Induction and recursion

Chapter 5

With Question/Answer Animations

Chapter Summary

- Weak Induction
- Strong Induction
- Recursive Definitions
- Structural Induction
- Recursive Algorithms

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

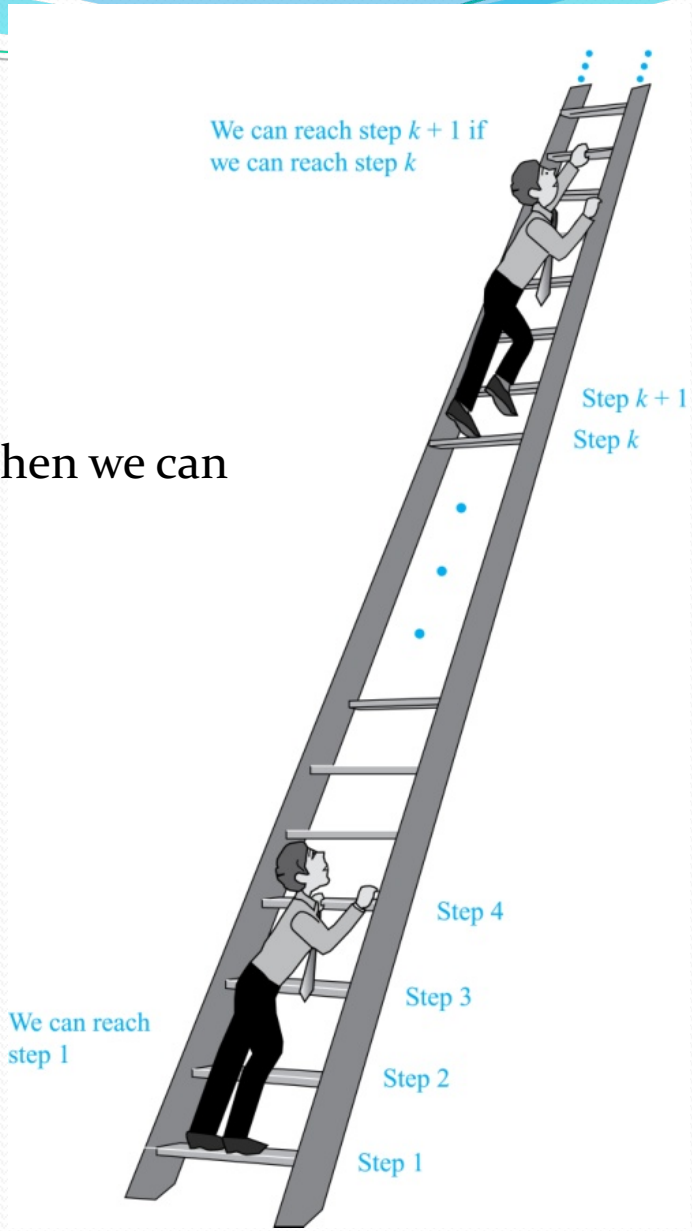
Climbing an Infinite Ladder

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:

- *Base Case*: Show that $P(1)$ is true.
- *INDUCTIVE STEP*: Show that $P(k\text{-}1) \rightarrow P(k)$ is true for all positive integers k.

To complete the INDUCTIVE STEP, assuming the *inductive hypothesis* that $P(k\text{-}1)$ holds for an arbitrary integer k, show that must $P(k)$ be true.

Climbing an Infinite Ladder Example:

- BASE CASE: By (1), we can reach rung 1.
- INDUCTIVE STEP: Assume the inductive hypothesis that we can reach rung $k\text{-}1$. Then by (2), we can reach rung k.

