1. **Genetic algorithms**
   
   (1) Tic-Tac-Toe
   
   **State description:** Assume that the agent plays “X”. There are \( n \) situations for Tic-Tac-Toe, and \( n < 3^9 \), so agent will have \( n \) rules. For each situation, there will be 9 possible moves, which means that the agent will place “X” in one of the 9 squares on the board. So the state is a string consisting of \( n \) characters, each character will be one of 1~9.

   **Fitness function:** For each of the characters, if the agent wins after applying the move corresponding to the value of the character, gains a score 1; if ties, gains score 0; if loses, gains a score -1.

   (2) Control an elevator
   
   **State description:** assume there are \( n \) floors; for each floor, there are 3 possible states: request to up, request to down, or no requests. So there are altogether \( 3^n \) states. For each state, the elevator may be at any level and going up or down, so it has \( 2n \) actions. We use each one number in 1-2\( n \) to indicate these actions. Our string is \( 3^n \) characters corresponding to \( 3^n \) states, each is one of 1-2\( n \).

   **Fitness function:** For each of the character, we get the smallest number of floors the elevator has to go before it stops for the current action. Sum this value over all characters to get the overall score for a string.

2. **Algorithm names**
   
   1. hill-climbing search
   2. Exhaustive search
   3. Hill-climbing search
   4. Random walk
   5. Random walk

3. **Positive linear transformation of leaf values in game tree**
   
   First of all, if \( x_1 \geq x_2 \), then obviously \( ax_1 + b \geq ax_2 + b \). Then we prove that if leaf values are transformed using positive linear transformation, then all the \( \text{minimax} \) values are transformed using the same positive linear transformation formula.

   We use \( \text{mm}(n) \) to denote the old minimax value, \( \text{mm'}(n) \) to denote the transformed minimax value; \( u(s) \) and \( u'(s) \) to denote the old and transformed utility value of state \( s \) respectively.
We prove by induction:

**Initial case:** assume the last move is MAX (assuming it to be MIN will yield the same result), so

\[
mm'(n) = \max_{s \in \text{successor}(n)} (u'(s))
\]

\[
= \max_{s \in \text{successor}(n)} (a \cdot u(s) + b)
\]

\[
= a \cdot \max_{s \in \text{successor}(n)} u(s) + b
\]

\[
= a \cdot mm(n) + b
\]

**Induction:** assume that for all \( n \)'s successors \( s \), \( mm'(s) = a \cdot mm(s) + b \), then

\[
mm'(n) = \max_{s \in \text{successor}(n)} (mm'(s))
\]

\[
= \max_{s \in \text{successor}(n)} (a \cdot mm(s) + b)
\]

\[
= a \cdot \max_{s \in \text{successor}(n)} mm(s) + b
\]

\[
= a \cdot mm(n) + b
\]

**Conclusion:** Minimax values for all the nodes in the game tree are transformed using the same positive linear transformation formula, so the relationship between successors of the same parent will remain the same as before, then choice of moves remain unchanged.
Here we handled \( ? \) by assuming that either MIN or MAX will avoid \( ? \) when they encounter it. So \( \min(-1, ?) = -1 \) and \( \max(1, ?) = 1 \). If all successors of a state are \( ? \) then the minimax value at that state is \( ? \).

(c) General minimax does a depth first traversal where the value of every internal node is defined by the min/max over its children (whose values are the min/max over their children, etc). So if it encounters a repeated state that is a descendant of itself, its value is not well defined because the value would be defined in terms of itself.

The strategy in (b) above sometimes works but not always. For example, some states could be draws (a utility of zero for both players, say). Then it is not clear if minimax should prefer those over \( ? \). For example, in this game tree, \((1,4)\) and \((2,4)\) are in fact winning positions so they should be preferred over drawn positions though they have a value of \( ? \).

The “right” way to solve such cyclic game trees is through dynamic programming. Here we would assign each state an unknown utility and recursively, iteratively update all the utilities until convergence. The key difference is that this cannot be done in just a single pass over the underlying state space, unlike minimax. We will see this in sequential decision making later in the class.

