Induction and recursion

Chapter 5

With Question/Answer Animations

Chapter Summary

- Mathematical Induction
- Strong Induction
- Well-Ordering
- Recursive Definitions
- Structural Induction
- Recursive Algorithms
- Program Correctness (*not yet included in overheads*)

Mathematical Induction

Section 5.1

Section Summary

- Mathematical Induction
- Examples of Proof by Mathematical Induction
- Mistaken Proofs by Mathematical Induction
- Guidelines for Proofs by Mathematical Induction

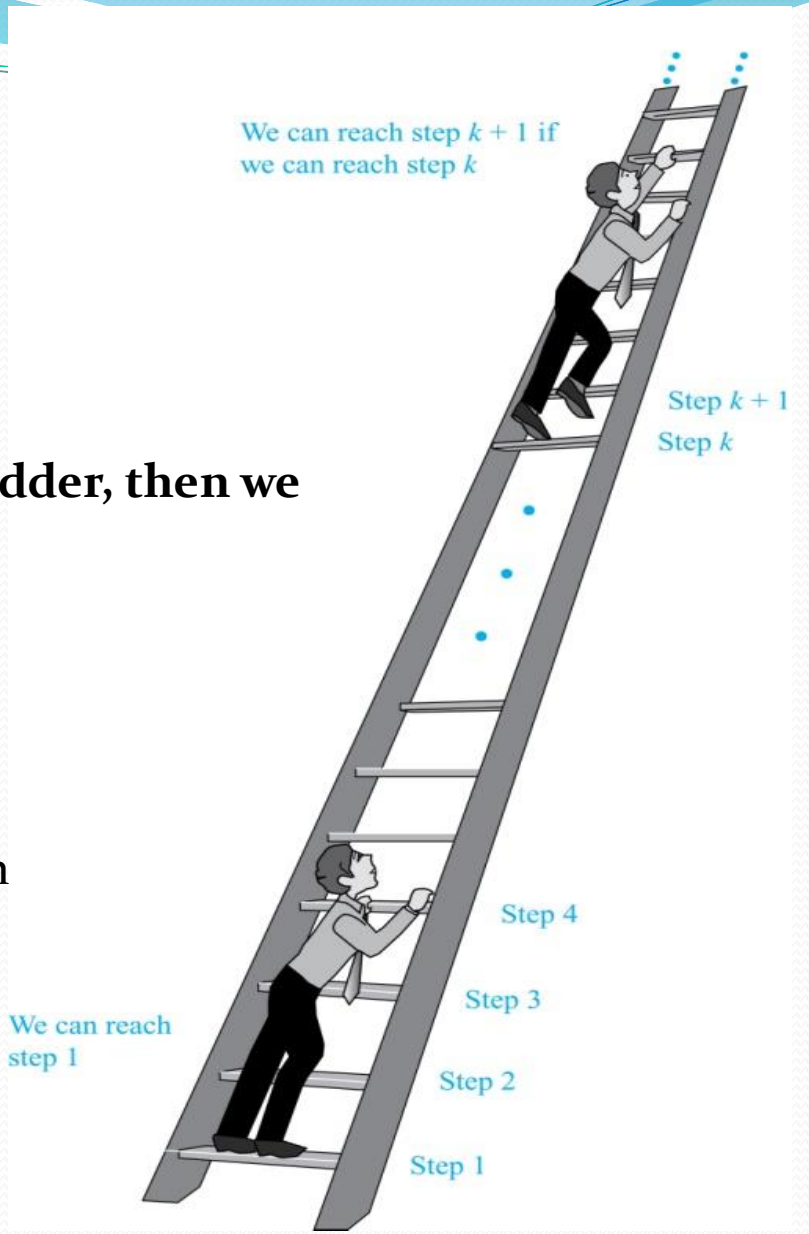
Climbing an Infinite Ladder

Suppose we have an infinite ladder:

1. We can reach **the first rung** of the ladder.
2. If we can reach **a particular rung of the ladder, then we can reach the next rung.**

From (1), we can reach the first rung. Then by applying (2), we can reach the second rung. Applying (2) again, the third rung. And so on. We can apply (2) any number of times to reach any particular rung, no matter how high up.

This example motivates proof by mathematical induction.



Principle of Mathematical Induction

Principle of Mathematical Induction: To prove that $P(n)$ is true for all positive integers n, we complete these steps:

- *Basis Step*: Show that $P(1)$ is true.
- *Inductive Step*: Show that $P(n) \rightarrow P(n+1)$ is true for all positive integers n.