Hence, $P(k\text{-}1) \rightarrow P(k)$ is true for all positive integers k. We can reach every rung on the ladder. ◀

Important Points About Using Mathematical Induction

- Mathematical induction can be expressed as the rule of inference

$$(P(1) \land \forall k\,(P(k\text{-}1) \rightarrow P(k))) \rightarrow \forall n >= 1\, P(n),$$

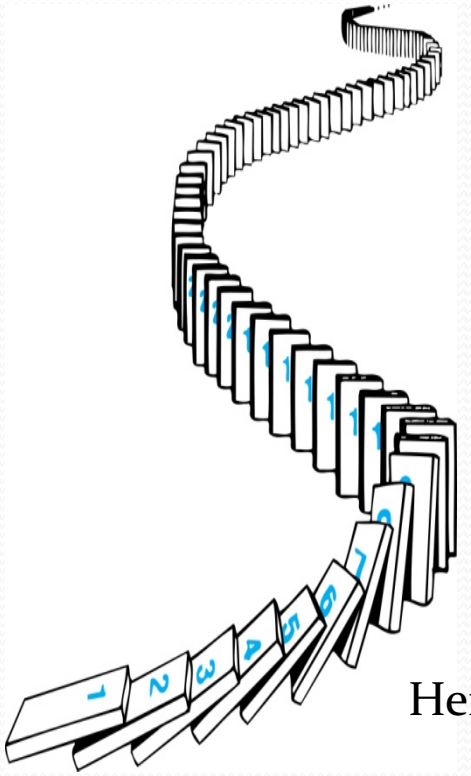
where the domain is the set of positive integers.

- In a proof by mathematical induction, we don't assume that $P(k\text{-}1)$ is true for all positive integers! We show that if we assume that $P(k\text{-}1)$ is true, then $P(k)$ must also be true.
- Proofs by mathematical induction do not always start at the integer 1. In such a case, the base case begins at a starting point b where b is an integer.

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-th domino, i.e, $P(k\text{-}1) \rightarrow P(k)$ 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: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

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

Solution:

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

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

Under this assumption,

$$\sum_{i=1}^{k} i = k + \sum_{i=1}^{k-1} i$$

$$= k + \frac{(k-1)k}{2} \qquad \text{By IH!}$$

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

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 is proved correct with mathematical induction.
- BASE CASE: $P(1)$ is true since $1^2 = 1$.
- INDUCTIVE STEP: $P(k\text{-}1) \rightarrow P(k)$ for every positive integer k.
  Assume the inductive hypothesis holds and then show that $P(k)$ holds has well.

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

- So, assuming $P(k\text{-}1)$, it follows that:

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

- Hence, we have shown that $P(k)$ follows from $P(k - 1)$. 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 $n < 2^n$.

- BASE CASE: $P(1)$ is true since $1 < 2^1 = 2$.
- INDUCTIVE STEP: Assume $P(k - 1)$ holds, i.e., $k - 1 < 2^{k-1}$, for an arbitrary positive integer k.
- Must show that $P(k)$ holds:

$$k = (k - 1) + 1$$
$$< 2^{k-1} + 1 \qquad \text{By IH}$$
$$\leq 2^{k-1} + 2^{k-1} = 2 \cdot 2^{k-1} = 2^k$$

◀

Proving Inequalities

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

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

- BASE CASE: $P(4)$ is true since $2^4 = 16 < 4! = 24$.
- INDUCTIVE STEP: [In Class!]

◀

Note that here the BASE CASE 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.

BC: $3 \mid 1 - 1 = 0$.

IH: $3 \mid (j-1)^3 - (j-1)$

IS: $j^3 - j = (j-1)^3 - (j-1) + 3j^2 - 3j$

$= 3x + 3j^2 - 3j$ BY IH, $(j-1)^3 - (j-1) = 3x$, for some integer x

Since $j^3 - j = 3y$, for some integer y, we know $3 \mid j^3 - j$

Number of Subsets of a Finite Set

Example: Use mathematical induction to show that if S is a finite set with n elements, where n is a nonnegative integer, then S has 2^n subsets.

(*Chapter 6 uses combinatorial methods to prove this result.*)

Solution: Let $P(n)$ be the proposition that a set with n elements has 2^n subsets.

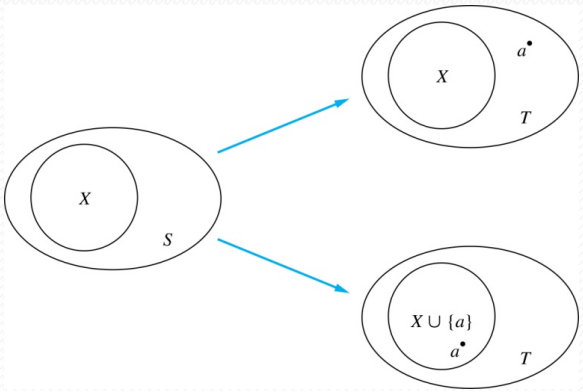
- BASE CASE: $P(0)$ is true, because the empty set has only itself as a subset and $2^0 = 1$.
- INDUCTIVE STEP: Assume $P(k-1)$ is true for an arbitrary nonnegative integer k.

continued →

Number of Subsets of a Finite Set

Inductive Hypothesis: For an arbitrary nonnegative integer k - 1, every set with k - 1 elements has 2^{k-1} subsets.

