Chapter 1

Mathematical Background

Broadly speaking, this text concerns what can and cannot be computed, and when something can be computed how simply and efficiently it can be done. As you will see later in this chapter, there are precisely defined problems which cannot be solved computationally, regardless of how fast or large a computer is used or how much time is spent on the computation. Likewise, as you will see in a later chapter, there are problems that are effectively infeasible although solvable in principle, for the resources needed would be prohibitive.

To formulate these issues precisely entails appropriate mathematical specifications. Accordingly, this chapter begins by reviewing notation and terminology that will be used throughout the book. This is followed by a quick overview of several proof techniques.

1.1 Definitions and Terminology

1.1.1 Sets

A set is a collection of items. To specify a set, the items are written as a list enclosed by braces, e.g. $A = \{1, 2, 4\}$. This means that $A$ is the set containing the items 1, 2, and 4. There is no notion of order in the set listing, so $\{1, 2, 4\} = \{4, 1, 2\} = \{2, 4, 1\}$, etc. Also, there is no notion of duplicate items, so $\{1, 2, 1\} = \{1, 2\}$ for example.

The empty set is the set containing no items; it is written as $\{\}$ or $\emptyset$ for short.

Notation

- $x \in A$ means that $x$ is one of the items contained in set $A$; $x$ is called an element of $A$.
- $A \cap B$, the intersection of $A$ and $B$, denotes the set containing those items that are in both $A$ and $B$.
- $A \cup B$, the union of $A$ and $B$, denotes the set containing those items that are in at least one of $A$ or $B$. 
• $A \subseteq B$ means that every item in $A$ is also in $B$; $A$ is called a subset of $B$. When, in addition, there is an item in $B$ which is not in $A$, we write $A \subset B$, or for emphasis $A \subsetneq B$. $A$ is then called a strict subset of $B$.

• $\overline{A}$ denotes the set containing those items not in $A$. For this definition to be meaningful we need a notion of the universe of items at hand, the set $U$ of items, where $A \subseteq U$. Then $\overline{A}$ is the set of items in $U$ but not in $A$.

• $|A|$ denotes the number of items in $A$.

• $A - B$ is the set consisting of items contained in $A$ that are not contained in $B$. Note that $A - B = A \cap \overline{B}$.

**Question 1.** What is the size of the empty set, $|\emptyset|$?

**Venn Diagrams.** These are diagrams that are used to illustrate the intersections of collections of sets. They can be useful in helping us see what is going on. Typically, the Universal Set, $U$, if present, is shown as a large rectangle. Other sets appear as circular shapes inside this rectangle.

![Venn Diagrams](image)

Figure 1.1: Venn Diagrams. (a) Sets $A$ and $B$ in a universal set $U$. (b) Sets $A$ and $\overline{A}$. (c) Sets $A$ and $B$ (represented by circles) and subsets $A - B$, $A \cap B$, $B - A$ (represented by the regions they label).
1.1. DEFINITIONS AND TERMINOLOGY

**Power Set.** The power set of \( A \) is the collection (or set) of all subsets of \( A \); it is written as \( 2^A \).

E.g. \( A = \{a, b\} \). \( \phi \subseteq A, a \subseteq A, b \subseteq A, \{a, b\} \subseteq A \). So \( 2^A = \text{PowerSet}(A) = \{\phi, \{a\}, \{b\}, \{a, b\}\} \).

**Question 2.** Show that if \( |A| = n \), then \( |\text{PowerSet}(A)| = 2^n \). Note that this provides some justification for the notation \( 2^A \).

1.1.2 Sequences

A sequence is an ordered list of items in which there may be repetitions. For example, here are two sequences of three items each: \( (10, 4, 13) \), and \( (10, 9, 10) \); the second sequence has a repetition. Conventionally, the items are regarded as being ordered in the left to right order in which they are written. A \( k \)-item sequence is often called a \( k \)-tuple.

Let \( A \) and \( B \) be sets. \( A \times B \) denotes the set of all possible 2-tuples whose first element is in \( A \) and whose second element is in \( B \). So \( |A \times B| = |A| \times |B| \). \( A \times B \) is called the cross-product or Cartesian product of \( A \) and \( B \). We can write

\[
A \times B = \{(a, b) \mid a \in A, b \in B\}.
\]

This is a first example of a notation for specifying sets. The expression to the left of the bar denotes a typical element of the set (in this case, a 2-tuple); the statement to the right of the bar explains what constraints such elements must obey (in this case that the first item in the tuple be an element of \( A \), and the second an element of \( B \)). It is understood that all the elements obeying the constraints (or condition or property) are in the set being defined. So 2-tuple \((e, f)\) will be an element of \( A \times B \) exactly if \( e \in A \) and \( f \in B \).

E.g. \( A = \{1, 3, 5\}, B = \{0, 1\} \). \( A \times B = \{(1, 0), (3, 0), (5, 0), (1, 1), (3, 1), (5, 1)\} \).

1.1.3 Strings

A string is simply a sequence of characters, e.g. 01001001, CTAGCTTAG, the\_dog\_chased\_the\_cat. In the third example, note that the blank (\_\_) is a distinct character.

Strings are everywhere. Text in documents provides an obvious example of strings. DNA sequences are another example. Programs are just (long) strings. Likewise, all inputs to programs are themselves strings. To be able to do anything useful with any of these various classes of strings one has to recognize and understand the useful units from which they are formed. Useful units could be individual words in text, subunits coding a protein in DNA, keywords in a program, mathematical operators (such as +), the digits forming a number, etc.

The study of strings forms a major part of this text: what collections of strings can be recognized, how easily this can be done for different sorts of collections, and what limits we face (for there are collections of strings that cannot be recognized as efficiently as we might wish, and other collections that cannot be recognized at all).
Alphabet. This is a set of one or more characters, e.g., \{0\}, \{0, 1\}, \{a, b, c, \ldots, z, \omega\}, \{A, C, G, T\}, etc. Note the use of the \ldots notation to indicate completion of the sequence in the natural way (in this case by including every letter of the English alphabet in the usual order). This notation is often used to specify infinite sets, e.g. \{1, 2, 3, \ldots\}, which denotes the set of all natural numbers, the integers greater than or equal to 1.

Strings. A string is a sequence of zero or more characters. The zero character string, written \(\lambda\), is a legitimate string. It has a role with strings analogous to the role 0 has with numbers as we will see in a bit. Incidentally, many authors use \(\epsilon\) to denote the empty string. I prefer to avoid this notation as it is rather similar to the symbol for set membership.

Notation Conventions. It is standard to use lower case letters from the end of the alphabet to name strings, typically \(u, v, w, x, y, z\). Lower case letters from the beginning of the alphabet are used for individual characters. Both can be indexed: e.g. \(a_1, b_2, u_3\), etc. Sometimes, though, \(u_i\) will indicate the \(i\)th character of string \(u\) (read from left to right). Typically, capital letters are used to name sets of strings.

More notation. \(\Sigma\) usually indicates an alphabet.

Language \(L\). This is a set of strings. e.g. \(L = \{aa, ab, ba, bb\}\), is the set of all 2-character strings over the alphabet \(\{a, b\}\). (The term "over" indicates the alphabet from which the characters are drawn.) \(\Sigma^*\) denotes the set of all strings over the alphabet \(\Sigma\); e.g. \(\{a, b\}^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, \ldots\}\).

