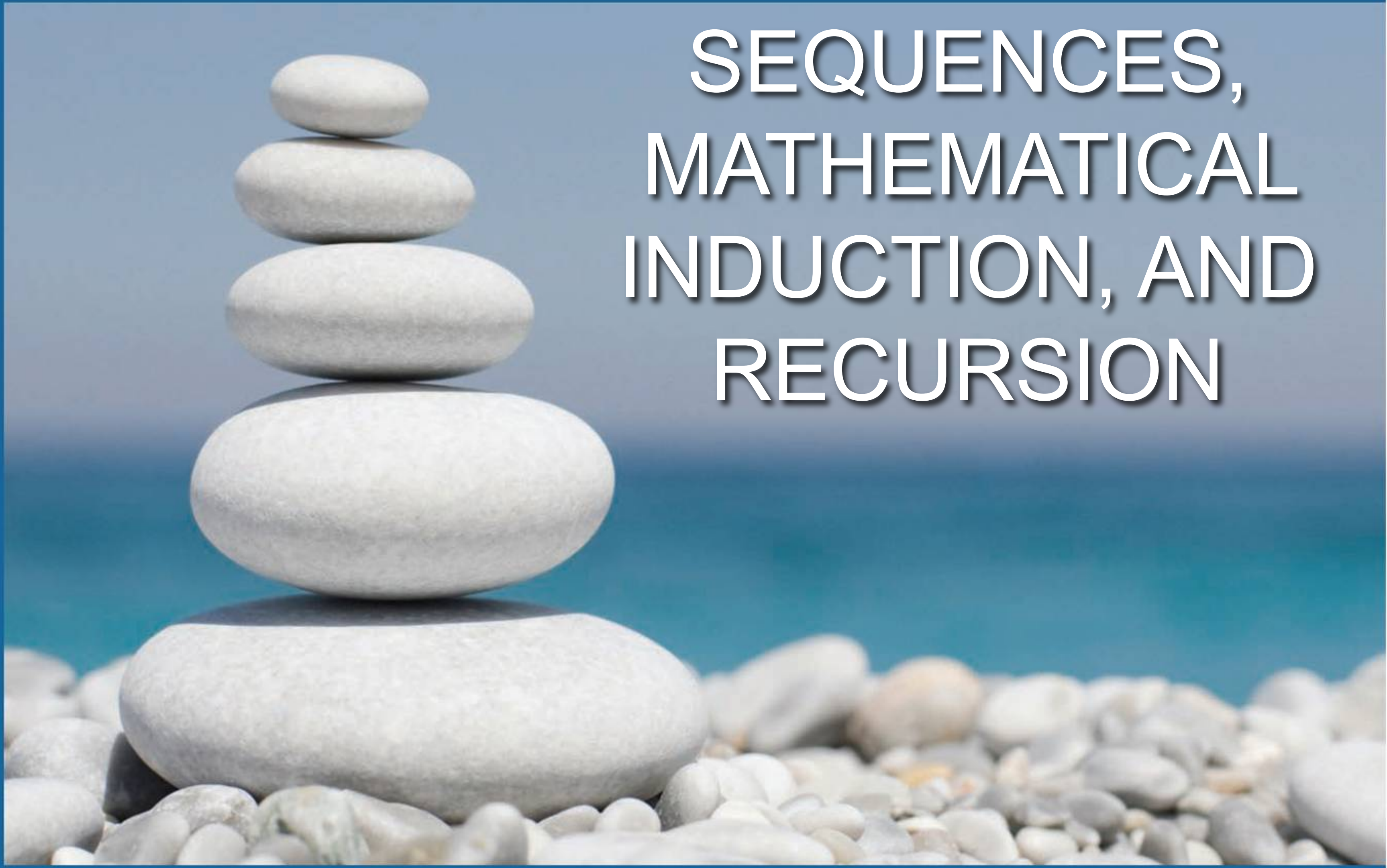
CHAPTER 5

SEQUENCES, MATHEMATICAL INDUCTION, AND RECURSION



SECTION 5.1

Sequences

Sequences

• Definition

A **sequence** is a function whose domain is either all the integers between two given integers or all the integers greater than or equal to a given integer.