- Let T be a set with k elements. Then $T = S \cup \{a\}$, where $a \in T$ and $S = T - \{a\}$. Hence $|S| = k - 1$.
- For each subset X of S, there are exactly two subsets of T, i.e., X and $X \cup \{a\}$.



- By the inductive hypothesis S has 2^{k-1} subsets. Since there are two subsets of T for each subset of S, the number of subsets of T is $2 \cdot 2^{k-1} = 2^k$.

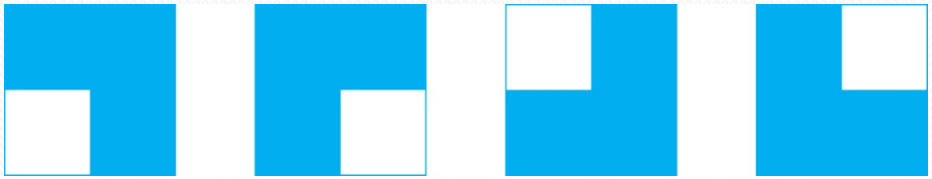
Tiling Checkerboards

Example: Show that every $2^n \times 2^n$ checkerboard with one square removed can be tiled using right triominoes.

A right triomino is an L-shaped tile which covers three squares at a time.

Solution: Let $P(n)$ be the proposition that every $2^n \times 2^n$ checkerboard with one square removed can be tiled using right triominoes. Use mathematical induction to prove that $P(n)$ is true for all positive integers n.

- BASE CASE: P(1) is true, because each of the four 2×2 checkerboards with one square removed can be tiled using one right triomino.



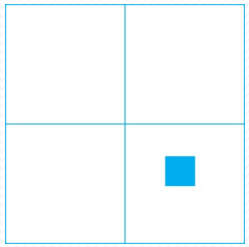
- INDUCTIVE STEP: Assume that $P(k - 1)$ is true for every $2^{k-1} \times 2^{k-1}$ checkerboard, for some positive integer $k - 1$.

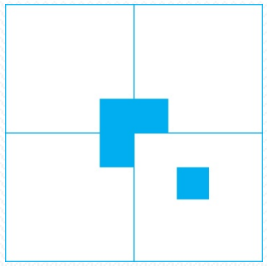
continued →

Tiling Checkerboards

Inductive Hypothesis: Every $2^{k-1} \times 2^{k-1}$ checkerboard, for some positive integer k, with one square removed can be tiled using right triominoes.

- Consider a $2^k \times 2^k$ checkerboard with one square removed. Split this checkerboard into four checkerboards of size $2^{k-1} \times 2^{k-1}$,by dividing it in half in both directions.





- Remove a square from one of the four $2^{k-1} \times 2^{k-1}$ checkerboards. By the inductive hypothesis, this board can be tiled. Also by the inductive hypothesis, the other three boards can be tiled with the square from the corner of the center of the original board removed. We can then cover the three adjacent squares with a triominoe.
- Hence, the entire $2^k \times 2^k$ checkerboard with one square removed can be tiled using right triominoes.

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:
  - *BASE CASE*: Verify that the proposition $P(1)$ is true.
  - *INDUCTIVE STEP*: Show the conditional statement $[P(1) \wedge P(2) \wedge \cdots \wedge P(k - 1)] \rightarrow P(k)$ 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

