Mathematical Induction

(Based on [Manber 1989])

Yih-Kuen Tsay

Department of Information Management
National Taiwan University

The Standard Induction Principle

- Let T be a theorem that includes a parameter n whose value can be any natural number.
- Here, natural numbers are positive integers, i.e., 1, 2, 3, . . ., excluding 0 (sometimes we may include 0).
- To prove T, it suffices to prove the following two conditions:
  - T holds for $n = 1$. (Base case)
  - For every $n > 1$, if T holds for $n - 1$, then T holds for n. (Inductive step)
- The assumption in the inductive step that T holds for $n - 1$ is called the *induction hypothesis*.

A Simple Proof by Induction

Theorem (2.1)

For all natural numbers x and n, $x^n - 1$ is divisible by $x - 1$.

Proof.

(Suggestion: try to follow the structure of this proof when you present a proof by induction.)

The proof is by induction on n.

Base case ($n = 1$): $x - 1$ is trivially divisible by $x - 1$.

Inductive step ($n > 1$): $x^n - 1 = x(x^{n-1} - 1) + (x - 1)$. $x^{n-1} - 1$ is divisible by $x - 1$ *from the induction hypothesis* and $x - 1$ is divisible by $x - 1$. Hence, $x^n - 1$ is divisible by $x - 1$. $\square$

Note: a is divisible by b if there exists an integer c such that $a = b \times c$. (0 is divisible by any integer, including 0 itself.)

Variants of Induction Principle

Theorem

If a statement T, with a parameter n, is true for $n = 1$, and if, for every $n \geq 1$, the truth of T for n implies its truth for $n + 1$, then T is true for all natural numbers.

Variants of Induction Principle

Theorem

If a statement T, with a parameter n, is true for $n = 1$, and if, for every $n \geq 1$, the truth of T for n implies its truth for $n+1$, then T is true for all natural numbers.

Theorem (Strong Induction)

If a statement T, with a parameter n, is true for $n = 1$, and if, for every $n > 1$, the truth of T for all natural numbers $< n$ implies its truth for n, then T is true for all natural numbers.

Variants of Induction Principle

IM NTU

Theorem

If a statement T, with a parameter n, is true for $n = 1$, and if, for every $n \geq 1$, the truth of T for n implies its truth for $n + 1$, then T is true for all natural numbers.

Theorem (Strong Induction)

If a statement T, with a parameter n, is true for $n = 1$, and if, for every $n > 1$, the truth of T for all natural numbers $< n$ implies its truth for n, then T is true for all natural numbers.

Theorem

If a statement T, with a parameter n, is true for $n = 1$ and for $n = 2$, and if, for every $n > 2$, the truth of T for $n - 2$ implies its truth for n, then T is true for all natural numbers.

Design by Induction: First Glimpse

- The selection sort, for instance, can be seen as constructed using design by induction:
  1. When there is only one element, we are done.
  2. When there are n (> 1) elements, we
     2.1 select the largest element,
     2.2 place it behind the remaining $n - 1$ elements, and
     2.3 sort the remaining $n - 1$ elements.

Design by Induction: First Glimpse

- The selection sort, for instance, can be seen as constructed using design by induction:
  1. When there is only one element, we are done.
  2. When there are n (> 1) elements, we
     2.1 select the largest element,
     2.2 place it behind the remaining $n - 1$ elements, and
     2.3 sort the remaining $n - 1$ elements.
- This looks simple enough, but the selection sort isn't very efficient.
- How can we obtain a more efficient algorithm via design by induction?

Design by Induction: First Glimpse

- The selection sort, for instance, can be seen as constructed using design by induction:
  1. When there is only one element, we are done.
  2. When there are n (> 1) elements, we
     2.1 select the largest element,
     2.2 place it behind the remaining $n - 1$ elements, and
     2.3 sort the remaining $n - 1$ elements.
- This looks simple enough, but the selection sort isn't very efficient.
- How can we obtain a more efficient algorithm via design by induction?
- To see the power of design by induction, let's look at a less familiar example.

Design by Induction: First Glimpse (cont.)

Problem

Given two sorted arrays $A[1..m]$ and $B[1..n]$ of positive integers, find their smallest common element; returns 0 if no common element is found.

- Assume the elements of each array are in ascending order.
- **Obvious solution**: take one element at a time from A and find out if it is also in B (or the other way around).

Design by Induction: First Glimpse (cont.)

IM NTU

Problem

Given two sorted arrays $A[1..m]$ and $B[1..n]$ of positive integers, find their smallest common element; returns 0 if no common element is found.

- Assume the elements of each array are in ascending order.
- **Obvious solution**: take one element at a time from A and find out if it is also in B (or the other way around).
- How efficient is this solution?
- Can we do better?

Design by Induction: First Glimpse (cont.)

- There are $m + n$ elements to begin with.
- Can we pick out one element such that either (1) it is the element we look for or (2) it can be ruled out from subsequent searches?
- In the second case, we are left with the same problem but with $m + n - 1$ elements?

