DEPARTMENT OF COMPUTER SCIENCE s ENGINEERING

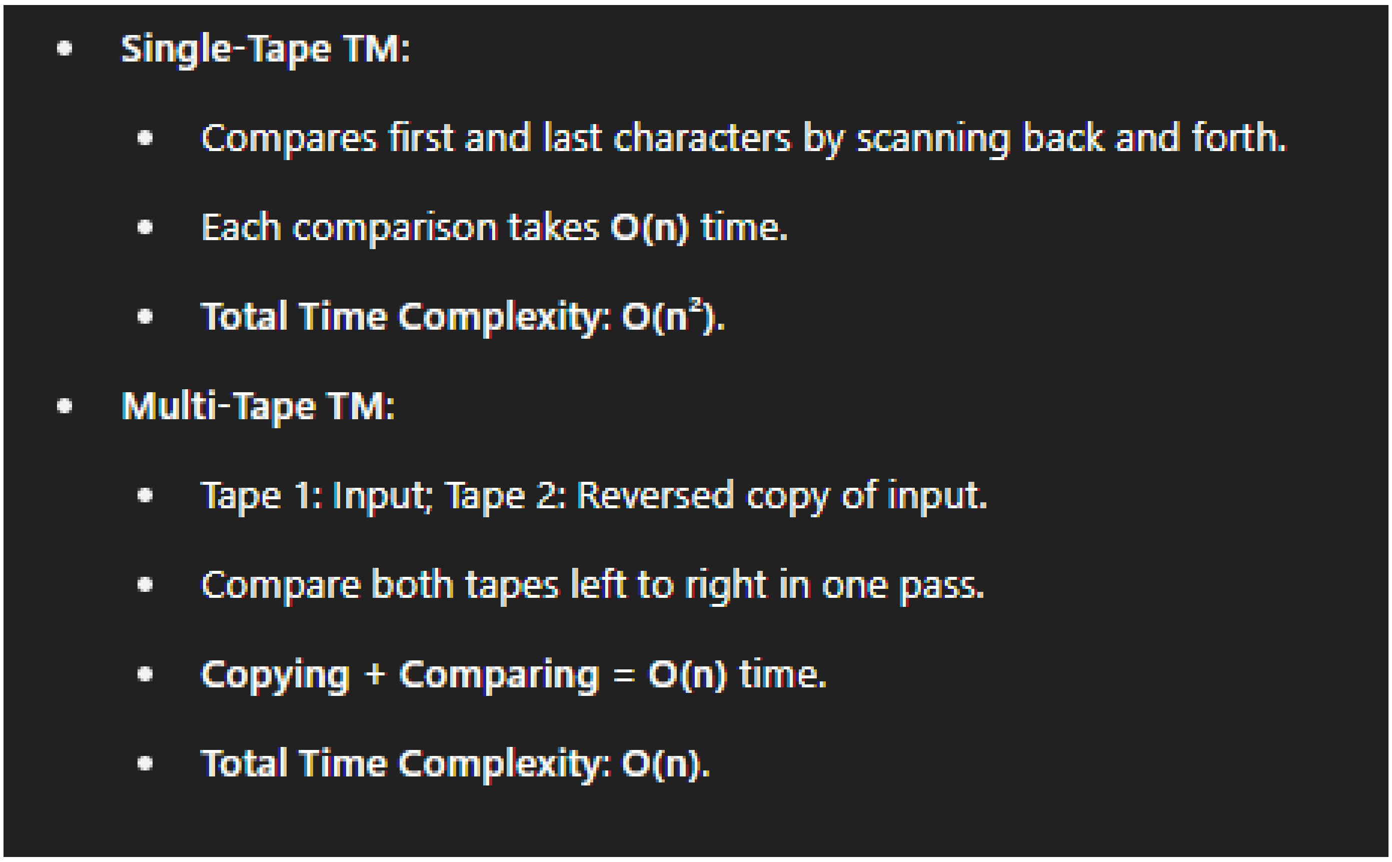
KONERU LAKSHMAIAH EDUCATION FOUNDATION

23MT2014 - THEORY OF COMPUTATION

Home Assignment - 4 (CO - 4)