We typically represent a sequence as a set of elements written in a row. In the sequence denoted

$$a_m, a_{m+1}, a_{m+2}, \ldots, a_n,$$

each individual element a_k is called a **term**.

Sequences

The notation:

$$a_m, a_{m+1}, a_{m+2}, \ldots$$

denotes an **infinite sequence**.

An **explicit formula** (or **general formula**) for a sequence is a rule that shows how the value a_k depends on k.

The following example shows that it is possible for two different explicit formulas to obtain sequences with the same terms.

Example 1 – *Finding Terms of Sequences Given by Explicit Formulas*

Define sequences $a_1, a_2, a_3, \ldots$ and $b_2, b_3, b_4, \ldots$ by the following explicit formulas:

$$a_k = \frac{k}{k+1} \quad \text{for all integers } k \geq 1,$$

$$b_i = \frac{i-1}{i} \quad \text{for all integers } i \geq 2.$$

Compute the first five terms of both sequences.

Solution:

$$a_1 = \frac{1}{1+1} = \frac{1}{2} \qquad\qquad b_2 = \frac{2-1}{2} = \frac{1}{2}$$

Example 1 – *Solution*

cont'd

$$a_2 = \frac{2}{2+1} = \frac{2}{3} \qquad b_3 = \frac{3-1}{3} = \frac{2}{3}$$

$$a_3 = \frac{3}{3+1} = \frac{3}{4} \qquad b_4 = \frac{4-1}{4} = \frac{3}{4}$$

$$a_4 = \frac{4}{4+1} = \frac{4}{5} \qquad b_5 = \frac{5-1}{5} = \frac{4}{5}$$

$$a_5 = \frac{5}{5+1} = \frac{5}{6} \qquad b_6 = \frac{6-1}{6} = \frac{5}{6}$$

As you can see, the first terms of both sequences are $\frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{4}{5}, \frac{5}{6}$; in fact, it can be shown that all terms of both sequences are identical.

Summation Notation

Summation Notation

In 1772 the French mathematician Joseph Louis Lagrange introduced the capital Greek letter sigma, Σ, to denote the word *sum* (or *summation*), and defined the summation notation as follows:

• Definition

If m and n are integers and $m \leq n$, the symbol $\sum_{k=m}^{n} a_k$, read the **summation from *k* equals *m* to *n* of *a*-sub-*k***, is the sum of all the terms $a_m, a_{m+1}, a_{m+2}, \ldots, a_n$. We say that $a_m + a_{m+1} + a_{m+2} + \ldots + a_n$ is the **expanded form** of the sum, and we write

$$\sum_{k=m}^{n} a_k = a_m + a_{m+1} + a_{m+2} + \cdots + a_n.$$

We call k the **index** of the summation, m the **lower limit** of the summation, and n the **upper limit** of the summation.

Summation Notation

Often, the terms of a summation are expressed using an explicit formula.

For instance, it is common to see summations such as

$$\sum_{k=1}^{5} k^2 \quad \text{or} \quad \sum_{i=0}^{8} \frac{(-1)^i}{i+1}.$$

Example 6 – *Changing from Summation Notation to Expanded Form*

Write the following summation in expanded form:

$$\sum_{i=0}^{n} \frac{(-1)^i}{i+1}.$$

Solution:

$$\sum_{i=0}^{n} \frac{(-1)^i}{i+1} = \frac{(-1)^0}{0+1} + \frac{(-1)^1}{1+1} + \frac{(-1)^2}{2+1} + \frac{(-1)^3}{3+1} + \cdots + \frac{(-1)^n}{n+1}$$

$$= \frac{1}{1} + \frac{(-1)}{2} + \frac{1}{3} + \frac{(-1)}{4} + \cdots + \frac{(-1)^n}{n+1}$$

$$= 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \cdots + \frac{(-1)^n}{n+1}$$

Summation Notation

A more mathematically precise definition of summation, called a *recursive definition*, is the following:

If *m* is any integer, then

$$\sum_{k=m}^{m} a_k = a_m \quad \text{and} \quad \sum_{k=m}^{n} a_k = \sum_{k=m}^{n-1} a_k + a_n \quad \text{for all integers } n > m.$$

When solving problems, it is often useful to rewrite a summation using the recursive form of the definition.