Design by Induction: First Glimpse (cont.)

- There are $m + n$ elements to begin with.
- Can we pick out one element such that either (1) it is the element we look for or (2) it can be ruled out from subsequent searches?
- In the second case, we are left with the same problem but with $m + n - 1$ elements?
- **Idea**: compare the current first elements of A and B.
  1. If they are equal, then we are done.
  2. If not, the smaller one cannot be the smallest common element.

Design by Induction: First Glimpse (cont.)

Below is the complete solution:

Algorithm

Algorithm SCE(A, m, B, n) : *integer*;
begin
  if $m = 0$ *or* $n = 0$ **then** $SCE := 0$;
  if $A[1] = B[1]$ **then**
    $SCE := A[1]$;
  else if $A[1] < B[1]$ **then**
      $SCE := SCE(A[2..m], m - 1, B, n)$;
  else $SCE := SCE(A, m, B[2..n], n - 1)$;
end

Why Induction Works

- Computations carried out by a computer/machine can, in essence, be understood as mathematical functions.

Why Induction Works

IM NTU

- Computations carried out by a computer/machine can, in essence, be understood as mathematical functions.
- To solve practical problems with computers,
  - objects/things in a practical domain must be modeled as (mostly discrete) mathematical structures/sets, and
  - various manipulations of the objects become functions on the corresponding mathematical structures.

Why Induction Works

IM NTU

- Computations carried out by a computer/machine can, in essence, be understood as mathematical functions.
- To solve practical problems with computers,
  - objects/things in a practical domain must be modeled as (mostly discrete) mathematical structures/sets, and
  - various manipulations of the objects become functions on the corresponding mathematical structures.
- Many mathematical structures are naturally defined by induction.

Why Induction Works

IM NTU

- Computations carried out by a computer/machine can, in essence, be understood as mathematical functions.
- To solve practical problems with computers,
  - objects/things in a practical domain must be modeled as (mostly discrete) mathematical structures/sets, and
  - various manipulations of the objects become functions on the corresponding mathematical structures.
- Many mathematical structures are naturally defined by induction.
- Functions on inductive structures are also naturally defined by induction (recursion).

Recursively/Inductively-Defined Sets

- The set $\mathbb{N}$ of natural numbers, including 0:
  1. Base case: 0 is a natural number ($0 \in \mathbb{N}$).
  2. Inductive step: if n is a natural number ($n \in \mathbb{N}$), then $n + 1$ is also a natural number ($(n + 1) \in \mathbb{N}$).

Recursively/Inductively-Defined Sets

- The set $\mathbb{N}$ of natural numbers, including 0:
  1. Base case: 0 is a natural number ($0 \in \mathbb{N}$).
  2. Inductive step: if n is a natural number ($n \in \mathbb{N}$), then $n + 1$ is also a natural number ($(n + 1) \in \mathbb{N}$).
- The set $\mathbb{N}_1$ of natural numbers, excluding 0:
  1. Base case: 1 is a natural number.
  2. Inductive step: if n is a natural number, then $n + 1$ is also a natural number.

Recursively/Inductively-Defined Sets

- The set $\mathbb{N}$ of natural numbers, including 0:
  1. Base case: 0 is a natural number ($0 \in \mathbb{N}$).
  2. Inductive step: if n is a natural number ($n \in \mathbb{N}$), then $n+1$ is also a natural number ($(n+1) \in \mathbb{N}$).
- The set $\mathbb{N}_1$ of natural numbers, excluding 0:
  1. Base case: 1 is a natural number.
  2. Inductive step: if n is a natural number, then $n+1$ is also a natural number.

Note: When $n = n' + 1$ for some $n' \in \mathbb{N}$, we write n' as $n - 1$; similarly, for $n' \in \mathbb{N}_1$. The factorial function $fac : \mathbb{N} \to \mathbb{N}_1$, for example, can be defined inductively as follows:

$$fac(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \times fac(n-1) & \text{otherwise} \end{cases}$$

Recursively/Inductively-Defined Sets (cont.)

- Binary trees:
  1. Base case: the empty tree is a binary tree.
  2. Inductive step: if L and R are binary trees, then a node with L and R as the left and the right children is also a binary tree.

Recursively/Inductively-Defined Sets (cont.)

- Binary trees:
  1. Base case: the empty tree is a binary tree.
  2. Inductive step: if L and R are binary trees, then a node with L and R as the left and the right children is also a binary tree.
- Nonempty binary trees:
  1. Base case: a single root node (without any child) is a binary tree.
  2. Inductive step: if L and R are binary trees, then a node with L as the left child and/or R as the right child is also a binary tree.

The height $H(t)$ of a binary tree t as an inductively defined function:

$$H(t) = \begin{cases} -1 & \text{if } t = \bot \text{ (the empty tree)} \\ 0 & \text{if } t = node(\cdot, \bot, \bot) \text{ (redundant)} \\ 1 + \max(H(t_l), H(t_r)) & \text{if } t = node(\cdot, t_l, t_r) \end{cases}$$

Structural Induction

- Structural induction is a generalization of mathematical induction on the natural numbers.
- It is used to prove that some proposition $P(x)$ holds for all x of some sort of recursively/inductively defined structure such as binary trees.

