11 The World of Algorithms

0905CH11

These days the word ‘algorithm’ pops up everywhere. What exactly is an algorithm? What does it have to do with mathematics?

11.1 Adding Numbers

We first learned to add numbers by counting the sum explicitly. To add 5 and 7, we would draw 5 dots to represent the number 5, then draw 7 more dots to represent the number 7, and finally count the dots, 12, to get the answer. Imagine adding large numbers this way, for instance 473 + 695!



$$\begin{array}{r} 473 \\ +\ 695 \\ \hline 8 \end{array} \qquad \begin{array}{r} {\scriptstyle 1\ \ } \\ 473 \\ +\ 695 \\ \hline 68 \end{array} \qquad \begin{array}{r} {\scriptstyle 1\ \ } \\ 473 \\ +\ 695 \\ \hline 1168 \end{array}$$

Fortunately, we discovered a much faster way to add numbers. We wrote the numbers one below the other in the Indian number system, with the rightmost digit aligned. We then added each column from right to left, remembering to add a carry to the left if the total was equal to or more than 10.

11.1.1 Adding Numbers Digit by Digit

This is an example of an **algorithm**, a step-by-step procedure to arrive at an answer. We describe an algorithm by writing down the steps as precisely as possible. For instance, here is one way to describe our addition algorithm.

Algorithm to add two numbers

1. Write the numbers one below the other in the Indian base-ten place-value system so that the digits are aligned from right to left.
2. Add the rightmost digits.
   - If the sum is less than 10, write the sum directly below the two digits. Set the value of carry to 0.
   - If the sum is more than 10, write the units digit of the sum directly below the two digits. Set the value of carry to 1.

© NCERT not to be republished

3. Move to the next column of digits on the left. Add the two digits and the current value of carry.
   - If the sum is less than 10, write the sum directly below the two digits. Set the value of carry to 0.
   - If the sum is more than 10, write the units digit of the sum directly below the two digits. Set the value of carry to 1.
4. Repeat Step 3 until there are no more digits on the left.
5. If the value of carry is 1, write 1 to the left of the bottom row.

Exercise Set 11.1

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?
2. What happens if you add a 5-digit number to a 3-digit number. Do our steps handle this situation correctly?
3. Why is it important to align the columns from right to left?
4. In Step 3, why cannot the value of carry be more than 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.

There is also an important assumption in this algorithm, namely, that we know how to add single digit numbers without counting them out. Typically, any algorithm is built using such basic steps that we assume we can perform directly.

The main advantage of this algorithm is that it is much faster than counting explicitly. For instance, if we move from adding 3-digit numbers to adding 4-digit numbers, we have 10 times as many dots to draw and count. On the other hand, if we add the digits right to left, as in the algorithm above, we only have to add 1 more column. Likewise, 5-digit numbers would generate 100 times as many dots as 3-digit numbers, but only 2 more columns to be added.

© NCERT not to be republished

In concrete terms, suppose we can count one dot per second. To count the dots in 33 + 27 would take a minute, to count 334 + 272 would take about ten minutes, while 3347 + 2729 would take over one hour and forty minutes.

On the other hand, suppose we work slowly and carefully and take 15 seconds to add two single digit numbers with a carry. Adding 33 + 27 would take about half a minute, while 334 + 272 would take about 45 seconds, while 3347 + 2729 would take about one minute.

Whenever we provide an algorithm, we also have to justify why it works. Combining the dots and counting the sum is how addition is defined. Why does adding individual digits from right to left result in the sum of the two numbers? The key idea is that we are grouping each number into units, tens, hundreds, ... and adding each group separately. Whenever a smaller group, say units, generates a large group of ten, we carry over the newly formed ten to the tens group.

Think and Reflect

1. See if you can complete the argument about grouping by units, tens, hundreds, ... to justify why the addition algorithm works.
2. How would you modify the algorithm to add two decimal fractions?

In the rest of this chapter, we will explore the idea of an algorithm in more detail and in some other contexts.

