Chapter 4:
Induction and Recursion
§4.1: Mathematical Induction

- A powerful, rigorous technique for proving that a predicate $P(n)$ is true for every positive integer $n$, no matter how large.
- Essentially a “domino effect” principle.
- Based on a predicate-logic inference rule:

\[
P(1) \\
\forall k \geq 1 \ (P(k) \rightarrow P(k+1)) \\
\therefore \forall n \geq 1 \ P(n)
\]

“The First Principle of Mathematical Induction”
Outline of an Inductive Proof

- **Want to prove** $\forall n P(n)\ldots$
- **Base case** (or basis step): Prove $P(1)$.
- **Inductive step**: Prove $\forall k P(k) \rightarrow P(k+1)$.
  - *E.g.* use a direct proof:
  - Let $k \in \mathbb{N}$, assume $P(k)$. (inductive hypothesis)
  - Under this assumption, prove $P(k+1)$.
- **Inductive inference rule then gives** $\forall n P(n)$.
Induction Example (1st princ.)

- **Prove that the sum of the first* $n$ odd positive integers is* $n^2$.* That is, prove:
  \[ \forall n \geq 1: \sum_{i=1}^{n} (2i - 1) = n^2 \]

- **Proof by induction.** $P(n)$
  - Base case: Let $n=1$. The sum of the first 1 odd positive integer is 1 which equals $1^2$.
  (Cont…)
Example cont.

- **Inductive step:** Prove $\forall k \geq 1: P(k) \rightarrow P(k+1)$.
  - Let $k \geq 1$, assume $P(k)$, and prove $P(k+1)$.

  $$\sum_{i=1}^{k+1} (2i - 1) = \left( \sum_{i=1}^{k} (2i - 1) \right) + (2(k + 1) - 1)$$

  $$= k^2 + 2k + 1$$

  By inductive hypothesis $P(k)$

  $$= (k + 1)^2$$
Another Induction Example

- Prove that \( \forall n > 0, n < 2^n \). Let \( P(n) = (n < 2^n) \)
  - Base case: \( P(1) = (1 < 2^1) = (1 < 2) = T \).
  - Inductive step: For \( k > 0 \), prove \( P(k) \rightarrow P(k+1) \).
    - Assuming \( k < 2^k \), prove \( k+1 < 2^{k+1} \).
    - Note \( k + 1 < 2^k + 1 \) (by inductive hypothesis)
      \( < 2^k + 2^k \) (because \( 1 < 2 = 2^0 \leq 2 \cdot 2^{k-1} = 2^k \))
      \( = 2^{k+1} \)
    - So \( k + 1 < 2^{k+1} \), and we’re done.
Validiy of Induction

Proof that $\forall k \geq 1 \ P(k)$ is a valid consequent:

Given any $k \geq 1$, $\forall n \geq 1 \ (P(n) \rightarrow P(n+1))$ (antecedent 2) trivially implies $\forall n \geq 1 \ (n < k) \rightarrow (P(n) \rightarrow P(n+1))$, or $(P(1) \rightarrow P(2)) \land (P(2) \rightarrow P(3)) \land \ldots \land (P(k-1) \rightarrow P(k))$. Repeatedly applying the hypothetical syllogism rule to adjacent implications $k-1$ times then gives $P(1) \rightarrow P(k)$; which with $P(1)$ (antecedent #1) and modus ponens gives $P(k)$. Thus $\forall k \geq 1 \ P(k)$. 

§ 4.1 – Mathematical Induction
The Well-Ordering Property

- The validity of the inductive inference rule can also be proved using the *well-ordering property*, which says:
  - Every non-empty set of non-negative integers has a minimum (smallest) element.
  - \( \forall \emptyset \subseteq S \subseteq \mathbb{N} : \exists m \in S : \forall n \in S : m \leq n \)

- Implies \( \{ n \mid \neg P(n) \} \) has a min. element \( m \), but then \( P(m-1) \rightarrow P((m-1)+1) \) contradicted.
Generalizing Induction

- Can also be used to prove $\forall n \geq c \ P(n)$ for a given constant $c \in \mathbb{Z}$, where maybe $c \neq 1$.
  - In this circumstance, the base case is to prove $P(c)$ rather than $P(1)$, and the inductive step is to prove $\forall k \geq c \ (P(k) \rightarrow P(k+1))$.
- Induction can also be used to prove $\forall n \geq c \ P(a_n)$ for an arbitrary series $\{a_n\}$.
- Can reduce these to the form already shown.
§4.2 : Strong Induction

