Chapter 11: The World of Algorithms

CONTENTS 4 exercises · 15 questions

11.1	Exercise Set 11.1	5 Q
11.2	Exercise Set 11.2	3 Q
11.3	Exercise Set 11.3	2 Q
Exercises	End-of-Chapter Exercises	5 Q

Exercise 11.1 — Exercise Set 11.1

Q1 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

Let us add 2,473 and 3,695 using the algorithm step by step.

Step 1: Write the numbers one below the other, aligning digits from right to left.

2 4 7 3
+ 3 6 9 5

Step 2: Add the rightmost digits (units column): 3 + 5 = 8. Since 8 < 10, write 8 below. Set carry = 0.

Step 3 (tens column): Add 7 + 9 + carry (0) = 16. Since 16 ≥ 10, write units digit 6 below, set carry = 1.

Step 3 (hundreds column): Add 4 + 6 + carry (1) = 11. Since 11 ≥ 10, write units digit 1 below, set carry = 1.

Step 3 (thousands column): Add 2 + 3 + carry (1) = 6. Since 6 < 10, write 6 below, set carry = 0.

Step 4: No more digits on the left.

Step 5: Carry = 0, so nothing extra to write.

Result: 2473 + 3695 = 6168. Yes, following the steps precisely gives the correct result.

The algorithm works correctly for any two 4-digit numbers, producing the correct sum by processing one column at a time from right to left.

Q2 What happens if you add a 5-digit number to a 3-digit number. Do our steps handle this situation correctly?

SOLUTION

Consider adding 12345 (5-digit) and 678 (3-digit).

Step 1 aligns the numbers from right to left:

```
  1 2 3 4 5
+     6 7 8
```

The 3-digit number has no digits in the hundreds-thousands and ten-thousands columns — these are treated as 0.

The algorithm as written says: in Step 3, 'add the two digits and the current value of carry'. When one number has no digit in a column, we treat that missing digit as 0.

However, the algorithm as stated does not explicitly say what to do when one number runs out of digits. It assumes both numbers have the same number of digits.

In practice the algorithm still works correctly IF we treat missing digits as 0: Steps 2 and 3 proceed normally for the 3 rightmost columns, and then for the remaining 2 columns of the 5-digit number we add 0 (from the 3-digit number) + the digit + carry.

The steps handle this situation correctly in spirit, but the written algorithm needs a small clarification: when one number has fewer digits, treat its missing (left) digits as 0 and continue applying Step 3 until all digits of both numbers have been processed.

Q3 Why is it important to align the columns from right to left?

SOLUTION

In the Indian (decimal) place-value system, the position of a digit determines its value: the rightmost digit represents units, the next represents tens, then hundreds, and so on.

When we add two numbers, we must add digits that represent the same place value together — units with units, tens with tens, hundreds with hundreds, etc.

Aligning the columns from right to left ensures that the digits in each column carry the same place value. For example, if we add 473 and 695, the 3 and 5 are both units, the 7 and 9 are both tens, and the 4 and 6 are both hundreds.

If we did not align from the right — say, we aligned from the left — a 3-digit number and a 5-digit number would have their units digits in different columns. We would then be adding units to hundreds or thousands, which gives a wrong answer.

Right-alignment guarantees that we are always adding place-value-equivalent digits, which is the fundamental reason the algorithm produces the correct sum.

Q4 In Step 3, why cannot the value of carry be more than 1?

SOLUTION

In Step 3, we add two single digits (each between 0 and 9) plus the current carry (which is 0 or 1).

The maximum possible value of this sum is: 9 + 9 + 1 = 19.

The minimum possible value is: 0 + 0 + 0 = 0.

So the sum in any column lies between 0 and 19.

The carry is set to 1 if the sum is ≥ 10 (i.e., the sum contributes a 'group of ten' to the next column), and to 0 otherwise.

Since the maximum sum is 19, which is less than 20, the sum can never contribute two groups of ten to the next column. Therefore the carry can never be 2 or more — it is always either 0 or 1.

This is why the carry in the addition algorithm is always at most 1.