11.2 Greatest Common Divisor

Let us look at a familiar problem: computing the **greatest common divisor (gcd) or highest common factor (hcf)** of two numbers. We have already learned different ways to solve this problem, using prime factorisation and other methods.

How can we solve the problem from the definition of gcd? How do we write down an algorithm for this?

The first step is to understand what is required and break down the problem into smaller pieces that we know how to solve.

- We want the greatest common divisor of two numbers.
- For this, we need to first compute the divisors of both numbers and note them down.
- Once we have done this, we need to compare the two collections of divisors and find the largest number that is present in both collections.

© NCERT not to be republished

11.2.1 Computing the Divisors of a Number

Suppose we want to compute gcd(375, 825).

How do we find the divisors of 375? The divisors of a number are just the factors of that number. Clearly any divisor, or factor, must lie between 1 and 375. The simplest way to find the divisors is to check each number between 1 and 375 and see if it divides 375.

We also have to keep track of the divisors of 375 so that we can compare them with the divisors of 825. We can maintain each of these in a sequence, or a **list**. How do we create the list of divisors of a number? When we start, we do not know any divisors, so the list of divisors is initially empty. Every time we find a new divisor, we add it at the end of the list.

Here is how we can write down this algorithm to find the list of divisors of any number *n*.

Algorithm to find the divisors of n

```
1. Start with an empty list-of-divisors.
2. For each number j in the sequence 1,2,3, ... ,n
   - if j divides n, add j to the list-of-divisors.
```

Notice how we use names `list-of-divisors` and `j` to identify the values that we keep track of in this algorithm. If we do not use `j` to identify the number we are checking next as a possible divisor, it would be clumsy to describe what to do each time we find a divisor.

Running through an algorithm step by step is also called **executing** the algorithm. Let us execute this algorithm for a small number, say 18. We use square brackets to denote a list of values — for instance, [1, 4, 9, 16].

Let us see how this algorithm works for a small number, say 18.

- Initially `list-of-divisors` is empty [ ].
- 1 divides 18, so we update `list-of-divisors` to [1].
- 2 divides 18, so we update `list-of-divisors` to [1, 2].
- 3 divides 18, so we update `list-of-divisors` to [1, 2, 3].
- 4 does not divide 18, so we leave `list-of-divisors` unchanged as [1, 2, 3].
- 5 does not divide 18, so we leave `list-of-divisors` unchanged as [1, 2, 3].

© NCERT not to be republished

- 6 divides 18, so we update `list-of-divisors` to [1, 2, 3, 6].
- 7 and 8 do not divide 18, so we leave `list-of-divisors` unchanged as [1, 2, 3, 6].
- 9 divides 18, so we update `list-of-divisors` to [1, 2, 3, 6, 9].
- 10, 11, 12, ..., 17 do not divide 18, so we keep `list-of-divisors` unchanged as [1, 2, 3, 6, 9].
- 18 divides 18, so we update `list-of-divisors` to [1, 2, 3, 6, 9, 18].

Thus, at the end of the algorithm we have divisors(18) = [1, 2, 3, 6, 9, 18]. Notice that the list is also in increasing order, because we checked for divisors systematically, from 1 to 18.

What is the basic step that we assumed in this algorithm? It is that we know how to compute when a smaller number divides a larger number.

If we go through the steps of the algorithm, we get the following answers.

- divisors(375) = [1, 3, 5, 15, 25, 75, 125, 375]
- divisors(825) = [1, 3, 5, 11, 15, 25, 33, 55, 75, 165, 275, 825]

Think and Reflect

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?

11.2.2 Finding the Greatest Common Divisor

The second step in our procedure to find the gcd is to compare the lists of divisors of the two numbers and find the largest number that appears in both lists.

If we look through the divisiors of 375 and 825 above, we see that the answer is 75. What if the numbers were 54000 and 81000?

