Chapter 1

Preliminaries

1.1 Induction

Let $\mathbb{N}$ be the set of natural numbers, including 0. Suppose that S is a subset of $\mathbb{N}$ with the following two properties: first $0 \in S$, and second, whenever $n \in S$, then $n+1 \in S$ as well. Then, invoking the *Induction Principle* (IP) we can conclude that $S = \mathbb{N}$. (See footnote[1].)

We shall use the IP with a more convenient notation; let P be a property of natural numbers, in other words, P is a unary relation such that $P(i)$ is either true or false. The relation P may be identified with a set S_P in the obvious way, i.e., $i \in S_P$ iff $P(i)$ is true. For example, if P is the property of being prime, then $P(2)$ and $P(3)$ are true, but $P(6)$ is false, and $S_P = \{2, 3, 5, 7, 11, \ldots\}$. Using this notation the IP may be stated as:

$$[P(0) \ \& \ \forall n(P(n) \rightarrow P(n+1))] \rightarrow \forall m P(m), \tag{1.1}$$

for any (unary) relation P over $\mathbb{N}$. In practice, we use (1.1) as follows: first we prove that $P(0)$ holds (this is the *basis case*). Then we show that $\forall n(P(n) \rightarrow P(n+1))$ (this is the *induction step*). Finally, using (1.1) and *modus ponens*, we conclude that $\forall m P(m)$.

As an example, let P be the assertion "the sum of the first i odd numbers equals i^2." We follow the convention that the sum of an empty set of numbers is zero; thus $P(0)$ holds as the set of the first zero odd numbers is an empty

[1] In fact, $\mathbb{N}$ and IP are very tightly related, for the rigorous definition of $\mathbb{N}$ is as the *unique* set satisfying the following three properties: (i) it contains 0, (ii) if n is in it, then so is $n+1$, and (iii) it has the IP.

set. $P(3)$ is also true as $1+3+5=9=3^2$. We want to show that in fact $\forall m P(m)$ (i.e., P is always true, and so $S_P = \mathbb{N}$).

We use induction. The basis case is $P(0)$ and we already showed that it holds. Suppose now that the assertion holds for n, i.e., the sum of the first n odd numbers is n^2, i.e., $1+3+5+\cdots+(2n-1)=n^2$ (this is our *inductive hypothesis* or *inductive assumption*). Consider the sum of the first $(n+1)$ odd numbers,

$$\boxed{1+3+5+\cdots+(2n-1)}+(2n+1)=\boxed{n^2}+(2n+1)=(n+1)^2,$$

and so we just proved the induction step, and by IP we have $\forall m P(m)$.

Exercise 1.1.1 Prove that $1+\sum_{j=0}^{i} 2^j = 2^{i+1}$.

Sometimes it is convenient to start our induction higher than at 0. We have the following generalized induction principle:

$$[P(k) \;\&\; \forall n(P(n) \to P(n+1))] \to (\forall m \geq k) P(m), \tag{1.2}$$

for any predicate P and any number k. Note that (1.2) follows easily from (1.1) if we simply let $P'(i)$ be $P(i+k)$, and do the usual induction on $P'(i)$.

Exercise 1.1.2 Use induction to prove that for $n \geq 1$,

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

Exercise 1.1.3 For every $n \geq 1$, consider a square of size $2^n \times 2^n$ where one square is missing. Show that the resulting square can be filled with "L" shapes.

Exercise 1.1.4 In the generalized IP (1.2) we can replace the induction step $\forall n(P(n) \to P(n+1))$ with $(\forall n \geq k)(P(n) \to P(n+1))$. Explain why both versions effectively yield the same principle.

Exercise 1.1.5 The *Fibonacci* sequence is defined as follows: $f_0 = 0$ and $f_1 = 1$ and $f_{i+2} = f_{i+1} + f_i$, $i \geq 0$. Prove that for all $n \geq 1$ we have:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} f_{n+1} & f_n \\ f_n & f_{n-1} \end{pmatrix}.$$

Exercise∗ 1.1.6 Prove the following: if m divides n, then f_m divides f_n, i.e., $m|n$ implies $f_m|f_n$.

The *Complete Induction Principle* (CIP) is just like IP except that in the induction step we show that if $P(i)$ holds for all $i \leq n$, then $P(n+1)$ also holds, i.e., the induction step is now $\forall n((\forall i \leq nP(k)) \rightarrow P(n+1))$.

Exercise 1.1.7 Use the CIP to prove that every number (in $\mathbb{N}$) greater than 1 may be written as a product of one or more prime numbers[2].

Exercise 1.1.8 Suppose that we have a (Swiss) chocolate bar consisting of a number of squares arranged in a rectangular pattern. Our task is to split the bar into small squares (always breaking along the lines between the squares) with a minimum number of breaks. How many breaks will it take? Make an educated guess, and prove it by induction.