Q5 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

Step 5 says: 'If the value of carry is 1, write 1 to the left of the bottom row.'

This step handles the situation where the final addition (in the leftmost column) produces a sum of 10 or more, generating a carry that needs to be recorded as an extra leading digit.

Case 1 — Algorithm works correctly WITHOUT Step 5:
Add 473 + 526 = 999.
Units: 3 + 6 = 9, carry = 0.
Tens: 7 + 2 + 0 = 9, carry = 0.
Hundreds: 4 + 5 + 0 = 9, carry = 0.
Final carry = 0, so Step 5 does nothing. Result is 999. Correct.

Case 2 — Algorithm FAILS without Step 5:
Add 473 + 695 = 1168.
Units: 3 + 5 = 8, carry = 0.
Tens: 7 + 9 + 0 = 16, write 6, carry = 1.
Hundreds: 4 + 6 + 1 = 11, write 1, carry = 1.
Without Step 5, we stop here and write only 168, missing the leading 1.
The correct answer is 1168, but the algorithm (without Step 5) gives 168. This is wrong.

Conclusion: The algorithm works correctly when the sum of the two numbers has the same number of digits as each addend (i.e., when there is no carry out of the leftmost column). It fails when the final carry is 1, which produces a result with one more digit than the addends.

Exercise 11.2 — Exercise Set 11.2

Q1 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) Algorithm to find elements in List 1 that are NOT present in List 2:

1. Start with an empty list, call it only-in-List1.
2. For each element x in List 1:
   — If x does not appear in List 2, add x to only-in-List1.
3. Report only-in-List1 as the result.

To check 'does x appear in List 2', we scan List 2 from left to right: if any element equals x, x is present; if we reach an element greater than x (using the increasing-order property), we can stop early and conclude x is absent.

Example: List 1 = [1, 3, 5, 15, 25, 75, 125, 375], List 2 = [1, 3, 5, 11, 15, 25, 33, 55, 75, 165, 275, 825].
Scanning: 1 → in List 2; 3 → in List 2; 5 → in List 2; 15 → in List 2; 25 → in List 2; 75 → in List 2; 125 → not in List 2 (List 2 jumps from 75 to 165); 375 → not in List 2.
Result: only-in-List1 = [125, 375].

(ii) Algorithm to find elements in List 2 that are NOT present in List 1:

1. Start with an empty list, call it only-in-List2.
2. For each element y in List 2:
   — If y does not appear in List 1, add y to only-in-List2.
3. Report only-in-List2 as the result.

Using the same example: 11 → not in List 1; 33 → not in List 1; 55 → not in List 1; 165 → not in List 1; 275 → not in List 1; 825 → not in List 1. All others (1, 3, 5, 15, 25, 75) are in List 1.
Result: only-in-List2 = [11, 33, 55, 165, 275, 825].

Q2 Describe an algorithm to compute the least common multiple (lcm) of two numbers.

SOLUTION

Recall that the lcm of two numbers m and n is the smallest positive number that is a multiple of both m and n.

Algorithm to find lcm(m, n):

1. Let multiples-of-m be the list of multiples of m: m, 2m, 3m, … up to m × n (since m × n is always a common multiple).
2. Let multiples-of-n be the list of multiples of n: n, 2n, 3n, … up to m × n.
3. Start with an empty list common-multiples.
4. For each number x in multiples-of-m:

— If x also appears in multiples-of-n, add x to common-multiples.

5. Report the leftmost (smallest) element of common-multiples as lcm(m, n).

Alternative (more efficient) algorithm using gcd:

We know that lcm(m, n) × gcd(m, n) = m × n.
Therefore: lcm(m, n) = (m × n) ÷ gcd(m, n).