- divisors(54000) = [1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20, 24, 25, 27, 30, 36, 40, 45, 48, 50, 54, 60, 72, 75, 80, 90, 100, 108, 120, 125, 135, 144, 150, 180, 200, 216, 225, 240, 250, 270, 300, 360, 375, 400, 432, 450, 500, 540, 600, 675, 720, 750, 900, 1000, 1080, 1125, 1200, 1350, 1500, 1800, 2000, 2160, 2250, 2700, 3000, 3375, 3600, 4500, 5400, 6000, 6750, 9000, 10800, 13500, 18000, 27000, 54000]

© NCERT not to be republished

- divisors(81000) = [1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 18, 20, 24, 25, 27, 30, 36, 40, 45, 50, 54, 60, 72, 75, 81, 90, 100, 108, 120, 125, 135, 150, 162, 180, 200, 216, 225, 250, 270, 300, 324, 360, 375, 405, 450, 500, 540, 600, 648, 675, 750, 810, 900, 1000, 1080, 1125, 1350, 1500, 1620, 1800, 2025, 2250, 2700, 3000, 3240, 3375, 4050, 4500, 5400, 6750, 8100, 9000, 10125, 13500, 16200, 20250, 27000, 40500, 81000]

It is no longer so simple to just inspect the lists to find the largest element common to both of them! We need to do this systematically.

We compute a new list, the list of common divisors. As before, when we compute the list of divisors, this list is initially empty. For each divisor in the first list, we check if it belongs to the second list. If it does, we add it to the list of common divisors. Notice that 1 divides every number, so the list of common divisors will have at least one entry, 1.

Using this, we can compute the list of common divisors of 54000 and 81000, to be [1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 18, 20, 24, 25, 27, 30, 36, 40, 45, 50, 54, 60, 72, 75, 90, 100, 108, 120, 125, 135, 150, 180, 200, 216, 225, 250, 270, 300, 360, 375, 450, 500, 540, 600, 675, 750, 900, 1000, 1080, 1125, 1350, 1500, 1800, 2250, 2700, 3000, 3375, 4500, 5400, 6750, 9000, 13500, 27000].

Since we know that each list of divisors is in increasing order, it must be the case that the list of common divisors is also in increasing order. Hence the largest common divisor is the rightmost element in the list. In other words the gcd of 54000 and 81000 is 27000.

11.2.3 First Algorithm for gcd

Combining the steps above, we can write out a complete algorithm to compute the gcd of two numbers m and n. This makes use of the earlier algorithm to compute the list of divisors of a number.

Algorithm to find gcd(m, n)

```
1. Let divisors-of-m be the list obtained by computing
   divisors(m)
2. Let divisors-of-n be the list obtained by computing
   divisors(n)
3. Start with an empty list common-divisors
```

© NCERT not to be republished

```
4. For each number x in the list of divisors-of-m
   - If x also appears in the list of divisors-of-n,add
   x to the list common-divisors
5. Report the rightmost element of the list common-
   divisors as gcd(m,n)
```

When we execute this algorithm for gcd(375, 825) we check whether each of the divisors [1, 3, 5, 15, 25, 75, 125, 375] of 375 appears in the list of divisors [1, 3, 5, 11, 15, 25, 33, 55, 75, 165, 275, 825] of 825. We see that 1, 3, 5, 15, 25 and 75 are in the second list while 125 and 375 are not. So we compute the list of common divisors to be [1, 3, 5, 15, 25, 75]. Since 75 is the rightmost element in this list, gcd(375, 825) = 75.

11.3 Data Structures

When we execute an algorithm, we compute intermediate quantities, such as the list `divisors-of-m` and the list `divisors-of-n`. To keep track of these values, we use names such as `divisors-of-m` and `divisors-of-n`, respectively, so that we can refer to them later in the algorithm. As we remarked earlier, assigning names to intermediate values is important for writing down algorithms concisely.

Also, these intermediate values are typically more complicated than just simple numbers. In this case, we maintain lists of numbers and, in fact, lists that are arranged in ascending order. A **list** is an example of a **data structure**. Data structures allow us to organise information in a manner that helps make an algorithm more efficient. We will explore this idea in higher grades.

Exercise Set 11.2

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

© NCERT not to be republished

*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?