Structural Induction

- Structural induction is a generalization of mathematical induction on the natural numbers.
- It is used to prove that some proposition $P(x)$ holds for all x of some sort of recursively/inductively defined structure such as binary trees.
- Proof by structural induction:
  1. Base case: the proposition holds for all the minimal structures.
  2. Inductive step: if the proposition holds for the immediate substructures of a certain structure S, then it also holds for S.

Another Simple Example

IM NTU

Theorem (2.4)

If n is a natural number and $1 + x > 0$, then $(1+x)^n \geq 1 + nx$.

- Below are the key steps:

$$
\begin{aligned}
(1+x)^{n+1} &= (1+x)(1+x)^n \\
&\quad \{\text{induction hypothesis and } 1 + x > 0\} \\
&\geq (1+x)(1+nx) \\
&= 1 + (n+1)x + nx^2 \\
&\geq 1 + (n+1)x
\end{aligned}
$$

- The main point here is that we should be clear about how conditions listed in the theorem are used.

Proving vs. Computing

IM NTU

Theorem (2.2)

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

- This can be easily proven by induction.
- Key steps: $1 + 2 + \cdots + n + (n+1) = \frac{n(n+1)}{2} + (n+1) = \frac{n^2+n+2n+2}{2} = \frac{n^2+3n+2}{2} = \frac{(n+1)(n+2)}{2} = \frac{(n+1)((n+1)+1)}{2}$.

Proving vs. Computing

Theorem (2.2)

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

- This can be easily proven by induction.
- Key steps: $1 + 2 + \cdots + n + (n+1) = \frac{n(n+1)}{2} + (n+1) = \frac{n^2+n+2n+2}{2} = \frac{n^2+3n+2}{2} = \frac{(n+1)(n+2)}{2} = \frac{(n+1)((n+1)+1)}{2}$.
- Induction seems to be useful only if we already know the sum.
- What if we are asked to compute the sum of a series?

Proving vs. Computing

Theorem (2.2)

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

- This can be easily proven by induction.
- Key steps: $1 + 2 + \cdots + n + (n+1) = \frac{n(n+1)}{2} + (n+1) = \frac{n^2+n+2n+2}{2} = \frac{n^2+3n+2}{2} = \frac{(n+1)(n+2)}{2} = \frac{(n+1)((n+1)+1)}{2}$.
- Induction seems to be useful only if we already know the sum.
- What if we are asked to compute the sum of a series?
- Let's try $8 + 13 + 18 + 23 + \cdots + (3 + 5n)$.

Proving vs. Computing (cont.)

- **Idea**: guess and then verify by an inductive proof!
- The sum should be of the form $an^2 + bn + c$.
- By checking $n = 1$, 2, and 3, we get $\frac{5}{2}n^2 + \frac{11}{2}n$.
- Verify this for all n (1, 2, 3, and beyond), i.e., the following theorem, by induction.

Theorem (2.3)

$$8 + 13 + 18 + 23 + \cdots + (3 + 5n) = \frac{5}{2}n^2 + \frac{11}{2}n.$$

A Summation Problem

$$\begin{array}{rcl} 1 & = & 1 \\ 3+5 & = & 8 \\ 7+9+11 & = & 27 \\ 13+15+17+19 & = & 64 \\ 21+23+25+27+29 & = & 125 \end{array}$$

Theorem

The sum of row n in the triangle is n^3.

A Summation Problem

$$\begin{array}{rcl} 1 & = & 1 \\ 3+5 & = & 8 \\ 7+9+11 & = & 27 \\ 13+15+17+19 & = & 64 \\ 21+23+25+27+29 & = & 125 \end{array}$$

Theorem

The sum of row n in the triangle is n^3.

The base case is clearly correct. For the inductive step, examine the difference between rows $i+1$ and i ...

A Summation Problem (cont.)

Suppose row i starts with an odd number j whose exact value is not important.

$$
\begin{array}{ccccccccccc}
 & & & & & & & & & & \text{sum of row } i \\
 & & & & & & & & & & \text{(conjectured)} \\
j & + & (j+2) & + & \cdots & + & (j+2(i-1)) & & & = & i^3 \\
(j+2i) & + & (j+2i+2) & + & \cdots & + & (j+2i+2(i-1)) & + & ? & = & (i+1)^3 \\
\hline
2i & + & 2i & + & \cdots & + & 2i & + & ? & = & 3i^2+3i+1
\end{array}
$$

So, ? (the last number of row $i+1$) must be
$3i^2+3i+1-2i\times i = i^2+3i+1$, if the conjecture is correct.

Lemma

The last number in row $i+1$ is i^2+3i+1.

A Simple Inequality

IM NTU

Theorem (2.7)

$\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{2^n} < 1$, *for all* $n \geq 1$.

