1. Write a recursive programme to calculate the faculty $n!$ of a natural number $n$.

2. Let’s consider binary representations (i.e., 2-based representations) and hexadecimal representations (i.e., 16-based representations) of non-negative integer decimal numbers (i.e., 10-based numbers). Leave out of consideration possible unnecessary leading zeros. For instance the number 50 has the binary representation 110010 and hexadecimal representation 32. The hexadecimal numbers are $0, 1, \ldots, 9, A, B, C, D, E, F$.

   (a) Derive a general formula for the length of a decimal number in binary representation.

   (b) Derive a general formula for the length of a decimal number in hexadecimal representation.

We use the following notation ($x$ is a real number):

\[ [x] = \max\{k \in \mathbb{Z} | k \leq x\} \]

\[ [x] = \min\{k \in \mathbb{Z} | k \geq x\} \]

3. Prove by induction that

   (a) \[ \sum_{i=1}^{n} (2i - 1) = n^2 \]

   (b) \[ \sum_{i=0}^{n} 2^i = 2^{n+1} - 1 \]

4. This exercise is about recursion.

   (a) Make a method that given as parameter a number, prints that number of stars (*) using recursion according to this model:

   ```java
   private static void stars(int n) {
       // print n stars
       // call the method to print n-1 starts
   }
   
   public static void main(String[] args) {
       stars(3);
   }
   ```

   Example:

   ```java
   ```

   gives as output:
(b) Change the program slightly to get the stars printed in reverse order like this
calling \texttt{stars}(3):

*  
**  
***  

You just need to place the recursive call in differently.

(c) A recursive method can call itself many times. Put two calls to itself in your
method, so that \texttt{stars}(2) would give the following result:

*  
**  
*  

and \texttt{stars}(3) would give:

*  
**  
*  
***  
*  
**  

5. It is well known that the binomial coefficients satisfy the recurrence

\[
\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}
\]

with the boundary conditions \(\binom{n}{0} = \binom{n}{n} = 1\) for all \(n \in \mathbb{N}\). These are the equations
that give rise to Pascal’s triangle.

(a) Write a recursive Java method that computes the binomial coefficient \(\binom{n}{k}\) based
directly on the recurrence given above.

(b) Write a Java method that computes the binomial coefficient \(\binom{n}{k}\) without recur-
sion. (One way to do this is to construct the appropriate part of Pascal’s triangle.)

Would you expect either the recursive or non-recursive solution to be significantly
faster?

6. We know that a certain algorithm takes 1 ms to run for input size \(n = 100\). How
long does it take for input size 10000, when the running time of the algorithm is

(a) \(\Theta(\log n)\)

(b) \(\Theta(n \log n)\)

(c) \(\Theta(n^2)\)

2
(d) $\Theta(2^n)$?

In each case assume that the lower order terms can be neglected. For example, in part (c) assume that the running time is $cn^2$ for some constant $c$.

7. (a) Sum $1 + 2 + 3 + \ldots + n$ can be calculated with the following algorithm:

```text
sum = 0
for i = 1 to n
    sum = sum + i
```

Define the running time complexity (the running time) and space complexity (the amount of extra memory) of the algorithm.

Define an invariant for the algorithm and use that to show that algorithm works correctly.

(b) Continuing from last week’s problem 4. You can calculate the sum $1+2+\ldots+n$ recursively as follows:

```text
Sum(k)
if k==0
    return 0
else
    return sum(k-1) + k
```

Define the time and space complexity for the algorithm with input $n$. Can you calculate the sum so that the time complexity is only $O(1)$?

8. The following algorithm evaluates the polynomial function $p(x) = a_0 + a_1x + a_2x^2 + \ldots + a_kx^k$, when the constant coefficients $a_0, \ldots, a_k$ are stored in an array $A[0..k]$:

```text
Evaluate-Polynomial(A,x)
1   s = 0
2   for i = 0 to k
3       z = 1
4       for j = 1 to i
5           z = z * x
6       s = s+A[i] * z
7   return s
```

Simulate algorithm with function $p(x) = 2x^3 - 3x^2 - 7$ when $x = 2$.

What is the running time of the algorithm? (Assume arithmetic operations take a constant time.) What is the loop invariant for the outer for loop?

9. We study people at a reception. A celebrity is defined as somebody all the others know, but the celebrity knows nobody else. Design an algorithm to find who of the present ones is a celebrity, if any, using questions "Does person x know person y?". The algorithm should work in time $O(n)$, if there are $n$ persons at the reception.

10. Which of the following claims are true, which not? Give a motivation for your answer.

(a) $\log n = O(10^9)$
(b) $\log(n^2) = \Theta(\log n)$
(c) $\log(n^2) = \Omega((\log n)^2)$
(d) $4^n = \mathcal{O}(2^n)$
(e) $\sum_{i=0}^{n} i^k = \Theta(n^{k+1})$

11. Let $f, g : \mathbb{N} \rightarrow \mathbb{R}^+ \cup \{0\}$. Define $f(n) = o(g(n))$, if $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$. Prove that if $f = o(g)$, then $f = \mathcal{O}(g)$, but $g \neq \mathcal{O}(f)$.

12. Discussion about the $\mathcal{O}$-notation:

(a) Consider an algorithm of $n$ numbers and the task is to count the sum of the numbers. Why is $\mathcal{O}(n)$ the best possible time complexity?

(b) Space complexity refers to how much memory the algorithm uses in addition to the input. It is not uncommon that the time complexity of an algorithm is $\mathcal{O}(n)$, but the space complexity only $\mathcal{O}(1)$. However, is it possible that an algorithm’s time complexity is $\mathcal{O}(1)$ but its space complexity is $\mathcal{O}(n)$?

