**Algorithm Complexity**

# Tutorial 1

**Exercise 1**

Consider a modern computer capable of executing approximately $10^{15}$ basic instructions per second. An algorithm operating on an input of size $n$ performs a number of instructions given by one of the following functions:
$f(n) \in \left\{\log_2 n, n, n\log_2 n, n^2, n^3, 2^n\right\}$.

1. For each function $f(n)$ and each input size $n \in 6, 25, 50, 100$, calculate the number of instructions performed by the algorithm and the corresponding execution time.
2. Express the execution times using an appropriate time unit and complete a table summarizing the results.
3. For each input size, arrange the algorithms from the fastest to the slowest according to their execution time.
4. Arrange the six functions according to their asymptotic growth rate, from the slowest-growing to the fastest-growing function.
5. Consider an algorithm with complexity $2^n$ and $n = 100$. Suppose that the computer used in the previous questions is replaced by a computer that is 1 000 times faster. Calculate the new execution time and compare it with the original execution time.
6. Moore's law is commonly expressed as an approximate doubling of computing capacity every two years. Suppose that a computer in 2005 could execute $40 \times 10^{12}$ instructions per second.
   (a) Estimate the computing capacity of this computer in 2025, assuming that the computing capacity doubles every two years.
   (b) Using the estimated 2025 computing capacity, calculate the execution time of an algorithm performing $2^{100}$ instructions.
   (c) Express the resulting execution time in years and discuss whether the improvement predicted by Moore's law makes the computation practically feasible.
7. Based on the previous results, explain why improvements in computing capacity cannot compensate indefinitely for algorithms with rapidly growing complexity.

**Additional Question**

Consider two algorithms solving the same problem:
$f_A(n) = n^3$ and $f_B(n) = 2^n$.
For small input sizes, their execution times may appear comparable. Explain what happens as $n$ increases and discuss which algorithm is more suitable for large-scale inputs.

**Exercise 2**

The Fibonacci sequence is defined by:

$$F_n = \begin{cases} 0 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \\ F_{n-1} + F_{n-2} & \text{if } n > 1. \end{cases}$$

Consider the following approaches for computing the $n$-th Fibonacci number:

1. A direct recursive approach based on the mathematical definition.
2. A dynamic programming approach using memoization (top-down).
3. A dynamic programming approach using tabulation (bottom-up).

1. Write the pseudocode for the direct recursive algorithm.
2. Write a pseudocode algorithm using the top-down dynamic programming approach with memoization.
3. Write a pseudocode algorithm using the bottom-up dynamic programming approach with tabulation.

4. For $n = 6$, illustrate the recursive calls generated by the direct recursive algorithm. Identify the subproblems that are solved more than once.
5. For each of the approaches, determine:
   (a) the time complexity;
   (b) the space complexity.
6. Explain how memoization and tabulation avoid the repeated computation observed in the direct recursive approach.
7. Calculate $F_{10}$ and $F_{20}$ using the bottom-up algorithm.
8. The Fibonacci number $F_n$ grows exponentially with $n$, although the dynamic programming algorithm computes it in linear time. Explain why the growth of the output value does not imply that the algorithm has exponential time complexity.

**Additional Question**

The bottom-up algorithm stores all Fibonacci values from $F_0$ to $F_n$. Propose a modified version that computes $F_n$ using only a constant amount of additional memory. Give its time and space complexities.

**Algorithm Complexity**

# Tutorial 2

**Exercise 1**

Consider an algorithm Selection Sort to sort $n$ numbers stored in an array $A$. Selection Sort starts by finding the smallest element in $A$ and swapping it with the first element of $A$. Then, the algorithm finds the second smallest element and swaps it with the second position. It continues this process for the remaining elements.

1. Write the algorithm in pseudo-code.
2. State the loop invariant of the algorithm and prove its correctness.
3. Find the best-case and worst-case time complexity.
4. Calculate the algorithm's complexity based on fundamental operations.

**Exercise 2**

Consider the Quick Sort algorithm for sorting $n$ numbers stored in a table $A$. The algorithm selects an element called the *pivot*, partitions the table so that elements smaller than the pivot are placed before it and elements greater than the pivot are placed after it, and then recursively sorts the two resulting subtables.