- There are at least two ways to select n terms from $n + 1$ terms.
  1. $(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{2^n}) + \frac{1}{2^{n+1}}$.

A Simple Inequality

Theorem (2.7)

$\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{2^n} < 1$, *for all* $n \geq 1$.

- There are at least two ways to select n terms from $n+1$ terms.
  1. $(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{2^n}) + \frac{1}{2^{n+1}}$.
  2. $\frac{1}{2} + (\frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{2^n} + \frac{1}{2^{n+1}})$.
- The second one leads to a successful inductive proof:

$$
\begin{aligned}
& \frac{1}{2} + (\frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{2^n} + \frac{1}{2^{n+1}}) \\
= & \frac{1}{2} + \frac{1}{2}(\frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^{n-1}} + \frac{1}{2^n}) \\
< & \frac{1}{2} + \frac{1}{2} \\
= & 1
\end{aligned}
$$

Euler's Formula

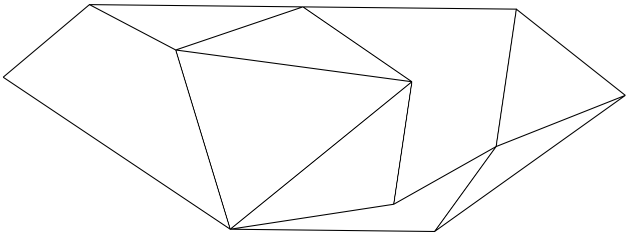

Figure: A planar map with 11 vertices, 19 edges, and 10 faces.

Source: redrawn from [Manber 1989, Figure 2.2].

Euler's Formula (cont.)

IM NTU

Theorem (2.8)

The number of vertices (V), edges (E), and faces (F) in an arbitrary connected planar graph are related by the formula $V + F = E + 2$.

Euler's Formula (cont.)

IM NTU

Theorem (2.8)

The number of vertices (V), edges (E), and faces (F) in an arbitrary connected planar graph are related by the formula $V + F = E + 2$.

The proof is by induction on the number of faces.
Base case ($F = 1$): connected planar graphs with only one face are trees. So, we need to prove the equality $V + 1 = E + 2$ or $V - 1 = E$ for trees, namely the following lemma:

Lemma

A tree with V vertices has $V - 1$ edges.

Inductive step ($F > 1$): for a graph with more than one faces, there must be a cycle in the graph. Remove one edge from the cyle . . .

Gray Codes

- A **Gray code** (after Frank Gray) for n objects is a binary-encoding scheme for naming the n objects such that the n names can be arranged in a *circular* list where *any two adjacent names, or code words, differ by only one bit*.
- Examples:
  - 00, 01, 11, 10
  - 000, 001, 011, 010, 110, 111, 101, 100
  - 000, 001, 011, 111, 101, 100

A Gray Code in Picture

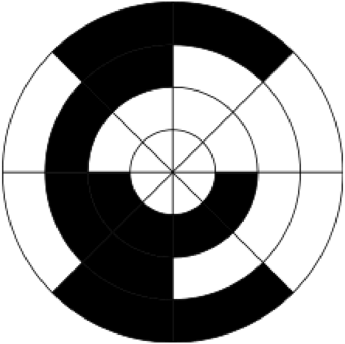

Figure: A rotary encoder using a 3-bit Gray code.

Source: Wikipedia.

Gray Codes (cont.)

Theorem (2.10)

There exist Gray codes of length $\frac{k}{2}$ for any positive even integer k.

Gray Codes (cont.)

Theorem (2.10)

There exist Gray codes of length $\frac{k}{2}$ for any positive even integer k.

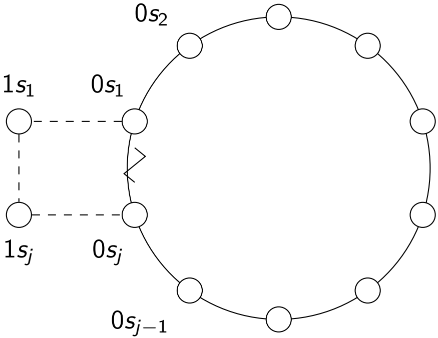

Figure: Constructing a Gray code of size $k = j + 2$, where j is even, from another of a smaller size j.

Source: adapted from [Manber 1989, Figure 2.3].

Gray Codes (cont.)

IM NTU

Theorem (2.10+)

There exist Gray codes of length $\log_2 k$ *for any positive integer* k *that is a power of* 2.

Gray Codes (cont.)