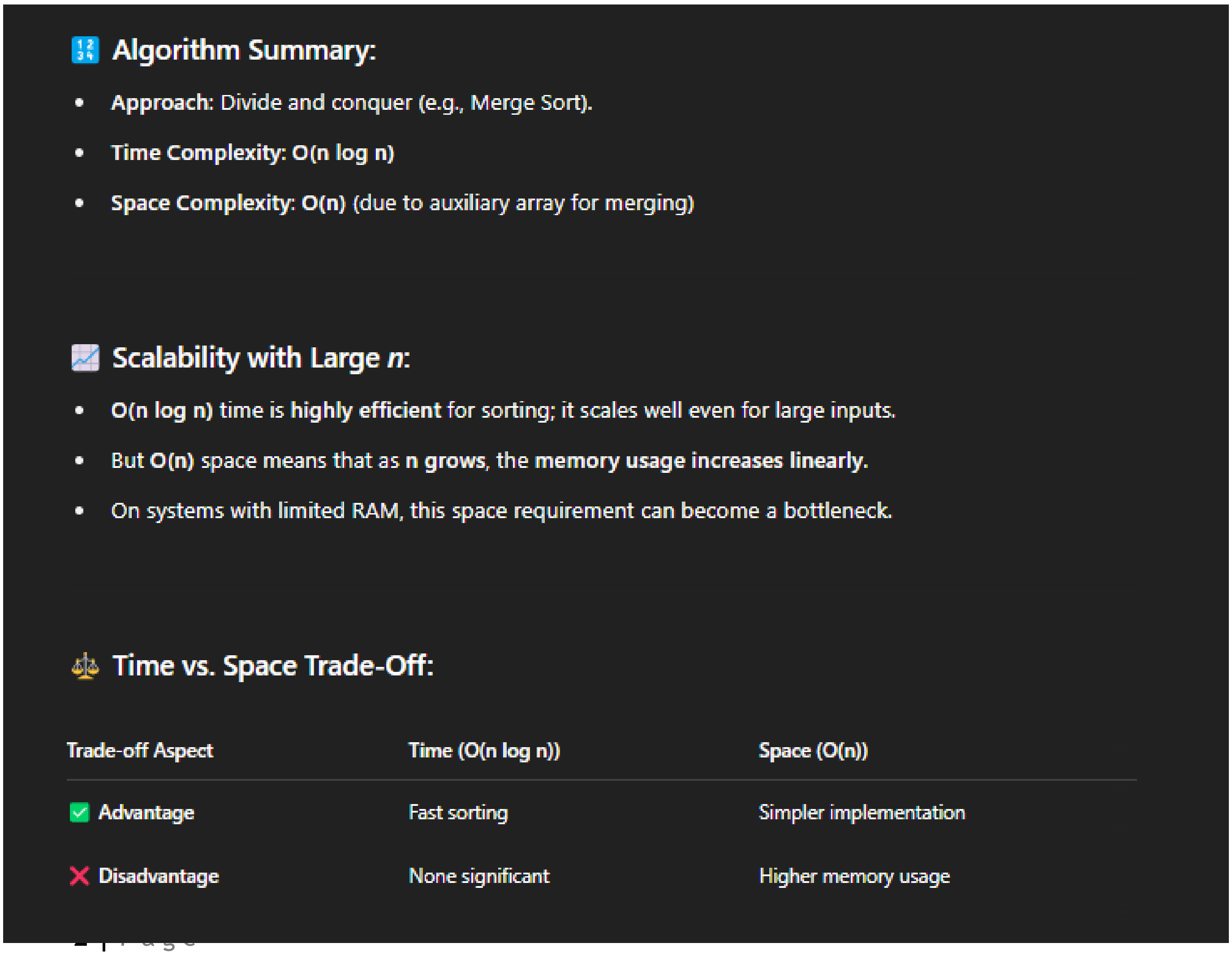
Full Marks: 25 **Deadline: 1G-04-2025**

1. Imagine you are tasked with designing a Turing Machine that operates in a multi-tape environment, where one tape is dedicated to input, and the other tapes are used for auxiliary computations. The input string has a length of ***n***, and the machine needs to decide if a given string is a palindrome. How would the time complexity of this multi-tape Turing Machine compare to that of a single-tape Turing Machine in terms of computational efficiency for the palindrome decision problem? Analyze how the use of multiple tapes influences the computational steps and whether it results in a significant improvement.