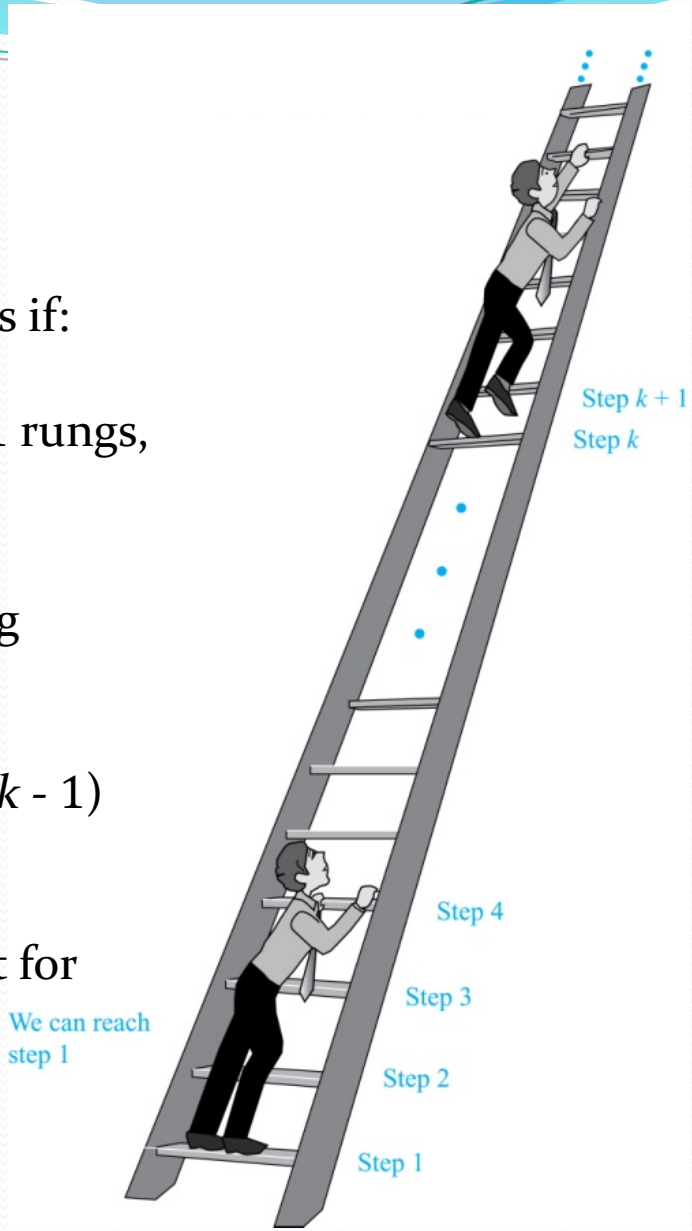
Strong induction tells us that we can reach all rungs if:

1. We can reach the first rung of the ladder.
2. For every integer k, if we can reach the first k - 1 rungs, then we can reach the k-th rung.

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

- BASE CASE: $P(1)$ holds
- INDUCTIVE STEP: Assume $P(1) \wedge P(2) \wedge \cdots \wedge P(k-1)$ holds for an arbitrary integer k, and show that $P(k)$ 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.
- I usually approach a problem with the idea of using strong induction, since it is the more powerful tool.
- However, for simple problems, weak induction is enough.

Completion of the proof of the Fundamental Theorem of Arithmetic

Example: Show that if n is an integer greater than 1, then n can be written as the product of primes.

Solution: Let $P(n)$ be the proposition that n can be written as a product of primes.

- BASE CASE: $P(2)$ is true since 2 itself is prime.
- INDUCTIVE STEP: The inductive hypothesis is $P(j)$ is true for all integers j with $2 \leq j < k$. To show that $P(k)$ must be true under this assumption, two cases need to be considered:
  - If k is prime, then $P(k)$ is true.
  - Otherwise, k is composite and can be written as the product of two positive integers a and b with $2 \leq a \leq b < k$. By the inductive hypothesis a and b can be written as the product of primes and therefore k can also be written as the product of those primes.

Hence, it has been shown that every integer greater than 1 can be written as the product of primes.

(*uniqueness proved in Section* 4.3)

◀

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.

- BASE CASE: $P(12)$, $P(13)$, $P(14)$, 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: The inductive hypothesis states that $P(j)$ holds for $12 \leq j < k$, where $k \geq 16$.
- Using the inductive hypothesis, $P(k-4)$ holds since $k - 4 \geq 12$. To form postage of k cents, add a 4-cent stamp to the postage for $k - 4$ cents.

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

Recursively Defined Functions

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

- BASE CASE: Specify the value of the function at zero.
- RECURSIVE STEP: Give a rule for finding its value at an integer from its values at smaller integers.

