CHAPTER 4
INDUCTION AND RECURSION

Outline

- **Content**
  - Mathematical induction
  - Strong induction and well-ordering
  - Recursive definitions and structural induction
  - Recursive algorithms
  - Program correctness
- **Reading**
  - Chapter 4

Most of the following slides are by courtesy of Prof. H.-C. Yen, Prof. J.-D. Huang, and Prof. M.P. Frank

Mathematical Induction

- **The first domino falls.**
- **If a domino falls,**
  so will the next domino.
- **All dominoes will fall!**

To see the world in a grain of sand,
And heaven in a wild flower,
Hold infinity in the palm of your hand,
And eternity in an hour.
- William Blake

Courtesy of Prof. C.L. Liu

Mathematical Induction

- **Given the propositional $P(n)$ where $n \in \mathbb{N}$, a proof by mathematical induction is of the form:**

1. **Basis step: The proposition $P(0)$ is shown to be true.**
2. **Inductive step: the implication $P(k) \rightarrow P(k+1)$ is shown to be true for every $k \in \mathbb{N}$**

- **In the inductive step, the statement $P(k)$ is called the inductive hypothesis**

- $(P(0) \wedge \forall k(P(k) \rightarrow P(k+1))) \rightarrow \forall n P(n)$

Example 1

- **Show that the sum of the first *n* odd positive integers is n^2**
- **Pf:**
  - Basis step: $P(1)$: the sum of the first one odd integer = $1 = 1^2$
  - Inductive step: Assume $P(k)$ is true; we shall claim $P(k+1)$ is true.
    $(1+ 3 + 5 + \ldots + (2k-1)) + (2k+1) = k^2 + 2k + 1 = (k+1)^2$
    Hence, $P(k+1)$ is true

- **Note that the starting point is not necessary to be 0**

Example 2

- **Show that $n < 2^n$ for all positive integers**
- **Pf:**
  - Basis step: $P(1)$: $1 < 2^1 = 2$
  - Inductive step: Assume $P(k)$ is true, i.e., $k < 2^k$. We would like to show $P(k+1)$ is true.

    Then $k+1 < 2^k+1 < 2^k + 2^k = 2^{k+1}$. Hence, $P(k+1)$ is true

Example 3

- **Prove $n^3 - n$ can be divided by 3 where $n \in Z^+$**
- **Pf:**
  - Basis step: $P(1)$: $1^3 - 1 = 0$, which can be divided by 3
  - Inductive step: Assume $P(k)$ is true, i.e., $3 \mid (k^3-k)$. We would like to show $P(k+1)$.
  - $(k+1)^3 - (k+1) = (k^3 + 3k^2 + 3k + 1) - (k+1) = (k^3 - k) + 3(k^2 + k)$, which can be divided by 3. Hence $P(k+1)$ is true

Example 4

- **Prove $2^0 + 2^1 + 2^2 + \ldots + 2^n = 2^{n+1} - 1$ where $n \in \mathbf{N}$**
- **Pf:**
  - Basis step: $P(0)$: $2^0 = 1 = 2^{0+1} - 1$
  - Inductive step: Assume $P(k)$ is true, then we will show $P(k+1)$ is true.
  - $(2^0 + 2^1 + 2^2 + \ldots + 2^k) + 2^{k+1} = (2^{k+1} - 1) + 2^{k+1} = 2^{k+2} - 1$. Hence, $P(k+1)$ is true

Example 5

- **Prove $1+2+3+\ldots+n = n(n+1)/2$, $n \in Z^+$**
- **Pf:**
  - Basis step: $P(1)$: $1 = 1(1+1)/2$
  - Inductive step: Assume $P(k)$ is true, i.e., $1+2+\ldots+k = k(k+1)/2$, then we shall claim $P(k+1)$ is true.

    $(1+2+\ldots+k)+ (k+1) = k(k+1)/2+(2k+2)/2 = (k^2+3k+2)/2 = (k+1)(k+2)/2$. Hence, $P(k+1)$ is true

Example 6

- **Prove $2^n < n!$ for every positive integer ≥ 4**
- **Pf:**
  - Basis step: $P(4)$: $2^4 = 16 < 4! = 24$
  - Inductive step: Assume $P(k)$ is true, i.e., $2^k < k!$, then we shall claim $P(k+1)$ is true.
  - $2^{k+1} = 2 \bullet 2^k < 2 \bullet k! < (k+1) * k! = (k+1)!$. Hence, $P(k+1)$ is true

- **Again, not starting from 1**

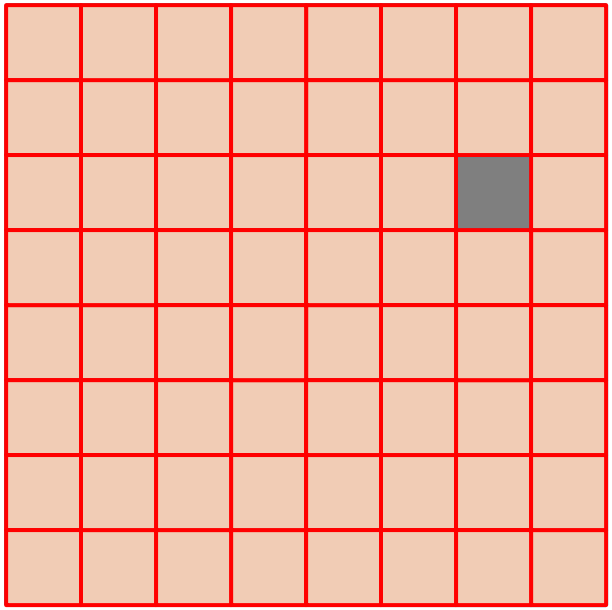
Example 7

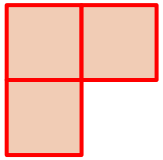
- **Show that if S is a finite set with n elements, then 2^S has 2^n subsets.**
- **Pf:**
  - Let $P(n)$ be the proposition "a set with n elements has 2^n subsets"
  - We use mathematical induction to prove $P(n)$ is true for all n.
  1. Basis step:
     - $P(0)$ is true. Clearly, the empty set has only one ($= 2^0$) subset.
  2. Inductive step:
     - Assume $P(k)$ is true. We would like to show $P(k+1)$ is true.
     - Let T be a set of $k + 1$ elements.

