The following exercises, taken from CLRS if not otherwise specified, are very good practice. They are not to be handed in.

**Practice Exercises**

1. Solve Exercises 3.1-1, -2, -3, -4, -5, -6
2. Solve Exercises 3.2-1, -2, -3, -6, -7
3. Solve Problems 3-1, -2, -3
4. Solve Exercises 4.1-1, -2, -3, -4, -5, -6
5. Solve Exercises 4.2-1, -2, -3, -4, -5
6. Solve Exercises 4.3-1, -2, -3
7. Solve Problems 4-1, -2, -3, -6
8. **Multiplying n-bit numbers.** Suppose we wish to multiply two n-bit numbers:

\[
X = \underbrace{x_n x_{n-1} \cdots x_{n/2+1} x_{n/2} \cdots x_2 x_1}_a = 2^{n/2}a + b
\]
\[
Y = \underbrace{y_n y_{n-1} \cdots y_{n/2+1} y_{n/2} \cdots y_2 y_1}_c = 2^{n/2}c + d
\]

The usual naive paper-and-pencil long-hand divide-and-conquer algo-
algorithm:

\[
\begin{array}{ccccccc}
  x_n & x_{n-1} & \cdots & x_2 & x_1 \\
  y_n & y_{n-1} & \cdots & y_2 & y_1 \\
  z^2_n & z^2_{n-1} & \cdots & z^2_2 & z^2_1 \\
  z^2_{n+1} & z^2_n & \cdots & z^2_3 & z^2_2 \\
  \vdots & \vdots & \ddots & \vdots & \vdots \\
  z_{2n} & z_{2n-1} & \cdots & z^n_{n+2} & z^n_{n+1} & z^n_n & z^n_{n-1} & \cdots & z_2 & z_1
\end{array}
\]

uses \( T(n) = \Theta(n^2) \) bit operations (multiplications and additions).

(i) We can devise a different divide-and-conquer algorithm based on the following decomposition:

\[
XY = 2^n ac + 2^{n/2}(ad + bc) + bd
\]

Work out a tight asymptotic bound on the number of bit operations (multiplications, additions (or subtractions), and shifts (multiplications by 2)) of the resulting algorithm.

(ii) We can devise yet another divide-and-conquer algorithm based on the following decomposition of Karatsuba:

\[
XY = (2^n + 2^{n/2}) ac + 2^{n/2}(a - b)(d - c) + (2^{n/2} + 1)bd
\]

Work out a tight asymptotic bound on the number of bit operations of the resulting algorithm.

9. **The Majority Problem.** For this problem, you are given an unsorted array of \( n \) elements, and asked to find the majority element, if it exists. (An element is a majority element if it appears more than \( n/2 \) times in the array.) Design an algorithm for the majority problem which runs in linear time.

10. **Horner’s Rule for the Derivative.** Horner’s Rule allows you to evaluate an \( n \)th-degree polynomial \( A(x) = \sum_{i=0}^{n} a_i x^i \) using \( n \) multiplications and \( n \) additions. Show that you can evaluate both \( A(x) \) and its derivative \( A'(x) = \sum_{i=1}^{n} ia_i x^{i-1} \) using \( 2n - 1 \) multiplications and \( 2n - 1 \) additions.

11. Solve Exercises 33.4-1, -3

12. Solve Exercises 15.1-1, -2, -3, -4, -5
13. Solve Exercises 15.2-1, -2, -3, -4, -5
14. Solve Exercises 15.3-1, -2, -3, -4, -5
15. Solve Exercises 15.4-1, -2, -3, -4, -5, -6
16. Solve Exercises 16.1-2, -3, -4
17. Solve Exercises 16.2-1, -2, -3, -4, -5, -6, -7
18. Solve Exercises 16.3-1, -2, -3, -4, -5, -6, -7, -8
19. Solve Exercises 22.1-1, -2, -3, -4, -5, -6, -7, -8
20. Solve Exercises 22.2-1, -2, -3, -4, -5, -6, -7, -8
21. Solve Exercises 22.3-1, -2, -3, -4, -5, -6, -7, -8, -9, -10
22. Solve Exercises 22.4-1, -2, -3, -4, -5
23. Solve Exercises 22.5-1, -2, -3, -4, -5, -6, -7
24. Solve Exercises 23.1-1, -2, -7
25. Solve Exercises 23.2-1, -4, -8

26. **Game of Matches.** Suppose there are 30 matches on a table. I begin by picking up 1, 2 or 3 matches. Then my opponent must pick up 1, 2 or 3 matches. We continue in this fashion until the last match is picked up. The player who picks up the last match loses. How can I (the first player) be sure of winning the game?

27. **The 0/1 Knapsack Problem.** A Thief robbing a store finds $n$ items, the $i$-th is worth $v_i$ dollars and weighs $w_i$ pounds, where $v_i$ and $w_i$ are integers. He wants to take as valuable a load as possible, but he can carry at most $W$ pounds in his knapsack. Which items should he take? (0/1 means drop or take, there is no possibility of taking fractions of the items.) Design an algorithm for this problem and analyse its complexity.

28. **The Knapsack Problem.** You plan to go hiking and you want to pack your stuff in a knapsack. Your knapsack has a total volume of $V$ cubic inches, and you have $n$ different items of volumes $v_1, \ldots, v_n$ that you want to take with you. Unfortunately, you can’t take them all, since their total volume exceeds the volume of your knapsack. You need to decide which items to bring and which items to leave at home.
Suppose that you want to optimize the packing so that the volume is utilized to its maximal extent. This means you want to find a subset of the items such that the sum of their volumes is maximal with the constraint that the sum is no more than $V$. We call this the optimum packing of $v_1, \ldots, v_n$ in $V$.

(i) Fix a list of items $v_1, \ldots, v_n$. For any $i \leq n$, and any $U \leq V$, denote by $OPT_i(U)$ the volume of the optimum packing of $v_1, \ldots, v_i$ in $U$. Prove that for all $i$ and $U$, we have

$$OPT_i(U) = \max\{OPT_{i-1}(U), v_i + OPT_{i-1}(U - v_i)\}$$

(ii) Assume that all the volumes $v_1, \ldots, v_n$ are integral numbers of units. Design a dynamic-programming algorithm to solve the knapsack problem. Analyze your algorithm.

Problems to hand in

1. **The Minimal Sum Sublist.** Given a list of $n$ integers, consider the problem of finding the minimal sum in any sublist of the input. For example, in the list

$$[-31, 41, -59, -26, 53, -58, -97, 93, 23, -84]$$

the minimal sum is achieved by summing the third through seventh element: $-59 - 26 + 53 - 58 - 97 = -187$. (Hint: When all numbers are negative, the entire array is the answer. When all numbers are positive, the empty array is the answer.)

   (i) Give a simple, clear and correct $\Theta(n^2)$ algorithm.

   (ii) Create a better, more efficient algorithm and analyze its time complexity, giving tight bounds.

2. **The Sysadmin Problem.** We wish to store programs $P_1, P_2, \ldots, P_n$ on a disk. Program $P_i$ needs $s_i$ kB and the disk has capacity $D < \sum_{i=1}^n s_i$ kB.

   (i) Prove or disprove: to maximize the number of programs held on the disk, we can use a greedy algorithm that takes programs in order of nondecreasing size.

   (ii) Prove or disprove: to maximize disk usage, we can use a greedy algorithm that takes programs in order of nonincreasing size.