(1) If \( n \) is prime, then it outputs \( \text{prime} \).
(2) If \( n \) is composite, then it outputs \( \text{composite} \) with probability at least \( 1 - 1/n \).

14.9 Exercises

14.1. Let \( p_1, p_2 \) and \( p_3 \) be three polynomials of degrees \( n, n \) and \( 2n \), respectively. Give a randomized algorithm to test whether \( p_3(x) = p_1(x) \times p_2(x) \).

14.2. Let \( n \) be a positive integer. Design an efficient randomized algorithm that generates a random permutation of the integers \( 1, 2, \ldots, n \). Assume that you have access to a fair coin. Analyze the time complexity of your algorithm.

14.3. Carefully define what it means for a randomized algorithm to run in expected time \( T(n) \) in the worst case (see Sec. 1.12).

14.4. Carefully define what it means for a randomized algorithm to run in expected time \( T(n) \) on the average (see Sec. 1.12).

14.5. In the discussion of Algorithm \textsc{randomizedquicksort}, it was stated that one possibility to obtain a \( \Theta(n \log n) \) expected time for Algorithm \textsc{quicksort} is by permuting the input elements so that their order becomes random. Describe an \( O(n) \) time algorithm to randomly permute the input array before processing it by Algorithm \textsc{quicksort}.

14.6. Show that Eq. 14.1 is maximum when \( k = \lceil n/2 \rceil \).

14.7. Consider the following modification of Algorithm \textsc{binarysearch} (see Sec. 1.3). Instead of halving the search interval in each iteration, select one of the remaining positions at random. Assume that every position between \( \text{low} \) and \( \text{high} \) is equally likely to be chosen by the algorithm. Compare the performance of this algorithm with that of Algorithm \textsc{binarysearch}.

14.8. Let \( A \) be a Monte Carlo algorithm whose expected running time is at most \( T(n) \) and gives a correct solution with probability \( p(n) \). Suppose the correctness of any solution of the algorithm can always be verified in time \( T'(n) \). Show that \( A \) can be converted into a Las Vegas algorithm \( A' \) for the same problem that runs in expected time at most \( (T(n) + T'(n))/p(n) \).

14.9. Suppose that a Monte Carlo algorithm gives the correct solution with probability at least \( 1 - \epsilon_1 \), regardless of the input. How many executions of the same algorithm are necessary in order to raise the probability to at least \( 1 - \epsilon_2 \), where \( 0 < \epsilon_2 < \epsilon_1 < 1 \).
14.10. Let \( L = x_1, x_2, \ldots, x_n \) be a sequence of elements that contains exactly \( k \) occurrences of the element \( x \) (\( 1 \leq k \leq n \)). We want to find one \( j \) such that \( x_j = x \). Consider repeating the following procedure until \( x \) is found. Generate a random number \( i \) between 1 and \( n \) and check whether \( x_i = x \). Which method is faster, on the average, this method or linear search? Explain.

14.11. Let \( S = \{x_1, x_2, \ldots, x_n\} \) be a set of \( n \) positive integers. We want to find an element \( x \) that is in the upper half when \( S \) is sorted, or in other words greater than the median. Give a randomized algorithm to solve this problem with probability \( 1 - 1/n \). Compare this algorithm with a deterministic algorithm for large values of \( n \), say more than a million.

14.12. Let \( L \) be a list of \( n \) elements that contains a majority element (see Sec. 5.7). Give a randomized algorithm that finds the majority element with probability \( 1 - \epsilon \), for a given \( \epsilon > 0 \). Is randomization suitable for this problem in view of the fact that there is an \( O(n) \) time algorithm to solve it?

14.13. Let \( A, B \) and \( C \) be three \( n \times n \) matrices. Give a \( \Theta(n^2) \) time algorithm to test whether \( AB = C \). The algorithm is such that if \( AB = C \), then it returns true. What is the probability that it returns true when \( AB \neq C \)? (Hint: Let \( x \) be a vector of \( n \) random entries. Perform the test \( A(BX) = CX \).)

14.14. Let \( A \) and \( B \) be two \( n \times n \) matrices. Give a \( \Theta(n^2) \) time algorithm to test whether \( A = B^{-1} \).

14.15. Consider the sampling problem in Sec. 14.7. Suppose we perform one pass over the \( n \) integers and choose each one with probability \( m/n \). Show that the size of the resulting sample has a large variance, and hence its size may be much smaller or larger than \( m \).

14.16. Modify Algorithm randomsampling in Sec. 14.7 to eliminate the need for the boolean array \( S[1..n] \). Assume that \( n \) is too large compared to \( m \), say \( n > m^2 \).

14.17. What are the time and space complexities of the algorithm in Exercise 14.16?


14.19. Consider \( F_n \) as defined on page 386. Suppose that \( n \) is neither a Carmichael number nor a prime number. Show that \( F_n \) under the operation of multiplication modulo \( n \) forms a group that is a proper subgroup of \( Z_n^* \).

14.20. Let \( G = (V, E) \) be a connected undirected graph with \( n \) vertices. A cut in \( G \) is a set of edges whose removal disconnects \( G \). A min-cut in \( G \) is a cut with the minimum number of edges. A straightforward deterministic approach to solve this problem is to apply the max-flow min-cut theorem
(Theorem 16.1). Give a simple Monte Carlo algorithm that finds a min-cut with reasonable probability.

14.21. A multigraph is a graph in which multiple edges are allowed between pairs of vertices. Show that the number of distinct min-cuts in a multigraph with \(n\) vertices is at most \(n(n - 1)/2\) (see Exercise 14.20).

14.22. Show that if there is a group of 23 people, then the probability that two of them will have the same birthday is at least 1/2.

14.10 Bibliographic notes

The real start of randomized algorithms was with the publication of Rabin’s paper “Probabilistic algorithms” (Rabin (1976)). In this paper, two efficient randomized algorithms were presented: one for the closest pair problem and the other for primality testing. The probabilistic algorithm of Solovay and Strassen (1977, 1978), also for primality testing, is another celebrated result. Motwani and Raghavan (1995) is a comprehensive book on randomized algorithms. Some good surveys in this field include Karp (1991), Welsh (1983) and Gupta, Smolka and Bhaskar (1994). Randomized quicksort is based on Hoare (1962). The randomized selection algorithm is due to Hoare (1961). The pattern matching algorithm is based on Karp and Rabin (1987).