       Without loss of generality, we may assume $T = \{a_0, a_1, \ldots, a_k\}$. Then $T' = \{a_0, a_1, \ldots, a_{k-1}\}$ is a set of k elements and $T = T' \cup \{a_k\}$
     - By inductive hypothesis, T' has 2^k subsets. Then any subset of T is either X or $\{a_k\} \cup X$ for some subset X of T'. Hence the number of subsets of $T = 2 \bullet 2^k = 2^{k+1}$.

Example 8: A Defective Chessboard

- **Any 8×8 defective chessboard can be covered with twenty-one triominoes**
- **Q: How?**





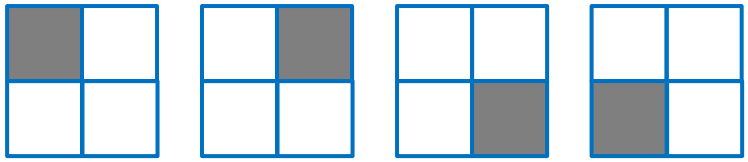
Triomino

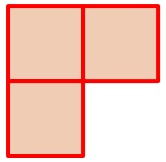
Example 8: A Defective Chessboard

- **Any 8×8 defective chessboard can be covered with twenty-one triominoes**
- **Any $2^n \times 2^n$ defective chessboard can be covered with $1/3(2^n \times 2^n - 1)$ triominoes**
- **Prove by mathematical induction!**

Example 8: Proof by Mathematical Induction

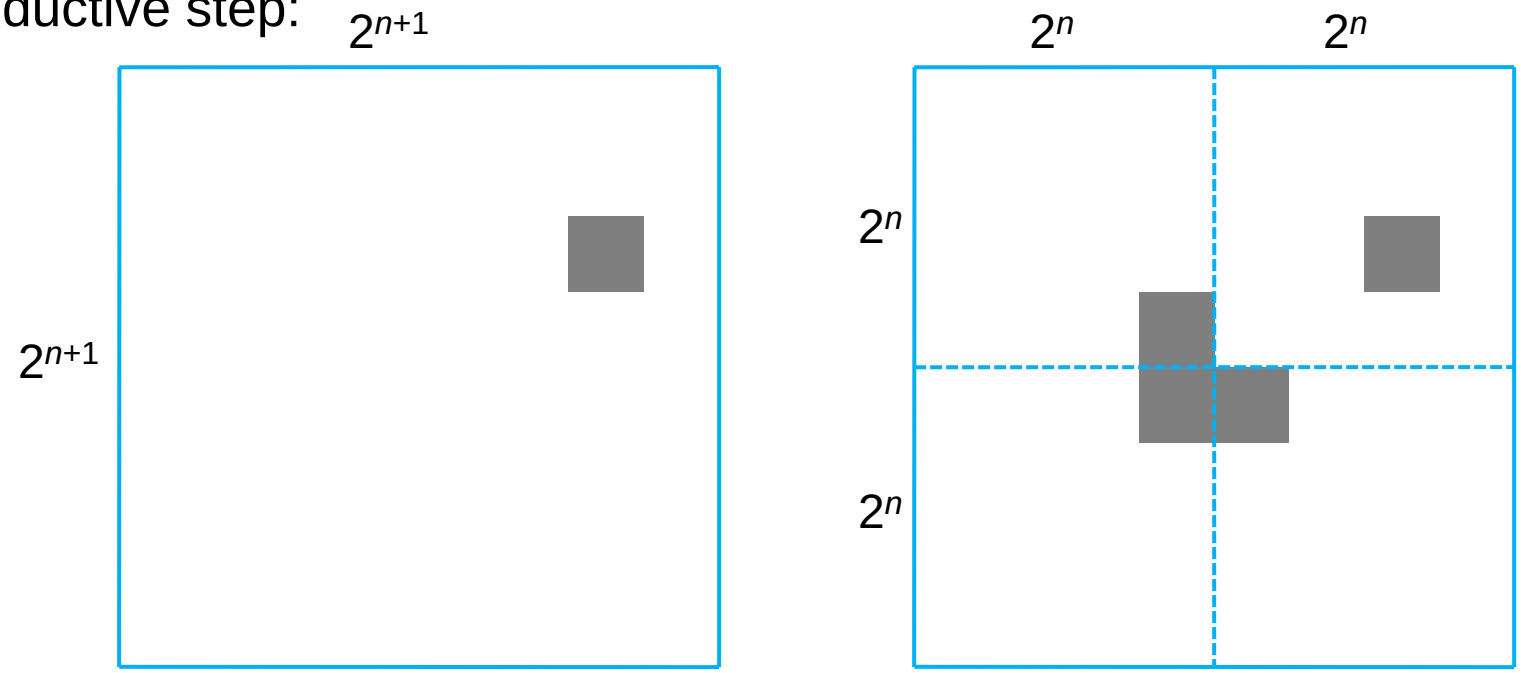
- **Any $2^n \times 2^n$ defective chessboard can be covered with $1/3(2^n \times 2^n - 1)$ trominoes**
  - Basis step:
    - $n=1$