- Characterized by another inference rule:
  $P(1)$ \quad $P$ is true in all previous cases
  $\forall k \geq 1: (\forall 1 \leq i \leq k \ P(i)) \rightarrow P(k+1)$
  $\therefore \forall n \geq 1: P(n)$

- Difference with 1st principle is that the inductive step uses the fact that $P(i)$ is true for all smaller $i<k+1$, not just for $i=k$. 
Example of Second Principle

• Show that every $n > 1$ can be written as a product $p_1 p_2 \ldots p_s$ of some series of $s$ prime numbers. Let $P(n) = "n has that property"

• Base case:

• Inductive step: Let $k \geq 2$. Assume $\forall 2 \leq i \leq k: P(i)$. Consider $k+1$. If prime, Else $k+1 = ab$, where $1 < a \leq k$ and $1 < b \leq k$. 

§ 4.2 – Strong Induction
Another 2nd Principle Example

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

• Base case:

• Inductive step: Let $k \geq 15$, assume $\forall 12 \leq i \leq k P(i)$. 

§ 4.2 – Strong Induction
Proofs By Well-Ordering Property

• Use the well-ordering property to prove the division algorithm: \( a = dq + r, \ 0 \leq r < |d|, \)
where \( q \) and \( r \) are unique.

- \( S = \{ n \mid n = a - dq \} \) is nonempty, so \( S \) has a least element \( r = a - dq_0 \). If \( r \geq 0 \), it is also the case that \( r < d \). If it were not,

- If \( a = dq_1 + r_1 = dq_2 + r_2, \ 0 \leq r_1, r_2 < |d|, \) then
§ 4.3 : Recursive Definitions

- In induction, we prove all members of an infinite set have some property \( P \) by proving the truth for larger members in terms of that of smaller members.

- In recursive definitions, we similarly define a function, a predicate or a set over an infinite number of elements by defining the function or predicate value or set-membership of larger elements in terms of that of smaller ones.
Recursion

• *Recursion* is a general term for the practice of defining an object in terms of *itself* (or of part of itself).

• An inductive proof establishes the truth of \( P(n+1) \) recursively in terms of \( P(n) \).

• There are also recursive algorithms, definitions, functions, sequences, and sets.
Recursively Defined Functions

• **Simplest case:** One way to define a function $f: \mathbb{N} \rightarrow S$ (for any set $S$) or series $a_n = f(n)$ is to:
  – Define $f(0)$.
  – For $n > 0$, define $f(n)$ in terms of $f(0), \ldots, f(n-1)$.

• **E.g.:** Define the series $a_n : \equiv 2^n$ recursively:
  – Let $a_0 : \equiv 1$.
  – For $n > 0$, let $a_n : \equiv 2a_{n-1}$.

§ 4.3 – Recursive Definitions
Another Example

• Suppose we define \( f(n) \) for all \( n \in \mathbb{N} \) recursively by:
  – Let \( f(0) = 3 \)
  – For all \( n \in \mathbb{N} \), let \( f(n+1) = 2f(n) + 3 \)

• What are the values of the following?
  – \( f(1) = \_\_ \), \( f(2) = \_\_ \), \( f(3) = \_\_ \), \( f(4) = \_\_ \)
Recursive definition of Factorial

- Give an inductive definition of the factorial function $F(n) :\equiv n! :\equiv 2 \cdot 3 \cdot \ldots \cdot n$.
  - Base case: $F(0) :\equiv 1$
  - Recursive part: $F(n) :\equiv n \cdot F(n-1)$.
    - $F(1)=1$
    - $F(2)=2$
    - $F(3)=6$
The Fibonacci Series

- The *Fibonacci series* $f_{n \geq 0}$ is a famous series defined by:

\[
\begin{align*}
    f_0 & : = 0, \\
    f_1 & : = 1, \\
    f_{n \geq 2} & : = f_{n-1} + f_{n-2}
\end{align*}
\]

Leonardo Fibonacci
1170-1250

§ 4.3 – Recursive Definitions
Inductive Proof about Fib. series

- **Theorem:** \( f_n < 2^n \). Implicitly for all \( n \in \mathbb{N} \)
- **Proof:** By induction.

**Base cases:**

**Inductive step:** Use 2\(^{nd}\) principle of induction (strong induction). Assume \( \forall i < k, f_i < 2^i \). Then

Section 4.3 – Recursive Definitions
Recursively Defined Sets

• An infinite set $S$ may be defined recursively, by giving:
  – A small finite set of *base* elements of $S$.
  – A rule for constructing new elements of $S$ from previously-established elements.
  – Implicitly, $S$ has no other elements but these.