The *Least Number Principle* (LNP) says that every non-empty subset of the natural numbers must have a least element. A direct consequence of the LNP is that every decreasing non-negative sequence of integers must terminate; that is, if $R = \{r_1, r_2, r_3, \ldots\} \subseteq \mathbb{N}$ where $r_i > r_{i+1}$ for all i, then R is a *finite* subset of $\mathbb{N}$. We are going to be using the LNP to show termination of algorithms.

Exercise 1.1.9 Show that IP, CIP, and LNP are equivalent principles.

There are three standard ways to list the nodes of a binary tree. We present them below, together with a recursive procedure that lists the nodes according to each scheme.

Infix: left sub-tree, root, right sub-tree.

Prefix: root, left sub-tree, right sub-tree.

Postfix: left sub-tree, right sub-tree, root.

For example, the tree given by figure 1.1 has the following representations: infix: 2164735, prefix: 1234675, and posftix: 2674531.

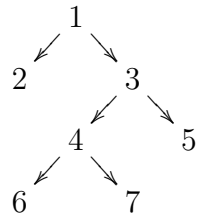

Figure 1.1: Binary tree.

[2]Note that this is almost the *Fundamental Theorem of Arithmetic*; what is missing is the fact that up to reordering of primes this representation is unique; to show uniqueness we need to prove first that if $p|(a_1a_2 \ldots a_n)$, and p is prime, then there exists an a_i such that $p|a_i$.

Note that some authors use a different name for infix, prefix, and postifix; they call it inorder, preorder, and postorder, respectively[3].

Exercise 1.1.10 Show that given any two representations we can obtain from them the third one, or, put another way, from any two representations we can reconstruct the tree. Show, using induction, that your reconstruction is correct. Then show that having just one representation is not enough.

1.2 Invariance

The *Invariance Technique* (IT) (see footnote[4]) is a technique for proving assertions about the outcomes of procedures. The IT identifies some property that remains true throughout the execution of a procedure. Then, once the procedure terminates, we use this property to prove assertions about the output.

As an example, consider an 8×8 board from which two squares from opposing corners have been removed (see Figure 1.2). The area of the board is 62 squares. Now suppose that we have 31 dominos of size 1×2. We want to show that the board cannot be covered by them[5].

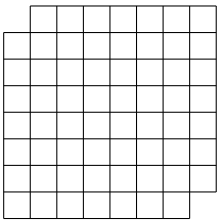

Figure 1.2: An 8×8 board.

Verifying this by "brute force" is a laborious job. However, using IT we argue as follows: color the squares as a chess board. Each domino, covering two adjacent squares, covers 1 white and 1 black square, and, hence, each placement covers as many white squares as it covers black squares. Note that the number of white squares and the number of black squares differ by 2—opposite corners lying on the same diagonal have the same color—and, hence, no placement of dominos yields a cover.

More precisely, we place the dominos one by one on the board, any way we want. The invariant is that after placing each new domino, the number of covered white squares is the same as the number of covered black squares.

[3]See [Knu97, §2.3.1, pg. 318] for more background on tree traversals.

[4]The exercises in this section come from [Eng98, Chapter 1].

[5]This example comes from [Dij89].

We prove that this *is* an invariant by induction on the number of placed dominos. The basis case is when zero dominos have been placed (so zero black and zero white squares are covered). In the induction step, we add one more domino which, no matter how we place it, covers one white and one black square, thus maintaining the property. At the end, when we are done placing dominos, we would have to have as many white squares as black squares covered, which is not possible due to the nature of the coloring of the board (i.e., the number of black and whites squares is not the same).

Note that this argument extends easily to the $n \times n$ board.

Exercise 1.2.1 Let n be an odd number. Write the numbers $\{1, 2, \ldots, 2n\}$ on a blackboard. Then pick any two numbers a, b, erase them, and write $|a-b|$ instead. Continue repeating this until just one number remains on the blackboard; show that this remaining number is odd.

Exercise 1.2.2 In the Parliament of some country, each member has at most three enemies (and being an enemy is a *reflexive relation*: if a is an enemy of b, then b is an enemy of a). Show that the house can be separated into two houses, so that each member has at most one enemy in his own house.

Exercise 1.2.3 $2n$ ambassadors are invited to a banquet. Every ambassador has at most $(n-1)$ enemies. Prove that the ambassadors can be seated at a round table so that nobody sits next to an enemy.

Exercise 1.2.4 Handshakes are exchanged at a big international congress. We call a person an *odd person* if he has exchanged an odd number of handshakes. Show that, at any moment, there is an even number of odd persons.