Operations on Strings

Concatenation. The concatenation of string \(u\) and string \(v\) is simply the string obtained by writing the characters of \(u\) followed by the characters of \(v\). It is written as \(u \circ v\) or simply \(uv\).

Examples: \(u = 01, v = 23\); then \(uv = 0123\). \(u = a_1a_2\cdots a_i, v = b_1b_2\cdots b_j\); then \(uv = a_1a_2\cdots a_ib_1b_2\cdots b_j\).

Note that the concatenation of string \(u\) and the empty string yields string \(u\): \(u \circ \lambda = u\); e.g. \(abc \circ \lambda = abc\). Likewise \(\lambda \circ u = u\). This is analogous to the addition of zero to a number: \(n + 0 = n\).

Reversal. The reversal of string \(u\), denoted \(u^R\), is the string obtained by writing the characters of \(u\) in reverse order. e.g. \(u = \) stop, \(u^R = \) pots; \(u = a_1a_2\cdots a_i, u^R = a_ia_{i-1}\cdots a_1\).

Formally, \(\lambda^R = \lambda\), and for \(v\) with \(|v| \geq 1\), i.e. \(v = au\) for some \(a \in \Sigma\) and \(u \in \Sigma^*\), \(v^R = u^Ra\).

Substring. String \(v\) is a substring of string \(z\) if \(z\) can be written as \(z = uvw\) where \(u\) and \(w\) are also strings. Note that one or both of \(u\) and \(w\) could be the empty string. So \(z\) is a (trivial) substring of \(z\) as is the empty string \(\lambda\).

Examples: Let \(z = \text{cat}\). \(ca, at\) are both substrings of \(z\), but \(ct\) is not.

Length of \(u\). This is the number of characters in \(u\), and is denoted by \(|u|\). So if \(u = a_1a_2\cdots a_i, |u| = i\).
1.1. DEFINITIONS AND TERMINOLOGY

1.1.4 Graphs

A graph consists of two sets, a set of vertices and a set of edges. We draw a graph using little circles to denote vertices, and line segments joining pairs of vertices to denote edges. Vertices are named by little letters, typically \( u, v, w \), or sometimes indexed, as in \( v_1, v_2, \ldots \). Edges are specified by naming the pair of vertices they join.

In the example in Figure 1.2, the vertex set \( V \) is given by \( V = \{ u, v, w, x, y \} \); the edge set \( E \) is given by \( E = \{ (u, v), (v, x), (v, w), (w, u) \} \). \( G = (V, E) \) denotes the graph.

There are two types of graphs, undirected and directed. In an undirected graph, the edges are unordered. In a directed graph, the edges are ordered, and here an edge \( e = (u, v) \) indicates an edge from vertex \( u \), called the tail of \( e \), to vertex \( v \), called the head of \( e \). Edges are sometimes named using letters \( e, f, g \), or using indices, as in \( e_1, e_2, \ldots \).

**Path.** A path in a graph is a sequence of vertices \( v_1, v_2, \ldots, v_k \) where successive vertices are joined by edges, i.e. \( (v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k) \) are all edges in the graph, whether directed or undirected. Sometimes we specify this path as the edge sequence \( e_1, e_2, \ldots, e_{k-1} \), where \( e_i = (v_i, v_{i+1}) \), for \( 1 \leq i < k \).

**Cycle.** This is a path that begins and ends at the same vertex.

**Simple path.** A path with no repeated vertex.

**Simple cycle.** A cycle with exactly one repeated vertex (its first and last vertex).

**Connected component.** This applies to undirected graphs only. It is a maximal collection of vertices such that for each pair of vertices in the component there is a path between them.

**Strongly connected component.** This applies to directed graphs only. It is a maximal collection of vertices such that for each ordered pair of vertices \( (u, v) \) in the component, there is a path starting at \( u \) and ending at \( v \).

In this text, by and large, the graphs we will be looking at are directed. Also, the edges will have labels, which will be either single characters, strings, or even sets of strings.

**Path label.** let \( P = (e_1, e_2, \ldots, e_r) \) be a path with edge \( e_i \) labeled by string \( s_i \), for \( 1 \leq i \leq r \). Then \( P \) is said to have label \( s_1s_2\cdots s_r \), the concatenation of strings \( s_1, s_2, \ldots, s_r \).
1.1.5 Trees

Perhaps the easiest way to define a tree is recursively.

- A tree can be empty.
- A tree can consist of a single node called the root.
- A tree can be a node, \( r \) say, called the root, together with a collection of non-empty trees, called the subtrees of the root.

![Figure 1.4: A Tree.](image)

Terminology

- The children of node \( u \) are the roots of \( u \)'s subtrees. \( u \) is the parent of its children.
- A leaf is a node with no children.
- \( v \) is a proper descendant of \( u \) if \( v \) is a node in one of \( u \)'s subtrees; likewise, \( u \) is a proper ancestor of \( v \).
- \( v \) is a descendant of \( u \) if \( v \) is \( u \) or if \( v \) is a proper descendant of \( u \); likewise, \( u \) is an ancestor of \( v \).
- \( v \) and \( w \) are siblings if they have the same parent.

1.1.6 Logic

Logic is the algebra of variables that take on just one of two values: \texttt{True} and \texttt{False} (\( T \) and \( F \) for short).

Operations

- \( x \land y \) (called \( x \) and \( y \)). This is \texttt{True} exactly if both \( x \) and \( y \) are \texttt{True}; otherwise it is \texttt{False}. 

1.2. PROOFS

- $x \lor y$ (called $x$ or $y$). This is True if at least one of $x$ or $y$ is True; otherwise (if both $x$ and $y$ are False) it is False.

- $\overline{x}$ (called not $x$, or sometimes “$x$ bar”); it is also written as $-x$. This is True exactly if $x$ is False; otherwise it is False.

Often, Boolean variables are defined with respect to a universe of objects. For example, the universe might be the set of integers and $x$ the property of being prime; then $x(n)$ equals True exactly if $n$ is a prime number.

There are two common and useful ways of expressing Boolean properties with respect to universes.

- $\forall x P(x)$. This is read as “for all $x$, $P(x)$ is true” and is itself True exactly if, for every $x$ in the relevant universe, $P(x)$ is True.

- $\exists x P(x)$. This is read as “there exists an $x$ for which $P(x)$ is true” and is itself True exactly if, for some $x$ in the relevant universe, $P(x)$ is True.

Most assertions we wish to prove involve an implicit quantification. For example the assertion “The merge sort algorithm rearranges its input in sorted order” really means “For all inputs $I$, the merge sort algorithm on input $I$ rearranges the input to put it in sorted order.”

Observe that $\neg(\forall x P(x)) = \exists x (\neg P(x))$ and $\neg(\exists x P(x)) = \forall x (\neg P(x))$.

There are two more useful Boolean operators, implies ($\Rightarrow$) and if and only if (iff or $\Leftrightarrow$).

- $x \Rightarrow y$. This is read as “$x$ implies $y$” and means that if $x$ is True then so is $y$. This is equivalent to $\overline{x} \lor y$. For if $x \Rightarrow y$ is True there are two possibilities:

  (i) $x$ is True and hence so is $y$; then $\overline{x} \lor y$ is True.

  (ii) $x$ is False; then there is no constraint on $y$. $x \Rightarrow y$ is still True and so is $\overline{x} \lor y$.

We now see that $x \Rightarrow y$ is False only if $x$ is True and $y$ is False; but then $\overline{x} \lor y$ is also False. Thus these two expressions ($x \Rightarrow y$ and $\overline{x} \lor y$) have the same truth value; they are then said to be equivalent.

