Exercises of The design and Analysis of Algorithms

Chapter 1

1. For each of the following pairs of functions, indicate whether the first function of each of the following pairs has a lower, same, or higher order of growth (to within a constant multiple) than the second function.
a. $n(n + 1)$ and $2000n^3$
b. $\log_2 n$ and $\ln n$
c. 2^{n-1} and 2^n

2. Use the informal definitions of O, Θ, and Ω to determine whether the following assertions are true or false.
a. $n(n + 1)/2 = O(n^3)$ **b.** $n(n + 1)/2 = O(n^2)$
c. $n(n + 1)/2 = \Theta(n^3)$ **d.** $n(n + 1)/2 = \Omega(n)$

3. Prove the following assertions by using the definitions of the notations involved, or disprove them by giving a specific counterexample.
a. If $t(n) = O(g(n))$, then $g(n) = \Omega(t(n))$.
b. $\Theta(\alpha g(n)) = \Theta(g(n))$, where $\alpha > 0$.

4. Consider the following algorithm.
ALGORITHM *Mystery*(n)
//Input: A nonnegative integer n
  $S \leftarrow 0$
  for $i \leftarrow 1$ **to** n **do**
    $S \leftarrow S + i * i$
  return S
a. What does this algorithm compute?
b. What is its basic operation?
c. Compute the complexity of this algorithm.
d. Suggest an improvement, or a better algorithm altogether, and indicate its complexity. If you cannot do it, try to prove that, in fact, it cannot be done.

5. Consider the following recursive algorithm for computing the sum of the first n cubes:
$$S(n) = 1^3 + 2^3 + \ldots + n^3.$$
ALGORITHM $S(n)$
//Input: A positive integer n
//Output: The sum of the first n cubes
  if $n = 1$ **return** 1
  else return $S(n - 1) + n * n * n$
a. Compute the complexity of this algorithm.
b. How does this algorithm compare with the straightforward nonrecursive algorithm for computing this sum?

6. Consider the following recursive algorithm

ALGORITHM $Q(n)$

//Input: A positive integer n

if $n = 1$ **return** 1

else return $Q(n - 1) + 2 * n - 1$

a. Set up a recurrence relation for the number of multiplications made by this algorithm and solve it.

b. Compute the complexity of the algorithm.

7. a. Design a recursive algorithm for computing 2^n for any nonnegative integer n that is based on the formula $2^n = 2^{n-1} + 2^{n-1}$.

b. Compute the complexity of this algorithm.

c. Is it a good algorithm for solving this problem?

8. Consider the following recursive algorithm.

ALGORITHM *Riddle*$(A[0..n - 1])$

//Input: An array $A[0..n - 1]$ of real numbers

if $n = 1$ **return** $A[0]$

else $temp \leftarrow Riddle(A[0..n - 2])$

if $temp \leq A[n - 1]$ **return** *temp*

else return $A[n - 1]$

a. What does this algorithm compute?

b. Compute the complexity of the algorithm.

Chapter 2

1. Write a brute-force algorithm to compute a^n then compute the time complexity of it.

2. a. Design a brute-force algorithm for computing the value of a polynomial

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

at a given point x_0 and determine its complexity.

b. If the algorithm you designed is in $\Theta(n^2)$, design a linear algorithm for this problem.

c. Is it possible to design an algorithm with a better-than-linear efficiency for this problem?

3. Consider the problem of counting, in a given text, the number of substrings that start with an A and end with a B. For example, there are four such substrings in CABAAXBYA.

a. Design a brute-force algorithm for this problem and determine its complexity.

b. Design a more efficient algorithm for this problem.

4. Can you design a more efficient algorithm than the one based on the bruteforce strategy to solve the closest-pair problem for n points $x_1, x_2, \ldots, x_n$ on the real line?

