \text{if } K \geq K_2 \end{aligned}$$

If we choose

$$K = K_3 \geq K_2$$

then we get (using the INDUCTION PRINCIPLE) that

There exists a constant K and another constant n_0 ($= 2$) such that for all $n \geq n_0$ it holds that

$$T_n \leq K n \log n$$

Hence, $T_n = O(n \log n)$, which is what we wanted to prove.

If we wanted to be really careful, then we would have

$$T_n \leq K_2 \cdot n + T_{\lfloor n/2 \rfloor} + T_{n/2^7}$$

Could still do similar calculations by using, e.g.)

$$\lfloor n/2 \rfloor \leq n/2$$

$$T_{n/2^7} \leq n/2 + 1$$

$$T_{n/2^7} \leq 3n/4$$

Calculations get messier, but final conclusion is the same. In computer science, we usually ignore this because we "know" these details don't matter.