1. Compute gcd(m, n) using Āryabhaṭa's division algorithm.
2. Compute m × n.
3. Report (m × n) ÷ gcd(m, n) as lcm(m, n).

Example: lcm(12, 18).
gcd(12, 18): 18 mod 12 = 6; gcd(12, 6): 12 mod 6 = 0; gcd = 6.
lcm(12, 18) = (12 × 18) ÷ 6 = 216 ÷ 6 = 36.
Verification: 36 = 3 × 12 = 2 × 18. ✓

Q3 (*) 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) If j is a divisor of n and j ≤ √n, then n/j is also a divisor of n and n/j ≥ √n. So divisors come in pairs (j, n/j) where one member is ≤ √n and the other is ≥ √n.

Therefore, to find all divisors of n in pairs, we only need to examine numbers from 1 to √n (i.e., approximately √n numbers), rather than from 1 to n.

More precisely, we examine $\lfloor \sqrt{n} \rfloor$ numbers (the floor of √n).

Special case: if n is a perfect square, √n pairs with itself (it appears only once in the list).

Example: For n = 36, √36 = 6. We check j = 1, 2, 3, 4, 5, 6 (6 numbers instead of 36). Pairs: (1, 36), (2, 18), (3, 12), (4, 9), (6, 6). All 9 divisors found by examining just 6 numbers.

(ii) If we list divisors in pairs (adding both j and n/j each time we find a divisor), the list of divisors will still contain all divisors of n, but they will no longer be in increasing order.

The original gcd algorithm's Step 5 relies on the list of common-divisors being in increasing order, so that the rightmost element is the greatest. If the list is not in order, the rightmost element is NOT necessarily the greatest.

Therefore, the gcd algorithm as described will NOT work correctly if we generate divisors in pairs (which produces an unordered list). We would need to either sort the list of common divisors after generating it, or modify Step 5 to explicitly find the maximum element rather than just take the rightmost element.

Exercise 11.3 — Exercise Set 11.3

Q1 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

In the original algorithm, we scan j from 1 up to n. The list-of-divisors is built in increasing order, because we find smaller divisors first.

If we reverse the scan and examine j from n down to 1, we find larger divisors first. The list-of-divisors would then be in decreasing order.

For the divisors(n) algorithm alone: this still finds ALL the same divisors of n — the set of divisors is unchanged. Only the order of the list changes (decreasing instead of increasing).

For the full gcd algorithm: the list common-divisors is built by checking, for each element x in divisors-of-m, whether x appears in divisors-of-n. If both lists are in decreasing order, the common-divisors list will also be in decreasing order.

Step 5 of the gcd algorithm says 'report the rightmost element of common-divisors as gcd(m, n)'. With a decreasing list, the rightmost element is the SMALLEST common divisor (which is 1), not the greatest.

Therefore, the gcd algorithm would give the wrong answer (always 1) if we scan in reverse order without changing Step 5. To fix it, Step 5 would need to say 'report the LEFTMOST element of common-divisors' (since the greatest common divisor now appears first in the decreasing list).

Q2 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

The last algorithm (without lists) uses a variable most-recent-common-divisor, starting at 1 and scanning k from 2 to min(m, n), updating it each time a larger common divisor is found.

If instead we scan k from min(m, n) down to 1, the first common divisor we encounter in this scan is the LARGEST common divisor, which is exactly the gcd.

So we can simplify the algorithm: scan k from min(m, n) down to 1. The very first k that divides both m and n is the gcd. We can stop immediately once we find it.

Modified algorithm:
1. If m < n, reverse and compute gcd(n, m).
2. For each k from min(m, n) down to 1:
   — If k divides both m and n, report k as gcd(m, n) and stop.

This is correct because we encounter the largest common divisor first (scanning downward). There is no need to maintain a variable that keeps updating — one check suffices.

Example: gcd(6, 12). min = 6. Scan: k = 6 divides both 6 and 12 → gcd = 6. Done in one step.

Note: In the worst case (e.g., gcd = 1), we still scan all the way down to 1, so the number of steps is the same as before. The advantage is early termination when the gcd is large relative to min(m, n).

Exercises — End-of-Chapter Exercises

Q1 Compute the following using the improved version of Euclid's algorithm (Āryabhaṭa's division algorithm). (i) gcd(375, 825) (ii) gcd(51000, 81000) (iii) gcd(1789287, 237656) (iv) gcd(2587392, 157656)

SOLUTION

The improved algorithm: (1) If m < n, reverse and compute gcd(n, m). (2) If n = 0, the answer is m. (3) Otherwise reduce to gcd(n, m mod n).