1.3 Correctness of algorithms

How can we prove that an algorithm is correct? We make two assertions, called the *pre-condition* and the *post-condition*; by correctness we mean that whenever the pre-condition holds *before* the algorithm executes, the post-condition will hold *after* it executes[6]. By *termination* we mean that whenever the pre-condition holds, the algorithm will stop running after finitely many steps. Correctness without termination is called *partial correctness*, and *correctness* per se is partial correctness *with* termination.

[6]For the history of the concept of pre and post-condition see [Knu97, pg. 17].

Consider the algorithm for integer division given below, with the pre and post-conditions given in the double curly-braces before and after the code.

In this section we assume that number inputs are in $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$; this assumption is implicitly part of the pre-conditions. Also note the indentation in lines 4 and 5 of algorithm 1.3.1; this indentation means that these two lines are part of the body of the while-loop, and so we get to line 6 only when the condition of the while-loop, $y \leq r$, is no longer true.

Algorithm 1.3.1 (Division)
$\{\!\{x \geq 0 \ \& \ y > 0\}\!\}$
1. $q \leftarrow 0$
2. $r \leftarrow x$
3. while $(y \leq r)$
4. $\quad r \leftarrow r - y$
5. $\quad q \leftarrow q + 1$
6. return q, r

$\{\!\{x = (q \cdot y) + r \ \& \ 0 \leq r < y\}\!\}$

The q and r returned by the division algorithm are usually denoted as $\mathrm{div}(x, y)$ and $\mathrm{rem}(x, y)$, respectively.

A *loop invariant* is some assertion that we make that stays true after each execution of a "while" (or "for") loop. This assertion is then used for proving partial correctness of the algorithm. In the case of the division algorithm the following is an appropriate loop invariant:

$$x = (q \cdot y) + r \ \& \ r \geq 0. \tag{1.3}$$

Note that in general many different loop invariants (and for that matter pre and post-conditions) may yield a desirable proof of correctness; the art of the analysis of algorithms consists in selecting them judiciously. We usually need induction to prove that a chosen loop invariant holds after each iteration of a loop, and we also usually need the pre-condition as an assumption in this proof.

We turn to (1.3) and show that it holds after each iteration of the loop. Basis case (i.e., zero iterations of the loop): $q = 0, r = x$, so $x = (q \cdot y) + r$ and since $x \geq 0$ and $r = x$, $r \geq 0$.

Induction step: suppose $x = (q \cdot y) + r \ \& \ r \geq 0$ and we go once more through the loop, and let q', r' be the new values of q, r, respectively. Since

we went through the loop one more time it follows that $y \leq r$, and since $r' = r - y$, we have that $r' \geq 0$. Since

$$x = (q \cdot y) + r = ((q+1) \cdot y) + (r - y) = (q' \cdot y) + r',$$

we have that the q', r', still satisfy the loop invariant (1.3).

Now we use the loop invariant to show that the post-condition of the division algorithm holds, *if* the pre-condition holds. This is very easy in this case since the loop ends when it is no longer true that $y \leq r$, i.e., when it is true that $r < y$, while (1.3) holds after each iteration, and in particular the last iteration. Putting together (1.3) and $r < y$ we get our post-condition, and hence partial correctness.

To show termination we use the least number principle (LNP). We need to relate some non-negative monotone decreasing sequence to the algorithm; just consider $r_0, r_1, r_2, \ldots$, where $r_0 = x$, and r_i is the value of r after the i-th iteration. Note that $r_{i+1} = r_i - y$. First, $r_i \geq 0$, because the algorithm enters the while loop only if $y \leq r$, and second, $r_{i+1} < r_i$, since $y > 0$. By LNP such a sequence "cannot go on for ever," (in the sense that the set $\{r_i | i = 0, 1, 2, \ldots\}$ is a subset of the natural numbers, and so it has a least element), and so the algorithm must terminate.

Thus we have full correctness of the division algorithm.

One of the oldest known algorithm is Euclid's algorithm[7], a process for finding the greatest common divisor of two numbers. It appeares in Euclid's *Elements* (Book 7, Propositions 1 and 2) around 300 BC. Given two positive integers a and b, the *gratest common divisor*, denoted as $\gcd(a, b)$ is the largest positive integer that divides them both.

Algorithm 1.3.2 (Euclid)
$\{\!\{a > 0$ & $b > 0\}\!\}$
1. $m \leftarrow a$
2. $n \leftarrow b$
3. $r \leftarrow \text{rem}(m, n)$
4. while $(r > 0)$
5. $\quad m \leftarrow n$
6. $\quad n \leftarrow r$
7. $\quad r \leftarrow \text{rem}(m, n)$
8. return n

[7]For an extensive study of Euclid's algorithm see [Knu97, §1.1].

$\{\!\{n = \gcd(a, b)\}\!\}$

Note that to compute $\mathrm{rem}(n, m)$ in lines 1 and 5 of Euclid's algorithm we need to use algorithm 1.3.1 (the division algorithm) as a subroutine; this is a typical "composition" of algorithms.