11.3.1 Improving the Algorithm

Compute both lists of divisors together

In the algorithm above, we first check all numbers between 1 and m to find the divisors of m. Then we start again with 1 and check all numbers between 1 and n to find the divisors of n. This means we have to scan $m + n$ possible divisors.

We can combine these two scans into one. Let max(m, n) be the maximum of m and n. We run through each `j` from 1 to max(m, n).

- If `j` divides m we add `j` to the list of divisors of m.
- If `j` divides n we add `j` to the list of divisors of n.

This reduces the number of possible divisors to be checked from $m + n$ to max(m, n).

Directly compute the list of common divisors

If we combine the scans, we can directly compute the list of common divisors, instead of first computing the divisors of each number and then combining the lists.

- For each number `j` from 1 to max(m, n), if `j` divides both m and n, add `j` to the list of common divisors.

In fact, a common divisor cannot be larger than the smaller of the two numbers, so we can check for common divisors up to the minimum of m and n rather than the maximum.

- For each number `j` from 1 to min(m, n), if `j` divides both m and n, add `j` to the list of common divisors.

Do away with lists

Do we need to maintain the list of all common divisors? Each time we find a larger common divisor, the ones that we have found earlier are useless!

© NCERT not to be republished

Let us keep track of a quantity called most-recent-common-divisor. We know that 1 is always a common divisor, so we can start with most-recent-common-divisor set to 1. We then scan all numbers from 2 to min(m, n).

- For each k from 2 to min(m, n), if k divides both m and n, update most-recent-common-divisor to k.

At the end of this scan, most-recent-common-divisor directly gives us the gcd.

Example: Let m = 6, n = 12.
- min(6, 12) = 6
- Initially set most-recent-common-divisor to 1
- k = 2 divides both 6 and 12.
  Update: most-recent-common-divisor to 2.
- k = 3 divides both 6 and 12.
  Update: most-recent-common-divisor to 3.
- k = 4 doesnot divide 6.
  most-recent-common-divisor remains 3.
- k = 5 doesnot divide 6.
  most-recent-common-divisor remains 3.
- k = 6 divides both 6 and 12.
  Update: most-recent-common-divisor to 6.

Scan complete. gcd(6, 12) = 6

Exercise Set 11.3

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?
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?

11.3.2 Analysing these Algorithms

We started with a very basic algorithm. Since this was derived directly from the definition of gcd, it is easy to justify that the result it computes is actually the gcd.

Each successive refinement that we have proposed improves the efficiency but does not substantially change what is being computed. Hence, we can argue that all of the algorithms we have seen so far compute the correct value.

How efficient are these algorithms? Even in the final version, where we eliminated lists, we had to examine all numbers from 1 to min(m, n). This is similar to what we saw when adding numbers by counting. If the smaller number goes from 3 digits to 4 digits, we have to scan 10 times as many values to find the common divisors. If we go from 3 digits to 5 digits, the work grows by a factor of 100.

© NCERT not to be republished

As we did for adding, can we find an algorithm for gcd where the effort is proportional to the number of digits in the number, rather than the value of the number?

11.3.3 Euclid's Subtraction Algorithm

An elegant solution to achieve this was written down by Euclid, in his book *Elements*. Historians believe that other Greek mathematicians knew of this algorithm before Euclid, but he was the first to put it down in writing.

Reducing a problem to a simpler one

The original version of Euclid's algorithm makes use of the following observation. Let us assume that $m \geq n$. Then, gcd(m, n) is the same as gcd(n, $m - n$).

Why is this the case?

- Let d be a common divisor of m and n. Then we can write m as ad and n as bd, where a and b are natural numbers. This means that $m - n = ad - bd = (a - b)d$, so d is also a divisor of $m - n$.
- On the other hand, if d divides both n and $m - n$, then $n = xd$ and $m - n = yd$ for some natural numbers x and y. This means that $n + (m - n) = m = xd + yd = (x + y)d$, so d also divides m.

This is shown visually below. If we can 'tile' m and n using blocks of size d, we can also tile $m - n$ by blocks of the same size. In fact, Euclid used a similar pictorial proof to justify his algorithm.