To complete the inductive step, assuming the *inductive hypothesis* that $P(n)$ holds for an arbitrary integer n, show that must $P(n+1)$ be true.

Climbing an Infinite Ladder Example:

- BASIS STEP: By (1), we can reach rung 1.
- INDUCTIVE STEP: Assume the inductive hypothesis that we can reach rung n. Then by (2), we can reach rung $n+1$.

Hence, $P(n) \rightarrow P(n+1)$ is true for all positive integers n. We can reach every rung on the ladder.

Important Points About Using Mathematical Induction

- Mathematical induction can be expressed as the rule of inference

$$(P(1) \land \forall k\,(P(k) \rightarrow P(k+1))) \rightarrow \forall n\,P(n),$$

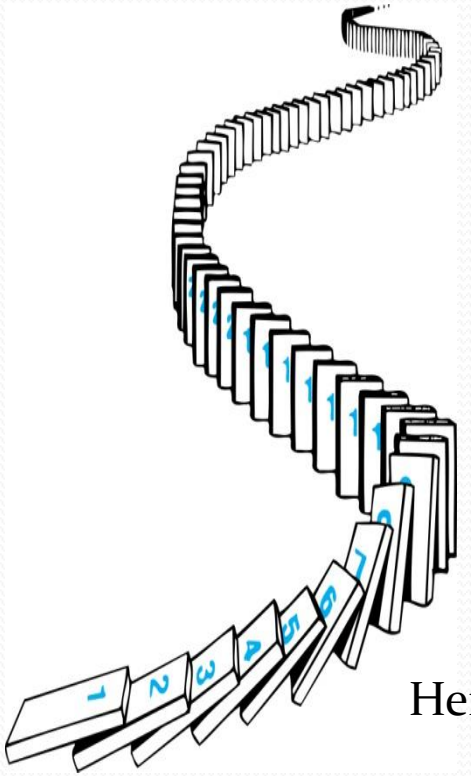
where the domain is the set of positive integers.

- In a proof by mathematical induction, we don't assume that $P(k)$ is true for all positive integers! We show that if we assume that $P(k)$ is true, then $P(k+1)$ must also be true.
- Proofs by mathematical induction do not always start at the integer 1. In such a case, the basis step begins at a starting point b where b is an integer. We will see examples of this soon.

Remembering How Mathematical Induction Works

Consider an infinite sequence of dominoes, labeled 1,2,3, …, where each domino is standing.

Let $P(n)$ be the proposition that the nth domino is knocked over.



We know that the first domino is knocked down, i.e., $P(1)$ is true .

We also know that if whenever the kth domino is knocked over, it knocks over the $(k + 1)$st domino, i.e, $P(k) \rightarrow P(k + 1)$ is true for all positive integers k.

Hence, all dominos are knocked over.

$P(n)$ is true for all positive integers n.

Proving a Summation Formula by Mathematical Induction

Example: Show that: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Solution:

Note: Once we have this conjecture, mathematical induction can be used to prove it correct

- BASIS STEP: $P(1)$ is true since $1(1 + 1)/2 = 1$.
- INDUCTIVE STEP: Assume true for $P(k)$.

The inductive hypothesis is $\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$

Under this assumption,

$$1 + 2 + \ldots + k + (k+1) = \frac{k(k+1)}{2} + (k+1)$$

$$= \frac{k(k+1) + 2(k+1)}{2}$$

$$= \frac{(k+1)(k+2)}{2}$$

Conjecturing and Proving Correct a Summation Formula

Example: Conjecture and prove correct a formula for the sum of the first n positive odd integers. Then prove your conjecture.

Solution: We have: $1 = 1, 1 + 3 = 4, 1 + 3 + 5 = 9, 1 + 3 + 5 + 7 = 16, 1 + 3 + 5 + 7 + 9 = 25.$

- We can conjecture that the sum of the first n positive odd integers is n^2,

$$1 + 3 + 5 + \cdots + (2n - 1) = n^2$$

- We prove the conjecture with mathematical induction.
- BASIS STEP: $P(1)$ is true since $1^2 = 1$.
- INDUCTIVE STEP: $P(k) \rightarrow P(k + 1)$ for every positive integer k.
  Assume the inductive hypothesis holds and then show that $P(k + 1)$ holds has well.