- $x \Leftrightarrow y$. This is read as “$x$ if and only if $y$.” It means that $x$ implies $y$ and $y$ implies $x$.

1.2 Proofs

This text is concerned with the demonstration, or proof, of mathematical claims, called theorems. The way this is done is to put together unassailable arguments, that is a flow of reasoning using clearly specified rules of deduction. At its most formal, this requires the specifications of axioms, which are the basic facts that are taken as given (e.g. the postulates
seen in high school geometry). Then each step in the argument proceeds by appeal to a standard fact (e.g. the angles of a triangle sum to 180°), a previously proven result, and a given collection of rules of deduction. We will not be this formal, but in principle any argument we give could be fleshed out in (much) greater detail so as to be of this form.

There are several styles of proof you will be encountering repeatedly. We begin by reviewing their use, by showing a few examples.

1.2.1 Proof by Construction

A proof by construction is a direct demonstration that a claimed object exists (a construction). For example:

**Theorem 1.2.1.** Suppose $n$ is the product of three distinct primes, $n = pqr$ say, where $p \neq q \neq r \neq p$. Then $n$ has (at least) 6 distinct factors.

**Proof.** The factors are $p$, $q$, $r$, $pq$, $pr$, and $qr$ and as $p$, $q$, $r$ are all distinct primes, these 6 numbers are all distinct.

In fact, these are the only non-trivial factors of $n$ (ruling out 1 and $n$ as factors), for each possible factor must be a product using some but not all of $p$, $q$, and $r$, and there are 6 such choices.

1.2.2 Proof by Contradiction

Suppose we want to prove assertion $A$. In a proof by contradiction, we begin by supposing ("assuming") that in fact $A$ is false. By a sequence of deductions we then manage to reach a contradiction, formally that $\text{FALSE} = \text{TRUE}$; an example would be that $0 = 1$. Since this is not possible (in any reasonable mathematical system), the only possibility is that the initial supposition was incorrect. That is, the assertion that $A$ is false was incorrect, which means that $A$ is actually true, and this constitutes a proof of that fact.

**Theorem 1.2.2.** $\sqrt{3}$ is irrational.

**Proof.** Recall that a number $r$ is rational if it can be written as the ratio of two integers, $r = n/m$ say. As you can cancel any common factors in $m$ and $n$, you can always assume that $m$ and $n$ have no common factors; such $m$ and $n$ are said to be coprime.

Now to the proof. Begin by assuming, for a contradiction, that $\sqrt{3}$ is in fact rational. This means that $\sqrt{3} = a/b$ for some coprime integers $a$ and $b$. Squaring both sides gives $3 = a^2/b^2$ or $3b^2 = a^2$. But as 3 divides $a^2$, 3 must divide $a$ (strictly, 3, being prime, must divide one of the factors of $a^2$, namely one of $a$ or $a$). In other words, $a = 3c$, where $c$ is also an integer. Substituting for $a$ (in $3b^2 = a^2$) yields $3b^2 = 9c^2$; simplifying this yields $b^2 = 3c^2$. By the same argument as before 3 must divide $b$. But then $a$ and $b$ are not coprime, for they are both divisible by 3. This contradicts the assumption that $\sqrt{3}$ is rational (namely that $\sqrt{3} = a/b$ where $a$ and $b$ are coprime integers). Thus $\sqrt{3}$ is not rational, that is $\sqrt{3}$ is irrational.
The Pigeonhole Principle. This states that if there are \( n - 1 \) slots and \( n \) objects, if each object is placed in a slot then some slot must receive two objects. This is a rather grand name for a statement of the obvious.

**Lemma 1.2.3.** Suppose there are \( n \) objects and \( n - 1 \) colors, and suppose each object is colored with one of the colors. Then at least two objects get the same color.

**Proof.** Suppose for a contradiction that each color is used for at most one object. Then take the at most one object colored with each of the \( n - 1 \) colors. This is at most \( n - 1 \) objects. Thus at least one object is uncolored, a contradiction. It follows that at least one color is reused or in other words two objects receive the same color.

As a rule, this level of painstaking precision is not needed. It would suffice to note that as there are \( n \) objects and only \( n - 1 \) colors, some color is used for at least two objects.

**Question 3.** What is the largest number of objects that can be colored with \( n \) colors where each color may be used twice, but not three times?

1.2.3 Induction

This is a powerful technique for proving results that have an integer index, e.g.

\[
P(n) : \sum_{i=1}^{n} i = \frac{1}{2}n(n+1).
\]

The goal is to prove that \( P(n) \) is true for every (sensible) value of \( n \), say for integers \( n \geq 1 \). (By the way, what is the meaning of \( \sum_{i=1}^{0} i \)? This is viewed as a sum of no terms and hence equals 0.) Proving that \( P(n) \) is true for all \( n \) is done in two steps.

In the first step, called the base case, we prove the claim for \( n = 1 \), that is we prove \( P(1) \): \( \sum_{i=1}^{1} i = \frac{1}{2} \cdot 1 \cdot 2 \). But the left hand side (LHS for short) of this expression equals 1, and the right hand side (RHS) also equals 1. So the claim is true in this case.

In the next step, called the inductive step, we prove the claim for \( n = k + 1 \), assuming it is true for \( n = k \), and we do this for all integer values \( k \geq 1 \). That is, we suppose that \( P(k) \) is true, namely that \( \sum_{i=1}^{k} i = \frac{1}{2}k(k+1) \): this is called the inductive hypothesis. We now need to show that \( P(k+1) \) is true, namely that \( \sum_{i=1}^{k+1} i = \frac{1}{2}(k+1)(k+2) \). But

\[
\sum_{i=1}^{k+1} i = k + 1 + \sum_{i=1}^{k} i = (k + 1) + \frac{1}{2}k(k+1),
\]

using the inductive hypothesis (namely that \( \sum_{i=1}^{k} i = \frac{1}{2}k(k+1) \)), and

\[
(k + 1) + \frac{1}{2}k(k+1) = (k+1)(1 + k/2) = (k+1)(2 + k)/2 = \frac{1}{2}(k+1)(k+2),
\]

which shows that \( \sum_{i=1}^{k+1} i = \frac{1}{2}(k+1)(k+2) \), as desired.
We claim this shows that \( P(n) \) holds for all values of \( n \geq 1 \). Why is this? Well suppose that \( P(n) \) did not hold for all values of \( n \). For example, say it did not hold for \( n = 5 \). But we have shown that \( P(1) \) is true, so \( P(n) \) holds for \( n = 1 \). We have also shown that \( P(1) \Rightarrow P(2) \) (since we have shown that \( P(n) \Rightarrow P(n + 1) \) for every \( n \geq 1 \)), and as \( P(1) \) is true it follows that \( P(2) \) is also true. We have further shown that \( P(2) \Rightarrow P(3) \), so \( P(3) \) must also be true. As we have shown that \( P(3) \Rightarrow P(4) \) and \( P(4) \Rightarrow P(5) \) it follows that \( P(4) \) and \( P(5) \) are also true. So the result does hold for \( P(5) \).

More generally, we have shown \( P(1) \) and \( P(1) \Rightarrow P(2) \Rightarrow P(3) \Rightarrow \cdots \Rightarrow P(n) \), from which we can conclude that \( P(n) \) is true for any given integer \( n \geq 1 \).