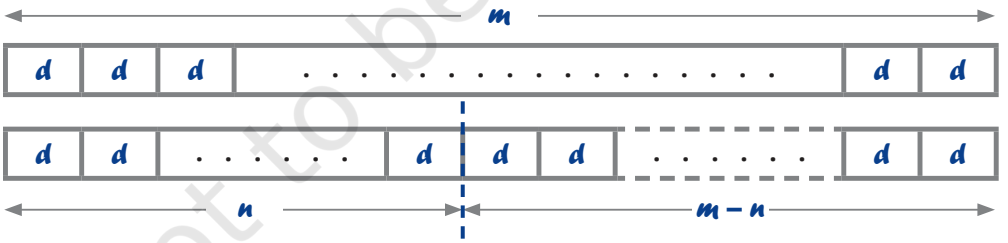


Fig. 11.1: Justification for Euclid's gcd algorithm

This observation tells us that any common divisor of m and n is also a common divisor of n and $m - n$, and the converse is also true. Hence the set of common divisors are the same in both cases, which means that gcd(m, n) = gcd(n, $m - n$).

Euclid's algorithm reduces the problem of finding gcd(m, n) to that of finding gcd(n, $m - n$). The second problem involves smaller numbers, so it should be a simpler problem to solve.

© NCERT not to be republished

How do we solve it? We apply the same strategy: reduce it to gcd(x, y) where x = min(n, $m - n$) and y is the difference between n and $m - n$.

When do we stop? Suppose we compute the gcd of a number with itself, say gcd(23, 23). The answer should clearly be 23. When we reduce this we get gcd(23, 0). In other words, if we reach gcd(m, 0), the answer is m.

Euclid's algorithm for gcd*(m,n)*

```
1. If m < n, reverse the numbers and compute gcd(n,m)
2. If n = 0, report the answer as m
3. Otherwise, reduce the problem to compute gcd(n,m–n)
```

Let us apply this algorithm to gcd(375, 825), the problem that we saw before.

- Since 375 < 825, Step 1 asks us to reverse the numbers and compute gcd(825, 375).
- Since 825 – 375 = 450, Step 3 asks us to reduce the problem to computing gcd(375, 450).
- Since 375 < 450, Step 1 asks us to reverse the numbers and compute gcd(450, 375).
- Step 3 asks us to reduce the problem to computing gcd(375, 75).
- Step 3 asks us to reduce the problem to computing gcd(75, 300).
- Since 75 < 300, Step 1 asks us to reverse the numbers and compute gcd(300, 75).
- Step 3 asks us to reduce the problem to computing gcd(75, 225).
- Since 75 < 225, Step 1 asks us to reverse the numbers and compute gcd(225, 75).
- Step 3 asks us to reduce the problem to computing gcd(75, 150).
- Since 75 < 150, Step 1 asks us to reverse the numbers and compute gcd(150, 75).
- Step 3 asks us to reduce the problem to computing gcd(75, 75).
- Step 3 asks us to reduce the problem to computing gcd(75, 0).
- Since n = 0, Step 2 tells us the answer is 75.

In the example above, we reduced the problem 7 times before finding the answer. Are we close to our original goal of finding an algorithm where the effort is proportional to the number of digits? Unfortunately, no.

© NCERT not to be republished

Consider gcd(99, 2). This will reduce successively to gcd(97, 2), gcd(95, 2), gcd(93, 2), ... , gcd(5, 2), gcd(3, 2), gcd(2, 1), gcd(1, 1), and gcd(1, 0) eventually giving us the answer 1. In general, if we start with an odd number $2k + 1$, gcd($2k + 1$, 2) will require about k reductions before we get the answer 1. This means the number of steps is proportional to the value we start with, rather than the number of digits used to write down the value.

11.3.4 Āryabhaṭa's Division Algorithm

In Indian treatises like Āryabhaṭa's Āryabhaṭīya (499 CE), we see the "long division" method for finding GCD using Indian place value numerals. It makes an efficient use of repeated "division by remainder" in place of repeated subtraction. For example, when one starts with 99 and subtracts 2 repeatedly, what remains, once one cannot subtract 2 anymore, is the remainder when 99 is divided by 2.