To prove the correctness of Euclid's algorithm we are going to show that after each iteration of the while loop the following assertion holds:

$$\gcd(a, b) = \gcd(m, n), \tag{1.4}$$

that is, (1.4) is our loop invariant. We prove this by induction on the number of iterations. Basis case: after zero iterations (i.e., just before the while loop starts—so after executing lines 1, 2, and 3, and before executing line 4) we have that $m = a$ and $n = b$, so (1.4) holds trivially.

For the induction step, suppose that $\gcd(a, b) = \gcd(m, n)$, and we go through the loop one more time, yielding m', n'. We want to show that $\gcd(m, n) = \gcd(m', n')$. Note that from lines 5, 6, and 7 of the algorithm we see that $m' = n, n' = r = \mathrm{rem}(m, n)$. In other words, it is enough to prove that in general $\gcd(m, n) = \gcd(n, \mathrm{rem}(m, n))$.

Exercise 1.3.3 Show that for all $m, n > 0$, $\gcd(m, n) = \gcd(n, \mathrm{rem}(m, n))$.

Now the correctness of Euclid's algorithm follows from (1.4), since the algorithm stops when $r = \mathrm{rem}(m, n) = 0$, so $m = q \cdot n$, and so $\gcd(m, n) = n$.

Exercise$*$ 1.3.4 Show that Euclid runs in time $O(\log(\min\{a, b\}))$. Try to give explicit constants instead of the big-Oh notation. Do you have any ideas to speed-up this algorithm?

Exercise$*$ 1.3.5 Given integers n, m, and $d = \gcd(n, m)$, it is possible to compute integers a, b such that $an + bm = d$. The algorithm for computing a, b (besides also computing the $\gcd(n, m)$) is called the *Extended* Euclid's algorithm. Design this algorithm and prove its correctness.

Exercise 1.3.6 Give an algorithm which when given a positive integer n (i.e., $n \in \mathbb{N} - \{0\}$), outputs "yes" if $n = 2^k$ (i.e., n is a power of 2), and "no" otherwise. Prove that your algorithm is correct.

The following algorithm tests strings for *palindromes*, which are strings that read the same backwards as forwards, for example, `madamimadam`[8] or `racecar`.

[8] This example comes from Joyce's *Ulysses*.

Algorithm 1.3.7 (Palindromes)
$\{\!\{ n \geq 1$ & $A[1 \ldots n]$ is a character array$\}\!\}$
1. $i \leftarrow 1$
2. while$(i \leq \lfloor \frac{n}{2} \rfloor)$
3. if $(A[i] \neq A[n-i+1])$
4. return FALSE
5. $i \leftarrow i+1$
6. return TRUE
$\{\!\{$ return True iff A is a palindrome $\}\!\}$

Let the loop invariant be: after the k-th iteration, $i = k+1$ & for all j such that $1 \leq j \leq k$, $A[j] = A[n-j+1]$. We prove that the loop invariant holds by induction on k. Basis case: before any iterations take place (i.e., after zero iterations), there are no j's such that $1 \leq j \leq 0$, so the second statement is (vacuously) true. The first statement holds since i is initially set to 1.

Induction step: we know that after k iterations, $A[j] = A[n-j+1]$ for all $1 \leq j \leq k$; after one more iteration we know that $A[k+1] = A[n-(k+1)+1]$, so the statement follows for all $1 \leq j \leq k+1$. This proves the loop invariant.

Exercise 1.3.8 Using the loop invariant argue the partial correctness of the palindromes algorithm. Show that the algorithm for palindromes always terminates. (Hint: consider $d_i = \lfloor \frac{n}{2} \rfloor - i$.)

Exercise 1.3.9 What does the following algorithm compute? Prove your claim.

Algorithm 1.3.10
1. $x \leftarrow m$; $y \leftarrow n$; $z \leftarrow 0$
2. while $(x \neq 0)$
3. if $(\mathrm{rem}(x,2) = 1)$
4. $z \leftarrow z + y$
5. $x \leftarrow \mathrm{div}(x,2)$
6. $y \leftarrow y \cdot 2$
7. return z

Exercise 1.3.11 What does the following algorithm compute? Assume that a, b are positive integers (i.e., assume that the pre-condition is that $a, b > 0$).

Algorithm 1.3.12
1. while $(a > 0)$
2. if $(a < b)$
3. $(a, b) \leftarrow (2a, b - a)$
4. else
5. $(a, b) \leftarrow (a - b, 2b)$

For which starting a, b does this algorithm terminate? In how many steps does it terminate, if it does terminate?

1.4 Stable marriage

The *Stable Marriage Problem*[9] was introduced by Gale and Shapley in 1962 and is related to the problem of college admissions. They gave an algorithm for solving the finite problem, which was later discovered to have been used in the matching of graduate medical students with hospitals since 1952. Other variants of the problem have been studied in computer science, economics, game theory and operations research.