5. **Prove the following assertions**
   a. If \( \alpha \) is valid, then \( \alpha \) is true for all models; so \( \alpha \) is true in all models where \( \text{True} \) is true (since \( \text{True} \) is also true for all models). So \( \text{True} \models \alpha \).
If True |= α, α is true in every model where True is true. Since True is true for all models, α is true for all models, so α is valid.

b. Suppose it is not the case that False |= α. Then there is some model that makes False true but makes α false. But there are no models that make False true, so this is impossible.

6. **Prove the deduction theorem in propositional logic**
   Proof: Using definitions, we can restate the theorem as α |= β if and only if α => β is true in every model. First, suppose α |= β. Then for every model in which α is true, so is β. This means that for every model, either α is false, making α => β true, or both α and β are true, making α => β true. Therefore, α => β is true in every model.

   On the other hand, suppose α => β is valid. Then the following truth assignments are possible for (α, β): (T,T),(F,T),(F,F). In these, whenever α is true, β is also true, so α |= β.

7. **How many models are there for the following sentences?**
   a. The models: assignments to A, B, C by {1, 1, 0}, {0, 1, 1} and {1, 1, 1}; considering values of D, there are 3*2=6 models altogether.
   b. The models: assignments to A, B by {0, 1}, {1, 0}, {1, 1}; considering values of C and D, there are 3*2*2=12 models altogether.
   c. The models: assignments to A, B, C by {1, 1, 1} and {0, 0, 0}; considering values of D, there are 2*2 = 4 models altogether.

8. **Unicorn: mythical, immortal, horned problem**
   We first define the following variables:

   My: mythical; Mo: mortal; Ma: mammal; H: horned; Mg: magical

   Then we can define the following rules:

   R1: Mg → ~Mo
   R2: ~Mg → (Mo ∧ Ma)
   R3: ~Mo ∨ Ma → H
   R4: H → My

   Transform the above rules as CNF, we get the following KB:

   ~Mg ∨ ~Mo; Mg ∨ Mo; Mg ∨ Ma; Mo ∨ H; ~Ma ∨ H; ~H ∨ My

   (1) Is unicorn magical?

   Add ~Mg to the KB:
No more new clauses can be generated. So we cannot conclude that unicorn is magical.

(2) Is unicorn mythical?

Add ~My to KB:

~My, ~H ∨ My

~H, ~Ma ∨ H

~Ma

~Ma, Mg ∨ Ma

Mg

Mg, ~Mg ∨ ~Mo

~Mo

~Mo, Mo ∨ H

H

H, ~H ∨ My

My

~My, My

□
So unicorn is mythical.

**3. Is unicorn horned?**

Add ~H to the KB.

~H, Mo ∨ H
Mo

~H, ~Ma U H

~Ma

~Ma, Mg U Ma

Mg

Mg, ~Mg U ~Mo

~Mo

~Mo, Mo

So unicorn is horned.

9. **Prove any sentence in propositional logic can be written in CNF**

Let f be a formula of propositional logic, and T be its truth table, where T maps possible variable assignment combination to True or False. The formula makes certain rows \( c_i = l_1 \wedge l_2 \wedge \ldots \wedge l_n \) of the truth table False (each \( l_i \) represents the \( i^{th} \) variable or its negation). Construct the formula \( f' = \neg c_1 \wedge \neg c_2 \wedge \ldots \wedge \neg c_n \). This formula represents the assertion that “each possible world in which f would be false is not the case.” We show that \( f' \) is identical to f.

Each row of the truth table that makes f False corresponds to a row \( c_i \) being True, so makes its negation False and \( f' \) False. Conversely, if \( f' \) is False, then some negated \( c_i \) is False, so some row is True. This assignment will make f False. So \( f \equiv f' \).

By applying DeMorgan’s law to each negated conjunction \( \neg c_i \), we get a disjunction of literals \( c' \) and \( f' \) can be written as a CNF formula: \( f' = (c'_1 \wedge c'_2 \wedge \ldots \wedge c'_n) \).

10. **Is a Random 5CNF with \( m \) clauses and \( n \) symbols more likely to be satisfiable than 4CNF formula?**

It is more likely to be solvable, since for each clause, having more variables means there are more possible assignments than make it True. This is because regardless of the number of variables, only
one assignment makes the clause False. Since it’s more likely that each clause is True, it’s more likely
that every clause (and the resulting formula) is True.