Inductive Hypothesis: $1 + 3 + 5 + \cdots + (2k - 1) = k^2$

- So, assuming $P(k)$, it follows that:

$$\begin{aligned} 1 + 3 + 5 + \cdots + (2k - 1) + (2k + 1) &= [1 + 3 + 5 + \cdots + (2k - 1)] + (2k + 1) \\ &= k^2 + (2k + 1) \quad (\textit{by the inductive hypothesis}) \\ &= k^2 + 2k + 1 \\ &= (k + 1)^2 \end{aligned}$$

- Hence, we have shown that $P(k + 1)$ follows from $P(k)$. Therefore the sum of the first n positive odd integers is n^2.

Proving Inequalities

Example: Use mathematical induction to prove that $n < 2^n$ for all positive integers n.

Solution: Let $P(n)$ be the proposition that $n < 2^n$.

- BASIS STEP: $P(1)$ is true since $1 < 2^1 = 2$.
- INDUCTIVE STEP: Assume $P(k)$ holds, i.e., $k < 2^k$, for an arbitrary positive integer k.
- Must show that $P(k + 1)$ holds. Since by the inductive hypothesis, $k < 2^k$, it follows that:

$$k + 1 < 2^k + 1 \leq 2^k + 2^k = 2 \cdot 2^k = 2^{k+1}$$

Therefore $n < 2^n$ holds for all positive integers n. ◀

Proving Inequalities

Example: Use mathematical induction to prove that $2^n < n!$, for every integer $n \geq 4$.

Solution: $P(\text{n}) = 2^n < n!$

- BASIS STEP: $P(4)$ is true since $2^4 = 16 < 4! = 24$.
- INDUCTIVE STEP: Assume $P(k)$ holds, i.e., $2^k < k!$ for an arbitrary integer $k \geq 4$. To show that $P(k+1)$ holds:

$$\begin{aligned} 2^{k+1} &= 2 \cdot 2^k \\ &< 2 \cdot k! \qquad (\textit{by the inductive hypothesis}) \\ &< (k+1)k! \\ &= (k+1)! \end{aligned}$$

Therefore, $2^n < n!$ holds, for every integer $n \geq 4$. ◀

Note that here the basis step is $P(4)$, since $P(0)$, $P(1)$, $P(2)$, and $P(3)$ are all false.

Proving Divisibility Results

Example: Use mathematical induction to prove that $n^3 - n$ is divisible by 3, for every positive integer n.

Solution: Let $P(n)$ be the proposition that $n^3 - n$ is divisible by 3.

- BASIS STEP: $P(1)$ is true since $1^3 - 1 = 0$, which is divisible by 3.
- INDUCTIVE STEP: Assume $P(k)$ holds, i.e., $k^3 - k$ is divisible by 3, for an arbitrary positive integer k. To show that $P(k+1)$ follows:

$$(k+1)^3 - (k+1) = (k^3 + 3k^2 + 3k + 1) - (k+1)$$
$$= (k^3 - k) + 3(k^2 + k)$$

By the inductive hypothesis, the first term $(k^3 - k)$ is divisible by 3 and the second term is divisible by 3 since it is an integer multiplied by 3. So by part (i) of Theorem 1 in Section 4.1 , $(k+1)^3 - (k+1)$ is divisible by 3.

Therefore, $n^3 - n$ is divisible by 3, for every integer positive integer n. ◀

Strong Induction and Well-Ordering

Section 5.2

Section Summary

- Strong Induction
- Example Proofs using Strong Induction

Strong Induction

- *Strong Induction*: To prove that $P(n)$ is true for all positive integers n, where $P(n)$ is a propositional function, complete two steps:
  - *Basis Step*: Verify that the proposition $P(1)$ is true.
  - *Inductive Step*: Show the conditional statement $[P(1) \wedge P(2) \wedge \cdots \wedge P(k)] \rightarrow P(k+1)$ holds for all positive integers k.

Strong Induction is sometimes called the *second principle of mathematical induction* or *complete induction*.