An instance of the stable marriage problem of size n consists of two disjoint finite sets of equal size $B = \{b_1, b_2, \ldots, b_n\}$ (the set of *boys*) and $G = \{g_1, g_2, \ldots, g_n\}$ (the set of *girls*). In addition, each boy b_i has a ranking or a linear ordering $<_i$ of G which reflects his preference for the girls that he wants to marry. That is, if $g_j <_i g_k$, then b_i would prefer to marry g_j over g_k. Similarly each girl g_j has a ranking or linear ordering $<^j$ of B which reflects her preference for the boys she would like to marry.

A *matching* (or *marriage*) M is a 1-1 correspondence between B and G. We say that b and g are *partners* in M if they are matched in M and write $p_M(b) = g$ and also $p_M(g) = b$. A matching M is *unstable* if there is a pair (b, g) from $B \times G$ such that b and g are not partners in M but b prefers g to $p_M(b)$ and g prefers b to $p_M(g)$. Such a pair (b, g) is said to *block* the matching M and is called a *blocking pair* for M (see figure 1.3). A matching M is stable if there is no blocking pair for M.

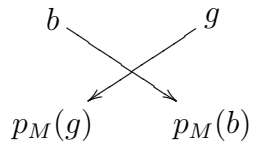

Figure 1.3: Blocking pair.

[9]This section is based on §2 in in the article *Proof-Theoretic Strength of the Stable Marriage Theorem and Other Problems*, by Douglas Cenzer and Jeffrey B. Remmel.

The result of Gale and Shapley is that any marriage problem has a solution, i.e., there always exists a stable matching, no matter what the lists of preferences. In fact, they give an algorithm which produces a solution in n stages and takes $\leq O(n^3)$ steps.

1.4.1 The Gale-Shapley algorithm

The matching M is produced in stages M_s so that b_t always has a partner at stage $s \geq t$ and $p_{M_t}(b_t) <_t p_{M_{t+1}}(b_t) <_t \cdots$. On the other hand, for each $g \in G$, if g has a partner at stage t, then g will have a partner at each stage $s \geq t$ and $p_{M_t}(g) >^t p_{M_{t+1}}(g) >^t \ldots$. Thus, as s increases, the partners of b_t become less preferable and the partners of g become more preferable.

Stage 1: At stage 1, b_1 chooses the first girl g in his preference list and we set $M_1 = \{(b_1, g)\}$.

Stage $(s+1)$: At the end of stage s, assume that we have produced a matching

$$M_s = \{(b_1, g_{i(1,s)}), \ldots, (b_s, g_{i(s,s)})\}.$$

We will say that partners in M_s are "engaged". The idea is that at stage $s+1$, b_{s+1} will try to get a partner by "proposing" to the girls in G in his order of preference. When b_{s+1} proposes to a girl g_j, g_j accepts his proposal if either g_j is not currently engaged or is currently engaged to a boy b such that $b_{s+1} <^j b$. In the case where g_j prefers b_{s+1} over her current partner b, then g_j breaks off an engagement with b and b then has to search for a new partner.

To be more precise, we begin stage $s+1$ by letting $M = M_s$ and letting $b^* = b_{s+1}$. Then we apply the following routine. We have b^* propose to the girls in order of his preference until one accepts. Here g will accept the proposal as long as she is either not engaged or prefers b^* to her current partner $p_M(g)$. Then we add (b^*, g) to M and proceed according to one of the following two cases: (i) If g was not engaged, then we terminate the procedure and let $M_{s+1} = M \cup \{(b^*, g)\}$. (ii) If g was engaged to b, then we set $M = (M - \{(b, g)\}) \cup \{(b^*, g)\}$ and $b^* = b$ and we continue.

Exercise 1.4.1 Show that each b need only propose at most once to each g.

From exercise 1.4.1 we see that we can make each boy keep a bookmark on his list of preference, and this bookmark is only moving forward. When a boy's turn to choose comes, he starts proposing from the point where his

bookmark is, and by the time he is done, his bookmark moved only forward. Note that at stage $(s+1)$ each boy's bookmark cannot move beyond the girl number s on the list without choosing someone (after stage s only s girls are engaged). As the boys take turns, each boy's bookmark is advancing, so some boy's bookmark (among the boys in $\{b_1, \ldots, b_{s+1}\}$) will advance eventually to a point where he must choose a girl.

Furthermore, this gives an upper bound of $(s+1)^2$ steps at stage $(s+1)$ in the procedure. This means that there are n stages, and each stage takes $O(n^2)$ steps, and hence $O(n^3)$ steps altogether.

Exercise 1.4.2 Show that there is exactly one girl that was not engaged at stage s but is engaged at stage $(s+1)$ and that, for each girl g_j that is engaged in M_s, g_j will be engaged in M_{s+1} and that $p_{M_{s+1}}(g_j) <^j p_{M_s}(g_j)$.