Often, we will simple state the result without specifying the induction index. The above result would be written as \( \sum_{i=1}^{n} i = \frac{1}{2} n(n + 1) \) with the label \( P(n) \) omitted. The statement is implicitly understood as holding for all sensible \( n \) (in this case, integers greater than or equal to 1).

The General Structure of a Proof by Induction

What is an effective way of presenting a proof by induction? I recommend the following approach.

First, you state the indexed result that you are going to show, including the range of integers for which it will be shown to hold; e.g.

We show \( P(n) \) is true for \( n \geq c \).

Note that \( c \) is an integer.

Second, you state and prove the result for the base case:

**Base Case: \( n = c \) (or in more detail: we show \( P(c) \)).**

Then prove that \( P(c) \) is true.

Sometimes you will need to have several base cases, e.g. you need to show both \( P(c) \) and \( P(c + 1) \).

Third, you state and show the inductive assertion.

**Inductive Step:** We show, for all integers \( k \geq c \), that if \( P(k) \) is true, then so is \( P(k + 1) \), or for short \( P(k) \Rightarrow P(k + 1) \).

Then give the proof.

Finally, you state the result.

**Conclusion:** It follows that \( P(n) \) is true for all \( n \geq c \).

**Question 4.** Prove that \( Q(n) \) is true for all \( n \geq 1 \), where \( Q(n) : \sum_{i=1}^{n} i(i + 1) = \frac{1}{3} n(n + 1)(n + 2) \).
A More General Form of Induction — Strong Induction

Consider the following problem: Let $T$ be an $n$-node binary tree, in which every internal node has two children. Let $l$ denote the number of leaves in $T$ and $t$ the number of internal nodes. Note that $n = l + t$. Let $P(n)$ be the assertion that $l = t + 1$.

We would like to use a proof by induction to prove this result. To do this, we need a stronger assumption in the inductive step, namely we seek to prove that:

If $P(h)$ is true for all $h \leq k$ then $P(k + 1)$ is also true.

Notice that this is fully consistent with the previous approach, for to prove $P(k)$ is true we needed that $P(k - 1)$ be true, and to prove $P(k - 1)$ is true we needed that $P(k - 2)$ be true, and so forth. So in fact if we assume $P(k)$ is true we might as well assume all of $P(c)$, $P(c + 1), \ldots, P(k)$ are true, where $c$ is the base case value.

Now back to the problem. We prove the result by induction on $n$.

**Base case.** $n = 1$.

Here $T$ is a tree with a single node. $T$ has one leaf ($l = 1$) and no internal nodes ($t = 0$), so in this case $l = t + 1$.

**Inductive step.** We show, for $k \geq 1$, that if $P(h)$ is true for all $h \leq k$ then $P(k + 1)$ is also true.

Let $T$ be a tree of $k + 1$ nodes. Then $T$’s root must have two children (as it cannot have zero children).

So $T$ has the form shown in Figure 1.5, where $r$ is the root and $A$ and $B$ are the subtrees rooted at $r$’s children. Let $A$ have $h_1$ nodes and $B$ have $h_2$ nodes. So $k + 1 = h_1 + h_2 + 1$. Hence $1 \leq h_1 \leq k$ and $1 \leq h_2 \leq k$. Let $l_1$ and $t_1$, and $l_2$ and $t_2$ be the numbers of the leaves and internal nodes in $A$ and $B$ respectively. By the inductive hypothesis applied to $A$, $l_1 = t_1 + 1$. And by the inductive hypothesis applied to $B$, $l_2 = t_2 + 1$. Now $l = l_1 + l_2$ and $t = t_1 + t_2 + 1$ (remember to count $T$’s root). But $l_1 + l_2 = (t_1 + 1) + (t_2 + 1) = (t_1 + t_2 + 1) + 1$, so $l = t + 1$ as desired. This shows that $P(k + 1)$ is true, assuming $P(h)$ is true for $1 \leq h \leq k$.

**Conclusion.** $P(n)$ is true for all $n \geq 1$, that is if $T$ is an $n$-node binary tree with $l$ leaves, $t$ 2-child internal nodes and no 1-child internal nodes, then $l = t + 1$. But as this is true for all $n$, we have shown that if $T$ is a binary tree with $l$ leaves, $t$ 2-child internal nodes and no 1-child internal nodes, then $l = t + 1$.

Note that $n = l + t = 2l - 1 = 2t + 1$.

**Structured Induction** Often, when using strong induction, the value of $n$ is not significant; rather, what matters is that the size of the sub-objects to which the inductive
hypothesis is being applied are smaller than the size \((n)\) of the original object. In fact, this is the case for the above problems on binary trees. This type of induction is often called structural induction.

Recursion and Induction

Recursion is simply an implementation of induction. Recall the Tower of Hanoi problem.

**Input.** \(n\) rings \(r_1, r_2, \ldots, r_n\), where \(r_i\) has radius \(i\), and 3 posts named \(A\), \(B\), and \(C\).

**Task.** Suppose the rings are all initially on post \(A\) with larger radii rings beneath smaller radii ones, so from bottom to top they are in the order \(r_n, r_{n-1}, \ldots, r_1\). The task is to move all the rings from post \(A\) to post \(B\) using post \(C\) to help as needed.

There are two rules:

1. Only a ring at the top of a post may be moved.

2. A ring when moved may be placed only on either a larger radius ring or an empty post.

The algorithm \(\text{ToH}\) to move the \(n\) rings is shown below.

\[
\text{ToH}(n, A, B, C)
\]

\[
\text{if } n = 1 \text{ then move ring 1 from Post } A \text{ to Post } B
\]

\[
\text{else do}
\]

\[
\text{ToH}(n - 1, A, C, B) (* \text{ moves the top } n - 1 \text{ rings from Post } A \text{ to Post } C *)
\]

\[
\text{move ring } n \text{ from Post } A \text{ to Post } B
\]

\[
\text{ToH}(n - 1, C, B, A) (* \text{ moves the } n - 1 \text{ rings on Post } C \text{ to Post } B *)
\]

\[
\text{end}
\]

We can argue that \(\text{ToH}\) moves the rings correctly using a proof by induction on \(n\), the number of rings.

To do this we need the correct inductive hypothesis, which is the following.

Suppose there is a legal arrangement of \(n + m\) rings on the 3 posts (legal in the sense that smaller rings are always on top of larger ones), with the \(n\) smallest rings on post \(A\). Then \(\text{ToH}(n, A, B, C)\) correctly moves the \(n\) smallest rings from Post \(A\) to Post \(B\).

We call this inductive hypothesis \(P(n)\). Note that for each fixed value of \(n\), \(P(n)\) holds for all possible values of \(m\) \((m = 0, 1, 2, \ldots)\).

**Comment.** The \(m\) rings play no role in the argument. However, in subproblems we have to account for the rings that do not move, and this is why the additional rings are present in the statement of the inductive hypothesis.
1.2. PROOFS

Base case. $n = 1$.
Here the algorithm correctly moves the smallest ring from Post $A$ to Post $B$. Regardless of
the positions of the other rings this is always a legal move.

Inductive step. Suppose that $\text{ToH}$ works correctly for $n = k$, for any $m$. We show that it
also works correctly for $n = k + 1$, for all $m$.