Product Notation

Product Notation

The notation for the product of a sequence of numbers is analogous to the notation for their sum. The Greek capital letter pi, $\prod$, denotes a product. For example,

$$\prod_{k=1}^{5} a_k = a_1 a_2 a_3 a_4 a_5.$$

• Definition

If m and n are integers and $m \leq n$, the symbol $\prod_{k=m}^{n} a_k$, read the **product from k equals m to n of a-sub-k**, is the product of all the terms $a_m, a_{m+1}, a_{m+2}, \ldots, a_n$. We write

$$\prod_{k=m}^{n} a_k = a_m \cdot a_{m+1} \cdot a_{m+2} \cdots a_n.$$

Product Notation

A recursive definition for the product notation is the following: If *m* is any integer, then

$$\prod_{k=m}^{m} a_k = a_m \qquad \text{and} \qquad \prod_{k=m}^{n} a_k = \left(\prod_{k=m}^{n-1} a_k\right) \cdot a_n \quad \text{for all integers } n > m.$$

Example 11 – *Computing Products*

Compute the following products:

a. $\prod_{k=1}^{5} k$

b. $\prod_{k=1}^{1} \frac{k}{k+1}$

Solution:

a. $\prod_{k=1}^{5} k = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120$

b. $\prod_{k=1}^{1} \frac{k}{k+1} = \frac{1}{1+1} = \frac{1}{2}$

Properties of Summations and Products

Properties of Summations and Products

The following theorem states general properties of summations and products.

Theorem 5.1.1

If $a_m, a_{m+1}, a_{m+2}, \ldots$ and $b_m, b_{m+1}, b_{m+2}, \ldots$ are sequences of real numbers and c is any real number, then the following equations hold for any integer $n \geq m$:

1. $$\sum_{k=m}^{n} a_k + \sum_{k=m}^{n} b_k = \sum_{k=m}^{n} (a_k + b_k)$$
2. $$c \cdot \sum_{k=m}^{n} a_k = \sum_{k=m}^{n} c \cdot a_k$$ generalized distributive law
3. $$\left(\prod_{k=m}^{n} a_k\right) \cdot \left(\prod_{k=m}^{n} b_k\right) = \prod_{k=m}^{n} (a_k \cdot b_k).$$

Factorial and “*n* Choose *r*” Notation

Factorial and "*n* Choose *r*" Notation

The product of all consecutive integers up to a given integer occurs so often in mathematics that it is given a special notation—*factorial* notation.

• Definition

For each positive integer n, the quantity ***n* factorial** denoted ***n*!,** is defined to be the product of all the integers from 1 to n:

$$n! = n \cdot (n-1) \cdots 3 \cdot 2 \cdot 1.$$

Zero factorial, denoted $0!$, is defined to be 1:

$$0! = 1.$$

Factorial and "*n* Choose *r*" Notation

A recursive definition for factorial is the following: Given any nonnegative integer *n*,

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{if } n \geq 1. \end{cases}$$

Factorial and "*n* Choose *r*" Notation

An important use for the factorial notation is in calculating values of quantities, called *n choose r*, that occur in many branches of mathematics and computer science, especially those connected with the study of counting techniques and probability.

• Definition

Let n and r be integers with $0 \leq r \leq n$. The symbol

$$\binom{n}{r}$$

is read "***n* choose *r***" and represents the number of subsets of size r that can be chosen from a set with n elements.

Observe that the definition implies that $\binom{n}{r}$ will always be an integer because it is a number of subsets.

Factorial and "*n* Choose *r*" Notation

The computational formula:

• Formula for Computing $\binom{n}{r}$

For all integers n and r with $0 \leq r \leq n$,

$$\binom{n}{r} = \frac{n!}{r!(n-r)!}.$$

The quantities $\binom{n}{r}$ are also called *combinations*. Sometimes they are referred to as *binomial coefficients* because of the connection with the binomial theorem.