• **Example:** Let $3 \in S$, and let $x+y \in S$ if $x,y \in S$. What is $S$?
The Set of All Strings

• Given an alphabet \( \Sigma \), the set \( \Sigma^* \) of all strings over \( \Sigma \) can be recursively defined as:
  
  \[ \varepsilon \in \Sigma^* \quad (\varepsilon \equiv "", \text{the empty string}) \]
  
  \[ w \in \Sigma^* \land x \in \Sigma \rightarrow wx \in \Sigma^* \]

• Exercise: Prove that this definition is equivalent to our old one:
  
  \[ \Sigma^* \equiv \bigcup_{n \in \mathbb{N}} \Sigma^n \]

Book uses \( \lambda \)
§4.4 : Recursive Algorithms

- Recursive definitions can be used to describe algorithms as well as functions and sets.
- Example: A procedure to compute $a^n$.

```plaintext
procedure power(a ≠ 0: real, n ∈ N)
    if n = 0 then return 1
    else return $a \cdot power(a, n-1)$
```

§ 4.4 – Recursive Algorithms
Efficiency of Recursive Algorithms

• The time complexity of a recursive algorithm may depend critically on the number of recursive calls it makes.

• Example: Modular exponentiation to a power $n$ can take $\log(n)$ time if done right, but linear time if done slightly differently.
  
  – Task: Compute $b^n \mod m$, where $m \geq 2$, $n \geq 0$, and $1 \leq b < m$. 
Chap. 4

Modular Exponentiation Alg. #1

Uses the fact that \( b^n = b \cdot b^{n-1} \) and that
\[ x \cdot y \mod m = x \cdot (y \mod m) \mod m. \]
(Prove the latter theorem at home.)

procedure \texttt{mpower}(b \geq 1, n \geq 0, m > b \in \mathbb{N})

\{Returns \( b^n \mod m \).\}
if \( n=0 \) then return 1 else
return \( (b \cdot \texttt{mpower}(b,n-1,m)) \mod m \)

Note this algorithm takes \( \Theta(n) \) steps!
Modular Exponentiation Alg. #2

- Uses the fact that \( b^{2k} = b^{k \cdot 2} = (b^k)^2 \).

\[
\text{procedure } mpower(b, n, m) \text{ \{same signature\}} \\
\text{if } n = 0 \text{ then return } 1 \\
\text{else if } 2 | n \text{ then} \\
\quad \text{return } mpower(b, n/2, m)^2 \mod m \\
\text{else return } (mpower(b, n-1, m) \cdot b) \mod m
\]

What is its time complexity? \( \Theta(\log n) \) steps
A Slight Variation

Nearly identical but takes $\Theta(n)$ time instead!

```plaintext
procedure mpower(b,n,m) {same signature}
    if \( n=0 \) then return \( 1 \)
    else if \( 2|n \) then
        return \( (mpower(b,n/2,m) \cdot mpower(b,n/2,m)) \mod m \)
    else return \( (mpower(b,n−1,m)\cdot b) \mod m \)
```

The number of recursive calls made is critical.
Recursive Euclid’s Algorithm

procedure $\text{gcd}(a, b \in \mathbb{N})$

if $a = 0$ then return $b$

else return $\text{gcd}(b \mod a, a)$

• Note recursive algorithms are often simpler to code than iterative ones…

• However, they can consume more stack space, if your compiler is not smart enough.
procedure \textit{sort}(L = l_1, \ldots, l_n) \\
\textbf{if} n > 1 \textbf{ then} \\
\quad m := \left\lfloor n/2 \right\rfloor \quad \text{\{this is rough 1/2-way point\}} \\
\quad L := \text{merge} (\text{sort}(l_1, \ldots, l_m), \\
\quad \quad \text{sort}(l_{m+1}, \ldots, l_n)) \\
\textbf{return} L \\
\textbullet \quad \text{The merge takes } \Theta(n) \text{ steps, and merge-sort takes } \Theta(n \log n).
Example: Sort the list 27, 10, 12, 20, 25, 13, 15, 22.
Merge Routine

procedure merge(A, B: sorted lists)
  \[ L = \text{empty list} \]
  \[ i := 0, j := 0, k := 0 \]
  while \( i < |A| \land j < |B| \) \{\(|A|\) is length of \(A\)}
    \[
    \begin{align*}
      &\text{if } i = |A| \text{ then } L_k := B_j; \ j := j + 1 \\
      &\text{else if } j = |B| \text{ then } L_k := A_i; \ i := i + 1 \\
      &\text{else if } A_i < B_j \text{ then } L_k := A_i; \ i := i + 1 \\
      &\text{else } L_k := B_j; \ j := j + 1 \\
    \end{align*}
    \]
  \[ k := k + 1 \]
return \(L\)

Takes \(\Theta(|A|+|B|)\) time