By the inductive hypothesis, the first recursive call, $\text{ToH}(k, A, C, B)$, correctly moves
the $k$ smallest rings from Post $A$ to Post $C$. So at this point ring $k + 1$, the $(k + 1)$th smallest
ring, is at the top of Post $A$, and on Post $B$ there are only larger radius rings. Therefore
the subsequent action, the move of ring $k + 1$ from Post $A$ to Post $B$ is legal. Finally, by
the inductive hypothesis, the second recursive call, $\text{ToH}(k, C, B, A)$, correctly moves the $k$
smallest rings from Post $C$ to Post $B$.

Conclusion. This shows $\text{ToH}(k + 1, A, B, C)$ correctly moves the $k + 1$ smallest rings from
Post $A$ to Post $B$.

1.2.4 Zero Knowledge Proofs

This section is included so as to demonstrate that arguments can be compelling and incon-
trovertible, without falling into one of the categories you have likely encountered before.

We illustrate this proof style by means of two examples. We begin with a very simple
problem. Suppose there are two envelopes, labeled $E_A$ and $E_B$. Each envelope holds a card.
$A$ labels the back of the card in $E_A$, $B$ the back of the card in $E_B$. Their whose fronts are
numbered either 0 or 1; these numbers are called the cards’ numbers. Suppose that a prover,
Pam, knows the cards have the same number and that she wants to prove this to Chris, a
checker.

What Chris sees:

Step 1.1 \[ \begin{array}{c} A \\ H \end{array} \quad \text{Step 2.1} \quad \begin{array}{c} B \\ H \end{array} \]

Step 1.2 \[ \begin{array}{c} 0 \\ 1 \end{array} \quad \text{Step 2.2} \quad \begin{array}{c} 0 \\ 1 \end{array} \]

Actual values:

Either: \[ \begin{array}{c} E_A \quad 0 \\ E_B \quad 0 \end{array} \]

Or: \[ \begin{array}{c} E_A \quad 1 \\ E_B \quad 1 \end{array} \]

Which is it? Cannot tell.

Figure 1.6: The Zero Knowledge Proof for Equal Values.
Pam proceeds as follows. She needs one additional card, whose back is labeled H (for Helper), and whose number will be whichever of 0 and 1 is not used by Cards A and B; but card H’s number will not be revealed. First, Pam puts Card H in envelope $E_A$. Then she removes the cards from $E_A$ and shows their fronts to Chris; he sees that one front has the number 0 and other has the number 1. Next, after shuffling, their backs are shown to Chris and he sees that they are Cards A and H. Next, Pam moves card H to envelope $E_B$ taking care not to show its front. Again, she removes the cards from $E_B$ and shows their fronts to Chris; once more, he sees one front has the number 0 and other has the number 1. Again, after shuffling, Pam shows their backs to Chris and he sees that they are Cards B and H. Chris can conclude that Cards A and B have the same number, for their numbers are both unequal to the number on Card H. However, he has gained no information regarding their actual number. For what has he seen? A pair comprising one card numbered 0 and one card numbered 1, twice. He would see this whether Cards A and B are both numbered 0 or both numbered 1. In other words, he has learned one bit of information: the fact that Cards A and B have the same number, and nothing more.

**Comment.** This procedure is called a *zero-knowledge proof*. A more precise term might be *minimal knowledge*, but the former is the standard name.

Next, we turn to a more substantial problem, namely graph coloring. The input is an undirected graph $G$. The task is to determine if the vertices of $G$ can be colored using 3 colors (Red, Green, and Blue, say) in such a way that no two adjacent vertices have the same color. Note that the output is simply one of the answers “Yes” or “No.” For example the triangle (see Figure 1.7a) can be 3-colored but the complete graph on 4 vertices (see Figure 1.7b) cannot be.

Now suppose that for a given $n$ vertex graph $G$ a *prover*, Pam, happens to know a 3-coloring ($n$ is a million, say). Suppose Pam wants to prove to Chris, a *checker*, that $G$ is indeed 3-colorable. She can use the following method, which does not reveal the 3-coloring of $G$, yet is still incontrovertible.

Suppose that $G$ has $n$ vertices and $m$ edges. Pam’s proof of 3-colorability uses $n + m$ physical cards, one card for each vertex and one card for each edge. On the back of each card, Pam writes the name of the corresponding edge or vertex. On the other side, if the card is for a vertex $u$, Pam writes $u$’s color in the coloring she knows, and if for an edge $(u, v)$ she writes the third color, the one not used by either of the vertices $u$ or $v$.

Next, Pam will demonstrate, for each edge $(u, v)$ in $G$, that $u$ and $v$ have been given distinct colors, but without revealing those colors. To do this, Pam simply takes the three
cards for $u$, $v$ and $(u,v)$, shows Chris the backs, with the names $u$, $v$, $(u,v)$ on them, then shuffles the three cards, turns them over and shows Chris the resulting 3 colors R,B,G, then takes the cards back, resuffles them, and turns them back side up once more. As a result, Chris knows that $u$ and $v$ have distinct colors, but does not know anything beyond this about the colors.

Once this has been done for every edge Chris knows that only 3 colors have been used to color the vertices and that every pair of adjacent vertices have distinct colors, that is, that $G$ is 3-colorable.

Yet, Chris knows nothing more. For Chris knows upfront that if the graph is 3-colorable then Pam’s demonstration of this will result in his seeing the triple R,G,B $m$ times, once for each edge, and this would be the case regardless of the details of the coloring, so long as it was a 3-coloring. So all that Chris knows is the following one bit of information: that the graph is 3-colorable.

On the other hand, if the demonstration fails (because one of the triples of cards uses just two or even one color, e.g. R,B,R) then what Chris learns is that the coloring of the graph is not a 3-coloring. However, this does not demonstrate that the graph is not 3-colorable. In sum:

**Theorem 1.2.4.** There is a (physical) procedure to demonstrate the 1-bit fact that a 3-colorable graph is 3-colorable, without revealing anything beyond this 1-bit.

## 1.3 Short Preview: Computationally Unsolvable Problems

Self-reference is well-known for creating paradoxes, as in the following conundrum:

This sentence is false.

For if the sentence is true then it is false, and if false then it is true.

Analogous difficulties arise with questions about programs, for programs (viewed as text) can be the input for other programs (e.g. compilers) including themselves (a compiler viewed as text could be the input to the same compiler viewed as a program).

In particular, this type of difficulty occurs with the *Halting Problem*, which seeks to answer the following class of questions:

Does the computation of program $P(\ )$ on input $x$ eventually terminate where $x$ is a possible input for $P$?

Note that $P$ not terminating on input $x$ might be the result of an infinite loop or an unbounded recursion.

We will prove that there is no algorithm which will answer this question correctly for every possible pair of $P$ and $x$. 
Such an algorithm, if it existed, on an input pair \((P, x)\), answers one of “Yes” or “No.” Note that an algorithm that simulates \(P\) on input \(x\) does not provide an adequate solution. For while if \(P\) does terminate on input \(x\), the simulation will eventually discover this, and then an answer of “Yes” is justified, if \(P\) does not terminate this is never known for sure, and thus neither a “Yes” nor a “No” answer is justified.

The question is not whether for some particular \(P\) and \(x\) termination can be determined, but whether there is a systematic procedure than can determine it for every possible pair of \(P\) and \(x\). As we will see, there is no such procedure.

Suppose our programs are written in binary and likewise for their inputs. (In practice they are written in the 256-character ASCII code, i.e. 8 bits per character.) Now imagine listing all strings of binary characters in lexicographic order: length 1 strings ordered alphabetically, then length 2 strings, and so forth. i.e., 0, 1, 00, 01, 10, 11, 000, 001, etc. Clearly, this listing includes all legitimate programs, as well as many other strings.