(i) gcd(375, 825):
Since 375 < 825, reverse: compute gcd(825, 375).
825 mod 375 = 75 → compute gcd(375, 75).
375 mod 75 = 0 → compute gcd(75, 0).
n = 0, answer is 75.
gcd(375, 825) = 75.

(ii) gcd(51000, 81000):
Since 51000 < 81000, reverse: compute gcd(81000, 51000).
81000 mod 51000 = 30000 → compute gcd(51000, 30000).
51000 mod 30000 = 21000 → compute gcd(30000, 21000).
30000 mod 21000 = 9000 → compute gcd(21000, 9000).
21000 mod 9000 = 3000 → compute gcd(9000, 3000).
9000 mod 3000 = 0 → compute gcd(3000, 0).
n = 0, answer is 3000.
gcd(51000, 81000) = 3000.

(iii) gcd(1789287, 237656):
1789287 mod 237656 = 125695 → gcd(237656, 125695).
237656 mod 125695 = 111961 → gcd(125695, 111961).
125695 mod 111961 = 13734 → gcd(111961, 13734).
111961 mod 13734 = 2089 → gcd(13734, 2089).
13734 mod 2089 = 1200 → gcd(2089, 1200).
2089 mod 1200 = 889 → gcd(1200, 889).
1200 mod 889 = 311 → gcd(889, 311).
889 mod 311 = 267 → gcd(311, 267).
311 mod 267 = 44 → gcd(267, 44).
267 mod 44 = 3 → gcd(44, 3).
44 mod 3 = 2 → gcd(3, 2).
3 mod 2 = 1 → gcd(2, 1).
2 mod 1 = 0 → gcd(1, 0).
n = 0, answer is 1.
gcd(1789287, 237656) = 1.

(iv) gcd(2587392, 157656):
2587392 mod 157656 = 64896 → gcd(157656, 64896).

157656 mod 64896 = 27864 → gcd(64896, 27864).
64896 mod 27864 = 9168 → gcd(27864, 9168).
27864 mod 9168 = 360 → gcd(9168, 360).
9168 mod 360 = 168 → gcd(360, 168).
360 mod 168 = 24 → gcd(168, 24).
168 mod 24 = 0 → gcd(24, 0).
n = 0, answer is 24.
gcd(2587392, 157656) = 24.

Q2 Assume that m ≥ n. Verify that d divides m and n if and only if d divides both n and m mod n.

SOLUTION

We need to show two things:
(A) If d divides both m and n, then d divides both n and m mod n.
(B) If d divides both n and m mod n, then d divides both m and n.

Let r = m mod n, so m = qn + r for some non-negative integer q, and 0 ≤ r < n.

Proof of (A):
Assume d divides m and d divides n.
Since m = qn + r, we have r = m − qn.
d divides m and d divides qn (because d divides n and therefore divides any multiple of n).
Therefore d divides m − qn = r.
So d divides r = m mod n, and d already divides n. ✓

Proof of (B):
Assume d divides n and d divides r = m mod n.
Since m = qn + r, and d divides qn (because d divides n) and d divides r,
d divides qn + r = m.
So d divides m, and d already divides n. ✓

Since both directions hold, we have shown: d divides m and n <=> d divides n and m mod n.

This means the set of common divisors of (m, n) equals the set of common divisors of (n, m mod n), so in particular gcd(m, n) = gcd(n, m mod n). This justifies the reduction step in Āryabhaṭa's division algorithm.

Q3 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

A prime number has exactly two distinct divisors: 1 and itself.

Algorithm prime(n):
1. Compute divisors(n) — the list of all divisors of n — using the divisors algorithm.
2. Count the number of elements in the list divisors(n).

