Finite automata and regular languages:

1a. Design a finite automaton for recognizing binary sequences that end with 0. Assume that the input strings contain only symbols 0 and 1. The easiest is to have two states:
   - the state z indicating that the last read symbol was 0, and
   - the state n indicating that either no symbol was read yet or the last symbol read was 1.

   You just need to describe transitions between these states, and which states are final. Show, step-by-step, how your automaton will accept the string 110 (that corresponds to binary number 011).

1b. Explain why in most computers binary numbers are represented starting with the lowest possible digit.

1c. On the example of the above automaton, show how the word 110 can be represented as xyz in accordance with the pumping lemma.

1d. Use a general algorithm to describe a regular expression corresponding to the finite automaton from the Problem 1a. (If you are running out of time, it is Ok not to finish, just eliminate the first state.)

1e-f. The resulting language can be described by a regular expression (0 U 1)*0. Use a general algorithm to transform this regular expression into a finite automaton: first a non-deterministic one, then a deterministic one.
2. Beyond finite automata: pushdown automata and context-free grammars:

2a. For a constitutional amendment to become official, it has to be approved by at least 3/4 of the states. For example, if the states voted YYNY or YNYYNYY, where Y stands for Yes, and N for no or undecided, the amendment will be approved. Prove that the language consisting of all sequences of Y and N that lead to the approval is not regular.

2b. Use a general algorithm to transform the finite automaton from the Problem 1a into a context-free grammar (CFG). Show, step-by-step, how this CFG will generate the word 110.

2c. For the context-free grammar from the Problem 2b, show how the word 110 can be represented as uvxyz in accordance with the pumping lemma.

2d. Use a general algorithm to translate the CFG from 2b into Chomsky normal form.

2e. Use a general algorithm to translate the CFG from 2b into an appropriate push-down automaton. Explain, step-by-step, how this automaton will accept the word 110.

2f. Use the general stack-based algorithms to show:

- how the compiler will transform a Java expression $7 - 1 - 3$ into inverse Polish (postfix) notation, and
- how it will compute the value of this expression.

2b

2c

2d

2e

2f

2b

2c

2d

2e

2f
2f) \[ 7 - 1 - 3 \]

\[ \frac{4}{3} - \frac{1}{3} \]
3. Beyond pushdown automata: Turing machines

3a. In a fictitious state of Saxet, four universities A, B, C, and D -- located at different cities -- compete for state funding. They want to make sure that each got an equal number of funds. So, e.g., if we the sequence of allocated grants is ABABCCDD, this is good. Prove that the set of all "good" sequences is not context-free and therefore, cannot be recognized by a pushdown automaton.

3b-c. Use a general algorithm to design a Turing machine that accepts exactly all sequences accepted by a finite automaton from Problem 1a. Show, step-by-step, how this Turing machine will accept the word 011. Describe, for each step, how the state of the tape can be represented in terms of states of two stacks.

3d-e. Design Turing machines for computing $a - 2$ in unary and in binary codes. Trace both for $a = 3$, i.e., for $a = 111$ in unary code and $a = 11$ in binary code.

\[
\text{Start, } \# \rightarrow \text{read, } X
\]
\[
\text{read, } 1 \rightarrow a
\]
\[
\text{read, } \# \rightarrow \text{sub } 2, \#L
\]
\[
\text{sub } 2, 1 \rightarrow \text{sub } 1, \#L
\]
\[
\text{sub } 2, \# \rightarrow \text{halt}
\]
\[
\text{sub } 1, 1 \rightarrow \text{done, } \#L
\]
\[
\text{sub } 1, \# \rightarrow \text{halt}
\]
\[
\text{done, } 1 \rightarrow L
\]
\[
\text{done, } \# \rightarrow \text{halt}
\]

\[
\text{Start, } \# \rightarrow \text{read, } a
\]
\[
\text{read, } 0 \rightarrow z, R
\]
\[
\text{read, } 1 \rightarrow \text{halt}
\]
\[
\text{z, 0 \rightarrow z, R}
\]
\[
\text{z, 1 \rightarrow \text{halt}}
\]
\[
\text{z, } \# \rightarrow \text{reject}
\]
\[
\text{v, } \# \rightarrow \text{accept}
\]
Start, # → prep, R
prep, 0 → sub2, R
prep, 1 → sub2, L
sub2, 0 → sub2, 1, A
sub2, 1 → done, 0, L
sub2, # → halt
done, 0 → R
done, 1 → 0
done, # → halt

$101_2 = 5_{10}$}

$101_2 = 5_{10}$

$011_2 = 3_{10}$
3a. assume L is CFG, then by pumping lemma

\[ \text{let } w \in L \text{ s.t. } |w| \geq p \Rightarrow \text{ exist } x, y, z \text{ s.t. } |yz| \leq p \text{ and } |y| > 0 \]

Let's take the word \( w = A^p B^p C^p D^p \text{ so the length is } 4p \geq p \) well, we can represent \( w = uu \cdots uwy ) \Rightarrow u \cdot A \cdot B \cdot C \cdot D \cdot D \cdot D \cdot D \cdot D \)

\( UV \) is not contain A, B, C, and D because its length would be greater than \( p \)

So we can add \( A^j, B^j, C^j \), and \( D^j \) both A, B, C, and D's, \( A^j, B^j, C^j \), or \( B^j, C^j \) and \( D^j \)

In each option the letters we add and by pumping lemma

the letter we add contain more than the letters we didn't add.

