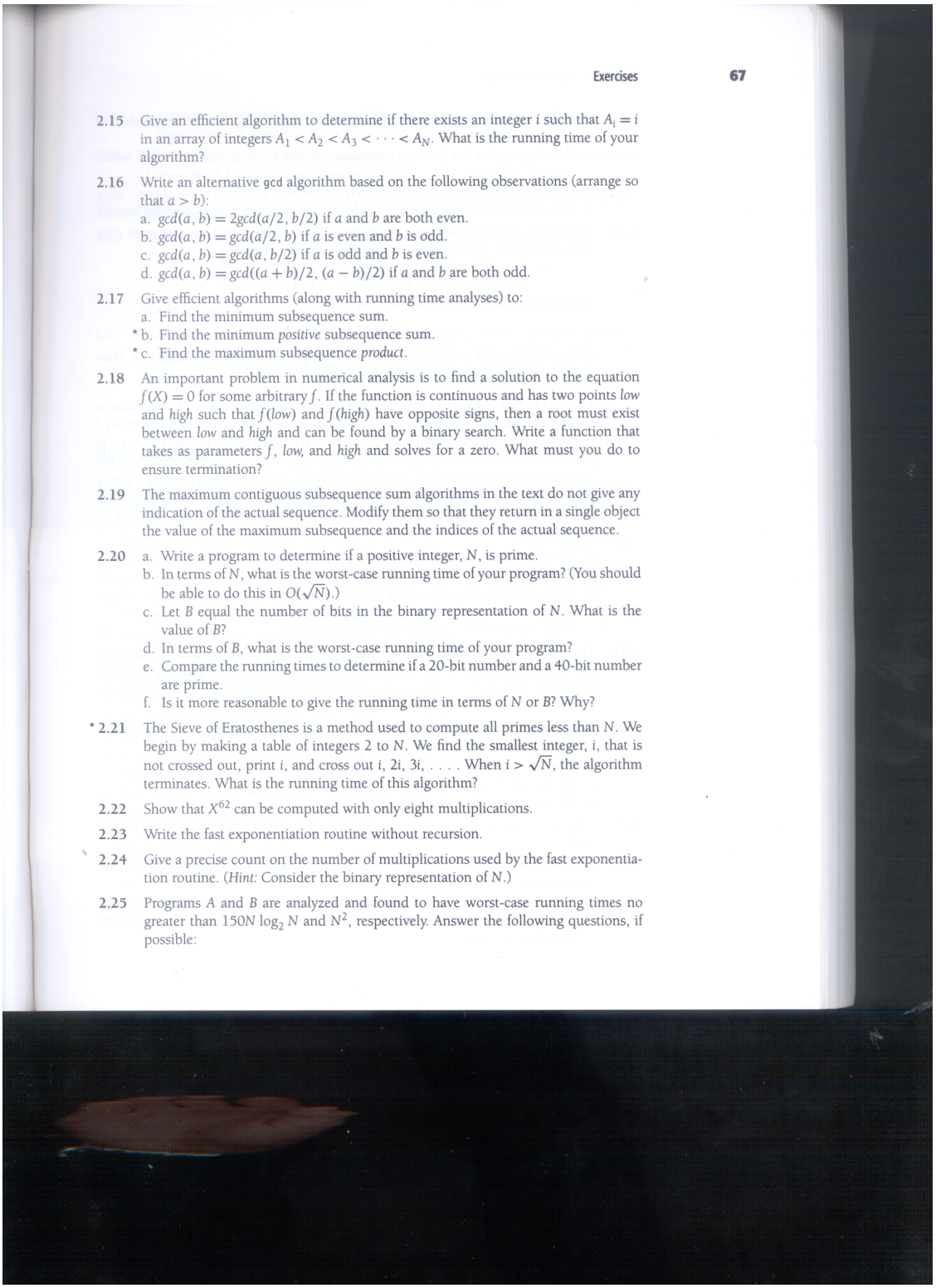
1. Do Qs 2.20, 2.22, 2.24, 2.25

2. Read all of §3 of *Weiss*



Exercises 67

2.15 Give an efficient algorithm to determine if there exists an integer i such that $A_i = i$ in an array of integers $A_1 < A_2 < A_3 < \cdots < A_N$. What is the running time of your algorithm?

2.16 Write an alternative `gcd` algorithm based on the following observations (arrange so that $a > b$):
a. $gcd(a, b) = 2gcd(a/2, b/2)$ if a and b are both even.
b. $gcd(a, b) = gcd(a/2, b)$ if a is even and b is odd.
c. $gcd(a, b) = gcd(a, b/2)$ if a is odd and b is even.
d. $gcd(a, b) = gcd((a + b)/2, (a - b)/2)$ if a and b are both odd.