- **Single-Tape TM:**
  - Compares first and last characters by scanning back and forth.
  - Each comparison takes O(n) time.
  - **Total Time Complexity: O(n^2).**
- **Multi-Tape TM:**
  - Tape 1: Input; Tape 2: Reversed copy of input.
  - Compare both tapes left to right in one pass.
  - **Copying + Comparing = O(n)** time.
  - **Total Time Complexity: O(n).**

2. You are analyzing an algorithm designed to solve a sorting problem on an input array of size ***n***. The algorithm uses a divide-and-conquer approach that recursively splits the array into two halves and then merges them in sorted order. If the algorithm runs in ***O(nlog n)*** time and requires ***O(n)*** space, how do these time and space complexities affect the scalability of the algorithm when ***n*** becomes very large? What are the trade-offs between time complexity and space complexity in the context of this algorithm?



🔢 Algorithm Summary:

- **Approach:** Divide and conquer (e.g., Merge Sort).
- **Time Complexity: O(n log n)**
- **Space Complexity: O(n)** (due to auxiliary array for merging)

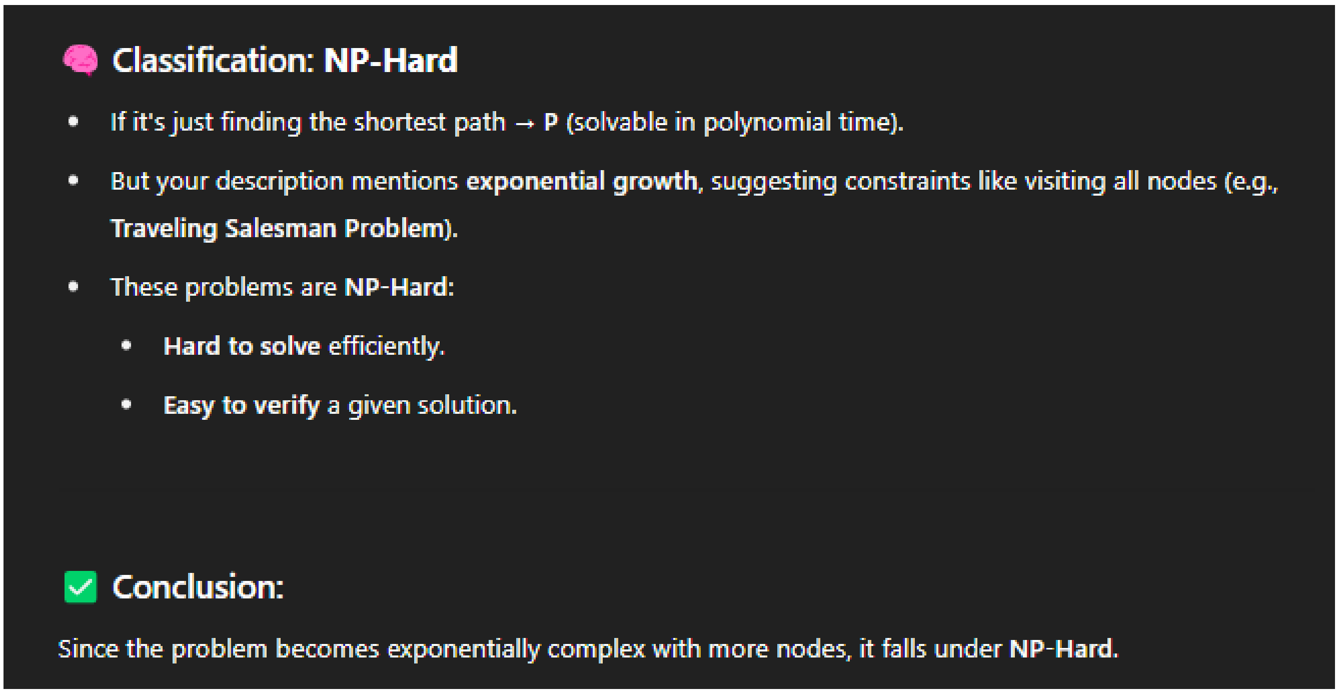
📈 Scalability with Large *n*:

- **O(n log n)** time is **highly efficient** for sorting; it scales well even for large inputs.
- But **O(n)** space means that as **n grows**, the **memory usage increases linearly.**
- On systems with limited RAM, this space requirement can become a bottleneck.