Strong Induction and the Infinite Ladder

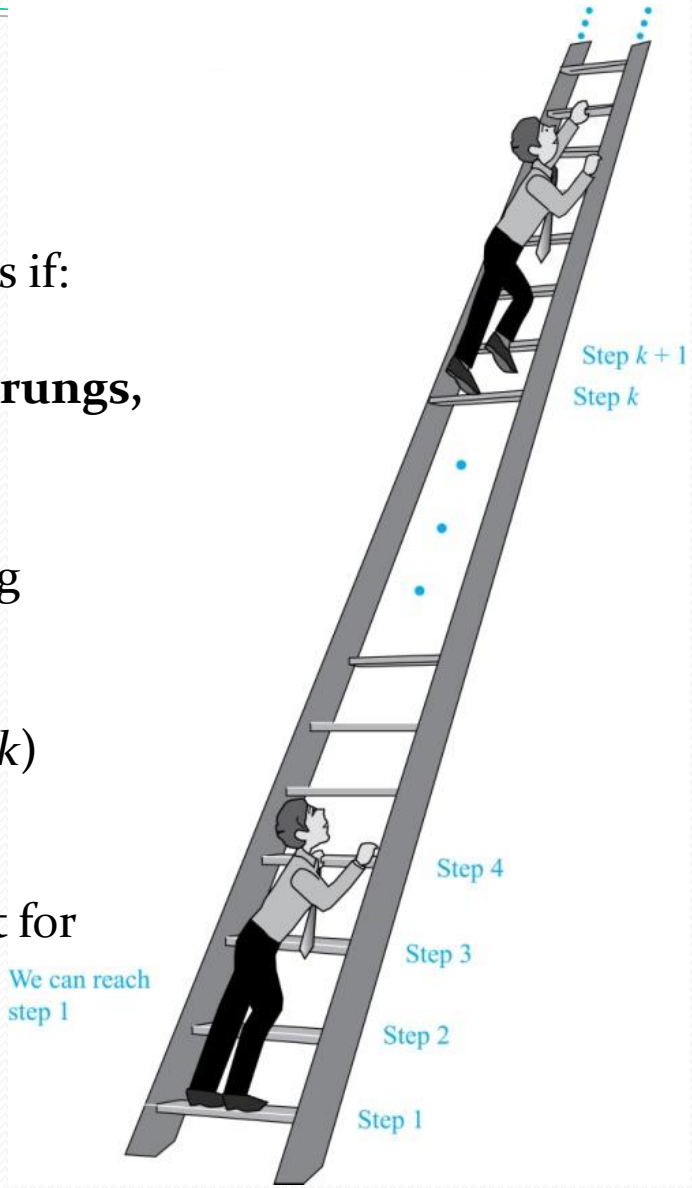
Strong induction tells us that we can reach all rungs if:

1. We can reach the **first rung** of the ladder.
2. For every integer k, **if we can reach the first k rungs, then we can reach the $(k + 1)$st rung.**

To conclude that we can reach every rung by strong induction:

- BASIS STEP: $P(1)$ holds
- INDUCTIVE STEP: Assume $P(1) \wedge P(2) \wedge \cdots \wedge P(k)$ holds for an arbitrary integer k, and show that $P(k + 1)$ must also hold.

We will have then shown by strong induction that for every positive integer n, $P(n)$ holds, i.e., we can reach the nth rung of the ladder.



Which Form of Induction Should Be Used?

- We can always use strong induction instead of mathematical induction. But there is no reason to use it if it is simpler to use mathematical induction.

Proof using Strong Induction

Example: Prove that every amount of postage of 12 cents or more can be formed using just 4-cent and 5-cent stamps.

Solution: Let $P(n)$ be the proposition that postage of n cents can be formed using 4-cent and 5-cent stamps.

- BASIS STEP: $P(12)$, $P(13)$, $P(14)$, and $P(15)$ hold.
  - $P(12)$ uses three 4-cent stamps.
  - $P(13)$ uses two 4-cent stamps and one 5-cent stamp.
  - $P(14)$ uses one 4-cent stamp and two 5-cent stamps.
  - $P(15)$ uses three 5-cent stamps.
- INDUCTIVE STEP: **Take any $n \geq 16$.** The inductive hypothesis states that $P(k)$ holds for all k: $12 \leq k < n$. Assuming the inductive hypothesis, it can be shown that $P(n)$ holds.
- **To form postage of n cents, add a 4-cent stamp to the postage for $n - 4$ cents. Using the inductive hypothesis, $P(n - 4)$ holds, since $n - 4 \geq 12$ for $n \geq 16$.**
- **Hence, $P(n)$ holds for all $n \geq 12$.**

Recursive Definitions and Structural Induction

Section 5.3

Section Summary

- Recursively Defined Functions
- Recursively Defined Sets and Structures
- Structural Induction
- Generalized Induction

Recursively Defined Functions

Definition: A *recursive* or *inductive definition* of a function consists of two steps.

- BASIS STEP: Specify the value of the function at zero (or some other small value).
- RECURSIVE STEP: Give a rule for finding its value at an integer from its values at smaller integers.