3. If the count equals exactly 2, report 'n is prime'.
4. Otherwise, report 'n is not prime'.

Equivalently, without building the full list:

Algorithm prime(n) [direct version]:
1. If n < 2, report 'n is not prime'.
2. For each j from 2 to n − 1:
   — If j divides n, report 'n is not prime' and stop.
3. If no such j was found, report 'n is prime'.

Example: prime(7).
Check j = 2: 7 mod 2 = 1 ≠ 0. Check j = 3: 7 mod 3 = 1 ≠ 0. Check j = 4: 7 mod 4 = 3 ≠ 0. Check j = 5: 7 mod 5 = 2 ≠ 0. Check j = 6: 7 mod 6 = 1 ≠ 0. No divisor found → 7 is prime. ✓

Example: prime(9).
Check j = 2: 9 mod 2 = 1. Check j = 3: 9 mod 3 = 0 → 3 divides 9 → 9 is not prime. ✓

Note: Using the paired-divisor idea, we only need to check j from 2 to √n, which is more efficient.

Q4 (*) 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

Algorithm primedivisors(n):
1. Compute divisors(n) using the divisors algorithm. This gives the list of ALL divisors of n.
2. Start with an empty list prime-divisors.
3. For each element d in divisors(n):
   — Run prime(d) using the prime-checking algorithm.
   — If prime(d) reports 'd is prime', add d to prime-divisors.
4. Report prime-divisors as the result.

Example: primedivisors(60).
divisors(60) = [1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60].
Check each: 1 → not prime; 2 → prime; 3 → prime; 4 → not prime; 5 → prime; 6 → not prime; 10 → not prime; 12 → not prime; 15 → not prime; 20 → not prime; 30 → not prime; 60 → not prime.
primedivisors(60) = [2, 3, 5].

Note: The prime divisors of a number are also called its prime factors. For instance, $60 = 2^2 \times 3 \times 5$, and its prime factors are 2, 3, and 5.

Q5 (*) 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 $2^2 \times 3^2 \times 5^1$. How would you represent this? (ii) How would you compare the prime factorisations of two numbers?

SOLUTION

Algorithm to compute the prime factorisation of n:
1. Start with an empty list factorisation. Let d = 2 (the smallest prime).
2. While d × d ≤ n:
   — While d divides n: record d as a factor (add d to factorisation), replace n with n ÷ d.
   — Increase d by 1.
3. If n > 1 after the loop, n itself is a prime factor; add n to factorisation.
4. Report factorisation.

Example: n = 180.
d = 2: 180 ÷ 2 = 90, 90 ÷ 2 = 45. 45 not divisible by 2. Record [2, 2].
d = 3: 45 ÷ 3 = 15, 15 ÷ 3 = 5. 5 not divisible by 3. Record [2, 2, 3, 3].
d = 4: 4 × 4 = 16 > 5, loop ends. Remaining n = 5 > 1, so record 5. Record [2, 2, 3, 3, 5].
Prime factorisation: $2^2 \times 3^2 \times 5^1$. ✓

(i) Representation: We can represent the prime factorisation as a list of (prime, exponent) pairs.
For 180 = $2^2 \times 3^2 \times 5^1$, we represent it as: [(2, 2), (3, 2), (5, 1)].
Each pair (p, e) means the prime p appears with exponent e. The list is arranged in increasing order of primes.

(ii) Comparing prime factorisations of two numbers m and n to find gcd(m, n):
Represent factorisations of both m and n as lists of (prime, exponent) pairs.
For each prime p that appears in BOTH factorisations, take the SMALLER exponent.
Multiply together all p^(min exponent) terms; the product is gcd(m, n).

Example: 180 = $2^2 \times 3^2 \times 5^1$ and 252 = $2^2 \times 3^2 \times 7^1$.
Common primes: 2 (min exponent = 2) and 3 (min exponent = 2).
gcd(180, 252) = $2^2 \times 3^2$ = 4 × 9 = 36.

For lcm(m, n): for each prime appearing in EITHER factorisation, take the LARGER exponent.
lcm(180, 252) = $2^2 \times 3^2 \times 5^1 \times 7^1$ = 4 × 9 × 5 × 7 = 1260.