⚖️ Time vs. Space Trade-Off:

Trade-off Aspect	Time (O(n log n))	Space (O(n))
✅ Advantage	Fast sorting	Simpler implementation
❌ Disadvantage	None significant	Higher memory usage

3. A research team is working on optimizing a large-scale network routing problem. The problem involves finding the shortest path through a network with a variable number of nodes and edges, but the problem's computational complexity increases exponentially as the number of nodes increases. Based on your understanding of complexity classes, classify the network routing problem into one of the following categories: P, NP, NP-Hard, or NP-Complete. Justify your classification by explaining the inherent characteristics of the problem, such as the ability to verify a solution quickly versus finding the solution efficiently.



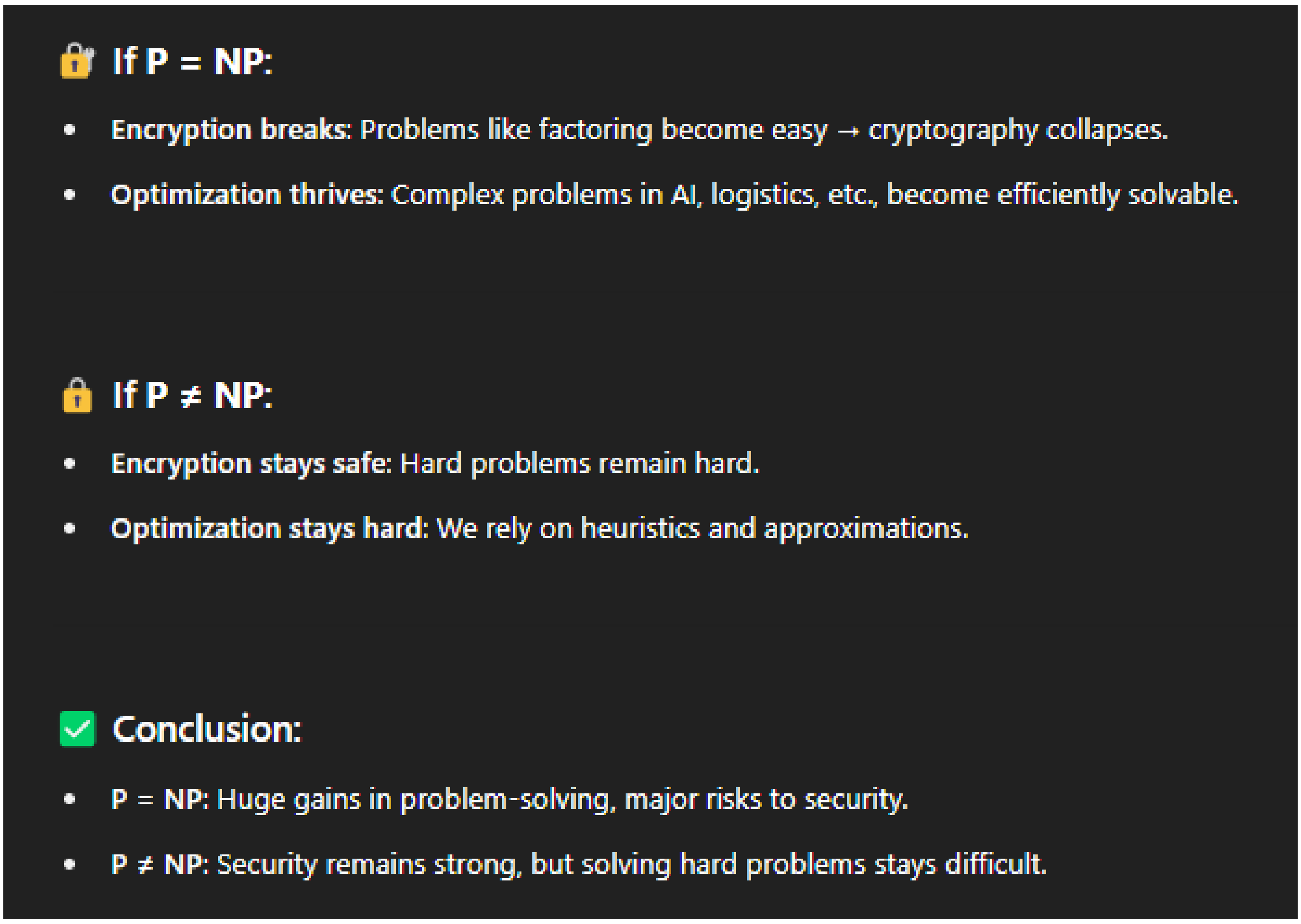
🧠 Classification: NP-Hard

- If it's just finding the shortest path → **P** (solvable in polynomial time).
- But your description mentions **exponential growth**, suggesting constraints like visiting all nodes (e.g., **Traveling Salesman Problem**).