The consequence of exercise 1.4.2 is that, for any girl g_j, once she becomes engaged, she will remain engaged and her partners will only gain in preference as the stages proceed.

Exercise 1.4.3 Suppose that $|B| = |G| = n$. Show that at the end of stage n, M_n will be a stable marriage.

Exercise 1.4.4 We say that a matching (b, g) is *feasible* if there exists a stable matching in which b, g are partners. We say that a matching is boy-*optimal* if every boy is paired with his highest ranked feasible partner. We say that a matching is boy-*pesimal* if every boy is paired with his lowest ranking feasible partner. Similarly, we define girl-*optimal/pesimal*. Show that our version of the algorithm produces a boy-optimal &girl-pesimal stable matching.

1.5 Answers to selected exercises

Exercise 1.1.2. This comes from [Knu97, §1.2.1 exercise #8, pg. 19]. Basis case: $n = 1$, then $1^3 = 1^2$. For the induction step:

$$\begin{aligned}
&(1+2+3+\cdots+n+(n+1))^2 \\
&= (1+2+3+\cdots+n)^2 + 2(1+2+3+\cdots+n)(n+1) + (n+1)^2 \\
&= (1^3+2^3+3^3+\cdots+n^3) + 2(1+2+3+\cdots+n)(n+1) + (n+1)^2 \qquad \text{IH} \\
&= (1^3+2^3+3^3+\cdots+n^3) + 2\frac{n(n+1)}{2}(n+1) + (n+1)^2 \\
&= (1^3+2^3+3^3+\cdots+n^3) + n(n+1)^2 + (n+1)^2 \\
&= (1^3+2^3+3^3+\cdots+n^3) + (n+1)^3
\end{aligned}$$

Exercise 1.1.3. It is important to interpret the statement of the exercise correctly; when it says that one square is missing, it means that any square can be missing. So the basis case is: given a 2×2 square, there are four possible ways for a square to be missing; but in each case, the remaining squares form an "L." Suppose the claim holds for n, and consider a square of size $2^{n+1} \times 2^{n+1}$. Divide it into four quadrants of equal size. No matter which square we choose to be missing, it will be in one of the four quadrants; that quadrant can be filled with "L" shapes by induction hypothesis. As to the remaining three quadrants, put an "L" in them in such a way that it straddles all three of them (the "L" wraps around the center staying in those three quandrants). The remaining squares of each quadrant can now be filled with "L" shapes by induction hypothesis.

Exercise 1.1.5. The basis case is $n = 1$, and it is immediate. For the induction step, assume the equality holds for exponent n, and show that it holds for exponent $n + 1$:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} f_{n+1} & f_n \\ f_n & f_{n-1} \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} f_{n+1}+f_n & f_{n+1} \\ f_n+f_{n-1} & f_n \end{pmatrix}$$

The right-most matrix can be simplified using the definition of Fibonacci numbers to be as desired.

Exercise 1.1.6. $m|n$ iff $n = km$, so show that $f_m|f_{km}$ by induction on k. If $k = 1$, there is nothing to prove. Otherwise, $f_{(k+1)m} = f_{km+m}$. Now, using a separate inductive argument, show that for $y \geq 1$, $f_{x+y} = f_y f_{x+1} + f_{y-1} f_x$, and finish the proof. To show this last statement, let $y = 1$, and note that $f_y f_{x+1} + f_{y-1} f_x = f_1 f_{x+1} + f_0 f_x = f_{x+1}$. Assume now that $f_{x+y} =$

$f_y f_{x+1} + f_{y-1} f_x$ holds. Consider

$$\begin{aligned} f_{x+(y+1)} &= f_{(x+y)+1} = f_{(x+y)} + f_{(x+y)-1} = f_{(x+y)} + f_{x+(y-1)} \\ &= (f_y f_{x+1} + f_{y-1} f_x) + (f_{y-1} f_{x+1} + f_{y-2} f_x) \\ &= f_{x+1}(f_y + f_{y-1}) + f_x(f_{y-1} + f_{y-2}) \\ &= f_{x+1} f_{y+1} + f_x f_y. \end{aligned}$$

Exercise 1.1.8. Let our assertion $P(n)$ be: the minimal number of breaks to break up a chocolate bar of n squares is $(n-1)$. Note that this says that $(n-1)$ breaks are sufficient, and $(n-2)$ are not. Basis case: only one square requires no breaks. Induction step: Suppose that we have $m+1$ squares. No matter how we break the bar into two smaller pieces of a and b squares each, $a+b = m+1$.

