CHAPTER: 11 - The World of Algorithms

Exercise Set 11.1 (Solutions)

1. Add two 4-digit numbers using the steps we have written down. Make sure you follow the steps precisely; do not perform any action that is not explicitly mentioned. Are you able to obtain the correct result?

Solution: Consider 4728 + 3695. The digit-by-digit process gives: units 8 + 5 = 13 (write 3, carry 1); tens 2 + 9 + 1 = 12 (write 2, carry 1); hundreds 7 + 6 + 1 = 14 (write 4, carry 1); thousands 4 + 3 + 1 = 8 (write 8). Thus 4728 + 3695 = **8423**. For this example, the stated steps produce the correct answer. Strictly, the wording of the algorithm should also specify what to do when a column sum equals 10; the book describes less than 10 and more than 10, but omits equality.

2. What happens if you add a 5-digit number to a 3-digit number. Do our steps handle

this situation correctly?

Solution: For example, 42536 + 178 = **42714**. Align the ones places. When the shorter number runs out of digits, treat each missing digit as zero and continue the carry. The stated algorithm does not explicitly cover missing digits, so it needs this additional rule to handle such inputs fully.

3. Why is it important to align the columns from right to left?

Solution: The rightmost digits represent units. Aligning from the right places units under units, tens under tens and hundreds under hundreds. This ensures that the values being added have the same place value and that carries go into the correct next column.

4. In Step 3, why cannot the value of carry be more than 1?

Solution: Two single-digit numbers can contribute at most 9 + 9, and the previous carry is at most 1. The largest column total is therefore 9 + 9 + 1 = 19. Only one ten can be carried to the next column, so the carry is 0 or 1.

5. What happens if we do not include the fifth step in the algorithm above? Give examples where the algorithm will work correctly and where it will fail to work.

Solution: If the final (leftmost) column produces no carry, omission of Step 5 causes no problem. Example: 123 + 456 = **579**. But when a carry remains beyond the leftmost column, omitting Step 5 loses that leading 1. Example: 999 + 1 = **1000**, but without

Step 5 the recorded digits would read only 000.

Think and Reflect - Adding Numbers

1. See if you can complete the argument about grouping by units, tens, hundreds, ... to justify why the addition algorithm works.

Solution: Every whole number can be decomposed into units, tens, hundreds and other powers of ten. The algorithm adds equal place-value groups. Whenever a column contributes ten units of its place value, it is regrouped as one unit of the next place value; this is the carry operation. Since regrouping does not change the total, the algorithm preserves the sum.

2. How would you modify the algorithm to add two decimal fractions?

Solution: Align the decimal points (so that corresponding place values match), add trailing or leading zeros where needed, and add from right to left using the same carry rules. Put the decimal point in the answer directly under those of the addends.

Example: 4.70 + 12.85 = **17.55**.

Think and Reflect - Computing Divisors

Try to execute the algorithm to compute the divisors of 15, 135, and 775. How does the amount of work increase as the numbers grow?

Solution: The divisors are shown below. The textbook algorithm tests every integer from 1 through n, so it performs respectively 15, 135 and 775 divisibility checks. The work grows in proportion to the value n, not simply the number of digits.

n	Divisors of n	Number of checks
15	1, 3, 5, 15	15
135	1, 3, 5, 9, 15, 27, 45, 135	135
775	1, 5, 25, 31, 155, 775	775

Exercise Set 11.2 (Solutions)

1. Suppose List 1 and List 2 are two lists of numbers in increasing order.

(i) Write an algorithm to find elements in List 1 that are not present in List 2.

(ii) Write an algorithm to find elements in List 2 that are not present in List 1.

Solution: (i) Create an empty output list. For each element x of List 1, test whether x occurs in List 2; append x only if it does not. **(ii)** Repeat with the roles of List 1 and List 2 interchanged. Since the lists are sorted, we can also move through both lists using two indices, advancing the smaller value and recording it only when absent from the other list.

2. Describe an algorithm to compute the least common multiple (lcm) of two numbers.

Solution: For positive inputs m and n, first compute g = gcd(m, n) using a gcd algorithm. Then return **lcm(m, n) = mn / g**. For instance, gcd(18,24) = 6, so lcm(18,24) = 18 × 24 / 6 = **72**. Equivalently, one can generate successive multiples of both numbers and take the least common positive one.

***3. Divisors occur in pairs. For instance, the divisors of 18 are (1, 18), (2, 9) and (3, 6).**

(i) If we write out divisors in pairs, how many numbers do we have to examine between 1 and n to find all the divisors of n?

(ii) If we list out the divisors in pairs, will our gcd algorithm still work in the manner we have described?

Solution: (i) We only need to test the integers j from 1 through $\lfloor\sqrt{n}\rfloor$. If j divides n, the paired divisor n/j is known at once; a divisor that equals √n is counted only once. **(ii) Not exactly in the way described.** Listing the divisors in pairs gives the same divisors, so the common divisors, and the gcd (the largest of them), do not change. But the lists are no longer in increasing order (for 18 they would read 1, 18, 2, 9, 3, 6), so the step “report the rightmost element” can give a wrong answer. We must either sort the lists first or choose the largest common divisor explicitly.