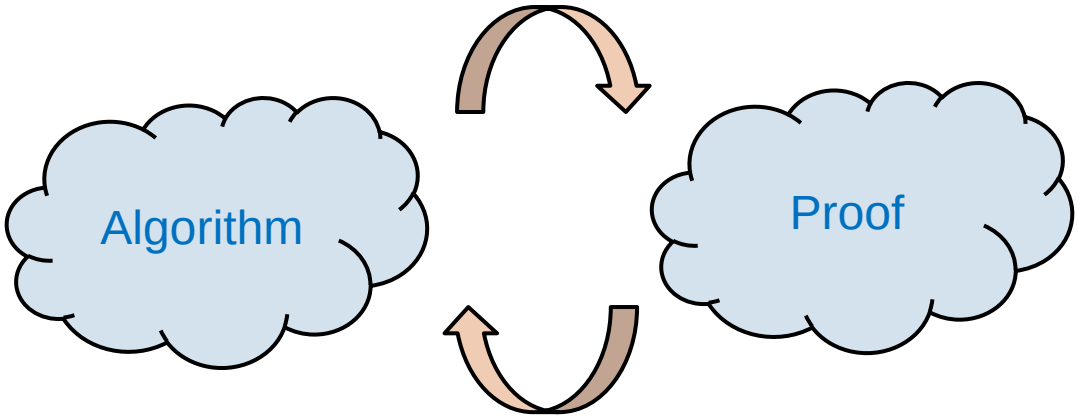
Triomino

  - Inductive step:



Example 8: Proof vs. Algorithm

- **Based on the defective chessboard, we can see**



Strong Induction

- **Given the propositional $P(n)$ where $n \in \mathbf{N}$, a proof by second principle of mathematical induction (or strong induction) is of the form:**

1. **Basis step: The proposition $P(0)$ is shown to be true.**
2. **Inductive step: the implication $[P(0) \wedge P(1) \wedge \ldots \wedge P(k)] \rightarrow P(k+1)$ is shown to be true for every $k \in \mathbf{N}$**

- **Two forms of mathematical induction are equivalent**

Example 1

- **Show that if *n* is an integer > 1, then *n* can be written as the product of primes.**
- **Pf:**
  - Basis step: $n = 2$
  - Inductive step:
    - Assume the proposition is true for $n = 2 \sim k$.
    - We shall claim that the proposition is true for $n = k+1$.
    - If $k+1$ is a prime, we are done!
    - If $k+1$ is composite, then $(k+1) = a*b$,
      where $2 \leq a, b < k+1$,
      and a, b can be written as the products of primes.

Example 2

- **Let $n \in \mathbf{N}$ and $n \geq 12$. Show that $n = 4i + 5j$ for some $i, j \in \mathbf{N}$.**
- **Pf:**
  - Let $P(n)$ be the proposition "$n = 4i + 5j$ for some $i, j \in \mathbf{N}$."
  - We use strong induction to show that $P(n)$ holds for $n \geq 12$.
  - Basis step: $P(12)$, $P(13)$, $P(14)$, $P(15)$ are easy to verify: $12 = 4 \bullet 3$; $13 = 4 \bullet 2 + 5$; $14 = 4 + 5 \bullet 2$; $15 = 5 \bullet 3$
  - Inductive step: Let $k \geq 15$. Assume $P(j)$ is true for $12 \leq j \leq k$.
  - Then $k + 1 = (k - 3) + 4$. By inductive hypothesis, $k - 3 = 4i + 5j$ for some $i, j \in \mathbf{N}$. Hence $k + 1 = 4(i + 1) + 5j$.
- **Remark. Can you prove the statement for $n \geq 4$? If not, why?**

The Well-Ordering Property

- **Well-ordering: a strict total order**

- **Let *S* be a set. We say *S* is well-ordered if there is a relation < defined over *S* such that**
  - For any $a, b \in S$, either $a = b$, $a < b$ or $b < a$
  - For any $a, b, c \in S$, if $a < b$ and $b < c$, $a < c$
  - Every nonempty subset of *S* has a least element (in terms of <)
- **We call 3 the well-ordering property**

- **For instance, N and Z^+ are well-ordered. But Z and Q^+ are not. (Why?)**

Math. Induction vs. Well-Ordering

- **Well-ordering → Math induction**
  - The validity of math induction depends on the well-ordering property.
  - Let us write the mathematical induction formally:

    $(P(0) \wedge \forall k(P(k) \rightarrow P(k+1))) \rightarrow \forall n P(n), n \in \mathrm{N}$
- **Pf:**
  - Suppose $\neg\forall n P(n)$. Let $S = \{m \mid \neg P(m)\}$, $S \subseteq \mathrm{N}$.
  - By our assumption, $S \neq \varnothing$. Hence S has a least element m_0 by the well-ordering property.
  - If $m_0 = 0$, we are done. (Why?)
  - On the other hand, $m_0 > 0$. Hence $m_0 - 1 \geq 0$ and $P(m_0 - 1)$ is true by the choice of m_0. Therefore, $P(m_0 - 1) \rightarrow P(m_0)$ is false. We conclude our proof.
- **Q: Can you prove the converse? That is, math. induction → well-ordering**

Recursion

- **Sometimes it is difficult to define an object explicitly**
  - We will see examples later
- **However, it may be easier to define this objects in terms of itself**
  - This process is called **recursion**

- **Recursive functions: 2 steps to define a recursive function with N as its domain**
  - Basis step (boundary conditions)
    - Specify value of the function at 0, i.e., $f(0)$
  - Recursive step
    - Give a rule for finding $f(n)$, $n > 1$, from values at smaller integers m where $m < n$

Example 1

- ***f* is defined recursively by**
  - $f(0) = 3$ // basis step
  - $f(n+1) = 2f(n) + 3$, $n \in \mathbf{N}$

- **Find f(1) ~ f(4)**
  - $f(1) = 2f(0)+3=9$
  - $f(2) = 2f(1)+3=21$
  - $f(3) = 2f(2)+3=45$
  - $f(4) = 2f(3)+3=93$

Example 2

- **Give a recursive definition of the factorial function $F(n) = n!$**
- **Ans:**
  - Basis step
    - $F(0) = 1$ // by definition
  - Recursive step
    - $F(n+1) = (n+1)F(n)$, $n \in \mathbf{N}$