Theorem (2.10+)

There exist Gray codes of length $\log_2 k$ *for any positive integer* k *that is a power of* 2*.*

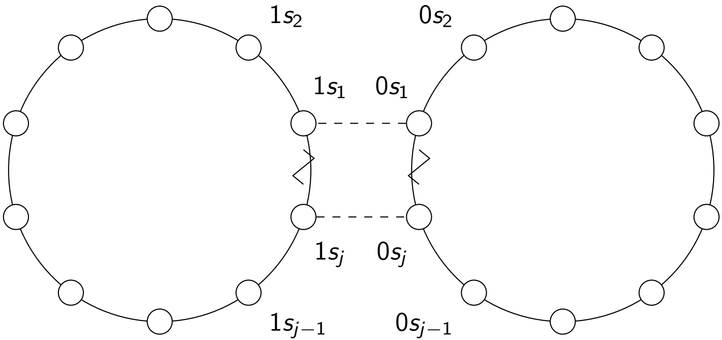

Figure: Constructing a Gray code from two smaller ones ($k = 2j$).

Source: adapted from [Manber 1989, Figure 2.4].

Gray Codes (cont.)

- 00, 01, 11, 10 (for 2^2 objects)
- 000, 001, 011, 010 (add a 0)
- 100, 101, 111, 110 (add a 1)
- Combine the preceding two codes (read the second in reversed order):
  000, 001, 011, 010, 110, 111, 101, 100 (for 2^3 objects)

Gray Codes (cont.)

IM NTU

Theorem (2.11−)

There exist Gray codes of length $\lceil \log_2 k \rceil$ for any positive even integer k.

Gray Codes (cont.)

IM NTU

Theorem (2.11−)

There exist Gray codes of length $\lceil \log_2 k \rceil$ for any positive even integer k.

To generalize the result and ease the proof, we allow a Gray code to be *open* where the last name and the first name may differ by more than one bit.

Gray Codes (cont.)

Theorem (2.11)

There exist Gray codes of length $\lceil \log_2 k \rceil$ for any positive integer $k \geq 2$. The Gray codes for the even values of k are closed, and the Gray codes for odd values of k are open.

Gray Codes (cont.)

Theorem (2.11)

There exist Gray codes of length $\lceil \log_2 k \rceil$ for any positive integer $k \geq 2$. The Gray codes for the even values of k are closed, and the Gray codes for odd values of k are open.

We in effect make the theorem stronger. A stronger theorem may be easier to prove, as we have a stronger induction hypothesis.

Gray Codes (cont.)

- 00, 01, 11 (open Gray code for 3 objects)
- 000, 001, 011 (add a 0)
- 100, 101, 111 (add a 1)
- Combine the preceding two codes (read the second in reversed order):
  000, 001, 011, 111, 101, 100 (closed Gray code for 6 objects)

Gray Codes (cont.)

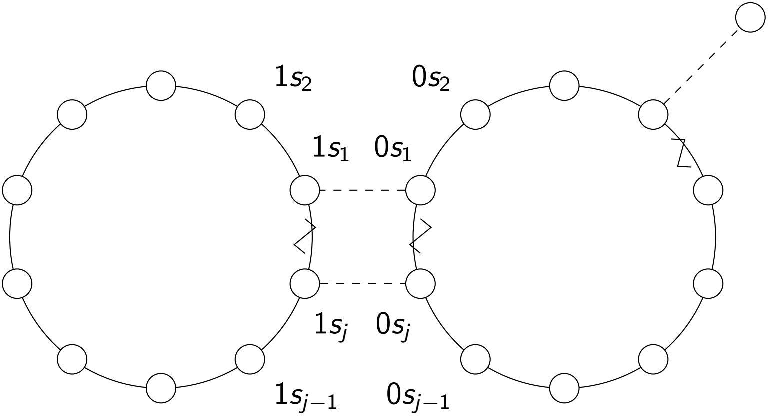

Figure: Constructing an open Gray code, for $k = 2j + 1$.

Source: adapted from [Manber 1989, Figure 2.5].

Arithmetic vs. Geometric Mean

IM NTU

Theorem (2.13)

If $x_1, x_2, \ldots, x_n$ *are all positive numbers, then*
$(x_1 x_2 \cdots x_n)^{\frac{1}{n}} \leq \dfrac{x_1 + x_2 + \cdots + x_n}{n}$.

Arithmetic vs. Geometric Mean

IM NTU

Theorem (2.13)

If $x_1, x_2, \ldots, x_n$ are all positive numbers, then
$$(x_1 x_2 \cdots x_n)^{\frac{1}{n}} \leq \frac{x_1 + x_2 + \cdots + x_n}{n}.$$

First use the standard induction to prove the case of powers of 2 and then use the reversed induction principle below to prove for all natural numbers.

Theorem (Reversed Induction Principle)

If a statement P, with a parameter n, is true for an infinite subset of the natural numbers, and if, for every $n > 1$, the truth of P for n implies its truth for $n - 1$, then P is true for all natural numbers.

Arithmetic vs. Geometric Mean (cont.)

- For all powers of 2, i.e., $n = 2^k$, $k \geq 1$: by induction on k.
- Base case: $(x_1 x_2)^{\frac{1}{2}} \leq \frac{x_1 + x_2}{2}$, squaring both sides ....

Arithmetic vs. Geometric Mean (cont.)