By induction hypothesis, the "a" piece requires $a-1$ breaks, and the "b" piece requires $b-1$ breaks, so together the number of breaks is

$$(a-1) + (b-1) + \boxed{\mathbf{1}} = a + b - 1 = m - 1,$$

and we are done (note that the **1** in the box comes from the initial break to divide the chocolate bar into the "a" and the "b" pieces).

Note that this tells us that the "boring" way of breaking up the chocolate (first into rows, and then each row separately into pieces) is in fact optimal.
Exercise 1.1.9. Let IND be: $[P(0)\&(\forall n)(P(n) \to P(n+1))] \to (\forall m)P(m)$ (where n, m range over natural numbers), and let LNP: *Every non-empty subset of the natural numbers has a least element.* These two principles are equivalent, in the sense that one can be shown from the other. Indeed:

[LNP $\Rightarrow$ IND] Suppose we have $[P(0)\&(\forall n)(P(n) \to P(n+1))]$, but that it is NOT the case that $(\forall m)P(m)$. Then, the set S of m's for which $P(m)$ is false is non-empty. By the LNP S has a least element. We know this element is not 0, as $P(0)$ was assumed. So this element can be expressed as $n+1$ for some natural number n. But since $n+1$ is the least such number, $P(n)$ must hold. This is a contradicion as we assumed that $(\forall n)(P(n) \to P(n+1))$, and here we have an n such that $P(n)$ but not $P(n+1)$.

[IND $\Rightarrow$ LNP] Suppose that S is a non-empty subset of the natural numbers. Suppose that it does not have a least element; let $P(n)$ say the following assertion "all elements up to and including n are not in S". We know that $P(0)$ must be true, for otherwise 0 would be in S, and it would then be the least element (by definition of 0). Suppose $P(n)$ is true (so

none of $\{0, 1, 2, \ldots, n\}$ is in S). Suppose $P(n+1)$ were false: then $n+1$ would necessarily be in S (as we know that none of $\{0, 1, 2, \ldots, n\}$ is in S), and thereby $n+1$ would be the smallest element in S. So we have shown $[P(0)\&(\forall n)(P(n) \rightarrow P(n+1))]$. By IND we can therefore conclude that $(\forall m)P(m)$. But this means that S is empty. Contradiction. Thus S must have a least element.

Exercise 1.1.10. For example, suppose that we want to obtain the tree from the infix (2164735) and prefix (1234675) encodings: from the prefix encoding we know that 1 is the root, and thus from the infix encoding we know that the left sub-tree has the infix encoding 2, and so prefix encoding 2, and the right sub-tree has the infix encoding 64735 and so prefix encoding 34675, and we recurse.

Exercise 1.2.1.[10] Consider the following invariant: let S be the sum of the numbers currently on the blackboard; then S is odd. Now we prove that this invariant holds. Basis case: $S = 1 + 2 + \cdots + 2n = n(2n+1)$ which is odd. Induction step: assume S is odd, let S' be the result of one more iteration, so $S' = S + |a - b| - a - b = S - 2\min(a, b)$, and since $2\min(a, b)$ is even, and S was odd by the induction hypothesis, it follows that S' must be odd as well. At the end, when there is just one number left, say x, $S = x$, so x is odd.

Exercise 1.2.2.[11] To solve this problem we must provide both an algorithm and an invariant for it. The algorithm works as follows: initially divide the parliament into any two groups. Let H be the total sum of enemies that each member has in his own house. Now repeat the following loop: while there is an m which has at least two enemies in his own house, move m to the other house (where m must have at most one enemy). Thus, when m switches houses, H decreases. Here the invariant is "H decreases monotonically." Now we know that a sequence of positive integers cannot decrease for ever, so when H reaches its absolute minimum, we obtain the required distribution.

Exercise 1.2.3.[12] Sit the ambassadors any way. Let H be the number of neighboring hostile pairs. We find an algorithm that reduces H whenever $H > 0$. Suppose $H > 0$, and let (A, B) be a hostile couple. Traverse the table until we find another couple (A', B') such that A, A' and B, B' are friends. Such a couple must exist, for A has $(n+1)$ friends A', and if besides

[10] [Eng98, §1, E2, pg. 2].
[11] [Eng98, §1, E4, pg. 2].
[12] [Eng98, §1, E8, pg. 5].

each A' there were only enemies of B' then B would have more than $(n-1)$ enemies. Now the situation around the table is $\ldots, A, \boxed{B, \ldots, A'}, B', \ldots$. Reverse everyone in the box (i.e., mirror image the box), to reduce H by 1.

Exercise 1.3.3. We show that $\gcd(m,n) = \gcd(n, \mathrm{rem}(m,n))$. First consider the case that $m < n$. Then $\mathrm{rem}(m,n) = m$, and it is certainly true that $\gcd(m,n) = \gcd(n,m)$. Now consider the case $m \geq n$. Suppose that $i|\gcd(m,n)$, then $i|m$ and $i|n$, and so $i|(m-qn)$ (where $m = q \cdot n + r$, $0 \leq r < n$), and so $i|r = \mathrm{rem}(m,n)$, and thus $i|\gcd(n, \mathrm{rem}(m,n))$. A similar argument shows that if $i|\gcd(n, \mathrm{rem}(m,n))$ then $i|\gcd(m,n)$, and so we have equality.