1. Write the pseudocode for the Quick Sort algorithm and its Partition procedure.
2. For the table
$$A = \langle 8, 3, 7, 4, 9, 2, 6, 5 \rangle,$$
trace the first partitioning step using the last element as the pivot. Show the resulting table and the final position of the pivot.
3. Establish the recurrence relation for the running time of Quick Sort in:
   (a) the best case, when the pivot divides the table into two approximately equal parts;
   (b) the worst case, when the pivot produces one subtable containing $n-1$ elements and another containing no elements.
4. Determine the asymptotic time complexity of Quick Sort in the best and worst cases. Explain the reason for the difference between these two cases.
5. State the loop invariant of the main loop in the Partition procedure and use it to justify that, when the procedure terminates, the pivot is placed in its correct position.

**Exercise 3**

Consider the Matrix-Multiplication algorithm for matrix multiplication.

**Algorithm 1:** Matrix-Multiplication($A$, $B$)

**Data:** $A = a_{ij}$, $B = b_{ij}$ two $n \times n$ matrices with elements in $\mathbb{N}$
**Result:** $C = A \times B$

```
1 for i ← 1 to n do
2 |   for j ← 1 to n do
3 |   |   c[i, j] ← 0 ;
4 |   |   for k ← 1 to n do
5 |   |   |   c[i, j] ← c[i, j] + a[i, k] ∗ b[k, j] ;
6 |   |   end
7 |   end
8 end
```

- Calculate the complexity of Matrix-Multiplication.
- Generalize the algorithm to the case where $A$ is an $m \times n$ matrix and $B$ is an $n \times p$ matrix, and compute its time complexity.

**Exercise 4**

Consider the SequentialSearch algorithm for sequential search of an element in a list.

**Algorithm 2:** SequentialSearch($L$, $X$)

**Data:** $L$ a list with $n$ elements and $X$ an arbitrary element
**Result:** Finds the position $j$ of element $X$ in $L$. If $X$ is not in the list, $j$ is set to zero

```
1 j ← 1 ;
2 while (j ≤ n) ∧ (L[j] ≠ X) do
3 |   j ← j + 1 ;
4 end
5 if (j > n) then
6 |   j ← 0 ;
7 end
```

- Determine its complexity in the best, worst, and average cases.

**Exercise 5**

Consider the Insertion-Sort algorithm for sorting an table $A$ containing $n$ distinct numbers in nondecreasing order. At each iteration, the algorithm takes one element from the unsorted part and inserts it into its appropriate position within the already sorted part.

1. Write the pseudocode for Insertion-Sort.
2. Apply the algorithm to the table
$$A = \langle 7, 4, 9, 2, 6 \rangle$$
and show the state of the table after each iteration of the outer loop.
3. Consider the comparison
$$A[j] > \text{element}$$
as the fundamental operation. Determine the exact number of executions of this operation for:
   (a) an already sorted table;
   (b) an table sorted in decreasing order.
4. Determine the best-case and worst-case time complexities of Insertion-Sort based on the number of executions of the fundamental operation.
5. Assume that all permutations of the $n$ elements are equally likely. Determine the average number of executions of the fundamental operation and derive the corresponding average-case time complexity.

**Algorithm Complexity**

# Tutorial 3

**Exercise 1**

For each of the following algorithms, determine the asymptotic time complexity. In each case, identify a suitable fundamental operation and justify your answer by analyzing the number of times it is executed.

**Algorithm 3:** $F_1(n)$

```
1 x ← 0;
2 for i ← 1 to n do
3 |   x ← x + 1;
4 end
5 return x;
```

**Algorithm 4:** $F_2(n)$

```
1 x ← 0;
2 i ← 1;
3 while i ≤ n do
4 |   x ← x + 1;
5 |   i ← 2 × i;
6 end
7 return x;
```

**Algorithm 5:** $F_3(n)$

```
1 x ← 0;
2 i ← 1;
3 while i ≤ n do
4 |   for j ← 1 to i do
5 |   |   x ← x + 1;
6 |   end
7 |   i ← 2 × i;
8 end
9 return x;
```

