**Question 1.** (a) Perform a BFS and a DFS on each of the graphs below. (b) Show 2 different spanning trees for each graph. (c) For any given graph, imagine that it is a labyrinth and that we are given as inputs a vertex that is an entrance and another one that is an exit. Write the pseudo-code for an algorithm that enters the labyrinth and exits it. (d) Design an ADT for a labyrinth and give the pseudo-code for the functions that it comprises. What is the asymptotic complexity of each function? Give tight lower and upper bounds. What if a graph was infinite?

![Graphs](image)

**Question 2.** Write pseudo-code for an algorithm that solves the longest common subsequence problem, that is, given two input strings over the alphabet $\Sigma = \{a, b\}$, find the longest succession of symbols (not necessarily consecutive) that appears in both strings. Hint: you may start by finding only the length of the longest such string. What is the complexity of this algorithm? Is it different from the one for the longest increasing subsequence? Why?

**Question 3.** Fibonacci numbers are defined as:

$$F_n = F_{n-2} + F_{n-1}$$

with $F_1 = F_2 = 1$. Write pseudo-code for an algorithm that evaluates the $n$th Fibonacci number. What is its running time (give a formal proof)? If it does not run in linear time, use dynamic programming to make it, and reanalyse your algorithm.

NB: The Master Theorem cannot be applied for this problem since the recurrence is not of the appropriate form.

**Question 4.** Consider the tokens in the mini programming language of the lectures of weeks 12 and 13. (a) Give a hash function that hashes all the reserved keywords perfectly. (b) Suppose that the compiler for the mini programming language needs to produce a syntax tree from the source code and the language’s vocabulary (reserved words, variable names, etc.) and grammar. Design an ADT that deals with such syntax trees. What is the complexity of its functions?

**Question 5.** Show that $n^2 \in O(2^n)$. Show that $2p \cdot \sum_{k=2}^{p} \left(1 - \frac{1}{k^2}\right) \in \Omega(p)$. Prove that for every $n \geq 1$, $1(1!) + \ldots + n(n!) = (n + 1)! - 1$. 

1
Question 6. Write an LL(1) grammar for the following context-free grammars and write the pseudo-code of a parser for it. (a) For $\Sigma = \{\langle, \rangle, \{, \}\}$ the grammar $G$ that generates the language:

$$L(G) = \{w : w \text{ is a string of balanced parentheses}\}$$

(b) For $\Sigma = \{a, b\}$ the grammar $G$ that generates the language:

$$L(G) = \{a^i b^j a^k b^l : i, j \geq 0\}$$

Question 7. Design an ADT for the secure storing of polynomials via RSA. Suppose that Alice is really paranoid about her polynomials, but still wants Bob to have access to them. No one else should be able to look them up easily. (a) Use a heap to store the encrypted polynomials. (b) Use a hash table to store them. (c) Use a doubly linked list. (d) Use a BST. (e) What if Alice wants to allow Bob read-only access and Charlie, read and write access (i.e. Charlie can also delete or modify the polynomials)? (f) What is the complexity of all these functions to be implemented?

Question 8. Trace, i.e. show the intermediary array states of a run, the counting sort algorithm on this list of integers between -9 and 9: [1, 4, -4, 3, 6, -2, -2]. What is the tight asymptotic lower bound for this sort?

Question 9. For-loops (which have a fixed iteration limit) are a special case of while-loops. A function which can be implemented using only for-loops (and no stacks!) is called primitive recursive. (In contrast, a computable function can be coded using a combination of for- and while-loops, or while-loops only.) Examples of primitive recursive functions include power, greatest common divisor, and $p(n)$ (the function giving the $n$th prime).

The Ackermann function is the simplest example of a well defined total function which is computable but not primitive recursive, providing a counterexample to the belief in the early 1900s that every computable function was also primitive recursive.

Give the pseudo-code for computing the Ackermann function which is defined as:

$$A(x, y) = \begin{cases} 
    y + 1 & \text{if } x = 0 \\
    A(x - 1, 1) & \text{if } y = 0 \\
    A(x - 1, A(x, y - 1)) & \text{otherwise}
\end{cases}$$

Question 10. TAK is a recursive function devised by I. Takeuchi. For integers $x, y,$ and $z$, and a function $h$, it is

$$TAK_h(x, y, z) = \begin{cases} 
    h(x, y, z) & \text{if } x \leq y \\
    TAK_h(TAK_h(x - 1, y, z), TAK_h(y - 1, z, x), TAK_h(z - 1, x, y)) & \text{if } x > y
\end{cases}$$

Give the pseudo-code for computing the TAK function for a general function $h$. Show that the number of function calls required to compute $TAK_0(a, b, 0)$ for $a, b > 0$ is

$$F_0(a, b, 0) = 4 \sum_{k=0}^{b} \frac{a - b}{a + b - 2k} \left( \frac{a + b - 2k}{b - k} \right) - 3 = 1 + 4 \sum_{k=0}^{b-1} \frac{a - b}{a + b - 2k} \left( \frac{a + b - 2k}{b - k} \right)$$