Recursively Defined Functions

- **Example**: Suppose f is defined by:
- $f(0) = 3$,
- $f(n + 1) = 2f(n) + 3$
- Find $f(1), f(2), f(3), f(4)$
- **Solution**:
  - $f(1) = 2f(0) + 3 = 2 \cdot 3 + 3 = 9$
  - $f(2) = 2f(1) + 3 = 2 \cdot 9 + 3 = 21$
  - $f(3) = 2f(2) + 3 = 2 \cdot 21 + 3 = 45$
  - $f(4) = 2f(3) + 3 = 2 \cdot 45 + 3 = 93$

- **Example:** Give a recursive definition of the factorial function $n!$:
- **Solution**:
  - $f(0) = 1$
  - $f(n + 1) = (n + 1) \cdot f(n)$

Recursively Defined Sets and Structures

Recursive definitions of sets have two parts:

- The *basis step* specifies an initial collection of elements.
- The *recursive step* gives the rules for forming new elements in the set from those already known to be in the set.

Recursively Defined Sets and Structures

Example: The natural numbers **N**.

BASIS STEP: $0 \in \mathbf{N}$.

RECURSIVE STEP: If n is in **N**, then $n + 1$ is in **N**.

- Initially 0 is in S, then $0 + 1 = 1$, then $1 + 1 = 2$, etc.

Example : **S: set of** positive integers divisible by 3

BASIS STEP: $3 \in S$.

RECURSIVE STEP: If $x \in S$ and $y \in S$, then $x + y$ is in S.

- Initially 3 is in S, then $3 + 3 = 6$, then $3 + 6 = 9$, etc.

Strings

Definition: The set Σ^* of *strings* over the alphabet Σ:

BASIS STEP: $\lambda \in \Sigma^*$ (λ is the empty string)

RECURSIVE STEP: If w is in Σ^* and x is in Σ,
then wx (concatenation of word w and symbol x) $\in \Sigma^*$

Example: If $\Sigma = \{0,1\}$, the strings in in Σ^* are the set of all bit strings, λ,0,1, 00,01,10, 11, etc.

Example: If $\Sigma = \{a,b\}$, show that aab is in Σ^*.

- Since $\lambda \in \Sigma^*$ and $a \in \Sigma$, $a \in \Sigma^*$.
- Since $a \in \Sigma^*$ and $a \in \Sigma$, $aa \in \Sigma^*$.
- Since $aa \in \Sigma^*$ and $b \in \Sigma$, $aab \in \Sigma^*$.

Balanced Parentheses

Example: Give a recursive definition of the set of balanced parentheses P.

Solution:

BASIS STEP: $() \in P$

RECURSIVE STEP: If $w \in P$, then $()w \in P$, $(w) \in P$ and $w() \in P$.

- Show that $((())())$ is in P.
- Why is $))((()$ not in P?

Well-Formed Formulae in Propositional Logic

Definition: The set of *well-formed formulae* in propositional logic involving **T**, **F**, propositional variables, and operators from the set $\{\neg,\wedge,\vee,\rightarrow,\leftrightarrow\}$.

BASIS STEP: **T**,**F**, and *s*, where *s* is a propositional variable, are well-formed formulae.

RECURSIVE STEP: If *E* and *F* are well formed formulae, then $(\neg E)$, $(E \wedge F)$, $(E \vee F)$, $(E \rightarrow F)$, $(E \leftrightarrow F)$, are well-formed formulae.

Examples: $((p \vee q) \rightarrow (q \wedge \mathbf{F}))$ is a well-formed formula.