Not without adjustment. The original gcd algorithm assumes divisor lists are in increasing order and reports the rightmost common divisor. Pair-generated lists need not be sorted. We must sort the lists or explicitly choose the greatest value among their common divisors.

Exercise Set 11.3 (Solutions)

1. How would our original algorithm change if we computed the divisors of n by examining the numbers from 1 to n in reverse order, from n down to 1?

Solution: The divisors are then listed in decreasing order. The largest common divisor would appear first rather than last in the list of common divisors. Thus the final selection step should report the **leftmost** common divisor, or compute the maximum explicitly.

2. What about the last algorithm described above? What happens when we look at common divisors starting from min(m, n) and work backwards to 1?

Solution: Starting from min(m,n) and descending, the first integer that divides both m and n is automatically their greatest common divisor. We can return this value immediately without storing lists or checking the remaining smaller integers. In the worst case, when gcd(m,n) = 1, we may still need to test all candidates down to 1.

End-of-Chapter Exercises (Solutions)

1. Compute the following using the improved version of Euclid's algorithm.

(i) gcd(375, 825)

(ii) gcd(51000, 81000)

(iii) gcd(1789287, 237656)

(iv) gcd(2587392, 157656)

Solution: Use the remainder rule gcd(a, b) = gcd(b, a mod b) repeatedly. The last nonzero remainder is the answer.

(i) 825 = 2 × 375 + 75; 375 = 5 × 75 + 0. Thus **gcd = 75**.

(ii) 81000 = 1 × 51000 + 30000; 51000 = 1 × 30000 + 21000; 30000 = 1 × 21000 + 9000; 21000 = 2 × 9000 + 3000; 9000 = 3 × 3000 + 0. Thus **gcd = 3000**.

(iii) 1789287 = 7 × 237656 + 125695; 237656 = 1 × 125695 + 111961; 125695 = 1 × 111961 + 13734; 111961 = 8 × 13734 + 2089; 13734 = 6 × 2089 + 1200; 2089 = 1 × 1200 + 889; 1200 = 1 × 889 + 311; 889 = 2 × 311 + 267; 311 = 1 × 267 + 44; 267 = 6 × 44 + 3; 44 = 14 × 3 + 2; 3 = 1 × 2 + 1; 2 = 2 × 1 + 0. Thus **gcd = 1**.

(iv) 2587392 = 16 × 157656 + 64896; 157656 = 2 × 64896 + 27864; 64896 = 2 × 27864 + 9168; 27864 = 3 × 9168 + 360; 9168 = 25 × 360 + 168; 360 = 2 × 168 + 24;

168 = 7 × 24 + 0. Thus **gcd = 24**.

2. Assume that m ≥ n. Verify that d divides m and n if and only if d divides both n and m mod n.

Solution: Write m = qn + r, where r = m mod n. If d divides both m and n, then d divides m − qn = r. Conversely, if d divides n and r, it divides qn + r = m. Hence each condition implies the other, and the sets of common divisors are identical.

3. Write an algorithm prime(n) to check if n is prime.

(Hint: A prime number p has exactly two distinct factors, 1 and p. Can you make use of divisors(n) to write out prime(n)?)

Solution: Algorithm prime(n): (1) If n < 2, return False. (2) Let D = divisors(n). (3) If D has exactly two elements, 1 and n, return True; otherwise return False. For example divisors(7) = [1,7], so 7 is prime, while divisors(8) = [1,2,4,8], so 8 is not prime.

***4. Write an algorithm primedivisors(n) to compute the list of divisors of n that are prime numbers.**

(Hint: Compute divisors(n) and then filter out the primes in this list.)

Solution: First compute D = divisors(n). Start with an empty list P. For every d in D, test prime(d). If true, append d to P. Return P after all divisors have been checked. For example, divisors(18) = [1,2,3,6,9,18] and primedivisors(18) = **[2,3]**.

***5. We can also find the gcd of two numbers by computing prime factorisation of both the numbers. Try to write an algorithm to compute the prime factorisation of a number.**

(i) The prime factorisation of 180 is 22 × 32 × 51. How would you represent this?

(ii) How would you compare the prime factorisations of two numbers?

Solution: Repeatedly divide the current number by a prime factor p, recording how many times it divides. Continue with the next prime until the remaining number becomes 1. **(i)** Represent 180 as a list of (prime, exponent) pairs: **[(2,2), (3,2), (5,1)]**.

(ii) Align the entries for each prime, inserting exponent 0 where a prime is absent. For gcd take the smaller of each pair of exponents; for lcm take the larger. For example, $180 = 2^2 \times 3^2 \times 5$ and $84 = 2^2 \times 3 \times 7$, so gcd(180,84) = $2^2 \times 3$ = **12**.