To prove our result, let us assume, in order to obtain a contradiction, that there is a 2-input program \(H\) that solves the halting problem.

Given the program for \(H\), we create a new one-input program \(D\), defined as follows:

\[
D(x) = \begin{cases} 
\text{Simulate } H(x, x); 
\text{If } H(x, x) \text{ outputs “Yes” then loop forever} \quad \text{else (* } H(x, x) \text{ outputs “No” and *)} \quad D(x) \text{ outputs “Yes”} 
\end{cases}
\]

Side note: If \(x\) is not a program, we define \(H(x, y) = “Yes.”\)

Observe that

\(D(x)\) terminates \(\iff\) \(H(x, x)\) outputs “No” \(\iff\) \(x(x)\) does not terminate.

Setting \(x = D\) yields

\(D(D)\) terminates \(\iff\) \(D(D)\) does not terminate,

which is the contradiction we seek.

This contradiction shows that the assumption that \(H\) exists must be false.

**Comment.** We have actually shown that there is no program \(H\) that can answer the question of “does program \(P\) on input \(P\) eventually halt?” for all possible programs \(P\).

Notice that the difficulties occurred when considering the self-referential \(D(D)\).

**Exercises**

1. What is the size of the empty set, \(|\emptyset|\)?

2. Show that if \(A \subseteq B\) then \(A \cap \overline{B} = \emptyset\).

   **Hint.** One way to show that \(A \cap \overline{B} = \emptyset\), is to show the following two results: (i) if \(x \in A\) then \(x \notin \overline{B}\) and (ii) if \(x \in \overline{B}\) then \(x \notin A\).
3. Show that if $|A| = n$, then $|\text{PowerSet}(A)| = 2^n$.

4. Show that $(u^R)^R = u$.
   
   Hint. Use a proof by induction on the length of $u$.

5. Show that:

   i. $(x \lor y) \land z = (x \land z) \lor (y \land z)$.

   **Sample solution.** We have to show that for every setting of the binary variables $x, y, z$ the LHS and RHS of the equation evaluate to the same truth value. We proceed as follows.

   If $z = \text{TRUE}$, then the LHS equals $(x \lor y) \land \text{TRUE} = x \lor y$. The RHS equals $(x \land \text{TRUE}) \lor (y \land \text{TRUE}) = x \lor y$. These are equal.

   While if $z = \text{FALSE}$, the LHS equals $(x \lor y) \land \text{FALSE} = \text{FALSE}$, and the RHS equals $(x \land \text{FALSE}) \lor (y \land \text{FALSE}) = \text{FALSE} \lor \text{FALSE} = \text{FALSE}$. Again, the LHS and RHS are equal.

   ii. $(x \land y) \lor z = (x \lor z) \land (y \lor z)$.

   iii. $\neg(x \land y) = (\neg x \lor \neg y)$.

   iv. $\neg(x \lor y) = (\neg x \land \neg y)$.

6. Show that:

   i. $(A \cup B) \cap C = (A \cap C) \cup (B \cap C)$.

   **Sample solution.** This result follows by showing both that $(A \cup B) \cap C \subseteq (A \cap C) \cup (B \cap C)$ and that $(A \cap C) \cup (B \cap C) \subseteq (A \cup B) \cap C$. We show each in turn.

   The first result means the following: if $x \in (A \cup B) \cap C$ then $x \in (A \cap C) \cup (B \cap C)$. To see this is true we argue as follows. $x \in (A \cup B) \cap C$ means that $x \in A \cup B$ and $x \in C$. $x \in A \cup B$ means that $x \in A$ or $x \in B$ (and possibly both). If $x \in A$, as $x \in C$ also, $x \in A \cap C$; similarly, if $x \in B$, then $x \in B \cap C$. As $x \in A$ or $x \in B$, it follows that $x \in (A \cap C) \cup (B \cap C)$.

   The second result is shown analogously. It means the following: if $y \in (A \cap C) \cup (B \cap C)$ then $y \in (A \cup B) \cap C$. Now $y \in (A \cap C) \cup (B \cap C)$ means that $y \in A \cap C$ or $y \in B \cap C$ (or possibly both). If $y \in A \cap C$ then $y \in A$ and $y \in C$. If $y \in A$ then $y \in A \cup B$. Thus, in this case, $y \in (A \cup B) \cap C$. Similarly, if $y \in B \cap C$ it follows that $y \in (A \cup B) \cap C$. Thus in either case $y \in (A \cup B) \cap C$.

   Both subsidiary results have been shown; it follows that $(A \cup B) \cap C = (A \cap C) \cup (B \cap C)$.

   ii. $(A \cap B) \cup C = (A \cup C) \cap (B \cup C)$.

   iii. $A \cap B = \overline{A} \cup B$.

   iv. $A \cup B = \overline{A} \cap B$. 
7. Find the mistake in the following “proof.”

**Claim.** Consider a girl scout camp attended by \( n \) girls, where \( n \geq 8 \). The camp counselors want to form teams for a scavenger hunt and believe that the ideal sizes for the teams are to have 4 or 5 girls per team. One of the counselors argues as follows that this is always possible with 8 or more girls.

**Argument (Proof).** Let \( P(n) \) be the assertion that we can do this team formation with camps of \( n \) girls. We show that \( P(n) \) is true for \( n \geq 8 \) by (strong) induction.

**Base case.** If there are 8 girls we make two teams of 4 girls each. If there are 9 girls we make a team of 4 and one of 5 girls. If there are 10 girls we make two teams of 5 girls each.

**Inductive Step.** Suppose that \( P(8), P(9), \ldots, P(k) \) are true. We show that \( P(k+1) \) is also true. Proceed as follows. Form one team of 4 girls; then, by the inductive hypothesis applied to \( P(k-3) \), the remaining \( n-3 \) girls can be put into teams of 4 and 5 girls. Thus the \( k+1 \) girls can also be so divided.

What is incorrect in the above argument? Is it just the argument that is incorrect or is the claim false? A natural way to show a claim is false is to give a counterexample, an example for which the claim does not hold. Even if the claim is false, you still need to explain what aspect of the argument is incorrect; a natural way to do this is to explain where the argument fails on the counterexample.

8. Find the mistake in the following “proof.”

Consider the people in the world and suppose that if \( p \) is a friend of \( q \) then \( q \) is also a friend of \( p \). Suppose further that everyone has at least one friend (not themselves).

**Claim.** For any two distinct people \( r \) and \( s \), there is a chain of friends connecting \( r \) and \( s \), that is there is a sequence \( r = p_1, p_2, \ldots, p_k = s \), with \( p_i \) and \( p_{i+1} \) being friends for \( 1 \leq i < k \).

**Proof.** We prove this by induction on \( n \), the number of people in the world.

**Base case.** \( n = 1 \).

Then the claim is trivially true (as there cannot be two distinct people \( r \) and \( s \)).

**Second base case.** \( n = 2 \).

Let \( r \) and \( s \) be the two people in the world. By assumption, they are friends, so again the claim is trivially true.

**Inductive step.** Suppose the claim is true for worlds of \( k \) or fewer people. We prove it for the world of \( k+1 \) people.

Let \( q \) be a person, other than \( r \) or \( s \).
Consider removing $q$ from the world and for each pair $u$, $v$ of friends of $q$, making $u$ and $v$ friends in the new world of the $k$ remaining people. Clearly, for each pair of people, there is a chain connecting them in the new world if and only if there is a chain connecting them in the old world; the new chain may require using a new $u-v$ link to replace a pair of links $u-q$ and $q-v$. By induction, there is a chain between any pair of people in the new $k$-person world; consequently, there is also a chain in the old world connecting any pair of people.