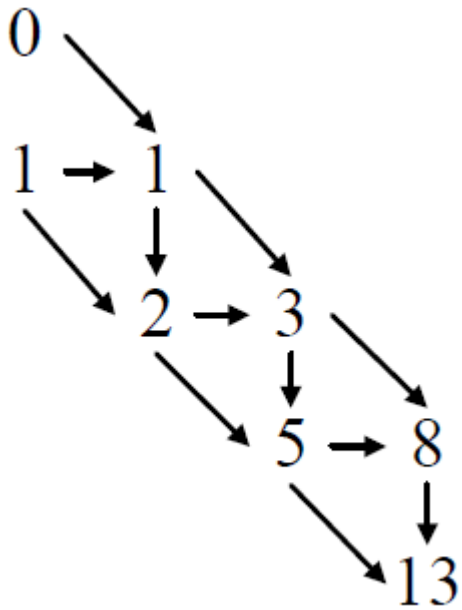
Example 3

- **Give a recursive definition of a^n, where a is a nonzero real and $n \in \mathbf{N}$**
- **Ans:**
  - Basis step
    - $a^0 = 1$ // by definition
  - Recursive step
    - $a^{n+1} = a * a^n$, $n \in \mathbf{N}$

Example 4: Fibonacci Sequence (1/2)

- **In some recursive definitions**
  - The function values of the **first *k*** integers are specified
  - A rule is given to determine the values of larger integers

- **The Fibonacci numbers, f_0, f_1, …, are defined as**
  - Basis step
    - $f_0 = 0$
    - $f_1 = 1$ // two initial terms are defined
  - Recursive step
    - $f_n = f_{n-1} + f_{n-2}$, $n = 2, 3, 4, \ldots$
  - Sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, …



Example 4: Fibonacci Sequence (2/2)

- **Show that whenever $n \geq 3$, $f_n > \alpha^{n-2}$, where $\alpha = (1 + \sqrt{5}) / 2$**
- **Pf:**
  - Basis step
    - $f_3 = 2 > \alpha$
    - $f_4 = 3 > \alpha^2 = (3 + \sqrt{5}) / 2$
  - Inductive step
    - Assume $f_j > \alpha^{j-2}$ for all j with $3 \leq j \leq k$, where $k \geq 4$
    - Based on a fact: $\alpha^2 = \alpha + 1$ (α is a root of $x^2-x-1 = 0$), $\alpha^{k-1} = \alpha^2 \alpha^{k-3} = (\alpha + 1) \alpha^{k-3} = \alpha^{k-2} + \alpha^{k-3}$
    - $f_{k+1} = f_k + f_{k-1} > \alpha^{k-2} + \alpha^{k-3} = \alpha^{k-1}$

Example 5: Recursively Defined Sets

- **A set can also be defined recursively**
- **E.g.,**
  - Basis step
    - $3 \in S$
  - Recursive step
    - If $x \in S$ and $y \in S$, then $x + y \in S$
  - Result
    - $S = \{3, 6, 9, 12, \ldots\}$

Example 6: String

- **An alphabet Σ is a set of symbols**
- **A string over an alphabet Σ is a finite sequence of symbols from Σ**
  - The empty string, denoted as λ, is a string containing no symbols

- **E.g.,**
  - English string (Σ = {a, A, b, B, …})
    - discrete, NCTU
  - Binary string (Σ = {0, 1})
    - 01100010

Example 6: The Set of Strings

- **The set Σ^* of strings over the alphabet Σ can be defined recursively by**
  - Basis step:
    - $\lambda \in \Sigma^*$
  - Recursive step:
    - If $w \in \Sigma^*$ and $x \in \Sigma$, then $wx \in \Sigma^*$
      - w: string; x: symbol

- **Example – binary string, Σ = {0, 1}**
  - Basis step: empty string λ
  - Recursive step: Σ^* = { λ, 0, 1, 00, 01, 10, 11, … }

Example 6: String Concatenation

- **Two strings can be combined via the operation of concatenation, denoted by ·**
  - Basis step:
    - If $w \in \Sigma^*$, then $w \cdot \lambda = w$
  - Recursive step:
    - If $w_1, w_2 \in \Sigma^*$, and $x \in \Sigma$, then $w_1 \cdot (w_2 x) = (w_1 \cdot w_2)x$
- **E.g.,**
  - If w_1 = abc, w_2 = def, then $w_1 \cdot w_2$ = abcdef