- A function $f(n)$ is the same as a sequence $a_0, a_1, \ldots,$ where a_i, where $f(i) = a_i$.

Recursively Defined Functions

Example: Suppose f is defined by:

$f(0) = 3,$

$f(n) = 2f(n - 1) + 3$

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:

$f(0) = 1$

$f(n) = n \cdot f(n)$

Solving Recurrences by Induction

Recurrence: $f(1) = 1$; $f(n) = 2f(n - 1) + 1$.

Fact: $f(n) = 2^n - 1$

Proof:

BC: $f(1) = 1 = 2^1 - 1$

IH: For all $0 < j < n$, $f(j) = 2^j - 1$

IS: $f(n) = 2f(n - 1) + 1$

$= 2\,(2^{n-1} - 1) + 1$ (by IH)

$= 2^n - 1$

Fibonacci
(1170- 1250)

Fibonacci Numbers

Example : The Fibonacci numbers are defined as follows:

$f_0 = 0$

$f_1 = 1$

$f_n = f_{n-1} + f_{n-2}$

Find f_2, f_3, f_4, f_5.

- $f_2 = f_1 + f_0 = 1 + 0 = 1$
- $f_3 = f_2 + f_1 = 1 + 1 = 2$
- $f_4 = f_3 + f_2 = 2 + 1 = 3$
- $f_5 = f_4 + f_3 = 3 + 2 = 5$

Fibonacci numbers were used to model population growth of rabbits.

Next, we use strong induction to prove a result about the Fibonacci numbers.

Solving Recurrences by Induction

Recurrence: $f(0) = 0; f(1) = 1; f(n) = f(n - 1) + f(n-2)$

Fact: $f(n) > \alpha^{n-2}$, for $n \geq 3$, where $\alpha = (1 + \sqrt{5})/2$

Proof:

BC: $f(3) = 2 > \alpha; f(4) = 3 > \alpha^2$

IH: For all $2 < j < n$, $f(j) > \alpha^{j-2}$

IS: $f(n) = f(n-1) + f(n-2)$

$> \alpha^{n-1} + \alpha^{n-2}$ (By IH)

$= \alpha^n$

Last step holds since $\alpha^2 = \alpha + 1$ (because α is a solution of $x^2 - x - 1 = 0$), implies that $\alpha^n = \alpha^{n-1} + \alpha^{n-2}$

Recursively Defined Sets and Structures

Recursive definitions of sets have two parts:

- The *BASE CASE* 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.

- Sometimes the recursive definition has an *exclusion rule*, which specifies that the set contains nothing other than those elements specified in the BASE CASE and generated by applications of the rules in the recursive step.
- We will always assume that the exclusion rule holds, even if it is not explicitly mentioned.
- We will later develop a form of induction, called *structural induction*, to prove results about recursively defined sets.

Recursively Defined Sets and Structures

Example : Subset of Integers S:

BASE CASE: $3 \in S$.

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.

Example: The natural numbers **N**.

BASE CASE: $0 \in \mathbf{N}$.

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.

Strings

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

BASE CASE: $\lambda \in \Sigma^*$ (λ is the empty string)

RECURSIVE STEP: If w is in Σ^* and x is in Σ, then $wx \in \Sigma^*$.

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 $b \in \Sigma$, $aab \in \Sigma^*$.

String Concatenation

Definition: Two strings can be combined via the operation of *concatenation*. Let Σ be a set of symbols and Σ^* be the set of strings formed from the symbols in Σ. We can define the concatenation of two strings, denoted by $\cdot$, recursively as follows.

BASE CASE: If $w \in \Sigma^*$, then $w \cdot \lambda = w$.

RECURSIVE STEP: If $w_1 \in \Sigma^*$ and $w_2 \in \Sigma^*$ and $x \in \Sigma$, then $w_1 \cdot (w_2 x) = (w_1 \cdot w_2)x$.

- Often $w_1 \cdot w_2$ is written as $w_1 w_2$.
- If $w_1 = abra$ and $w_2 = cadabra$, the concatenation $w_1 w_2 = abracadabra$.

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 $x \in \Sigma$.