2.17 Give efficient algorithms (along with running time analyses) to:
a. Find the minimum subsequence sum.
*b. Find the minimum *positive* subsequence sum.
*c. Find the maximum subsequence *product*.

2.18 An important problem in numerical analysis is to find a solution to the equation $f(X) = 0$ for some arbitrary f. If the function is continuous and has two points *low* and *high* such that $f(low)$ and $f(high)$ have opposite signs, then a root must exist between *low* and *high* and can be found by a binary search. Write a function that takes as parameters f, *low*, and *high* and solves for a zero. What must you do to ensure termination?

2.19 The maximum contiguous subsequence sum algorithms in the text do not give any indication of the actual sequence. Modify them so that they return in a single object the value of the maximum subsequence and the indices of the actual sequence.

2.20 a. Write a program to determine if a positive integer, N, is prime.
b. In terms of N, what is the worst-case running time of your program? (You should be able to do this in $O(\sqrt{N})$.)
c. Let B equal the number of bits in the binary representation of N. What is the value of B?
d. In terms of B, what is the worst-case running time of your program?
e. Compare the running times to determine if a 20-bit number and a 40-bit number are prime.
f. Is it more reasonable to give the running time in terms of N or B? Why?

*2.21 The Sieve of Eratosthenes is a method used to compute all primes less than N. We begin by making a table of integers 2 to N. We find the smallest integer, i, that is not crossed out, print i, and cross out i, $2i$, $3i$, ... When $i > \sqrt{N}$, the algorithm terminates. What is the running time of this algorithm?

2.22 Show that X^{62} can be computed with only eight multiplications.

2.23 Write the fast exponentiation routine without recursion.

2.24 Give a precise count on the number of multiplications used by the fast exponentiation routine. (*Hint:* Consider the binary representation of N.)

2.25 Programs A and B are analyzed and found to have worst-case running times no greater than $150N \log_2 N$ and N^2, respectively. Answer the following questions, if possible:

a. Which program has the better guarantee on the running time, for large values of N ($N > 10{,}000$)?
b. Which program has the better guarantee on the running time, for small values of N ($N < 100$)?
c. Which program will run faster *on average* for $N = 1{,}000$?
d. Is it possible that program B will run faster than program A on *all* possible inputs?

2.26 A majority element in an array, A, of size N is an element that appears more than $N/2$ times (thus, there is at most one). For example, the array

3, 3, 4, 2, 4, 4, 2, 4, 4

has a majority element (4), whereas the array

3, 3, 4, 2, 4, 4, 2, 4

does not. If there is no majority element, your program should indicate this. Here is a sketch of an algorithm to solve the problem:

First, a candidate majority element is found (this is the harder part). This candidate is the only element that could possibly be the majority element. The second step determines if this candidate is actually the majority. This is just a sequential search through the array. To find a candidate in the array, A, form a second array, B. Then compare A_1 and A_2. If they are equal, add one of these to B; otherwise do nothing. Then compare A_3 and A_4. Again if they are equal, add one of these to B; otherwise do nothing. Continue in this fashion until the entire array is read. Then recursively find a candidate for B; this is the candidate for A (why?).

a. How does the recursion terminate?
* b. How is the case where N is odd handled?
* c. What is the running time of the algorithm?
d. How can we avoid using an extra array B?
* e. Write a program to compute the majority element.

2.27 The input is an N by N matrix of numbers that is already in memory. Each individual row is increasing from left to right. Each individual column is increasing from top to bottom. Give an $O(N)$ worst-case algorithm that decides if a number X is in the matrix.

2.28 Design efficient algorithms that take an array of positive numbers `a`, and determine:
a. the maximum value of `a[j]+a[i]`, with `j` $\geq$ `i`.
b. the maximum value of `a[j]-a[i]`, with `j` $\geq$ `i`.
c. the maximum value of `a[j]*a[i]`, with `j` $\geq$ `i`.
d. the maximum value of `a[j]/a[i]`, with `j` $\geq$ `i`.

* **2.29** Why is it important to assume that integers in our computer model have a fixed size?

2.30 Consider the word puzzle problem described in Chapter 1. Suppose we fix the size of the longest word to be 10 characters.
a. In terms of R and C, which are the number of rows and columns in the puzzle, and W, which is the number of words, what are the running times of the algorithms described in Chapter 1?