

The monks of Hanoi would have discovered long ago what this signifies in practice. Their task entails making $2^{64} - 1$ or about 18 million million million moves, an enormous number. At the rate of one move per second, their task would take about 570 billion years, or about 40 times the estimated age of the universe!

Summary

In this chapter:

- We have seen that some problems can be solved by algorithms, but others cannot.
- We have seen that algorithms must be capable of being performed by a processor, one step at a time, and that they must eventually terminate.
- We have introduced a notation, based on English, suitable for expressing algorithms.
- We have seen how we can measure the time efficiency of algorithms in terms of the number of characteristic operations performed.
- We have seen how to determine the time complexity of algorithms, expressed in terms of O -notation, and what this tells us about the growth rate of the algorithms' time requirements.
- We have studied recursive algorithms, how we can ensure that they terminate, and how to determine their time complexity.

Exercises

- 2.1 Hand-test the simple and smart power algorithms (Algorithms 2.3 and 2.5 respectively). Use the test case $b = 2$, $n = 11$. How many multiplications are performed by each algorithm?
- 2.2 What is the time complexity of the geometric algorithm given as Algorithm 1.1?
- 2.3 Create a spreadsheet to reproduce the table of growth rates given in Table 2.10, and extend it to $n = 100$.
- 2.4 The following Java methods implement matrix addition and multiplication. Each matrix is represented by an $n \times n$ two-dimensional array of **float** numbers.

```
static void matrixAdd (int n, float[][] a,
    float[][] b, float[][] sum) {
// Set sum to the sum of the nxn matrices a and b.
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            sum[i][j] = a[i][j] + b[i][j];
        }
    }
}
```