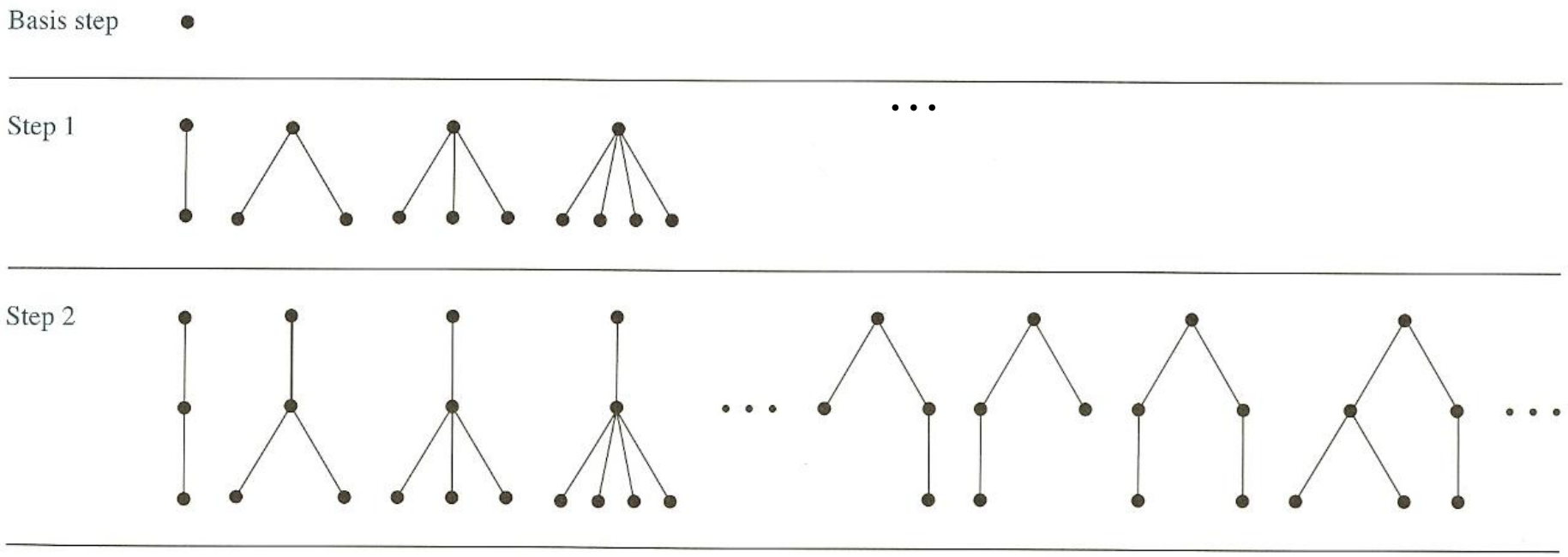
Example 6: Length of a String

- **Give a recursive definition of *l*(*w*), the length of the string *w***
  - Basis step:
    - $l(\lambda) = 0$ // by definition
  - Recursive step:
    - $l(wx) = l(w) + 1$, where $w \in \Sigma^*$ and $x \in \Sigma$

Example 7: Rooted Trees (1/2)

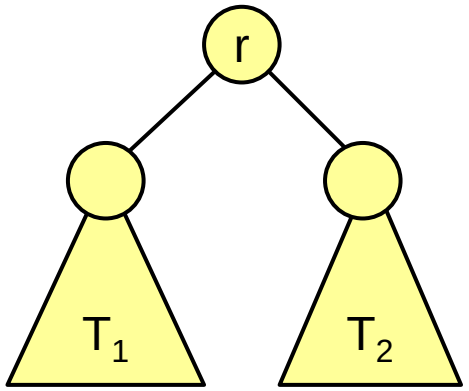
- **A rooted tree consists of**
  - A set of vertices containing a distinguished vertex called the root
  - And a set of edges connecting these vertices
- **The set of rooted trees can be recursively defined**
  - Basis step:
    - A single vertex r is a rooted tree
  - Recursive step:
    - Suppose that $T_1, T_2, \ldots, T_n$ are disjoint rooted trees with roots $r_1, r_2, \ldots, r_n$. Then the graph formed by starting with a root r, which is not in $T_1, T_2, \ldots, T_n$, and adding an edge from r to $r_1, r_2, \ldots, r_n$, is also a rooted tree

Example 7: Rooted Trees (2/2)

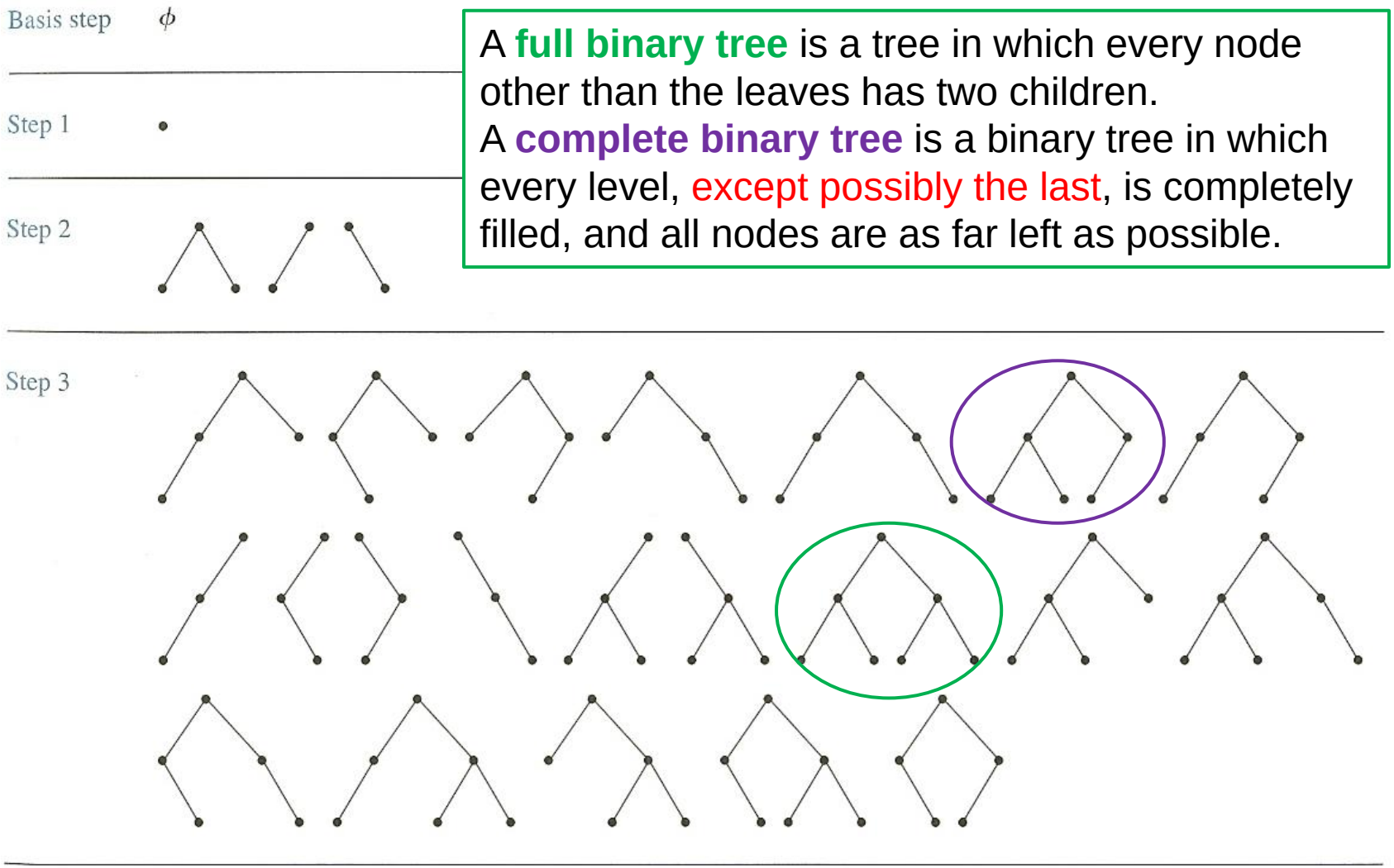


Example 8: Binary Trees (1/2)