- For all powers of 2, i.e., $n = 2^k$, $k \geq 1$: by induction on k.
- Base case: $(x_1 x_2)^{\frac{1}{2}} \leq \frac{x_1+x_2}{2}$, squaring both sides ....
- Inductive step:

$$\left(x_1 x_2 \cdots x_{2^{k+1}}\right)^{\frac{1}{2^{k+1}}}$$

Arithmetic vs. Geometric Mean (cont.)

- For all powers of 2, i.e., $n = 2^k$, $k \geq 1$: by induction on k.
- Base case: $(x_1 x_2)^{\frac{1}{2}} \leq \frac{x_1+x_2}{2}$, squaring both sides ....
- Inductive step:

$$
\begin{aligned}
& \left(x_1 x_2 \cdots x_{2^{k+1}}\right)^{\frac{1}{2^{k+1}}} \\
= \; & \left[\left(x_1 x_2 \cdots x_{2^{k+1}}\right)^{\frac{1}{2^k}}\right]^{\frac{1}{2}}
\end{aligned}
$$

Arithmetic vs. Geometric Mean (cont.)

- For all powers of 2, i.e., $n = 2^k$, $k \geq 1$: by induction on k.
- Base case: $(x_1x_2)^{\frac{1}{2}} \leq \frac{x_1+x_2}{2}$, squaring both sides ....
- Inductive step:

$$
\begin{aligned}
& (x_1x_2\cdots x_{2^{k+1}})^{\frac{1}{2^{k+1}}} \\
= \; & \left[(x_1x_2\cdots x_{2^{k+1}})^{\frac{1}{2^k}}\right]^{\frac{1}{2}} \\
= \; & \left[(x_1x_2\cdots x_{2^k})^{\frac{1}{2^k}}(x_{2^k+1}x_{2^k+2}\cdots x_{2^{k+1}})^{\frac{1}{2^k}}\right]^{\frac{1}{2}}
\end{aligned}
$$

Arithmetic vs. Geometric Mean (cont.)

- For all powers of 2, i.e., $n = 2^k$, $k \geq 1$: by induction on k.
- Base case: $(x_1 x_2)^{\frac{1}{2}} \leq \frac{x_1 + x_2}{2}$, squaring both sides ....
- Inductive step:

$$
\begin{aligned}
& (x_1 x_2 \cdots x_{2^{k+1}})^{\frac{1}{2^{k+1}}} \\
= \; & \left[(x_1 x_2 \cdots x_{2^{k+1}})^{\frac{1}{2^k}}\right]^{\frac{1}{2}} \\
= \; & \left[(x_1 x_2 \cdots x_{2^k})^{\frac{1}{2^k}} (x_{2^k+1} x_{2^k+2} \cdots x_{2^{k+1}})^{\frac{1}{2^k}}\right]^{\frac{1}{2}} \\
\leq \; & \frac{(x_1 x_2 \cdots x_{2^k})^{\frac{1}{2^k}} + (x_{2^k+1} x_{2^k+2} \cdots x_{2^{k+1}})^{\frac{1}{2^k}}}{2}, \text{ from the base case}
\end{aligned}
$$

Arithmetic vs. Geometric Mean (cont.)

- For all powers of 2, i.e., $n = 2^k$, $k \geq 1$: by induction on k.
- Base case: $(x_1 x_2)^{\frac{1}{2}} \leq \frac{x_1 + x_2}{2}$, squaring both sides ....
- Inductive step:

$$
\begin{array}{cl}
 & (x_1 x_2 \cdots x_{2^{k+1}})^{\frac{1}{2^{k+1}}} \\
= & \left[(x_1 x_2 \cdots x_{2^{k+1}})^{\frac{1}{2^k}}\right]^{\frac{1}{2}} \\
= & \left[(x_1 x_2 \cdots x_{2^k})^{\frac{1}{2^k}} (x_{2^k+1} x_{2^k+2} \cdots x_{2^{k+1}})^{\frac{1}{2^k}}\right]^{\frac{1}{2}} \\
\leq & \frac{(x_1 x_2 \cdots x_{2^k})^{\frac{1}{2^k}} + (x_{2^k+1} x_{2^k+2} \cdots x_{2^{k+1}})^{\frac{1}{2^k}}}{2}, \text{ from the base case} \\
\leq & \frac{\frac{x_1 + x_2 + \cdots + x_{2^k}}{2^k} + \frac{x_{2^k+1} + x_{2^k+2} + \cdots + x_{2^{k+1}}}{2^k}}{2}, \text{ from the Ind. Hypo.}
\end{array}
$$

Arithmetic vs. Geometric Mean (cont.)