Example 17 – *Computing* $\binom{n}{r}$ *by Hand*

Use the formula for computing $\binom{n}{r}$ to evaluate the following expressions:

a. $\binom{8}{5}$ **b.** $\binom{4}{0}$ **c.** $\binom{n+1}{n}$

Solution:

a. $$\binom{8}{5} = \frac{8!}{5!(8-5)!}$$

$$= \frac{8 \cdot 7 \cdot \cancel{6} \cdot \cancel{5} \cdot \cancel{4} \cdot \cancel{3 \cdot 2 \cdot 1}}{(\cancel{5} \cdot \cancel{4} \cdot \cancel{3} \cdot \cancel{2} \cdot 1) \cdot (\cancel{\cdot 3 \cdot 2 \cdot 1})}$$

always cancel common factors before multiplying

$$= 56.$$

Example 17 – *Solution*

cont' d

b. $$\binom{4}{4} = \frac{4!}{4!(4-4)!}$$

$$= \frac{4!}{4!0!}$$

$$= \frac{\cancel{4\cdot 3\cdot 2\cdot 1}}{\cancel{(4\cdot 3\cdot 2\cdot 1)}(1)}$$

$$= 1$$

The fact that 0! = 1 makes this formula computable. It gives the correct value because a set of size 4 has exactly one subset of size 4, namely itself.

c. $$\binom{n+1}{n} = \frac{(n+1)!}{n!((n+1)-n)!} = \frac{(n+1)!}{n!1!} = \frac{(n+1)\cdot \cancel{n!}}{\cancel{n!}} = n+1$$

Sequences in Computer Programming

Sequences in Computer Programming

An important data type in computer programming consists of finite sequences. In computer programming contexts, these are usually referred to as *one-dimensional arrays*.

For example, consider a program that analyzes the salaries paid to a sample of 50 workers. Such a program might compute the average salary and the difference between each individual salary and the average.

Each salary is stored in a one-dimensional array:

$$W[1], W[2], W[3], \ldots, W[50].$$

Sequences in Computer Programming

The recursive definitions for summation, product, and factorial lead naturally to computational algorithms.

For instance, we present two sets of pseudocode to find the sum of $a[1], a[2], \ldots, a[n]$.

Sequences in Computer Programming

In both cases the output is $\sum_{k=1}^{n} a[k]$.

The one on the left exactly mimics the recursive definition; the one on the right is instead an iterative algorithm.

Recursive

int funct S(m,n)
    if m=n **then return** a[m]
    else return S(m,n-1)+a[n]

Iterative

$s := 0$
for $k := 1$ **to** n
    $s := s + a[k]$
next k

Application: Algorithm to Convert from Base 10 to Base 2 Using Repeated Division by 2

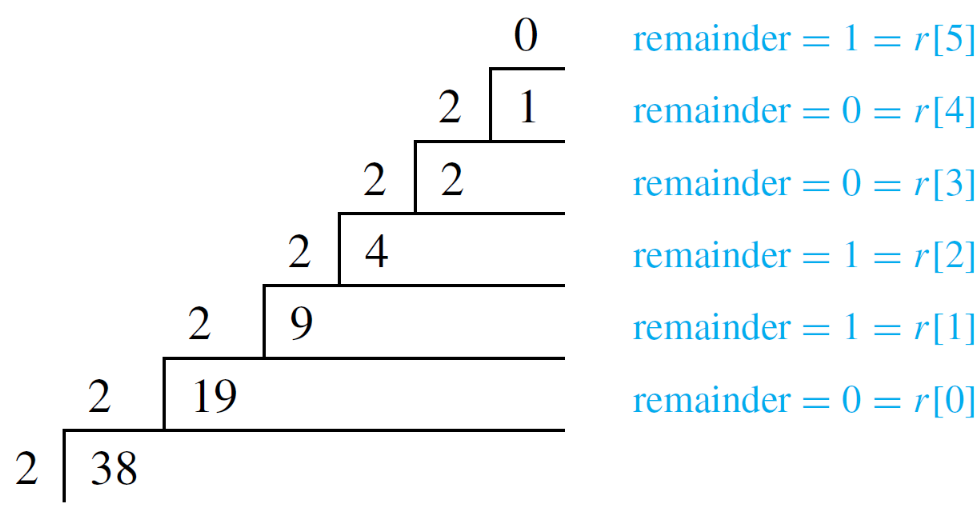
Application: Algorithm to Convert from Base 10 to Base 2 Using Repeated Division by 2

A systematic algorithm to convert any non-negative integer to binary notation uses repeated division by 2.