- **The set of extended binary trees can be recursively defined**
  - Basis step:
    - The empty set is an extended binary tree
  - Recursive step:
    - If T_1 and T_2 are disjoing extended binary trees, there is an extended binary tree, denoted by $T_1 \cdot T_2$, consisting of a root r together with edges connecting the root to each of the roots of the left subtree T_1 and the right subtree T_2



Example 8: Binary Trees (2/2)



A **full binary tree** is a tree in which every node other than the leaves has two children.
A **complete binary tree** is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.

Example 8: Height of a Binary Tree

- **The height $h(T)$ of a nonempty binary tree T can be defined recursively**
  - Basis step:
    - The height of the binary tree T consisting of only a root r is $h(T) = 0$
  - Recursive step:
    - If T_1 and T_2 are binary trees, then the binary tree $T = T_1 \cdot T_2$ has height $h(T) = 1 + \max(h(T_1), h(T_2))$

Example 8: Vertices in a Binary Tree

- **The number of vertices $n(T)$ in a nonempty binary tree T can be found recursively**
  - Basis step:
    - $n(T)$ of the binary tree T consisting of only a root r is 1
  - Recursive step:
    - If T_1 and T_2 are binary trees, then the number of vertices in the binary tree $T = T_1 \cdot T_2$ is $n(T) = 1 + n(T_1) + n(T_2)$

Structural Induction

- **Structural induction can be used to prove results about recursively defined sets**
- **A proof by structural induction consists of 2 parts**
  - Basis step: show that the results holds for all elements specified in the basis step
  - Recursive step: show that if the statement is true for each of the elements used to construct new elements in the recursive step, then the results holds for these new elements

Example 6: Strings

- **Assume that $P(w)$ is a propositional function over the set of strings $w \in \Sigma^*$, we need to**
  - Basis step: show that $P(\lambda)$ = true
  - Recursive step: Assume that $P(w)$ is true where $w \in \Sigma^*$, show that if $x \in \Sigma$, then $P(wx)$ must also be true

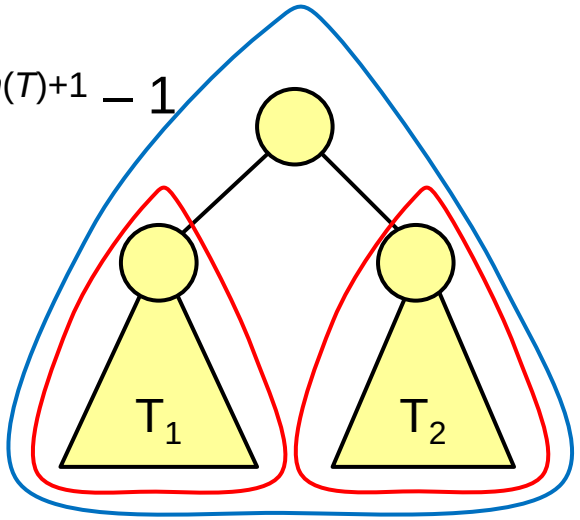
- **Use structural induction to prove that $l(xy) = l(x) + l(y)$ where x and $y \in \Sigma^*$**
  - Let $P(y)$ be the statement that $l(xy) = l(x) + l(y)$
  - Basis step: show that $P(\lambda)$ is true
    - $l(x\lambda) = l(x) = l(x) + 0 = l(x) + l(\lambda)$
  - Recursive step:
    - Assume $l(xy) = l(x) + l(y)$ and $a \in \Sigma$, then $l(xya) = l(xy)+1 = l(x) + l(y) + 1 = l(x) + l(ya)$

Example 8: Binary Trees (1/2)

- **To prove a result about binary trees using structural induction**
  - Basis step:
    - Show that the result is true for the tree consisting of a single vertex
  - Recursive step:
    - Show that if the result is true for T_1 and T_2, then it must be true for tree $T_1 \cdot T_2$ consisting of a root r

Example 8: Binary Trees (2/2)

- **Show that if T is a nonempty binary tree, then $n(T) \le 2^{h(T)+1} - 1$**
- **Pf:**
  - Basis step:
    - For the binary tree T consisting of only a root, $n(T) = 1 \le 2^{0+1}-1$
  - Recursive step:
    - Assume $n(T_1) \le 2^{h(T1)+1} - 1$ and $n(T_2) \le 2^{h(T2)+1} - 1$,
      then $n(T) = 1 + n(T_1) + n(T_2)$
      $\le 1 + (2^{h(T1)+1} - 1) + (2^{h(T2)+1} - 1)$
      $\le 2 * \max(2^{h(T1)+1}, 2^{h(T2)+1}) - 1$
      $= 2 * 2^{\max(h(T1), h(T2))+1} - 1 = 2 * 2^{h(T)} - 1 = 2^{h(T)+1} - 1$



Recursive Algorithms

- **An algorithm is called recursive if it solves a problem by reducing it to an instance of the same problem with smaller input**

- **E.g., give a recursive algorithm to compute a^n where a is a nonzero real and n is a nonnegative integer**
  - $a^n = a \cdot a^{n-1}$

ALGORITHM 1 A Recursive Algorithm for Computing a^n.

procedure *power*(a: nonzero real number, n: nonnegative integer)
if $n = 0$ **then** $power(a, n) := 1$
else $power(a, n) := a \cdot power(a, n - 1)$

