1-2. Prove that the language \( L = \{ a^n b^2 c^n : n = 0, 1, 2, \ldots \} \) is not context-free.

\[
L = \{ a^n b^2 c^n : n = 0, 1, 2, \ldots \} = \{ \Lambda, abc, aabbcc, aaabbbccc, \ldots \}
\]

is not CFG.

Proof by contradiction. Let's assume that \( L \) is CFG. Then, by pumping lemma, there exists a natural number \( p \) such that every word \( w \) from \( L \) can be represented as \( uvxyz \), where \( \text{len}(vy) > 0 \), \( \text{len}(vxy) \leq p \), and for each \( c \), \( uv^i xy^i z \in L \).

Let's take \( w = a^p b^2 c^p = a \ldots a b b \ldots b c \ldots c \)

\[
\text{len}(w) = 4p > p,
\]

so \( w \) can be represented as \( uv^i x y^i z \). \( \text{len}(vxy) \leq p \), so \( vxy \) cannot cover all 3 parts, else we will have \( \text{len}(vxy) > p \).

We have several options:

1. \( vxy \) is in \( a \)'s. Then, when we pump, we add \( a \)'s but not \( b \)'s & \( c \)'s, so we got a disbalance, so \( uv^i x y^i z \notin L \).
2. \( vxy \) can be in \( a \)'s & \( b \)'s but we don't add \( c \)'s - so \( uv^i x y^i z \notin L \). We get a disbalance.
3. \( vxy \) is in \( b \)'s. We add \( b \)'s but not \( a \)'s & \( c \)'s so when we pump, we get a disbalance and \( uv^i x y^i z \notin L \).
4. \( vxy \) in \( b \)'s & \( c \)'s - add \( b \)'s & \( c \)'s, don't add \( a \)'s, \( uv^i x y^i z \notin L \).
5. \( vxy \) in \( c \)'s, add \( c \)'s but not \( a \)'s & \( b \)'s \( uv^i x y^i z \notin L \).

but by pumping lemma \( uv^i x y^i z \notin L \) - a contradiction.
3. Trace the following Turing machine on the example of the word 10: start, _ → work, R (here, _ means blank); work, 0 → R; work, 1 → 0, R; work, _ → back, L; back, 0 → L; back, _ → halt. Explain how each step will be represented if we interpret the Turing machine as a finite automaton with two stacks.
4. Design a Turing machine that subtracts 1 from a binary number. Trace your Turing machine, step-by-step, on the example of the string 100. Why in Turing machines (and in most actual computers) the representation of a binary number starts with the least significant digit?

```
Rules:
Start: # → go, R
go: 0 → 1, L
1 → 0, rewind, R
rewind, # → halt.
```

```
Start:

# 1 0 1

Start: go

# 1 0 1

Start: go

# 1 1 1

Start: go

# 1 1 0

# 1 1 0

# 1 1 0

rewind, rewind, rewind.

rewind, rewind, rewind.

rewind, rewind, rewind.

rewind, rewind, rewind.

rewind, rewind, rewind.

Why? Every operation always starts with the least significant digit. So we do it here as well to save us time.
```
5. The following finite automaton describes odd binary numbers. This automaton has the starting state s, the final state d, an additional state e, and the following transitions: from each of the three states, 0 leads to e, while 1 leads to d. Use the general algorithm to transform this finite automaton into a Turing machine. Show, step-by-step, how your Turing machine will accept the string 01.

START, # → S, R
s, 0 → e, R
s, 1 → d, R
s, # → reject

e, 0 → e, R
e, 1 → d, R
e, # → reject
d, 0 → e, R
d, 1 → d, R
d, # → accept

# 0 1 #
↑ start

# 0 1 #
↑
S

# 0 1 #
↑
e

# 0 1 1 #
↑
d → accept

# 0 1 1 1 1 #
↑
START S e d d accept
6. Give the formal definition of a feasible algorithm. Give two examples:

1. of a computation time which is formally feasible, but not practically feasible, and
2. a computation time which is practically feasible but not formally feasible.

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 $P(len(x))$

1. $10^{100}n$
2. $e^{10^{-20}}n$
7. What is P? What is NP? What does it mean for a problem to be NP-hard? NP-complete? Give brief definitions. Give an example of an NP-complete problem. Is P equal to NP?

**P** - Problems that can be solved in feasible time.

**NP** - A problem of instance: given x, find y such that \( C(x, y) \) is true & lengthy \( \leq P(\text{len}(x)) \) or give a message that there is no such y.

**NP-Hard** - A problem (not necessarily NP) is defined if every problem from the class NP can be reduced to it.

**NP-Complete** - A problem from the class NP that is also NP-Hard.

Is P equal to NP? No one really knows.

Example of NP-Complete: SAT - Propositional Satisfiability

We have a propositional formula (boolean expression), everything obtained from boolean variables \( x_1, x_2, \ldots, x_n \) by using \&, (\text{or}), 1 (\text{!}) \text{ Ex:} (x_1 \& x_2) \text{ or } (x_7 \text{ or } \overline{x_2} \& x_3)

We want: Find the values of these boolean variables that make the formula true (or a message that it's not possible).

\[
x_1 = 1 \quad x_2 = 0 \quad x_3 = 1
\]
8. Prove that the square root of 7 is not a rational number.

Theorem $\sqrt{7}$ is not rational

Proof: by contradiction. Let's assume it is rational, i.e. $\sqrt{7} = \frac{a}{b}$ for some $a$ and $b$.

If $a$ and $b$ have a common factor, we can divide both numerator and denominator by this factor and get a ratio $\frac{a'}{b'}$ where $a'$ and $b'$ have no common factor.

We square both sides $7 = \frac{a^2}{b^2}$, we multiply by $b^2$, $7b^2 = a^2$.

LHS is divisible by 7, so RHS ($a^2$) must be divisible by 7.

$a \cdot a$ is divisible by 7, this means that $a = 7p$ for some integer $p$. Plug in $a = 7p$ into $7b^2 = a^2$, we get

$$7b^2 = (7p)^2 = 49p^2$$

we can divide both sides by 7, $b^2 = 7p^2$.

Similar arguments show that $b$ is also divisible by 7, so $a$ and $b$ have a common factor, which contradicts to our assumption.

So, $\sqrt{7}$ is not rational.
9. Formulate the halting problem. Prove that it is not possible to check whether a given program halts on given data.

Theorem: no algorithm is possible for solving the halting problem

no algorithm is possible that, given a program $p$, and data $d$, checks whether $p$ halts on $d$.

Proof: (by contradiction) Suppose that there is a program

```java
public static boolean ex(String x)
    if (h(x, x))
        while (true) x = x + 1
    else return true;
//Main
boolean t = h(ex, ex);
```

What is $h(ex, ex)$?

Case 1: $ex$ halts on $ex$, then the $h(ex, ex)$ is true, so the the program goes into an infinite loop and so it doesn't halt.

Case 2: $ex$ doesn't halt on $ex$, then the program will skip the if-statement and halt.

In both cases, we get a contradiction. So, our initial assumption is false, and $h$ (haltechecker) does not exist.
10. Formulate Church-Turing thesis. Is it a mathematical theorem? Is it a statement about the physical world?

Church-Turing Thesis: anything that can be computed on any physical device can also be computed on a Turing Machine / by a Java program

Is it a mathematical thesis?
- No, it's not a mathematical thesis,
  it's a statement about the physical world