- For all powers of 2, i.e., $n = 2^k$, $k \geq 1$: by induction on k.
- Base case: $(x_1 x_2)^{\frac{1}{2}} \leq \frac{x_1+x_2}{2}$, squaring both sides ....
- Inductive step:

$$
\begin{aligned}
& (x_1 x_2 \cdots x_{2^{k+1}})^{\frac{1}{2^{k+1}}} \\
= \; & \left[(x_1 x_2 \cdots x_{2^{k+1}})^{\frac{1}{2^k}}\right]^{\frac{1}{2}} \\
= \; & \left[(x_1 x_2 \cdots x_{2^k})^{\frac{1}{2^k}} (x_{2^k+1} x_{2^k+2} \cdots x_{2^{k+1}})^{\frac{1}{2^k}}\right]^{\frac{1}{2}} \\
\leq \; & \frac{(x_1 x_2 \cdots x_{2^k})^{\frac{1}{2^k}} + (x_{2^k+1} x_{2^k+2} \cdots x_{2^{k+1}})^{\frac{1}{2^k}}}{2}, \text{ from the base case} \\
\leq \; & \frac{\frac{x_1+x_2+\cdots+x_{2^k}}{2^k} + \frac{x_{2^k+1}+x_{2^k+2}+\cdots+x_{2^{k+1}}}{2^k}}{2}, \text{ from the Ind. Hypo.} \\
= \; & \frac{x_1+x_2+\cdots+x_{2^{k+1}}}{2^{k+1}}
\end{aligned}
$$

Arithmetic vs. Geometric Mean (cont.)

- For all natural numbers: by reversed induction on n.
- Base case: the theorem holds for all powers of 2.

Arithmetic vs. Geometric Mean (cont.)

- For all natural numbers: by reversed induction on n.
- Base case: the theorem holds for all powers of 2.
- Inductive step: observe that

$$\frac{x_1 + x_2 + \cdots + x_{n-1}}{n-1} = \frac{x_1 + x_2 + \cdots + x_{n-1} + \frac{x_1+x_2+\cdots+x_{n-1}}{n-1}}{n}.$$

Arithmetic vs. Geometric Mean (cont.)

$$\left(x_1 x_2 \cdots x_{n-1}\left(\frac{x_1+x_2+\cdots+x_{n-1}}{n-1}\right)\right)^{\frac{1}{n}} \quad \leq \quad \frac{x_1+x_2+\cdots+x_{n-1}+\frac{x_1+x_2+\cdots+x_{n-1}}{n-1}}{n}$$

(from the Ind. Hypo.)

Arithmetic vs. Geometric Mean (cont.)

$$\left(x_1 x_2 \cdots x_{n-1}\left(\tfrac{x_1+x_2+\cdots+x_{n-1}}{n-1}\right)\right)^{\frac{1}{n}} \leq \frac{x_1+x_2+\cdots+x_{n-1}+\frac{x_1+x_2+\cdots+x_{n-1}}{n-1}}{n}$$

(from the Ind. Hypo.)

$$\left(x_1 x_2 \cdots x_{n-1}\left(\tfrac{x_1+x_2+\cdots+x_{n-1}}{n-1}\right)\right)^{\frac{1}{n}} \leq \frac{x_1+x_2+\cdots+x_{n-1}}{n-1}$$

Arithmetic vs. Geometric Mean (cont.)

$$\left(x_1 x_2 \cdots x_{n-1}\left(\frac{x_1+x_2+\cdots+x_{n-1}}{n-1}\right)\right)^{\frac{1}{n}} \leq \frac{x_1+x_2+\cdots+x_{n-1}+\frac{x_1+x_2+\cdots+x_{n-1}}{n-1}}{n}$$

(from the Ind. Hypo.)

$$\left(x_1 x_2 \cdots x_{n-1}\left(\frac{x_1+x_2+\cdots+x_{n-1}}{n-1}\right)\right)^{\frac{1}{n}} \leq \frac{x_1+x_2+\cdots+x_{n-1}}{n-1}$$

$$\left(x_1 x_2 \cdots x_{n-1}\left(\frac{x_1+x_2+\cdots+x_{n-1}}{n-1}\right)\right) \leq \left(\frac{x_1+x_2+\cdots+x_{n-1}}{n-1}\right)^n$$

Arithmetic vs. Geometric Mean (cont.)

$$\left(x_1 x_2 \cdots x_{n-1}\left(\frac{x_1+x_2+\cdots+x_{n-1}}{n-1}\right)\right)^{\frac{1}{n}} \leq \frac{x_1+x_2+\cdots+x_{n-1}+\frac{x_1+x_2+\cdots+x_{n-1}}{n-1}}{n}$$

(from the Ind. Hypo.)

$$\left(x_1 x_2 \cdots x_{n-1}\left(\frac{x_1+x_2+\cdots+x_{n-1}}{n-1}\right)\right)^{\frac{1}{n}} \leq \frac{x_1+x_2+\cdots+x_{n-1}}{n-1}$$

$$\left(x_1 x_2 \cdots x_{n-1}\left(\frac{x_1+x_2+\cdots+x_{n-1}}{n-1}\right)\right) \leq \left(\frac{x_1+x_2+\cdots+x_{n-1}}{n-1}\right)^{n}$$

$$\left(x_1 x_2 \cdots x_{n-1}\right) \leq \left(\frac{x_1+x_2+\cdots+x_{n-1}}{n-1}\right)^{n-1}$$