5. The closest-pair problem can be posed in the k-dimensional space, in which the Euclidean distance between two points $p'(x'_1, \ldots, x'_k)$ and $p(x''_1, \ldots, x''_k)$

is defined as $d(p', p'')= \sqrt{\sum_{s=1}^{k}(x_s^{'} - x_s^{''})^2}$

Chapter 3

1. a. Write pseudocode for a divide-and-conquer algorithm for finding the position of the largest element in an array of n numbers.
b. Compute the complexity of the algorithm made by you.
c. How does this algorithm compare with the brute-force algorithm for this problem

2. a. Write pseudocode for a divide-and-conquer algorithm for finding values of the smallest elements in an array of n numbers.
b. Compute the complexity of the algorithm made by you.
c. How does this algorithm compare with the brute-force algorithm for this problem?

3. a. Write pseudocode for a divide-and-conquer algorithm for the exponentiation problem of computing a^n where n is a positive integer.
b. Compute the complexity of the algorithm made by you.
c. How does this algorithm compare with the brute-force algorithm for this problem?

4. Design an algorithm to rearrange elements of a given array of n real numbers so that all its negative elements precede all its positive elements. Your algorithm should be both time efficient and space efficient.

5. a. For the one-dimensional version of the closest-pair problem, i.e., for the problem of finding two closest numbers among a given set of n real numbers, design an algorithm that is directly based on the divide-and-conquer technique and determine its complexity.
b. Is it a good algorithm for this problem?

Chapter 4

1. Consider the problem of finding the distance between the two closest numbers in an array of n numbers. (The distance between two numbers x and y is computed as $|x - y|$).
a. Design a presorting-based algorithm for solving this problem and determine its complexity.
b. Compare the efficiency of this algorithm with that of the brute-force algorithm

2. Let $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_m\}$ be two sets of numbers. Consider the problem of finding their intersection, i.e., the set C of all the numbers that are in both A and B.
a. Design a brute-force algorithm for solving this problem and determine its complexity.
b. Design a presorting-based algorithm for solving this problem and determine its complexity.

3. Consider the problem of finding the smallest and largest elements in an array of n numbers.
a. Design a presorting-based algorithm for solving this problem and determine its complexity.
b. Compare the efficiency of the three algorithms: (i) the brute-force algorithm, (ii) this presorting-based algorithm, and (iii) the divide-and-conquer algorithm.

4. You have an array of n real numbers and another integer s. Find out whether the array contains two elements whose sum is s. (For example, for the array 5, 9, 1, 3 and $s = 6$, the answer is yes, but for the same array and $s = 7$, the answer is no.) Design an algorithm for this problem with a better than quadratic time efficiency.

5. Consider the following brute-force algorithm for evaluating a polynomial.

```
ALGORITHM BruteForcePolynomialEvaluation(P[0..n], x)
 p←0.0
 for i ←n downto 0 do
     power ←1
    for j ←1 to i do
        power ←power ∗ x
    p←p + P[i] ∗ power
 return p
```

a. Compute the complexity of the algorithm
b. Use transforn-and-conquer or dynamic programming to design an algorithm with more efficiency

6. Is it a good idea to use a general-purpose polynomial-evaluation algorithm such as Horner's rule to evaluate the polynomial $p(x) = x^n + x^{n-1} + . . . +x + 1$?

7. Consider the problem of finding, for a given positive integer n, the pair of integers whose sum is n and whose product is as large as possible. Design an efficient algorithm for this problem and indicate its complexity.

Chapter 5

1. Binomial coefficient: Design an efficient algorithm for computing the binomial coefficient $C(n, k)$ that uses no multiplications. Compute the complexity of algorithms designed.

2. Design a dynamic programming algorithm to find the minimum number of coins of denominations $d_1 < d_2 < . . . < d_m$ with $d_1=1$ so that the sum of their values is n.

3. **a.** Find a recurrence relation and the initial conditions for the number of ways to climb n stairs if the person climbing the stairs can take one stair or two stairs at a time.
a. Design a brute force algorithm for computing the number of ways to climb n stairs established by the recurrence relation in question a.
b. Design a dynamic programming algorithm for computing the number of ways to climb n stairs established by the recurrence relation in question a.
c. Design a transform-and conquer algorithm for computing the number of ways to climb n stairs established by the recurrence relation in question a.
d. Design a algorithm with the complexity $O(\log_2 n)$ for computing the number of ways to climb n stairs established by the recurrence relation in question a.