Example 1: Recursive Algorithm for GCD

- **Give a recursive algorithm to compute gcd(*a*, *b*) where *a* and *b* are 2 nonnegative integers with *a* < *b***
  - gcd(*a*, *b*) = gcd(*a*, *b*-*a*)

```
// Recursive algorithm
procedure gcd(a, b: nonnegative integers with a < b)
        if a = 0 then gcd(a, b) := b
        else gcd(a, b) := gcd(b mod a, a)
```

```
// Iterative algorithm
procedure gcd(a, b: positive integers)
        while b ≠ 0
                r := a mod b; a := b; b := r
        gcd(a, b) := a
```

Example 3: Linear Search

- **Search for x in the sequence $a_1, a_2, \ldots, a_n$**
- **Develop search(i, j, x) to search for x in the sequence $a_i, \ldots, a_j$**

ALGORITHM 4 A Recursive Linear Search Algorithm.

procedure $search(i, j, x)$
if $a_i = x$ **then**
    $location := i$
else if $i = j$ **then**
    $location := 0$
else
    $search(i + 1, j, x)$

Example 4: Recursive Binary Search

ALGORITHM 5 A Recursive Binary Search Algorithm.

procedure *binary search*(x, i, j)
$m := \lfloor (i + j)/2 \rfloor$
if $x = a_m$ **then**
    $location := m$
else if $(x < a_m$ and $i < m)$ **then**
    binary search$(x, i, m - 1)$
else if $(x > a_m$ and $j > m)$ **then**
    binary search$(x, m + 1, j)$
else $location := 0$

Example 5: Recursive and Iterative Factorial

ALGORITHM 6 A Recursive Procedure for Factorials.

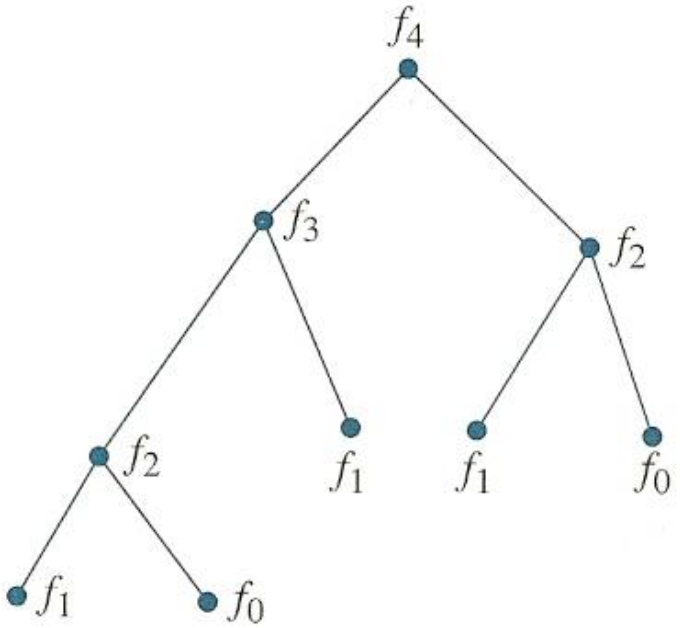
procedure *factorial*(n: positive integer)
if $n = 1$ **then**
    $factorial(n) := 1$
else
    $factorial(n) := n \cdot factorial(n - 1)$

procedure iterative_factorial (*n*: positive integer)
x := 1
for *i* := 1 **to** *n*
    x := *i* * *x*
iterative_factorial(n) := x

Example 6: Calculation for Fibonacci (1/2)

ALGORITHM 8 A Recursive Algorithm for Fibonacci Numbers.

procedure *fibonacci*(n: nonnegative integer)
if $n = 0$ **then** *fibonacci*$(0) := 0$
else if $n = 1$ **then** *fibonacci*$(1) := 1$
else *fibonacci*$(n) :=$ *fibonacci*$(n-1) +$ *fibonacci*$(n-2)$