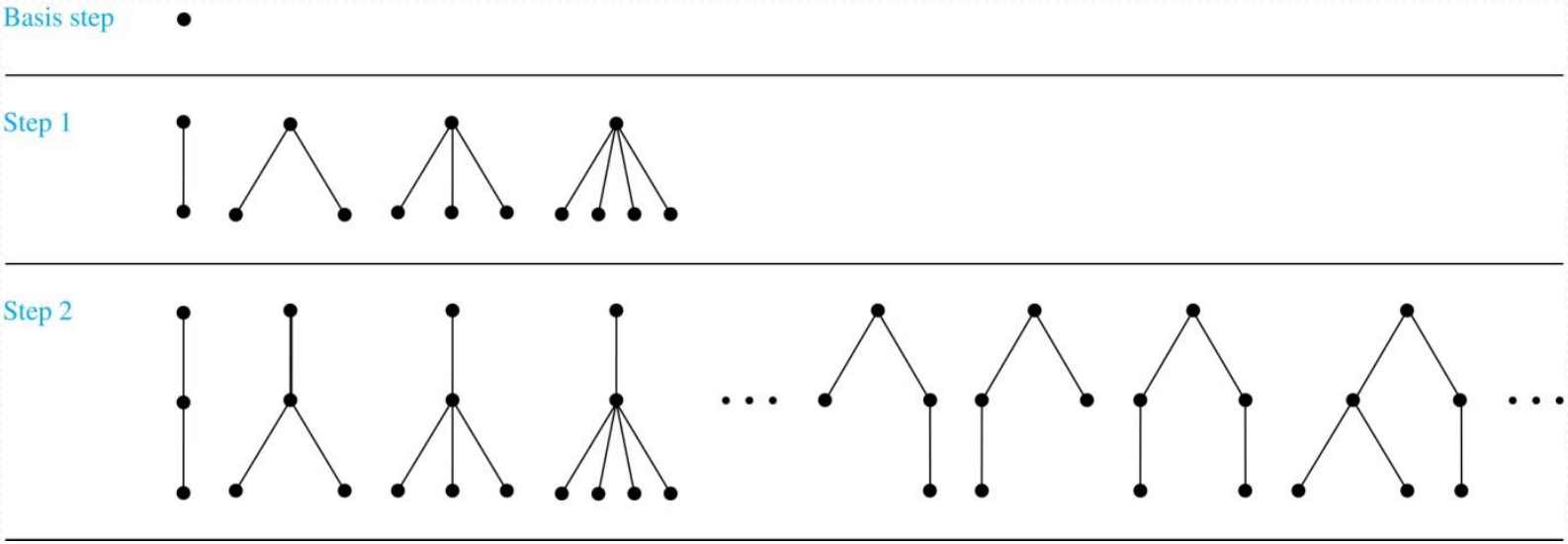
$pq \wedge$ is not a well formed formula.

Rooted Trees

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 $r_1, r_2, \ldots, r_n$, respectively. Then the graph formed by starting with a root r, which is not in any of the rooted trees $T_1, T_2, \ldots, T_n$, and adding an edge from r to each of the vertices $r_1, r_2, \ldots, r_n$, is also a rooted tree.

Building Up Rooted Trees



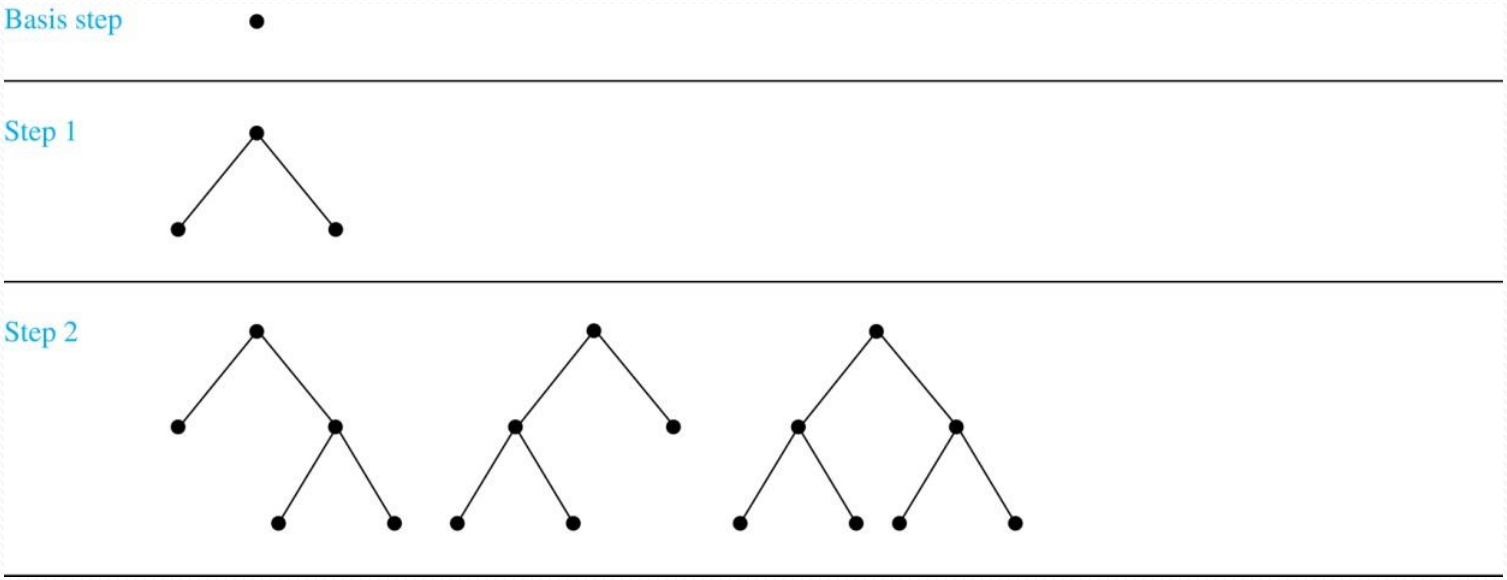
Binary Trees

Definition: The set of *binary trees* can be defined recursively by these steps.