(c) Consider a table of integers:

| 5 | 2 | 1 | 3 | 1 |

The table can be encoded as an integer using prime numbers, in the following manner: $2^5 \cdot 3^2 \cdot 5^1 \cdot 7^3 \cdot 11^1 = 5433120$. This procedure is reversible: from an encoding as a number we can retrieve the original numbers. So why use five places in the table, when one is enough:

| 5433120 | - | - | - | - |

It seems that the original table had space complexity $\mathcal{O}(n)$, and the new one $\mathcal{O}(1)$. Correct?

13. Analyze the time and space complexities of the following recursive functions.

(a) private static void rek1(int n) {
    System.out.println(n);
    if (n != 0) {
        rek1(n-1);
    }
}

(b) private static void rek2(int n) {
    System.out.println(n);
    if (n != 0) {
        rek2(n-1);
        rek2(n-1);
    }
}

14. Given a singly linked list, make a fast algorithm to reverse the list, i.e., to produce a list with the elements in opposite order.

15. In a singly linked list there are only next pointers, no prev pointers. This makes it more difficult to implement the DELETE operation. Let’s implement lazy deletion: We
add a field deleted to each node of the list, and deletion of an element is performed simply by switching its deleted value into True. In addition, we keep count of deleted nodes in the structure. When the number of deleted nodes exceeds half the total length of the whole list structure, we clean up the structure by actually removing the deleted nodes.

Give in pseudocode implementations for the list operations SEARCH, INSERT and DELETE. Estimate the efficiency of the implementations compared to the doubly linked implementation presented in Chapter 10.2 of the book.

16. Consider the following game: N persons are sitting in a circle. The persons are considered in order. Every second one has to leave the circle. Give an algorithm that tells which person remains as the last remaining person. What is its time complexity?

   For instance, if N = 5, the persons leaves in the following order: 2, 4, 1, 5. Thus the answer in this case is 3.

17. Show how to implement the queue as a linked structure so that the operations take constant time. Implementation here means a similar presentation in pseudo-code, or a version written 'as clearly as possible' in some language. It is not allowed to use the existing libraries for real languages.

   Do not forget to illustrate the function of the queue operations with figures etc.

18. Show how to simulate a queue with the help of two stacks. Assume here that the queue and stacks are unlimited in size. What is the time complexity for the queue operations?

19. A linked list can be represented as an array, where links to the nodes are simply indices of the array. For example, the (unordered doubly-linked) list (68, 24, 15, 17) might have an array representation

   \[
   \begin{array}{ccc}
   key & next & prev \\
   1: & & \\
   2: & & \\
   3: & 24 & 7 \\
   4: & & \\
   5: & 68 & 3 \\
   6: & & \\
   7: & 15 & 8 \\
   8: & 17 & 0 \\
   9: & & \\
   10: & & \\
   \end{array}
   \]

   head: 5

   Show in pseudocode how to implement INSERT and DELETE using such an array representation. Your solution should be such that an array with n rows is sufficient if the list never contains more than n elements. In other words, the INSERT procedure must be able to re-use the rows released earlier by DELETE. Try also to avoid unnecessarily moving elements around.

   Note: maintain a separate list of free rows.
20. It is possible to store a polynomial in an array having the size of the maximum degree of the polynomial. However, this may lead to the waste of space. For example, consider the polynomial \( p(x) = 2x^{20000} + x^{10000} + 2 \). We need an array of size 20001 although the polynomial contains only three terms. An alternative way is to use a linked list. Each node contains a degree (exponent of \( x \)) and a coefficient of one term.

Write in Java an algorithm which calculates the sum of two polynomials which are stored like described above. The algorithm should not update the original lists but create a new list for the sum. What is the time complexity of your algorithm (in \( O \) notation)?

21. Program Java methods to compute the

(a) height  
(b) number of leaves  
(c) sum of key values of the nodes

of a binary tree. \textit{Hint:} using recursion will help.

22. We consider the implementation (with Java or some other programming language) of a stack based on an array. Implement the push operation. Increase the size of the array if there is no place for a new element in the array when a push operation is made.

Implement two versions of the stack: one where the size of the array is always doubled, and another where the size always increases with 100. Both arrays start out with the size 100.

Compare the performance of your implementations empirically. Again, we are most interested in large inputs. Draw an figure of the time spent by each version in relation to the number of performed push operations. What conclusions can you draw from an empirical analysis of the stack performances? Why is there a difference in performance?

23. Write the following unambiguous binary tree (it is not a search tree):

Its nodes listed by preorder tree walk are: 8, 13, 7, 11, 4, 10, 1, 5, 14, 17, 3, 18
Its nodes listed by inorder tree walk are: 7, 13, 4, 11, 10, 8, 5, 1, 17, 3, 14, 18

24. Give an algorithm that checks if two binary trees are similar.

A and B below are similar. A and B are not since the structures differ. A and D are not similar since the keys differ.

```
   3     3     3     3
  / \   / \   / \   / \  
 2  5  2  5  2  5  2  5
 / \ / \ / \ / \ / \ 
1  4 1  4 1  4 4  1

A   B   C   D
```

25. Give an algorithm (in pseudocode) that performs an inorder tree walk without using recursion.
26. Assume a binary tree has $m$ leaves $l_1, \ldots, l_m$, and the depths of the leaves are $d_1, \ldots, d_m$. Show that

$$\sum_{i=1}^{m} 2^{-d_i} \leq 1.$$ 

When does this hold as equality? (This result is a special case of Kraft’s inequality, which is central in coding theory.)