What is incorrect in the above argument? Is it just the argument that is incorrect or is the claim false? A natural way to show a claim is false is to give a counterexample, an example for which the claim does not hold. Even if the claim is false, you still need to explain what aspect of the argument is incorrect; a natural way to do this is to explain where the argument fails on the counterexample.

9. We will call a number $r$ a square rational if it can be written in the form $r = a^2/b^2$ where $a$ and $b$ are integers. Show that 5 is not a square rational.

10. Let $G = (V, E)$ be an undirected graph. Show that if every vertex of $G$ has odd degree then $G$ has an even number of vertices. (The degree of a vertex $v$ is the number of edges having $v$ as an endpoint.)

11. i. What is the largest number of objects that can be colored with $n$ colors where each color may be used twice, but not three times?
ii. What is the smallest number of objects that cannot be colored with $n$ colors where each color may be used twice, but not three times?

12. i. Show by induction that $Q(n)$ is true for all $n \geq 1$, where $Q(n): \sum_{i=1}^{n} i(i + 1) = \frac{1}{3}n(n + 1)(n + 2)$.
ii. Show that $\sum_{i=1}^{n} i(i + 1)(i + 2) = \frac{1}{4}n(n + 1)(n + 2)(n + 3)$.
iii. Show that $\sum_{i=1}^{n} i(i + 1)(i + 2) \cdots (i + h) = \frac{1}{h+2}n(n + 1)(n + 2) \cdots (n + h + 1)$ for any $h \geq 0$.

13. Prove by induction that $\sum_{i=0}^{k} r^i = \frac{r^{k+1} - 1}{r-1}$, for $r \neq 1$.

14. Let $H_k = \sum_{i=1}^{k} \frac{1}{i}$. $H_k$ is called the $k$th harmonic number. Show by induction on $m$ that $1 + m/2 \leq H_{2^m} \leq m + 1$.

15. Show by induction that $e \left(\frac{n}{e}\right)^n \leq n!$ for all integers $n \geq 1$. You may assume that $(1 + 1/x)^x \leq e$ for all $x \geq 1$. Recall that $e \approx 2.72$ is the natural logarithm base.

Hint. Assuming the inductive hypothesis for $n = k$, what do you need to show to obtain the result for $n = k + 1$?
16. Consider the recurrence equation

\[ T(1) = 1 \]
\[ T(n) = n + \max_{1 \leq i,j < n} \{ T(i) + T(j) \} \quad n > 1. \]

Show by induction that \( T(n) \) is maximized by choosing \( i = 1, j = n - 1 \).

Comment. This suggests that the worst case for a divide-and-conquer procedure is the most unbalanced case.

17. Let \( F \) be a Boolean formula constructed from Boolean variables and the operators “and,” “or” and “not.” Prove, by structural induction, that there is an equivalent formula \( F' \) in which all the not operators are applied solely to variables. e.g. The formula \( \neg x \land \neg y \) is in the desired form, but \( \neg(x \lor y) \) is not. Finally, recall that two formulae are equivalent if for every truth setting of the Boolean variables they evaluate to the same value.

18. The game of \( \textit{Nim} \) is played as follows by two players, \( A \) and \( B \) say.

\( n \) sticks are placed in a pile.

In turn, starting with player \( A \), each player remove one or two sticks from the pile. The player to remove the last stick loses.

Show by induction on \( k \) that \( B \) can force a win if there are \( 3k + 1 \) sticks in the pile initially; in addition, show that \( A \) can force a win otherwise.

b. In \( \textit{Variant-Nim} \) each player can remove one, two, or three sticks. When can \( B \) force a win now? Justify your answer.

19. This is a Tower of Hanoi variant. Let \( R \) be a legal arrangement of the \( n \) rings. \( R[i] \) denotes the post on which ring \( i \) is sitting. Give a recursive algorithm \( \text{VAR-ToH} \) which on input \( R \) moves all the rings to Post \( C \) using legal moves. \( \text{VAR-ToH} \) takes 5 parameters: \( \text{VAR-ToH}(P, k, A, B, C) \) where \( P \) is the current arrangement of rings, and \( k \) indicates that the \( k \) smallest rings are being moved in this procedure call, with the \( k \) smallest rings ending up on post \( C \) (so the order of parameters \( A \) and \( B \) does not matter).

Hints: What is the base case? Consequently, what must the recursive call or calls do? Note that you need to update \( P \) when moving a ring.

Comment: It is very hard to solve this problem without using recursion. But so long as you remember that recursive subproblems take care of themselves, this is not a difficult problem. A recursive call can be treated as a black box; you don’t have to work out how it is carrying out its task.

20. This is another Tower of Hanoi variant. Let \( R \) and \( S \) be two legal arrangements of the \( n \) rings. \( R[i] \) denotes the post on which ring \( i \) is sitting initially. \( S[i] \) denotes the post on
which ring \( i \) is to sit at the end of the algorithm. Give a recursive algorithm \( \text{VAR2-ToH} \) which given inputs \( R \) and \( S \) moves the rings from arrangement \( R \) to arrangement \( S \) using legal moves. \( \text{VAR2-ToH} \) takes 6 parameters: \( \text{VAR2-ToH}(P, S, k, A, B, C) \), where \( P \) is the current arrangement of rings, and \( k \) indicates that the \( k \) smallest rings are being moved in this procedure call, with the \( k \) smallest rings ending up on post \( C \) (so the order of parameters \( A \) and \( B \) does not matter).

Hints: What is the base case? Consequently, what must the recursive call or calls do? Note that you need to update \( P \) when moving a ring.

21. Recall the Binary Search algorithm for testing if an item \( x \) lies in a sorted array \( A[i : k] \).

\[
\text{BS}(A, i, k, x) - \text{returns True if } x = A[j] \text{ for some } j, \quad i \leq j \leq k,
\]

and returns \text{FALSE} otherwise.

\[
\text{if } i > k \text{ then return FALSE}
\]

\[
\text{else if } i = k \text{ then}
\]

\[
\text{if } A[i] = x \text{ then return TRUE else return FALSE}
\]

\[
\text{else do}
\]

\[
\text{mid} \leftarrow \lfloor \frac{i+k}{2} \rfloor;
\]

\[
\text{if } A[mid] > x \text{ then return BS}(A, i, \text{mid } - 1);\]

\[
\text{else if } A[mid] = x \text{ then return TRUE}
\]

\[
\text{else (* } A[mid] < x *) \text{ return BS}(A, \text{mid } + 1, k);
\]

\[
\text{end (* else do *)}
\]

Prove by induction that \( \text{BS} \) reports correctly whether \( x \) lies in array \( A[i : k] \), using the following inductive hypothesis:

\[
\text{BS reports correctly on arrays of } n = \max\{0, k - i + 1\} \text{ items.}
\]

Sample solution. We use strong induction on \( n \).

Base case. When \( n = 0 \), i.e. \( i > k \), \( \text{BS} \) correctly returns \text{FALSE}.

Inductive step. Suppose that the inductive hypothesis holds for \( n \leq l \); we show it also holds for \( n = l + 1 \).

Case 1. \( A[mid] = x \), where \( mid = \lfloor \frac{i+k}{2} \rfloor \). Then \( \text{BS} \) correctly returns \text{TRUE}.