BASIS STEP: There is a binary tree consisting of only a single vertex r.

RECURSIVE STEP: If T_1 and T_2 are disjoint binary trees, there is a 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.

Building Up Binary Trees



Define a property of recursively defined set

To define/compute a property of the elements of a recursively defined set, we do the following:

BASIS STEP: define/compute a property for all elements specified in the basis step of the recursive definition.

RECURSIVE STEP: determine the property for the new element from the properties for the elements used to construct new elements in the recursive step of the definition.

Length of a String

Example: Give a recursive definition of $l(w)$, the length of the string w.

Solution: The length of a string can be recursively defined by:

$l(\lambda) = 0$;

$l(wx) = l(w) + 1$ if $w \in \Sigma^*$ and $x \in \Sigma$.

Height of a Binary Tree

The *height* $h(T)$ of a binary tree T is defined recursively as follows:

- BASIS STEP: The height of a 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 $h(T) = 1 + \max(h(T_1), h(T_2))$.

Number of Vertices of a Binary Tree

- The number of vertices $n(T)$ of a binary tree T satisfies the following recursive formula:
  - **BASIS STEP**: The number of vertices of a binary tree T consisting of only a root r is $n(T) = 1$.
  - **RECURSIVE STEP**: If T_1 and T_2 are binary trees, then the binary tree $T = T_1 \cdot T_2$ has the number of vertices
    $$n(T) = 1 + n(T_1) + n(T_2).$$

Recursive Algorithms

Section 5.4

Section Summary

- Recursive Algorithms
- Proving Recursive Algorithms Correct
- Merge Sort

Recursive Algorithms

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

- For the algorithm to terminate, the instance of the problem must eventually be reduced to some initial case for which the solution is known.

Recursive Factorial Algorithm

Example: Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

- **Solution**: Use the recursive definition of the factorial function.

procedure *factorial*(n: nonnegative integer)
if $n = 0$ **then return** 1
else return $n \cdot factorial\ (n - 1)$
{output is $n!$}

Recursive Exponentiation Algorithm

Example: Give a recursive algorithm for computing a^n, where a is a nonzero real number and n is a nonnegative integer.

Solution: Use the recursive definition of a^n.

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

Recursive GCD Algorithm

Example: Give a recursive algorithm for computing the greatest common divisor of two nonnegative integers a and b with $a < b$.

Solution: Use the reduction

$$\gcd(a,b) = \gcd(b \bmod a, a)$$

and the condition $\gcd(0,b) = b$ when $b > 0$.

```
procedure gcd(a,b: nonnegative integers
                          with a < b)
if a = 0
    return b
else
    return gcd (b mod a, a)
```

Recursive Binary Search Algorithm

Example: Construct a recursive version of a binary search algorithm.

Solution: Assume we have $a_1, a_2, \ldots, a_n$, an increasing sequence of integers. Initially i is 1 and j is n. We are searching for x.

procedure *binary search*(i, j, x : integers, $1 \leq i \leq j \leq n$)
$m := \lfloor (i + j)/2 \rfloor$
if $x = a_m$ **then**
    return m
else if ($x < a_m$ and $i < m$) **then**
    return *binary search*($i, m-1, x$)
else if ($x > a_m$ and $j > m$) **then**
    return *binary search*($m+1$,j,x)
else
    return 0
{output is location of x in $a_1, a_2, \ldots, a_n$ if it appears, otherwise 0}

Proving Recursive Algorithms Correct

Example: Prove that the algorithm for computing the powers of real numbers is correct.

```
procedure power(a: nonzero real number, n: nonnegative integer)
if n = 0 then return 1
else return a · power (a, n − 1)
{output is a^n}
```

Solution: Use mathematical induction on the exponent n.

BASIS STEP: $a^0 = 1$ for every nonzero real number a, and $power(a,0) = 1$.