**Algorithm 6:** $F_4(n)$

```
1 x ← 0;
2 for i ← 1 to n do
3 |   for j ← 1 to i do
4 |   |   x ← x + 1;
5 |   end
6 end
7 return x;
```

**Algorithm 7:** $F_5(n)$

```
1  x ← 0;
2  i ← n;
3  while i > 1 do
4  |   j ← i;
5  |   while j > 1 do
6  |   |   x ← x + 1;
7  |   |   j ← ⌊j/2⌋;
8  |   end
9  |   i ← ⌊i/2⌋;
10 end
11 return x;
```

1. Determine the number of executions of the fundamental operation for each algorithm.
2. Determine the asymptotic time complexity of each algorithm.
3. Arrange the five algorithms in increasing order of asymptotic complexity.
4. For each algorithm, indicate whether its complexity changes depending on the input values, or whether the best, average, and worst cases have the same asymptotic order.
5. Explain briefly why $F_3(n)$ has a lower asymptotic complexity than $F_4(n)$, even though both algorithms contain nested loops.

**Exercise 2**

Consider a sorted table $A[1..n]$ containing distinct elements in ascending order and a value $X$. Binary Search uses the ordering of the table to reduce the search interval by approximately half at each step.

1. Write an iterative pseudocode algorithm BinarySearch$(A, X)$ that returns the position of $X$ if it is present in $A$, and $-1$ otherwise. Apply the algorithm to

$$A = \langle 4, 9, 15, 21, 28, 34, 41, 47, 53, 60, 68 \rangle$$

   for $X = 41$ and $X = 30$, showing the successive search intervals and middle positions.

2. Identify the fundamental comparison of the algorithm and determine its best-case, average-case, and worst-case time complexities. Justify the worst-case complexity by expressing the size of the search interval after $p$ iterations.

**Exercise 3**

Consider the following functions:

$$T_1(n) = 4n \log_2 n + 3n + 7,$$

$$T_2(n) = 2^n + n^4 + 10n^2 + 25n,$$

$$T_3(n, k) = n + k + \log_2 n, \qquad 1 \le k \le n,$$

$$T_4(n) = n^2 + 3^{\log_4 n},$$

and

$$T_5(n) = n \log_2 n + 2n \log_2(\log_2 n) + n.$$

1. Prove that
$$T_1(n) = O(n \log n).$$
Give suitable values for the constants $c$ and $n_0$.

2. Prove that
$$T_2(n) = O(2^n).$$
Justify how the polynomial terms compare with $2^n$.

3. Given that
$$1 \leq k \leq n,$$
prove that
$$T_3(n, k) = O(n).$$
Then determine whether
$$T_3(n, k) = \Theta(n)$$
holds.

4. Simplify
$$3^{\log_4 n}$$
as a function of $n$, and determine the tight asymptotic complexity of
$$T_4(n).$$

5. Determine the tight asymptotic complexity of
$$T_5(n).$$
Justify your answer by comparing the growth rates of
$$n \log_2(\log_2 n)$$
and
$$n \log_2 n.$$

**Exercise 4**

Let $f(n)$, $g(n)$, and $h(n)$ be positive functions for sufficiently large $n$.

1. Prove that
$$\max\{f(n), g(n)\} = \Theta(f(n) + g(n)).$$
Your demonstration should establish both the upper and lower bounds required by the $\Theta$ notation.

2. Suppose that
$$f(n) = \Theta(g(n)) \qquad \text{and} \qquad g(n) = \Theta(h(n)).$$
Prove that
$$f(n) = \Theta(h(n)).$$
Then determine whether the result remains valid if $\Theta$ is replaced by $O$.

3. Let

$$T(n) = n^3 + n^2 2^{\log_2 n} + 4^n + n(\log_2 n)^2.$$

Determine a tight asymptotic bound for $T(n)$. Your solution must first simplify the non-standard terms and then justify both the upper and lower bounds.

4. Let

$$T(n,k) = n^2 + kn + \log n, \qquad 1 \le k \le n.$$

Determine a tight asymptotic bound for $T(n,k)$ with respect to $n$. Then investigate whether the bound changes when:

$$k = 1, \qquad k = n.$$

**Exercise 5**

Can we write: $2^{n+1} = O(2^n)$? $2^{2n} = O(2^n)$?
Justify your answer based on the Landau notation $O$.