Arithmetic vs. Geometric Mean (cont.)

$$\left(x_1 x_2 \cdots x_{n-1}\left(\frac{x_1+x_2+\cdots+x_{n-1}}{n-1}\right)\right)^{\frac{1}{n}} \leq \frac{x_1+x_2+\cdots+x_{n-1}+\frac{x_1+x_2+\cdots+x_{n-1}}{n-1}}{n}$$

(from the Ind. Hypo.)

$$\left(x_1 x_2 \cdots x_{n-1}\left(\frac{x_1+x_2+\cdots+x_{n-1}}{n-1}\right)\right)^{\frac{1}{n}} \leq \frac{x_1+x_2+\cdots+x_{n-1}}{n-1}$$

$$\left(x_1 x_2 \cdots x_{n-1}\left(\frac{x_1+x_2+\cdots+x_{n-1}}{n-1}\right)\right) \leq \left(\frac{x_1+x_2+\cdots+x_{n-1}}{n-1}\right)^{n}$$

$$\left(x_1 x_2 \cdots x_{n-1}\right) \leq \left(\frac{x_1+x_2+\cdots+x_{n-1}}{n-1}\right)^{n-1}$$

$$\left(x_1 x_2 \cdots x_{n-1}\right)^{\frac{1}{n-1}} \leq \left(\frac{x_1+x_2+\cdots+x_{n-1}}{n-1}\right)$$

Loop Invariants

- An *invariant* at some point of a program is an assertion that holds whenever execution of the program reaches that point.
- Invariants are a bridge between the static text of a program and its dynamic computation.

Loop Invariants

- An *invariant* at some point of a program is an assertion that holds whenever execution of the program reaches that point.
- Invariants are a bridge between the static text of a program and its dynamic computation.
- An invariant at the front of a while loop is called a *loop invariant* of the while loop.
- A loop invariant is formally established by induction.
  - Base case: the assertion holds right before the loop starts.
  - Inductive step: assuming the assertion holds before the i-th iteration ($i \geq 1$), it holds again after the iteration.

A Variant of Euclid's Algorithm

Algorithm

Algorithm myEuclid(m, n);
begin
  // *assume that* $m > 0$ *and* $n > 0$
  $x := m;$
  $y := n;$
  while $x \neq y$ **do**
    if $x < y$ **then** *swap(x,y)*;
    $x := x - y;$
  od
  $\dots$
end

where swap(x,y) exchanges the values of x and y.

A Variant of Euclid's Algorithm (cont.)

Theorem (Correctness of myEuclid)

When Algorithm myEuclid terminates, x or y stores the value of $\gcd(m, n)$ (assuming that $m, n > 0$ initially).

A Variant of Euclid's Algorithm (cont.)

Theorem (Correctness of myEuclid)

When Algorithm myEuclid terminates, x or y stores the value of $\gcd(m,n)$ *(assuming that $m,n>0$ initially).*

Lemma

Let $Inv(m,n,x,y)$ denote the assertion:

$$x>0 \wedge y>0 \wedge \gcd(x,y)=\gcd(m,n).$$

Then, $Inv(m,n,x,y)$ is a loop invariant of the while loop, assuming that $m,n>0$ initially.

A Variant of Euclid's Algorithm (cont.)

Theorem (Correctness of myEuclid)

When Algorithm myEuclid terminates, x or y stores the value of $\gcd(m, n)$ (assuming that $m, n > 0$ initially).

Lemma

Let $Inv(m, n, x, y)$ denote the assertion:

$$x > 0 \wedge y > 0 \wedge \gcd(x, y) = \gcd(m, n).$$

Then, $Inv(m, n, x, y)$ is a loop invariant of the while loop, assuming that $m, n > 0$ initially.

The loop invariant is sufficient to deduce that, when the while loop terminates, i.e., when $x = y$, either x or y stores the value of $\gcd(x, y)$, which equals $\gcd(m, n)$.

Proof of a Loop Invariant

- The proof is by induction on the number of times the loop body is executed.
- More specifically, we show that
  1. the assertion is true when the flow of control reaches the loop for the first time and
  2. given that the assertion is true and the loop condition holds, the assertion will remain true after the next iteration (i.e., after the loop body is executed once more).

Proof of a Loop Invariant (cont.)

- Base case: $x = m > 0$ and $y = n > 0$, so the loop invariant $Inv(m, n, x, y)$, i.e., $x > 0 \wedge y > 0 \wedge \gcd(x, y) = \gcd(m, n)$, holds.
- Inductive step:
  Given $Inv(m, n, x, y)$ (the Induction Hypothesis), $x \neq y$ (the loop condition), and the effects after the next iteration

$$\begin{array}{ll} & ((x < y) \rightarrow (x' = y - x) \wedge (y' = x)) \\ \wedge & ((x > y) \rightarrow (x' = x - y) \wedge (y' = y)) \\ \wedge & m' = m \\ \wedge & n' = n, \end{array}$$

  it can be shown that $Inv(m', n', x', y')$ also holds.