Exercise 1. Suppose there exists a \( O^*(1.2^n) \) time algorithm, which, given a graph \( G \) on \( n \) vertices, computes the size of a largest independent set of \( G \).

Design an algorithm, which, given a graph \( G \), finds a largest independent set of \( G \) in time \( O^*(1.2^n) \).

Exercise 2. Let \( A \) be a branching algorithm, such that, on any input of size at most \( n \) its search tree has height at most \( n \) and for the number of leaves \( L(n) \), we have
\[
L(n) = 3 \cdot L(n - 2)
\]
Upper bound the running time of \( A \), assuming it spends only polynomial time at each node of the search tree.

Exercise 3. Same question, except that
\[
L(n) \leq \max \left\{ \begin{array}{l}
2 \cdot L(n - 3) \\
L(n - 2) + L(n - 4) \\
2 \cdot L(n - 2) \\
L(n - 1)
\end{array} \right. 
\]

Exercise 4. Consider the Max 2-CSP problem

Max 2-CSP

Input: A graph \( G = (V, E) \) and a set \( S \) of score functions containing
- a score function \( s_e : \{0, 1\}^2 \to \mathbb{N}_0 \) for each edge \( e \in E \),
- a score function \( s_v : \{0, 1\} \to \mathbb{N}_0 \) for each vertex \( v \in V \), and
- a score “function” \( s_\emptyset : \{0, 1\}^0 \to \mathbb{N}_0 \) (which takes no arguments and is just a constant convenient for bookkeeping).

Output: The maximum score \( s(\phi) \) of an assignment \( \phi : V \to \{0, 1\} \):
\[
s(\phi) := s_\emptyset + \sum_{v \in V} s_v(\phi(v)) + \sum_{uv \in E} s_{uv}(\phi(u), \phi(v)).
\]

1. Design simplification rules for vertices of degree \( \leq 2 \).
2. Using the simple analysis, design and analyze an \( O^*(2^{m/4}) \) time algorithm, where \( m = |E| \).
3. Use the measure \( \mu := w_e \cdot m + (\sum_{v \in V} w_{deg(v)}) \) to improve the analysis to \( O^*(2^{m/5}) \).