**Algorithm Complexity**

# Tutorial 4

**Exercise 1**

Consider the algorithm RechMax which searches for the largest element in a list.

**Algorithm 8:** RechMax($L$)

**Data:** $L$: a non-empty list with $n$ integer elements
**Result:** Finds the largest element $M$ in the non-empty list $L$

```
1 M ← L[1] ;
2 for i ← 2 to n do
3 |   if L[j] > M then
4 |   |   M ← L[j] ;
5 |   end
6 end
```

Regardless of the evaluation measure considered, the number of comparisons is equal to $n-1$:

$$Min_A(n) = Avg_A(n) = Max_A(n) = n-1$$

- Prove that this algorithm is optimal in terms of the number of comparisons.

**Exercise 2**

Consider the following recurrence equation:

$$T(n) = \begin{cases} 2 & \text{if } n = 2 \\ 2T(\frac{n}{2}) + n & \text{if } n = 2^k, \text{ for } k > 1 \end{cases}$$

Given that $n$ is a power of 2:

1. Find the solution for $T(n)$ using the "Plug-and-Chug" method (illustrate an algorithm that exhibits this behavior).
2. Prove the correctness of the obtained solution through induction.

**Exercise 3**

Consider the following recurrence equations:

$$T(n) = \begin{cases} 1 & \text{if } n = 0 \\ T(n-1) + n & \text{for } n > 0 \end{cases}$$

$$S(n)^1 = \begin{cases} 0 & \text{if } n = 0 \\ 2S(n-1) + 1 & \text{for } n > 0 \end{cases}$$

- Find the solution for $T(n)$ and $S(n)$.
- Prove by induction that the solutions are correct.

**Exercise 4**

Consider the following recurrence relation:

$$T(n) = \begin{cases} b & \text{if } n < 2 \\ 2T(\frac{n}{2}) + bn & \text{for } n \geq 2 \end{cases}$$

- Solve this equation by drawing its recurrence tree.

[1] The number of moves needed to solve the Tower of Hanoi problem with $n$ disks.

**Exercise 5**

Let $F(n)$ be the number of additions performed by the recursive algorithm to calculate the Fibonacci number at position $n$:

$$F(n) = \begin{cases} 0 & \text{if } n = 0 \text{ or } n = 1 \\ F(n-1) + F(n-2) + 1 & \text{for } n > 1 \end{cases}$$

- Calculate the complexity of this algorithm.

**Exercise 6**

Let $F(n)$ and $T(n)$ be non-uniform linear recurrence relations of degree $k$ ($k \geq 1$) with constant coefficients:

$$F(n) = \begin{cases} 1 & \text{if } n = 0 \\ 8F(n-1) + 10^{n-1} & \text{for } n > 0 \end{cases}$$

$$T(n) = \begin{cases} 2 & \text{if } n = 1 \\ 3 & \text{if } n = 2 \\ 5 & \text{if } n = 3 \\ 5T(n-1) - 8T(n-2) + 4T(n-3) + n^2 2^n & \text{for } n > 3 \end{cases}$$

- Find the solution for each recurrence by following the five resolution steps discussed in class.

**Exercise 7**

For each of the following recurrences, provide an expression for the execution time $T(n)$ if the recurrence can be solved using the general theorem (master theorem). Otherwise, explain why the theorem does not apply.

1. $T(n) = 9T\left(\frac{n}{3}\right) + n$
2. $T(n) = T\left(\frac{2n}{3}\right) + 1$
3. $T(n) = 3T\left(\frac{n}{4}\right) + n \log n$
4. $T(n) = 2^n T\left(\frac{n}{2}\right) + n$
5. $T(n) = 2T\left(\frac{n}{2}\right) + n \log n$
6. $T(n) = 0.5T\left(\frac{n}{2}\right) + n$
7. $T(n) = 4T\left(\frac{n}{2}\right) + \frac{n^2}{\log n}$
8. $T(n) = 64T\left(\frac{n}{2}\right) + n^2 \log n$