Balanced Parentheses

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

Solution:

BASE CASE: $() \in P$

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 $\{\neg,\wedge,\vee,\rightarrow,\leftrightarrow\}$.

BASE CASE: **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.

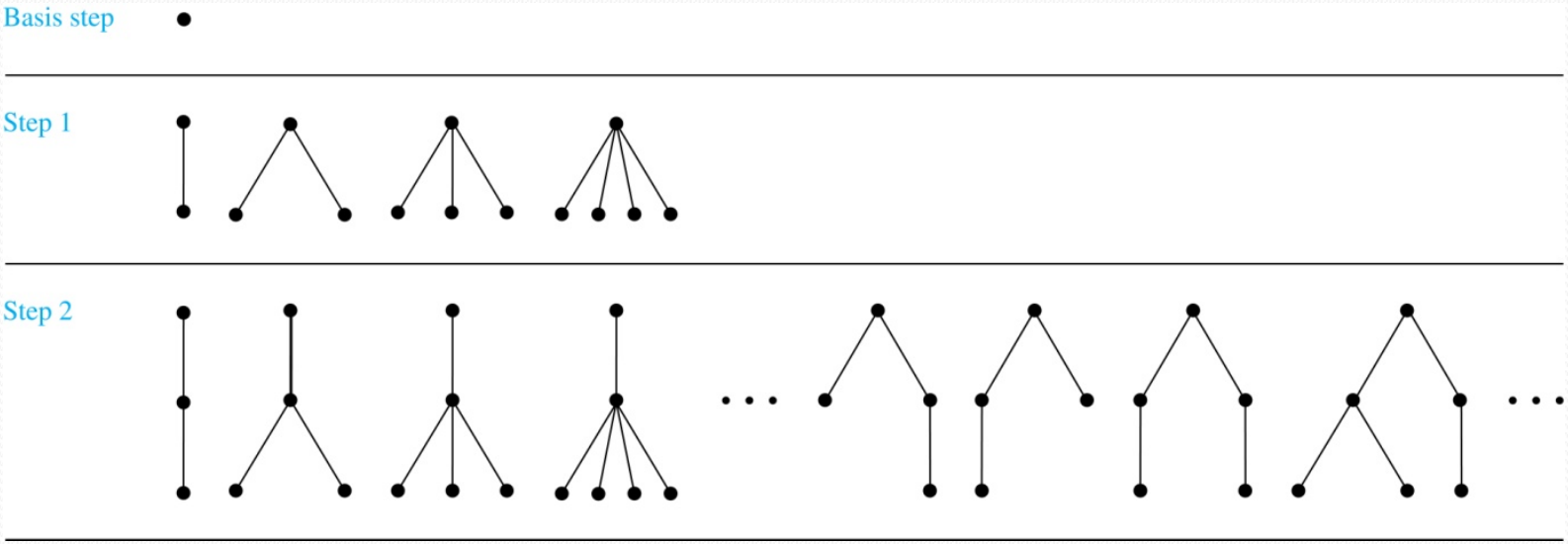
Rooted Trees

Definition: The set of *rooted trees*, where a rooted tree consists of a set of vertices containing a distinguished vertex called the *root*, and edges connecting these vertices, can be defined recursively by these steps:

BASE CASE: 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



- Next we look at a special type of tree, the full binary tree.

