**Total possible marks:** 100. Homeworks must be solved individually. This set covers chapters 1–4 of the textbook *Introduction to Algorithms*, 3rd. ed., by Cormen et al.

**Exercise 1: asymptotic behavior (20 points).**

1. (10 points) Assume you have two computers, $C_A$ and $C_B$, capable of performing $10^7$ and $10^9$ operations per second, respectively. Both computers run a set of algorithms whose precise complexities $f(n)$ are given below. Determine the size $n^*$ of the biggest input that can be processed in 1 second for each computer, as in the example.

<table>
<thead>
<tr>
<th>$f(n)$</th>
<th>$n^*$ for $C_A$</th>
<th>$n^*$ for $C_B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lg \lg n$</td>
<td></td>
<td>$10^{14}$</td>
</tr>
<tr>
<td>$\sqrt{n}$</td>
<td></td>
<td>$10^{18}$</td>
</tr>
<tr>
<td>$14n + 4$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n \log_3 n + n$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n^2 + 7n$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n^3$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n^n$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The precise complexity tells you how many operations are performed to solve an instance of size $n$. Assume each operation takes the same time and that the input sizes are natural numbers $1, 2, 3, \ldots$

The point of this exercise is to see how much can we gain by going from $C_A$ to $C_B$.

2. (10 points) (Exercise 3.1-1 in the textbook.) Let $f$ and $g$ be asymptotically nonnegative functions of $n = 1, 2, 3 \ldots$ (that is, $f(n) \geq 0$ and $g(n) \geq 0$ if $n$ is sufficiently large). Using the basic definition of $\Theta$-notation, prove that $\max(f, g) = \Theta(f + g)$. Hint: find values for the constants $c_1$, $c_2$, $n_0$ in the definition of $\Theta(\cdot)$ and show it holds.

**Exercise 2: (34 points).** The following functions compute the factorial $n!$ of a natural number $n$ using iteration and recursion, respectively:

**ITERATIVE-FACTORIAL($n$)**

1. $f = 1$
2. for $i = 2$ to $n$
3. $f = i \times f$
4. return $f$

**RECURSIVE-FACTORIAL($n$)**

1. if $n == 1$
2. return 1
3. return $n \times$ RECURSIVE-FACTORIAL($n - 1$)

1. (10 points) Write a recursive function to compute the factorial using divide and conquer as in MERGE-SORT. That is, assuming $n \geq m$, write a function FACTORIAL($n, m$) that computes $n(n-1)(n-2) \cdots (m+1)m$. The top-level call will be FACTORIAL($n, 1$).
2. (6 points) Prove the algorithm is correct, i.e., that \textsc{Factorial}(n, 1) = n!. \textit{Hint:} use
induction to prove that \textsc{Factorial}(n, m) is correct.

3. (6 points) Write the runtime \( T(n) \) of \textsc{Factorial}(n, 1) as a recurrence.

4. (6 points) Solve the recurrence and obtain \( T(n) \) in asymptotic notation.

5. (6 points) Is the runtime \( T(n) \) better than for \textsc{Iterative-Factorial}(n) or \textsc{Recursive-Factorial}(n)? Explain.

Exercise 3: sorting algorithms and runtime (10 points). (Exercise 2.3-6 in the textbook.)
Observe that the while loop of lines 5–7 of the \textsc{Insertion-Sort} procedure in Section 2.1 of the
book uses a linear search to scan (backward) through the sorted subarray \( A[1 \ldots j-1] \). Can we
use a binary search (see Exercise 2.3-5) instead to improve the overall worst-case running time
of insertion sort to \( \Theta(n \lg n) \)?

Exercise 4: recurrent equations (36 points). (This is based on textbook problems 4.1
and 4.3.) Give asymptotic upper and lower bounds for \( T(n) \) in each of the following recurrences.
Assume that \( T(n) \) is constant for \( n \leq 2 \).

1. (6 points) \( T(n) = 9T(n/3) + n + n^2 + 1 \)
2. (6 points) \( T(n) = 2T(n^{1/4}) + 1 \)
3. (6 points) \( T(n) = 2T(2n/9) + \sqrt{n} \)
4. (6 points) \( T(n) = 2T(n) + n^2 \)
5. (6 points) \( T(n) = T(n-2) + \Omega(1) \)
6. (6 points) \( T(n) = 2T(n/11) + \sqrt[3]{n} \)

Justify your answers. If you use the master theorem, specify which case and show that its
hypotheses are satisfied. If you use recursion trees to find a good guess, verify the guess with
the substitution method.

Bonus exercise: recurrent equations (10 points). Consider the recurrence

\[ T(n) = aT(n/b) + f(n). \]

Case 3 of the master theorem requires that \( f(n) = \Omega(n^{\log_b a + \epsilon}) \) for some constant \( \epsilon > 0 \) and
that \( f(n) \) satisfies the regularity condition \( af(n/b) \leq cf(n) \) for some constant \( c < 1 \) and all
sufficiently large \( n \). Prove that the regularity condition is always satisfied if \( f(n) = n^k \). (This
means we don’t need to check it when \( f \) is a polynomial.)
Bonus exercise: recurrent equations (30 points). Consider the following particular type of recurrence:

\[
T(n) = \begin{cases} 
1, & n = 1 \\
\frac{aT(n/b) + n^c}{2^k}, & n > 1 
\end{cases}
\]

where \( a \geq 1 \) and \( b > 1 \) are integers, \( c \geq 0 \) is real and \( k = \log_b n \) is integer (that is, we can only pick sizes \( n \) of the form \( n = b^k \) where \( k \geq 0 \) is an integer). Use mathematical induction to prove that

\[
T(n) = \begin{cases} 
n^c(1 + \log_b n), & \text{if } c = \log_b a \\
\frac{a \cdot n^{\log_b a - n^c}}{2^k - 1}, & \text{if } c \neq \log_b a. 
\end{cases}
\]

**Hint:** as a simpler example, see exercise 2.3-3.

Consequently, prove the master theorem for this recurrence, that is, prove that

\[
T(n) = \begin{cases} 
\Theta(n^{\log_b a}), & \text{if } c < \log_b a \\
\Theta(n^c \lg n), & \text{if } c = \log_b a \\
\Theta(n^c), & \text{if } c > \log_b a. 
\end{cases}
\]