Case 2. \( A[mid] > x \). If \( x \) is present in \( A[i : k] \), as \( A \) is in sorted order, \( x \) would be in \( A[i : mid - 1] \). In this case \( \text{BS} \) recursively calls \( \text{BS}(A, i, mid - 1) \). As \( mid - 1 - i + 1 = mid - i < n \), by the inductive hypothesis, \( \text{BS} \) returns the correct answer.

Case 3. \( A[mid] < x \). The argument is completely analogous to the one for Case 2.

In every case, \( \text{BS} \) returns the correct answer, which proves the inductive hypothesis for \( n = l + 1 \).

We conclude that \( \text{BS} \) reports correctly whether \( x \) lies in array \( A[i : k] \).
22. Recall the Binary Search algorithm for locating an item \( x \) in a sorted array \( A[i : k] \), when present.

\[
\text{FBS}(A, i, k, x) \rightarrow \text{returns } j \text{ if } x = A[j] \text{ for some } j, i \leq j \leq k, \\
\text{and returns } \text{FALSE} \text{ otherwise.}
\]

\[
\begin{align*}
\text{if } i > k & \text{ then return } \text{FALSE} \\
\text{else if } i = k & \text{ then } \\
& \text{if } A[i] = x \text{ then return } i \text{ else return } \text{FALSE} \\
\text{else do} & \\
& \text{mid} \leftarrow \floor{\frac{i + k}{2}}; \\
& \text{if } A[mid] > x \text{ then return } \text{FBS}(A, i, mid - 1); \\
& \text{else if } A[mid] = x \text{ then return } mid \\
& \text{else (* A[mid] < x *) return } \text{FBS}(A, mid + 1, k); \\
\text{end (* else do *)}
\end{align*}
\]

Prove by induction that BS reports correctly: if \( x = A[j] \) for some \( j, i \leq j \leq k \), it returns \( j \), and otherwise it returns \( \text{FALSE} \), by using the following inductive hypothesis:

BS reports correctly on arrays of \( n = \max\{0, k - i + 1\} \) items.

23. Recall the Merge Sort algorithm:

\[
\text{MERGESORT}(A, i, j) \rightarrow \text{sorts the items in array } A[i : j] \\
\text{if } i \geq j \text{ then return} \\
\text{else do} \\
& \text{mid} \leftarrow \floor{\frac{i + j}{2}}; \\
& \text{MERGESORT}(A, i, mid); \\
& \text{MERGESORT}(A, mid + 1, j); \\
& \text{MERGE}(A, i, mid, j) \\
\text{end (* else do *)}
\]

Prove by induction that MergeSort sorts correctly, using the following inductive hypothesis:

MERGE方式进行排序的数组的 \( n \) 项。


24. Recall Dijkstra’s single source shortest path algorithm. The input is a directed graph \( G = (V, E) \) together with a source vertex \( s \in V \) and for each edge \( e = (i, j) \) a length \( \ell(i, j) \) which gives the length of the edge. You may assume all edge lengths are positive values. The algorithm finds, for each vertex \( v \in V \), the length of a shortest path from \( s \) to \( v \).
Dijkstra\((G, s)\)

initialize: \(D(v) \leftarrow \infty\) for all \(v \in V - \{s\}\); \(D(s) \leftarrow 0\); \(S \leftarrow \phi\);

Create empty Priority Queue \(Q\); for all \(v \in V\) \(\text{INSERT}(Q, v)\) with key \(D(v)\);

while \(Q\) not empty do
  \(v \leftarrow \text{DELETE_MIN}(Q);\) \(S \leftarrow S \cup \{v\}\);
  for each edge \((v, w)\) do
    if \(D(v) + \ell(v, w) < D(w)\) then do
      \(D(w) \leftarrow D(v) + \ell(v, w); \text{REDUCE_KEY}(Q, w, D(w))\)
    end (* then do *)
  end (* while *)

Show by induction that Dijkstra’s algorithm is correct, using the following inductive hypothesis.

After \(k\) iterations of the while loop \(S\) contains the \(k\) vertices in \(V\) nearest to \(s\) (with ties broken arbitrarily). Further, for every vertex \(v \in V\), \(D(v)\) is the length of a shortest path from \(s\) to \(v\) among the following paths: those paths all of whose vertices are in the set \(S\), except possibly for \(v\).

Recall that \(\text{INSERT}(Q, v)\) inserts item \(v\) into the Priority Queue \(Q\) where \(v\) has an associated key value, that \(\text{REDUCE_KEY}(Q, w, k)\) reduces the key value for item \(w\) in \(Q\) to \(k\) (the data structure works correctly only if \(k\) is less than or equal to the current value of \(w\)’s key), and that \(\text{DELETE_MIN}(Q)\) removes and returns the item in \(Q\) with the smallest value key.

25. Suppose you are given two envelopes each holding a card whose front is numbered either 0 or 1. Describe a zero-knowledge proof to show that two cards have distinct numbers.

26. Recall the game of Sudoku. You are faced with a \(9 \times 9\) grid of cells, which is further subdivided into nine \(3 \times 3\) grids. Some of the cells have integers in the range 1–9 written in them, while others are blank. The task is to fill in the blank cells so that each row, each column, and each of the nine \(3 \times 3\) grids contains each of the nine numbers 1–9 exactly once. Well constructed puzzles have exactly one solution.

Suppose that Pam knows a solution. Describe a zero-knowledge proof she can use to convince Chris of this fact without revealing anything else about her solution.

27. Let \(G = (V, E)\) be an undirected graph. \(I \subset V\) is an independent set if every edge \(e \in E\) has at most one endpoint in \(I\).

Suppose that Pam knows an independent set of size \(k\). Describe a zero knowledge proof she can use to convince Chris of this fact without revealing anything else about the independent set.
28. Let \( G = (V, E) \) be an undirected graph. \( C \) is a Hamiltonian Cycle of \( G \) if \( C \) goes through each vertex of \( G \) exactly once.

i. Suppose that Pam knows a Hamiltonian cycle \( C \) for graph \( G \). Why is the following procedure not a zero-knowledge proof of this fact?

The proof uses one card per edge. On its back, Pam writes the name of the corresponding edge. On the front, she writes a “Y” if the edge is in the circuit and she writes an “N” otherwise.

For each vertex \( v \) in turn, Pam gathers up the cards for the inedges (the edges into vertex \( v \)), shows the backs of these cards to Chris (so that he sees she has taken exactly these cards), she shuffles them, turns them over and thereby demonstrates that exactly one of these cards has a “Y”; she then takes the cards back, resuffles them and turns them back over. This process is repeated for the cards for the outedges from \( v \) (the edges leaving \( v \)).

Pam thereby shows Chris that there is one edge entering and one edge leaving each vertex, and consequently that there is a circuit which goes through all the vertices exactly once.

*ii. Give a correct zero-knowledge proof to show that \( G \) has a Hamiltonian cycle.

Hint. One solution uses \( n \) copies of each edge, where \( G \) has \( n \) vertices. If \( e_1, e_2, \ldots, e_n \) is a Hamiltonian cycle expressed as a path of \( n \) edges, then the proof will use the \( i \)th copy of \( e_i \) in demonstrating that there is a Hamiltonian cycle.

**Bibliographic Notes**

Problem 26 and its solution are due to Michael Rabin. Problems 19 and 20 are based on Tower of Hanoi problems developed by Alan Siegel.