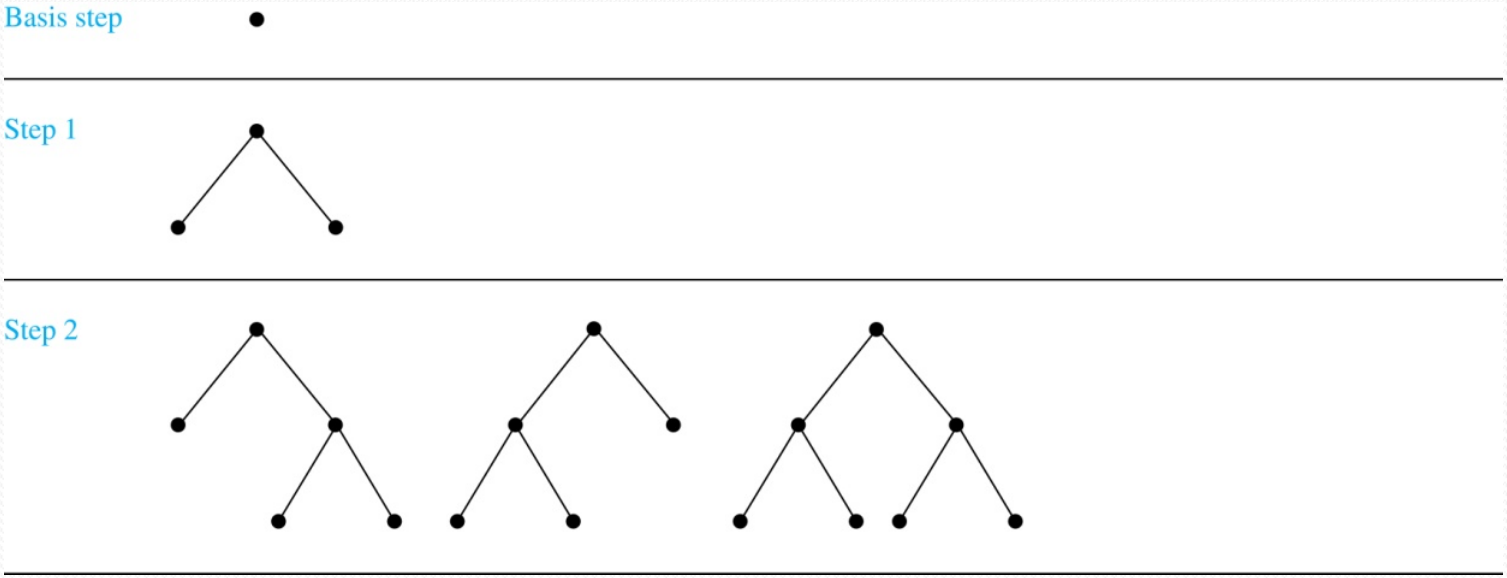
Full Binary Trees

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

BASE CASE: There is a full binary tree consisting of only a single vertex r.

RECURSIVE STEP: If T_1 and T_2 are disjoint full binary trees, there is a full 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 Full Binary Trees



Induction and Recursively Defined Sets

Example: Show that the set S defined by specifying that $3 \in S$ and that if $x \in S$ and $y \in S$, then $x + y$ is in S, is the set of all positive integers that are multiples of 3.

Solution: Let A be the set of all positive integers divisible by 3. To prove that $A = S$, show that A is a subset of S and S is a subset of A.

- $A \subset S$: Let $P(n)$ be the statement that $3n$ belongs to S.

  BASE CASE: $3 \cdot 1 = 3 \in S$, by the first part of recursive definition.

  INDUCTIVE STEP: Assume $P(k\text{-}1)$ is true. By the second part of the recursive definition, if $3k \in S$, then since $3 \in S$, $3(k\text{-}1) + 3 = 3k \in S$. Hence, $P(k)$ is true.

- $S \subset A$:

  BASE CASE: $3 \in S$ by the first part of recursive definition, and $3 = 3 \cdot 1$.

  INDUCTIVE STEP: The second part of the recursive definition adds $x + y$ to S, if both x and y are in S. If x and y are both in A, then both x and y are divisible by 3. By part (i) of Theorem 1 of Section 4.1, it follows that $x + y$ is divisible by 3.

Structural Induction

Definition: To prove a property of the elements of a recursively defined set, we use *structural induction*.

BASE CASE: Show that the result holds for all elements specified in the BASE CASE of the recursive definition.

RECURSIVE STEP: Show that if the statement is true for each of the elements used to construct new elements in the recursive step of the definition, the result holds for these new elements.

- The validity of structural induction can be shown to follow from the principle of mathematical induction.

Full Binary Trees

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

- BASE CASE: The height of a full binary tree T consisting of only a root r is $h(T) = 0$.
- RECURSIVE STEP: If T_1 and T_2 are full binary trees, then the full binary tree $T = T_1 \cdot T_2$ has height $h(T) = 1 + \max(h(T_1), h(T_2))$.

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

Structural Induction and Binary Trees

Theorem: If T is a full binary tree, with $h(T) \geq 0$, then $n(T) \leq 2^{h(T)+1} - 1$.