Suppose a is a non-negative integer. Divide a by 2 to obtain a quotient $q[0]$ and a remainder $r[0]$. If the quotient is nonzero, divide by 2 again to obtain a new quotient $q[1]$ and a new remainder $r[1]$.

Continue this process until a quotient of 0 is obtained. At each stage, the remainder must be less than the divisor, which is 2. Thus each remainder is either 0 or 1.

The process is illustrated below for *a* = 38. (Read the divisions from the bottom up.)



The results of all these divisions can be written as a sequence of equations:

$$38 = 19 \cdot 2 + 0,$$

Application: Algorithm to Convert from Base 10 to Base 2 Using Repeated Division by 2

$$
\begin{aligned}
19 &= 9 \cdot 2 + 1, \\
9 &= 4 \cdot 2 + 1, \\
4 &= 2 \cdot 2 + 0, \\
2 &= 1 \cdot 2 + 0, \\
1 &= 0 \cdot 2 + 1.
\end{aligned}
$$

By repeated substitution, then,

$$
\begin{aligned}
38 &= 19 \cdot 2 + 0 \\
&= (9 \cdot 2 + 1) \cdot 2 + 0 = 9 \cdot 2^2 + 1 \cdot 2 + 0 \\
&= (4 \cdot 2 + 1) \cdot 2^2 + 1 \cdot 2 + 0 = 4 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2 + 0 \\
&= (2 \cdot 2 + 0) \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2 + 0
\end{aligned}
$$

Application: Algorithm to Convert from Base 10 to Base 2 Using Repeated Division by 2

$$= 2 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2 + 0$$

$$= (1 \cdot 2 + 0) \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2 + 0$$

$$= 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2 + 0.$$

Note that each coefficient of a power of 2 on the right-hand side is one of the remainders obtained in the repeated division of 38 by 2.

This is true for the left-most 1 as well, because $1 = 0 \cdot 2 + 1$. Thus

$$38_{10} = 100110_2 = (r[5]r[4]r[3]r[2]r[1]r[0])_2.$$

Example 19 – *Converting from Decimal to Binary Notation Using Repeated Division by 2*

Use repeated division by 2 to write the number 29_{10} in binary notation.

Solution:

$$
\begin{array}{rrl}
 & 0 & \text{remainder} = r[4] = 1 \\
2 & \underline{\big|\ 1} & \text{remainder} = r[3] = 1 \\
2 & \underline{\big|\ 3} & \text{remainder} = r[2] = 1 \\
2 & \underline{\big|\ 7} & \text{remainder} = r[1] = 0 \\
2 & \underline{\big|\ 14} & \text{remainder} = r[0] = 1 \\
2 & \big|\ 29 &
\end{array}
$$

Hence $29_{10} = (r[4]\ r[3]\ r[2]\ r[1]\ r[0])_2 = 11101_2$.

Application: Algorithm to Convert from Base 10 to Base 2 Using Repeated Division by 2

Input: *n [a nonnegative integer]*

Algorithm Body:

$q := n$, $i := 0$

[Repeatedly perform the integer division of q by 2 *until q becomes* 0. *Store successive remainders in a one-dimensional array* $r[0], r[1], r[2], \ldots, r[k]$.

Even if the initial value of q equals 0, *the loop should execute one time* (*so that* $r[0]$ *is computed*).

Thus the guard condition for the **while** *loop is* $i = 0$ or $q \neq 0$.*]*

Application: Algorithm to Convert from Base 10 to Base 2 Using Repeated Division by 2

while ($i = 0$ or $q \neq 0$)

$r[i] := q \textit{ mod } 2$

$q := q \textit{ div } 2$

[r [i] and q can be obtained by calling the division algorithm.]

$i := i + 1$

end while

Application: Algorithm to Convert from Base 10 to Base 2 Using Repeated Division by 2

[After execution of this step, the values of $r[0], r[1], \ldots, r[i-1]$ *are all* 0's and 1's, *and* $a = (r[i-1]\,r[i-2] \ldots r[2]\; r[1] r[0])_2$.*]*

Output: $r[0], r[1], r[2], \ldots, r[i-1]$ *[a sequence of integers]*