Hence, to improve the performance of Euclid's algorithm, we can change the reduction step: instead of reducing gcd(m, n) to gcd(n, $m – n$), we reduce it to gcd(n, m mod n), where m mod n stands for the remainder when m is divided by n. For example, 17 mod 5 is 2 and 33 mod 7 is 5.

We saw earlier that a divisor d 'tiles' both m and n if and only if it 'tiles' both n and $m – n$. In a similar way, we can argue that d 'tiles' m and n if and only if it tiles m and m mod n. This justifies reducing gcd(m, n) to gcd(n, m mod n).

Here is the modified algorithm. Notice that the only change is in the third statement. The stopping condition is the same as before.

```
1. If m < n, reverse the numbers and compute gcd(n,m)
2. If n = 0, report the answer as m
3. Otherwise, reduce the problem to computing gcd(n, m mod n)
```

Consider the earlier example gcd(99, 2) that took a long time to converge. Here is what happens when we execute the improved algorithm.

- Step 3 asks us to reduce the problem to computing gcd(2, 1).
- Step 3 asks us to reduce the problem to computing gcd(1, 0).
- Since $n = 0$, Step 2 tells us the answer is 1.

In other words, we needed only two reduction steps to get to the final answer. Check that this would happen for any similar problem of the form gcd($2k + 1$, 2).

© NCERT not to be republished

The above modified algorithm can be carried out concisely using the "long division" method given by Indians at least from the time of Āryabhaṭa. We illustrate the method through the following examples—

1. gcd(99, 2) = gcd(2, 1)

```
2)99(49
  98
 1)2(2
   2
   0
```

gcd

2. gcd(825, 375) = gcd(375, 75)

```
375)825(2
  750
 75)375(5
  375
    0
```

gcd

Here is another example—

3. gcd(60, 16) = gcd(16, 12) = gcd(12, 4)

```
16)60(3
  48
 12)16(1
   12
  4)12(3
    12
     0
```

gcd

In fact, it can be shown that the number of reductions needed in Āryabhaṭa's division algorithm is now proportional to the number of digits in m and n. Unfortunately, the justification for this fact is too involved to present here, but you will learn about it in future grades.

A Pinch of History

The influential Persian mathematician Al-Khwārizmī (780–850 CE), a scholar at the "House of Wisdom" in Baghdad, developed a great liking for Indian mathematics. He learnt Sanskrit, and around 820 CE wrote a detailed treatise expounding the Indian place value number system and the Indian procedures for the four fundamental operations — addition, subtraction, multiplication, and division. His original is lost, but its 12th-century Latin translation, *Liber Algorismi de numero Indorum* ("The Book of Al-Khwārizmī on Indian Numerals"), survives and played a major role in spreading Indian arithmetic through medieval Europe. Al-Khwārizmī's name became so closely tied to these methods that the word algorismus — from Algoritmi, the Latin form of his name — came to mean any Indian method for numerical computation; it later contracted to algorism and then algorithm. Etymologically, then, "algorithm" originally denoted an Indian method of arithmetic, before it broadened to mean any well-defined, step-by-step systematic procedure for solving a class of problems.

© NCERT not to be republished

End-of-Chapter Exercises

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)
2. Assume that $m \geq n$. Verify that d divides m and n if and only if d divides both n and m mod n.
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)?)

*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.)

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

© NCERT not to be republished

Chapter Summary

- An **algorithm** is a systematic procedure to solve a problem.
- Algorithms consist of basic steps that we know how to execute.
- We may execute the same step a number of times; for instance, for each `j` in 1, 2, ..., n, check if `j` is a divisor of n.
- A step in an algorithm may be conditional; for instance, add `j` to the list of divisors of n only if `j` divides n.
- When we describe an algorithm to solve a problem, we need to justify that it is correct.
- We can also analyse how many steps an algorithm takes, in terms of the size of its input, and decide how efficient it is.