Proof: (by induction on h(T))

BC: For $h(T) = 0$, $n(T) = 1$ and $0 \leq 2^1 - 1$

IH: For all $1 \leq j < h$, if $h(T) = h$ then $n(T) \leq 2^{h(T)+1} - 1$

IS: Consider a tree T with $h(T) = h$. Let T_1 be the left subtree of the root and T_2 be the right subtree of the root. Then we know:

$$n(T) = n(T_1) + n(T_2) + 1$$
$$\leq (2^{h(T_1)+1} - 1) + (2^{h(T_1)+1} - 1) + 1 \quad \text{(by IH)}$$
$$\leq (2^{h(T)} - 1) + (2^{h(T)} - 1) + 1 \quad \text{(since } h(T_1), h(T_2) < h(T)\text{)}$$
$$= 2^{h(T)+1} - 1 \quad \text{(by algebra)}$$

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 then return b
else return gcd (b mod a, a)
{output is gcd(a, b)}
```

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

- Both mathematical and strong induction are useful techniques to show that recursive algorithms always produce the correct output.

Exercise: 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 $a \cdot power\ (a, n - 1)$
{output is a^n}

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.

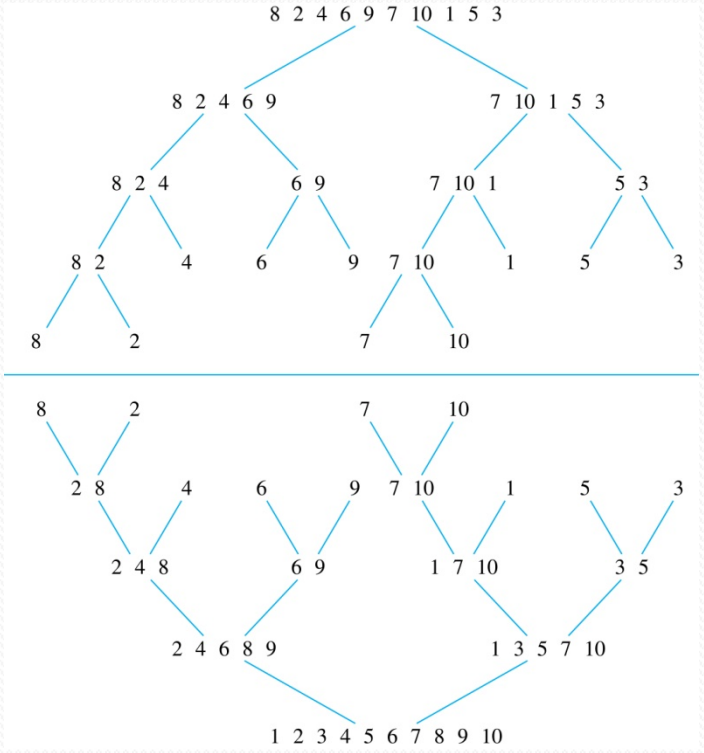
Merge Sort

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

continued →

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;
          put at the right end of 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}
```

Complexity of Merge: Two sorted lists with m elements and n elements can be merged into a sorted list using no more than $m + n - 1$ comparisons.

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	

Complexity of Merge Sort

Complexity of Merge Sort: The number of comparisons needed to merge a list with n elements is $O(n \log n)$.

- Then runtime is a recurrence relation given by $f(2) = 2$; $f(n) = 2\, f(n/2) + n$; n power of 2.
- In next slide, we show that the solution to this recurrence is no more than n log n.

continued →

MergeSort Runtime

Recurrence: $f(2) = 2$; $f(n) = 2\, f(n/2) + n$; n power of 2.

Fact: $f(n) \leq n \log n$, for all $n > 1$.

Proof (by Strong Induction):

BC: $f(2) = 2 \leq 2 \log 2$

IH: For all $2 \leq j < n$, $f(j) \leq j \log j$

IS: $f(n) = 2\, f(n/2) + n$

$\leq 2\,(n/2\,(\log n/2)) + n$ (by IH)

$= n \log n$ (since $\log x/y = \log x - \log y$)

MergeSort Runtime

We've shown that MergeSort is a O(n log n) runtime sorting algorithm for lists of size n.

Much better than BubbleSort and InsertionSort which have $O(n^2)$ runtime

It turns out that O(n log n) is optimal sorting time for a large class of algorithms.