INDUCTIVE STEP: The inductive hypothesis is that $power(a,k) = a^k$, for all $a \neq 0$. Assuming the inductive hypothesis, the algorithm correctly computes a^{k+1}, since

$$power(a,k+1) = a \cdot power\ (a, k) = a \cdot a^k = a^{k+1}\ .$$

Proving Recursive Algorithms Correct

Example: Prove correctness of the recursive version of a binary search algorithm **by strong induction.**

procedure *binary search*$(i, j, x :$ integers, $1 \le i \le j \le n)$
$m := \lfloor (i + j)/2 \rfloor$
if $x = a_m$ **then**
    return m
else if $(x < a_m$ and $i < m)$ **then**
    return *binary search*$(i, m-1, x)$
else if $(x > a_m$ and $j > m)$ **then**
    return *binary search*$(m+1, j, x)$
else
    return 0
{output is location of x in $a_1, a_2, \ldots, a_n$ if it appears, otherwise 0}

Merge Sort

- *Merge Sort* works by iteratively splitting a list (with an even number of elements) into two sublists of equal length until each sublist has one element.
- Each sublist is represented by a balanced binary tree.
- At each step a pair of sublists is successively merged into a list with the elements in increasing order. The process ends when all the sublists have been merged.
- The succession of merged lists is represented by a binary tree.

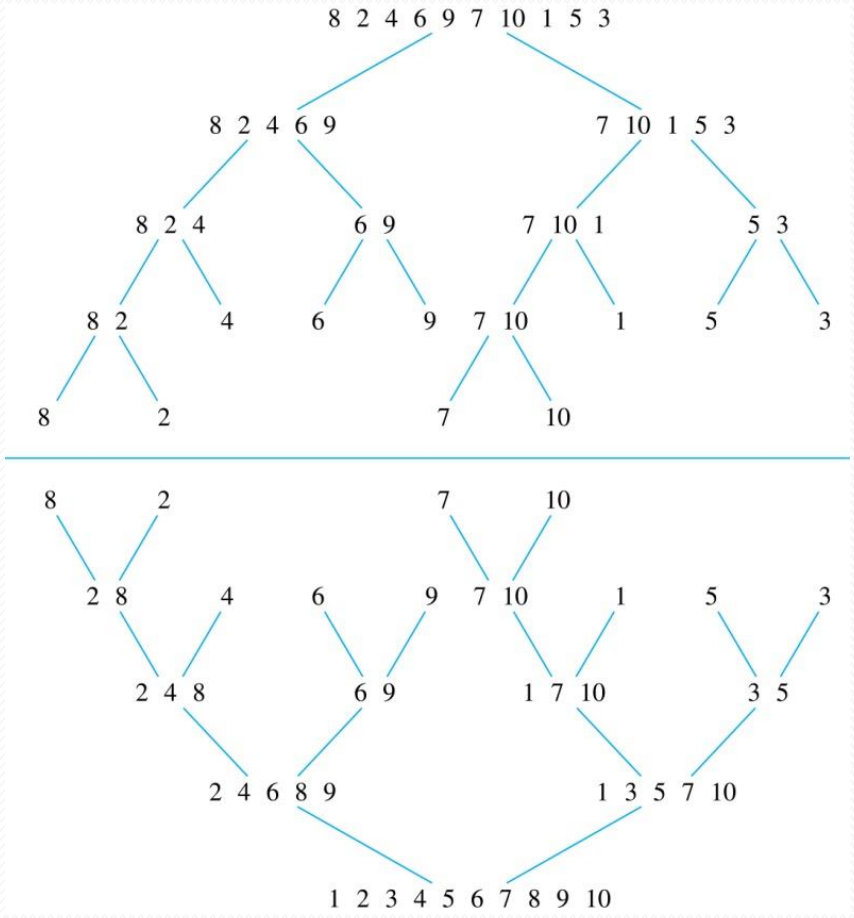
Merge Sort

Example: Use merge sort to put the list

8,2,4,6,9,7,10, 1, 5, 3

into increasing order.

Solution:



Recursive Merge Sort

Example: Construct a recursive merge sort algorithm.

Solution: Begin with the list of n elements L.

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

continued →

Merging Two Lists

Example: Merge the two lists 2,3,5,6 and 1,4.

Solution:

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	

Recursive Merge Sort

- Subroutine *merge*, which merges two sorted lists.

```
procedure merge(L1, L2 :sorted lists)
L := empty list
while L1 and L2 are both nonempty
    remove smaller of first elements of L1 and L2 from its list;
          add it to L
    if this removal makes one list empty
       then remove all elements from the other list and append them to L
return L
{L is the merged list with the elements in increasing order}
```

Correctness of Recursive Merge Sort

Example: Prove correctness of merge sort algorithm (assuming merge procedure correctly merges 2 sorted arrays into 1 sorted array).

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

Use strong induction.

continued →