Exercise 1.3.4. One idea to speed-up this algorithm is the following: notice that when $m < n$ then all the loop does is to swap them; that is, if $m < n$, and m', n' are m, n after one iteration, respectively, then $m' = n$ and $n' = m$. So instead of going through the entire while loop to do this swapping, assume $a \geq b$ (swap them at the beginning if that is not the case), and maintain the property that $m \geq n$ throughout the execution.

Exercise 1.3.6.

```
1. x ← n
2. while (x > 1)
3.         if (2|x) then
4.             x ← x/2
5.         else
6.             stop and return "no"
7. return "yes"
```

Pre-condition: $n \geq 1$; post-condition: return "yes" $\iff$ n is a power of 2; loop invariant: x is a power of 2 iff n is a power of 2.

We show the loop invariant by induction on the number of iterations of the main loop. Basis case: zero iterations, and since $x \leftarrow n$, $x = n$, so obviously x is a power of 2 iff n is a power of 2. For the induction step, note that if we ever get to update x, we have $x' = x/2$, and clearly x' is a power of 2 iff x is. Note that the algorithm always terminates (let $x_0 = n$, and $x_{i+1} = x_i/2$, and apply the LNP as usual). We can now prove correctness: if the algorithms returns "yes", then after the last iteration of the loop $x = 1 = 2^0$, and by the loop invariant n is a power of 2. If, on the other hand, n is a power of 2, then so is every x, so eventually $x = 1$, and so the algorithm returns "yes".

Exercise 1.3.8. To show that the algorithm terminates, let $d_i = \lfloor \frac{n}{2} \rfloor - i$. By the precondition, we know that $n \geq 1$. The sequence $d_1, d_2, d_3, \ldots$ is a

decreasing sequence of natural numbers (because $i \leq \lfloor \frac{n}{2} \rfloor$), so by the fact it is finite, and so the loop terminates.

Exercise 1.4.1. If some b wants to propose to the same g for a second time, it means that they already once disengaged, and since the g's disengage only when they get a better offer, it means that now g still has a better partner than b, and so even if b proposed, he would be rejected. Therefore, it is not necessary for the b's to propose twice to the same g.

Exercise 1.4.2. b_{s+1} proposes to the g's according to his list of preference; a g ends up accepting, and if the g who accepted b_{s+1} was free, she is the new one with a partner. Otherwise, some $b^* \in \{b_1, \ldots, b_s\}$ became disengaged, and we repeat the same argument. The g's disengage only if a better b proposes, so it is true that $p_{M_{s+1}}(g_j) <^j p_{M_s}(g_j)$.

Exercise 1.4.3. Suppose that we have a blocking pair $\{b, g\}$ (meaning that $\{(b, g'), (b', g)\} \subseteq M_n$, but b prefers g to g', and g prefers b to b'). Either b came after b' or before. If b came before b', then g would have been with b or someone better when b' came around, so g would not have become engaged to b'. On the other hand, since (b', g) is a pair, no better offer has been made to g after the offer of b', so b could not have come after b'. In either case we get an impossibility, and so there is no blocking pair $\{b, g\}$.

Exercise 1.4.4. To show that the matching is boy-optimal, we argue by contradiction. (Let *g is an optimal partner for b* mean that among all the stable matchings g is the best partner that b can get.)

We run the Gale-Shapley algorithm, and let b be the first boy who is rejected by his optimal partner g. This means that g has already been paired with some b', and g prefers b' to b. Furthermore, g is at least as desirable to b' as his own optimal partner (since the proposal of b is the first time during the run of the algorithm that a boy is rejected by his optimal partner). Since g is optimal for b, we know (by definition) that there exists some stable matching S where (b, g) is a pair. On the other hand, the optimal partner of b' is ranked (by b' of course) at most as high as g, and since g is taken by b, whoever b' is paired with in S, say g', b' prefers g to g'. This gives us an unstable pairing, because $\{b', g\}$ prefer each other to the partners they have in S.

To show that the Gale-Shapley algorithm is girl-pessimal, we use the fact that it is boy-optimal (which we just showed). Again, we argue by contradiction. Suppose there is a stable matching S where g is paired with b, and g prefers b' to b, where (b', g) is the result of the Gale-Shapley algorithm. By boy-optimality, we know that in S we have (b', g'), where g' is not higher

on the preference list of b' than g, and since g is already paired with b, we know that g' is actually lower. This says that S is unstable since $\{b', g\}$ would rather be together than with their partners.