Submission Instructions

1. This assignment is to be solved in teams of two students. Take our warning on plagiarism very seriously. We assume that by submitting a solution you are certifying that it is solely the work of your team, except where explicitly stated otherwise, and that each teammate can individually explain any part of the solution.

2. Write clear answers. Justify all answers, except where explicitly not required. State any assumptions you make. Thoroughly spell-check and grammar-check your answers. Meticulously follow all instructions in this document and at the homework web-page.

3. Document each algorithm according to the coding convention of the course.

4. Name your report $\text{LastName}_1.\text{FirstName}_1 - \text{LastName}_2.\text{FirstName}_2.pdf$, but without using any special characters. The report must be in PDF format and must have the cover page of the course.

5. Submit your report via the Course Manager server (whose clock may differ from yours) by the deadline given above.

6. Each team member must submit a copy of the same solution. Only one of the submitted solution copies of a team will be graded, so make sure they are identical. The lateness penalty, if any, for a team will be determined by the moment of arrival of its last submitted solution copy.

Failure to follow the instructions above may result in 0 (zero) points, and we reserve the right to process your solutions automatically.
Question 1: Comparison of running times (16 points)


For each function \( f(n) \) and time \( t \) in the following table, determine the largest size \( n \) of a problem that can be solved in time \( t \), assuming that the algorithm to solve the problem takes \( f(n) \) microseconds.

<table>
<thead>
<tr>
<th></th>
<th>1 second</th>
<th>1 minute</th>
<th>1 day</th>
<th>1 year</th>
<th>1 century</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lg n )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \sqrt{n} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( n )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( n \log n )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( n^2 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( n^3 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( 2^n )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( n! )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Complete and correct answers to each row give 2 points. Exhibit your reasoning; otherwise you get zero points for correct rows.

Question 2: Integer-sort (24 points)

It can be proven that, over arbitrary arrays of length \( n \), the cost in time of any sorting algorithm is \( \Omega(n \log n) \) in the worst-case. To this respect, algorithms like MERGE-SORT are optimal.

On the other hand, if the arrays to be sorted satisfy certain (additional) properties, faster algorithms may exist. As an example, consider the following idea for sorting an array \( A[1..n] \) of \( n \) integers, each one known in advance to belong to the interval \( [1..k] \). This algorithm is known as INTEGER-SORT.

1. Create an auxiliary integer array \( Y[1..k] \) of \( k \) elements. Initialise this array with zeros.
2. Scan array \( A \) from left to right. At every step, if the current position of \( A \) holds value \( x \), increment \( Y[x] \) by 1.
3. To return the ordered array, scan \( Y \). At any position \( x \), if \( Y[x] = t \), put value \( x \) a total of \( t \) times into the result array in the proper order.

Given the high-level description above:

a (7 points) Write pseudocode for INTEGER-SORT, having as arguments an array \( A[1..n] \) and a positive integer \( k \). Elements in array \( A \) are known to belong to the interval \([1..k] \).

b (7 points) Show with some pictures the behaviour of the algorithm over an array of size \( n = 5 \) with \( k = 4 \).

c (7 points) Compute the worst-case running time of the algorithm in terms of \( \Theta \) notation. Note that the size of the input is made of two numbers, \( n \) and \( k \).

d (3 points) What is the worst-case running time of INTEGER-SORT if \( k = O(n) \)?
Question 3: Binary multiplication (35 points)

Consider the problem of computing the product of two integers, represented in binary notation as two arrays $A[1..n]$ and $B[1..n]$ of $n$ digits in $\{0, 1\}$, with $A[n]$ and $B[n]$ representing the least significant digits.

**Input:** Two 0/1 arrays $A[1..n]$ and $B[1..n]$ of the same length $n$.

**Output:** A 0/1 array $C[1..2n]$ encoding the product of $A$ and $B$ in binary notation.

a (8 points) Write pseudocode for a recursive version of the algorithm, following the divide-and-conquer approach. (Hint: split the two arrays into two halves, recursively multiply the halves in a proper way, and then proceed by merging the partial results using summations and shifts.)

b (8 points) Show with some pictures the behaviour of the algorithm over numbers having $n = 5$ digits.

c (5 points) Make an educated guess on the worst-case running time of the algorithm in terms of $\Theta$ notation using the recursion-tree method.

d (8 points) Prove your conjecture by using the substitution method. If your conjecture is not true, return to step 3.

e (6 points) Briefly discuss whether the Master Theorem (MT) is applicable to your algorithm or not. In the positive case, check that the answer obtained by applying MT is the one you obtained in steps 3 and 4.

Question 4: Recurrence equations (25 points)

[This problem is inspired from Problem 4-1 of (Cormen et al., 2001).]

Compute asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for $n \leq 2$. Make your bounds as tight as possible.

a $T(n) = 2T(n/2) + n^3$.
b $T(n) = T(9n/10) + n$.
c $T(n) = T(n - 1) + n$.
d $T(n) = 2T(n/4) + \sqrt{n}$.
e $T(n) = 7T(n/3) + n^2$.

The correct closure of each recurrence gives 5 points.