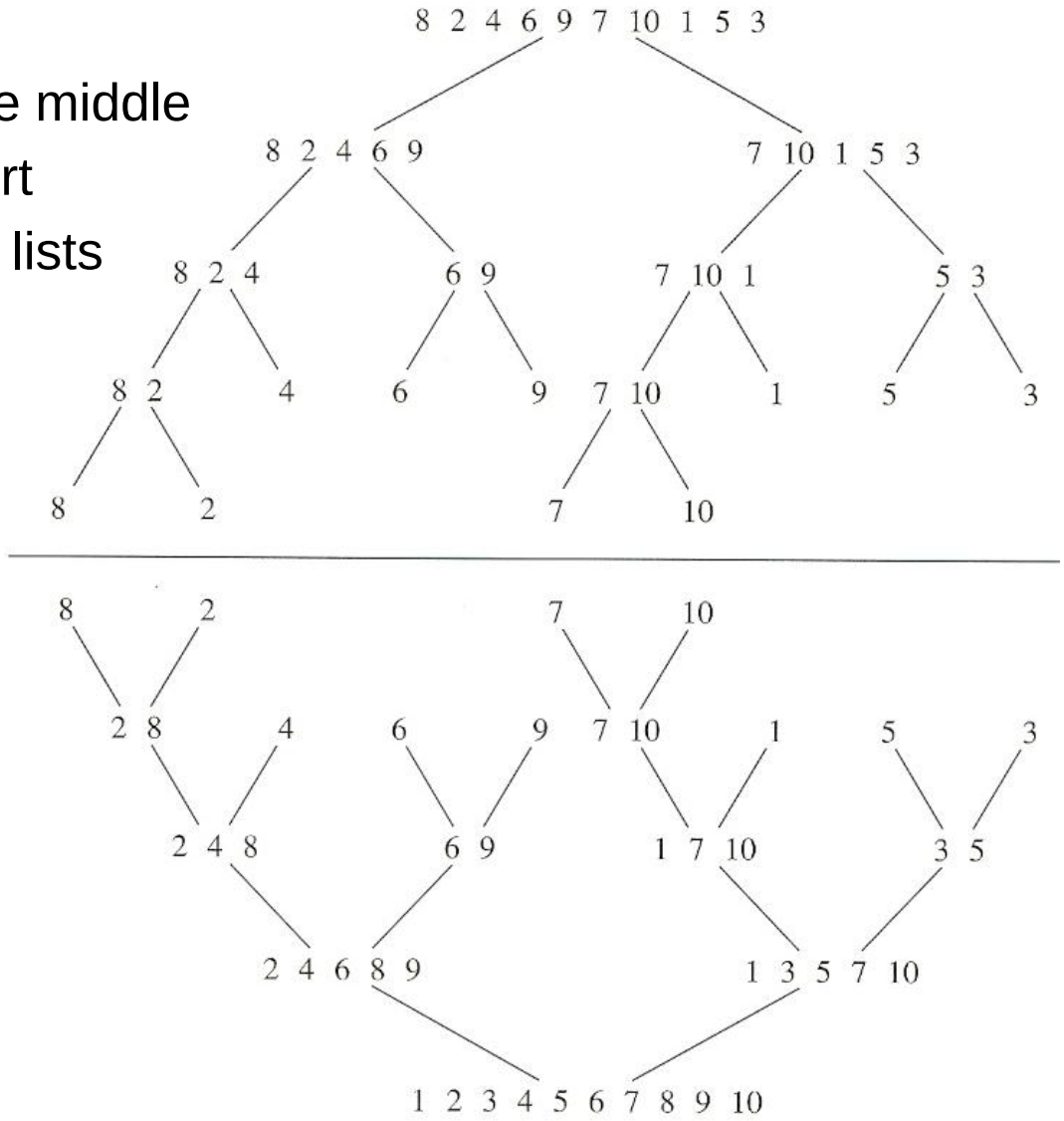


Example 6: Calculation for Fibonacci (2/2)


```
procedure iterative fibonacci(n: nonnegative integer)
if n = 0 then y := 0
else
begin
      x := 0
      y := 1
      for i := 1 to n − 1
      begin
            z := x + y
            x := y
            y := z
      end
end
{y is the nth Fibonacci number}
```

Example 7: Merge Sort (1/3)

- **Divide and conquer**
  - Divide: partition w.r.t. the middle
  - Conquer: recursively sort
  - Combine: merge sorted lists
- **Learn more D&C in "Data Structures" or "Algorithms"**



Example 7: Merge Sort (2/3)

ALGORITHM 10 A Recursive Merge Sort.

procedure $mergesort(L = a_1, \ldots, a_n)$
if $n > 1$ **then**
    $m := \lfloor n/2 \rfloor$
    $L_1 := a_1, a_2, \ldots, a_m$
    $L_2 := a_{m+1}, a_{m+2}, \ldots, a_n$
    $L := merge(mergesort(L_1), mergesort(L_2))$
{L is now sorted into elements in nondecreasing order}

Example 7: Merge Sort (3/3)

2 sorted lists with n_1 and n_2 elements can be merged into a sorted list using no more than $n_1 + n_2 - 1$ comparisons

ALGORITHM 11 Merging Two Lists.

procedure *merge*(L_1, L_2: lists)
L := empty list
while L_1 and L_2 are both nonempty
begin
  remove smaller of first element of L_1 and L_2 from the list it is in and put it at the left end of L
  if removal of this element makes one list empty **then** remove all elements from the other list and append them to L
end {L is the merged list with elements in increasing order}

TABLE 1 Merging the Two Sorted Lists 2, 3, 5, 6 and 1, 4.

First List	*Second List*	*Merged List*	*Comparison*
2 3 5 6	1 4		1 < 2
2 3 5 6	4	1	2 < 4
3 5 6	4	1 2	3 < 4
5 6	4	1 2 3	4 < 5
5 6		1 2 3 4	
		1 2 3 4 5 6	

Program Correctness

- **A program segment *S* is said to be partially correct with respect to the initial assertion *p* and the final assertion *q*, write *p*{*S*}*q*, if whenever *p* is true for the input values of *S* and *S* terminates, then *q* is true for the output values of *S*.**

- **Remark: The notation *p*{*S*}*q* is known as Hoare triple after tony Hoare.**
  - The inventor of Quicksort

Inference Rules

- **With Hoare triples, it is possible to define the semantics of programming languages. Here, we give some examples:**
- **Consider the inference rules:**

$$\frac{p\{S_0\}q \qquad r\{S_1\}t \qquad q \rightarrow r}{p\{S_0; S_1\}t}$$

// consecutive program segments

  - and

$$\frac{(p \wedge c)\{S_0\}q \quad (p \wedge \neg c)\{S_1\}q}{p\{\textbf{if } c \textbf{ then } S_0 \textbf{ else } S_1\}q}$$

// conditional statement

  - In addition to conditionals, we have inference rules for loops.

$$\frac{(p \wedge c)\{S\}p}{p\{\textbf{while } c \textbf{ do } S \textbf{ od}\}(\neg c \wedge p)}$$

Example

- **A program segment for Computing |*x*|**
  1. **if** $x < 0$ **then**
  2. $abs := -x$
  3. **else**
  4. $abs := x$

- **Verify the program segment with respect to the initial assertion T and final assertion *abs* = |*x*|.**
- **Pf:**
  - We have (**T** $\wedge$ $x < 0$) {*abs* := -*x*} (abs = |*x*|) and
    (**T** $\wedge$ $x \not< 0$) {*abs* := *x*} (*abs* = |*x*|).
  - By the inference rule, we conclude *abs* = |*x*| at the end of the program segment.

Semantics

- **In practice, we do not write programs as Hoare triples. However, we can add assertions to help us catch bugs. If you write C (or C++) programs, please try to use assert() as often as possible. They can save you a lot of time!**

- **The subject of understanding the meaning of programs is called program semantics. As we have seen in the example, Hoare triples can be used to specify the semantics of programs. We call them axiomatic semantics. There are other ways to define the semantics of programs. The semantics of programming languages is yet another interesting field of theoretical computer science.**