- These problems are **NP-Hard**:
  - **Hard to solve** efficiently.
  - **Easy to verify** a given solution.

✅ Conclusion:

Since the problem becomes exponentially complex with more nodes, it falls under **NP-Hard**.

4. You are participating in a theoretical computer science debate on the P vs NP conjecture. One participant argues that P = NP, stating that there is likely an efficient algorithm for solving NP-complete problems. Another participant argues that P ≠ NP, claiming that NP-complete problems are inherently difficult and cannot be solved in polynomial time. How would you analyze the implications of the P vs NP conjecture for real-world applications, such as cryptography and optimization? Discuss the potential consequences if it were proven that P = NP or P ≠ NP in terms of computational security and problem-solving efficiency.



🔓 If P = NP:

- **Encryption breaks:** Problems like factoring become easy → cryptography collapses.
- **Optimization thrives:** Complex problems in AI, logistics, etc., become efficiently solvable.

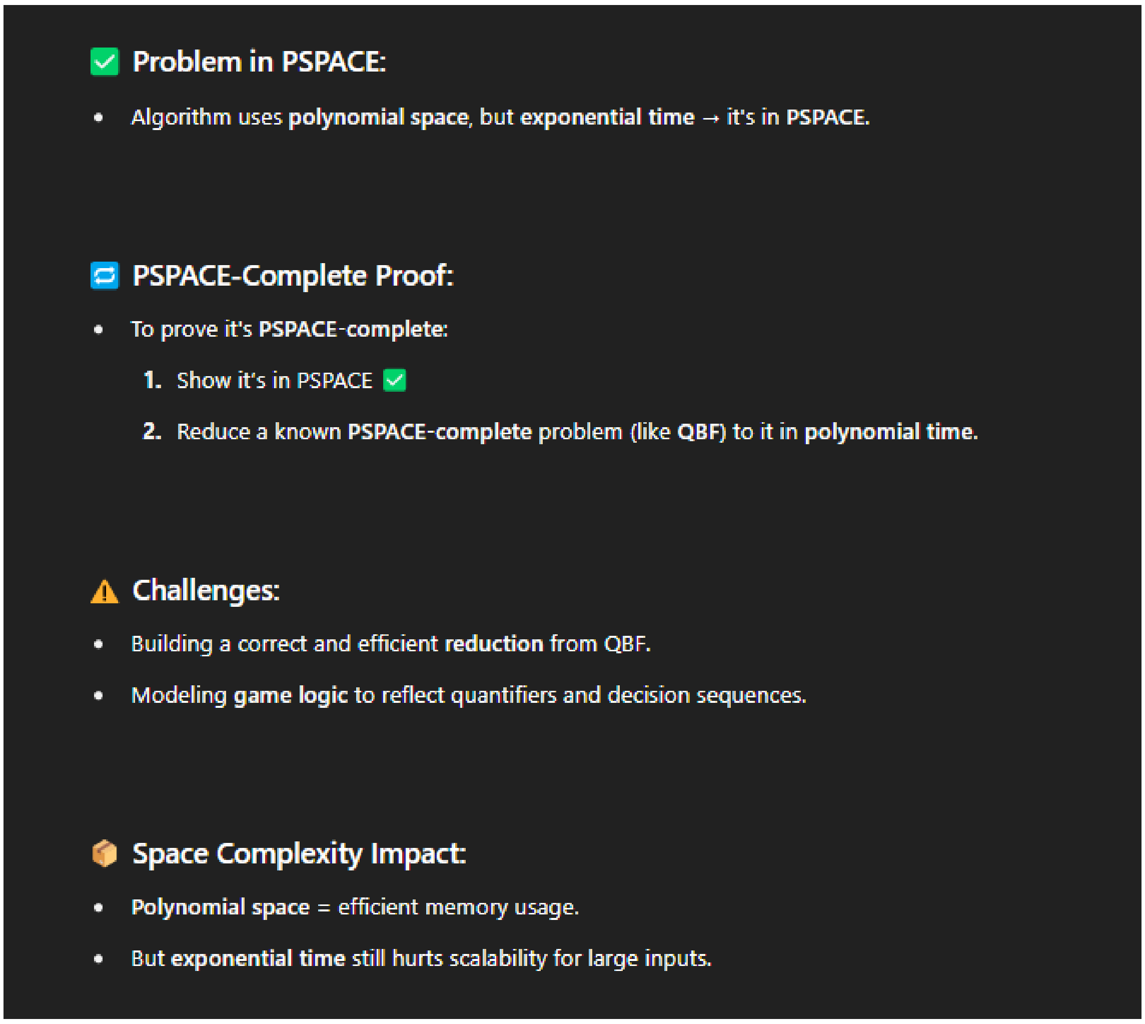
🔒 If P ≠ NP:

- **Encryption stays safe:** Hard problems remain hard.
- **Optimization stays hard:** We rely on heuristics and approximations.

✅ Conclusion:

- **P = NP:** Huge gains in problem-solving, major risks to security.
- **P ≠ NP:** Security remains strong, but solving hard problems stays difficult.

5. You are tasked with analyzing a new algorithm for a game-solving problem, where the goal is to determine the winner of a game given a series of moves. The algorithm runs in polynomial space, but it requires exponential time to explore all possible moves in the game tree. Given that the algorithm runs in polynomial space (i.e., it is in PSPACE), analyze how the problem can be reduced to a known PSPACE-complete problem. What challenges would you face in proving that the problem is PSPACE-complete, and how does space complexity impact the algorithm's ability to scale with input size?



✅ Problem in PSPACE:

- Algorithm uses **polynomial space**, but **exponential time** → it's in **PSPACE.**

🔁 PSPACE-Complete Proof:

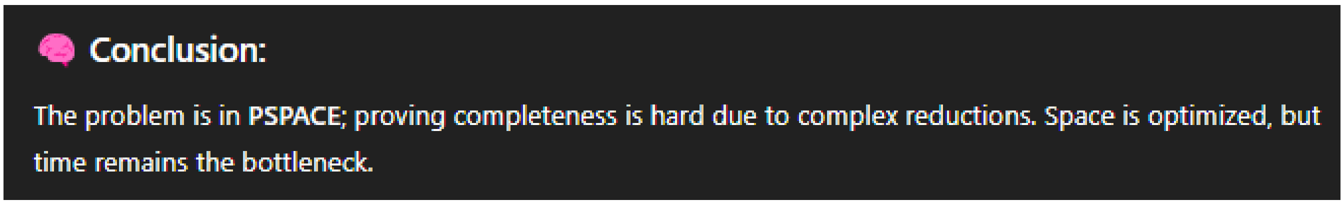
- To prove it's **PSPACE-complete:**
  1. Show it's in PSPACE ✅
  2. Reduce a known **PSPACE-complete** problem (like **QBF**) to it in **polynomial time.**

⚠️ Challenges:

- Building a correct and efficient **reduction** from QBF.
- Modeling **game logic** to reflect quantifiers and decision sequences.

📦 Space Complexity Impact:

- **Polynomial space** = efficient memory usage.
- But **exponential time** still hurts scalability for large inputs.



🧠 Conclusion:

The problem is in **PSPACE**; proving completeness is hard due to complex reductions. Space is optimized, but time remains the bottleneck.