4. Present executing of the dynamic programming algorithm that pick up the maximum amount of coin with the coin row of denominations 7, 1, 3, 8, 5, 3.

5. Present executing of the dynamic programming algorithm for knapsack problem with weights $w_1 = 3$, $w_2 = 2$, $w_3 = 5$, $w_4 = 2$ and values $v_1=13$, $v_2=11$, $v_3=22$, $v_4=17$ and the knapsack of capacity $W=7$.

6. Design a dynamic programming algorithm for the version of the knapsack problem in which there are unlimited quantities of copies for each of the n item kinds given. Indicate Compute the complexity of algorithms designed.

7. Shortest path counting A chess rook can move horizontally or vertically to any square in the same row or in the same column of a chessboard. Design a dynamic programming algorithm to find the number of shortest paths by which a rook can move from one corner of a chessboard to the diagonally opposite corner.

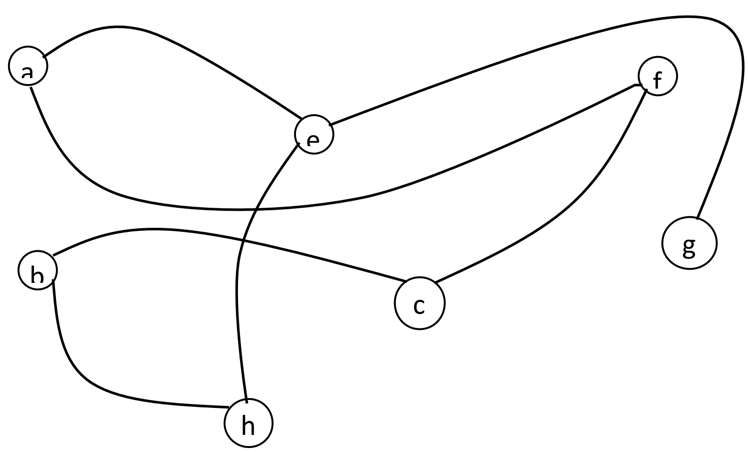
Chapter 6

1. There are n people who need to be assigned to execute n jobs, one person per job. (That is, each person is assigned to exactly one job and each job is assigned to exactly one person). The cost that would accrue if the ith person is assigned to the jth job is a known quantity $C[i, j]$ for each pair $i, j = 1, 2, \ldots, n$. The problem is to find an assignment with the minimum total cost. Design a greedy algorithm for the assignment problem. Execute the greedy algorithm with the table entries representing the assignment costs $C[i, j]$ as follows:

	Job 1	Job 2	Job 3	Job 4
Person 1	9	2	7	8
Person 2	6	4	3	7
Person 3	5	8	1	8
Person 4	7	6	9	4

2. Design a a greedy algorithm to find the minimum number of coins of denominations $d_1<d_2 < \ldots < d_m$ with $d_1=1$ so that the sum of their values is n.

3. Present executing of the algorithm that colours the graph below



Chapter 7

1. Design a backtracking algorithm for finding a subset of a given set $A = \{a_1, \ldots, a_n\}$ of n positive integers whose sum is equal to a given positive integer d. For example, for $A = \{1, 2, 5, 6, 8\}$ and $d = 9$, there are two solutions: $\{1, 2, 6\}$ and $\{1, 8\}$. Of course, some instances of this problem may have no solutions.

2. Design a backtracking algorithm for generating all permutations of $\{1, 2, ..., n\}$. Compute the complexity of the algorithm.

3. Design a backtracking algorithm for generating all bit strings of length n that do not have two consecutive 0s. Compute the complexity of the algorithm.

4. Design a backtracking algorithm for generating all bit strings of length n that do not have two consecutive 1s. Compute the complexity of the algorithm.

5. Present executing of the branch-and-bound algorithm to find the shortest path of the traveler with the cost matrix as follows.

$$\begin{bmatrix} 0 & 10 & 7 & 8 & 19 \\ 5 & 0 & 3 & 18 & 20 \\ 43 & 28 & 0 & 12 & 56 \\ 13 & 20 & 23 & 0 & 17 \\ 7 & 10 & 13 & 19 & 0 \end{bmatrix}$$