So \( UV \) is a \( L \) but by pumping lemma \( UV \) is \( L \)

Therefore by Contradiction \( L \) is not CFG.
4. Beyond Turing machines: computability

4a. Formulate Church-Turing thesis. Is it a mathematical theorem? Is it a statement about the physical world?

4b. Prove that the halting problem is not algorithmically solvable.

4c. Not all algorithms are feasible, but, unfortunately, we do not have a perfect definition is feasibility. Give a current formal definition of feasibility and give two examples:

- an example of an algorithm's running time which is feasible according to the current definition but not practically feasible, and
- an example of an algorithm's running time which is practically feasible but not feasible according to the current definition.

4d. Briefly describe what is P, what is NP, what is NP-hard, and what is NP-complete. Is P equal to NP?

4e. Give an example of an NP-complete problem: what is given, and what we want to find.

4f. Give definitions of a recursive (decidable) language and of a recursively enumerable (Turing-recognizable) language.

4a. Thesis: Anything that can be computed on any physical device can also be computed on a Turing machine.

- it is not a mathematical theorem it is a statement about the physical world.

4b. Proof (by contradiction) lets assume such an algorithm exists, where given a program P and data d checks whether P halts on d.

Suppose that there is a program haltchecker (P, d) that given P halts on d

```java
public static void ex(String x) {
    if (haltchecker(x, x)) {
        while (true) {
            x = x;
        }
    }
}
```

Case 1: ex halts on ex, then haltchecker(ex, ex) is true, the program enters an infinite loop and does not halt.

Case 2: ex does not halt on ex then the program will skip the if statement and halt.

\[ \therefore \text{both cases we get a contradiction, so our initial assumption is false and haltchecker does not exist.} \]
4c. Feasible algorithm: An algorithm is feasible if there exists a polynomial \( P(n) \) such that for every input \( x \), the algorithm finishes on or before time \( t_A(x) \leq P(\text{len}(x)) \). Feasible by definition \( \Rightarrow 0 \leq t_A(n) = 10^{100} \cdot n \)

Practically \( 0 \leq t_A(n) = \exp(10^{-20} \cdot n) = e^{10^{-20} \cdot n} \) feasible

4d. \( P \): Class of problems that can be solved in polynomial time.

\( \text{NP} \): Class of problems for which, once you have a candidate for a solution, you can feasibly check whether it is a solution.

\( \text{NP-Hard} \): Every problem from class \( \text{NP} \) can be reduced to this problem. That is, problems harder than those of \( \text{NP} \).

\( \text{NP-complete} \): A problem is \( \text{NP-complete} \) if it is \( \text{NP-hard} \) and in the class \( \text{NP} \).

\( P \equiv \text{NP} \): \( P = \text{NP} \) has not been proven to be true or proven to be false.

4e. Propositional Satisfiability - given a boolean expression, find values of variables that make it true, needed for testing
4f. recursive language $\equiv$ decidable language
if there is an algorithm that given a word $w$ checks whether $w \in L$ through boolean outputs

an example of a language which is not decidable:
set of all pairs $(p, d)$ such that $p$ halts on $d$

$\exists (p, d) : p$ halts on $d$?

Recursive enumerable - a language is Turing-reconizable if there is a Turing machine that produces the results "accept" if and only if $w \in L$. (eventually prints all elements)
5. (for extra credit) Parallelism

5a. If you had an unlimited number of processors, how can you find the smallest of \( n \) numbers by using the smallest possible time? Explain step by step. How much time did this computation take? How many processors did you use?

5b. In general, how many processors do you need to compute the smallest of \( n \) numbers in parallel? How much time will it take?

5c. Describe the class NC. Is it equal to P? Give an example of a P-complete problem.

5d. What if we take into account communication time? How much time will you then need to find the smallest of \( n \) numbers? Explain your answer.

\( \leq, \geq \) means comparison

\[
\begin{align*}
\leq, \geq & \quad \leq, \geq & \quad \leq, \geq & \quad \leq, \geq \\
\leq, \geq & \quad \leq, \geq & \quad \leq, \geq & \\
\end{align*}
\]

\( n = 8 \)

![Diagram of comparisons]

Used 8 processors,
Time taken = 3

5b) To compute the smallest of \( n \) number, the time it takes is \( t = \log_2(n) \).
You need \( n \) processors.

5c) NC (Nick's Class): Anything computable on polynomial number of processors in polylog time \( \equiv \text{time} = P(\log(\log(n))) \).
We don't know if NC \( \subseteq \) P.

P-Complete problem example: Linear Programming \( \Rightarrow \) Finding a solution to a system of linear inequalities.

5d) we are here

\[
R = C \cdot T_{\text{par}}
\]

Sphere \( V = \frac{4}{3} \pi R^3 = \frac{4}{3} \pi C^3 \cdot T_{\text{par}} \)

\( \Delta V = \text{vol of processors} \)

\[
N_{\text{proc}} = \frac{4}{3} \pi C^3 \cdot T_{\text{par}} \cdot \frac{\Delta V}{\Delta V}
\]

\( T_{\text{seq}} \leq C \cdot T_{\text{par}} \)

\[
T_{\text{par}} \geq C \cdot T_{\text{seq}}^{\frac{1}{14}}
\]

\[\sqrt[14]{n} \Rightarrow \log_2(n)\]