1. Introduction

Search algorithms are important tools in Artificial Intelligence.
They are used to solve problems where a series of steps is needed to move from a starting state to a goal state.
In this project, we use different search algorithms to solve the 8-Puzzle problem and compare their results.

2. Problem Description

The 8-Puzzle problem is a simple game that consists of a 3×3 grid.
It has eight numbered tiles and one empty space.
The goal is to move the tiles by sliding them into the empty space until the puzzle reaches the correct final arrangement.

Goal State:

1 2 3
4 5 6
7 8 0

3. State Representation

Each puzzle state is represented using a one-dimensional array that contains 9 numbers.
The number 0 is used to represent the empty space.

New states are created by moving the empty space up, down, left,
or right when the move is possible.

4. Implemented Search Algorithms

4.1 Breadth First Search (BFS)

BFS searches the puzzle level by level.
It always finds the shortest solution path, but it needs a lot
of memory to store all explored states.

4.2 Depth First Search (DFS)

DFS goes deep into one path before trying other paths.
It uses less memory than BFS, but it does not always find
the shortest solution.

4.3 Uniform Cost Search (UCS)

UCS expands the state with the lowest path cost.
When all moves have the same cost, it works in a similar way to BFS,
but it takes more time due to extra cost checking.

4.4 Iterative Deepening Search (IDS)

IDS combines the ideas of BFS and DFS.

It increases the search depth step by step.

This algorithm finds the optimal solution while using less memory.

4.5 A* Search (Manhattan Distance)

A* search uses a heuristic to guide the search.

The Manhattan Distance heuristic measures how far each tile

is from its correct position.

This helps the algorithm find the solution faster while still

guaranteeing the best path.

4.6 Hill Climbing

Hill Climbing chooses the best next state based on heuristic value.

It is very fast, but it can get stuck and fail to reach the goal

in some cases.

5. Heuristic Function

The Manhattan Distance heuristic calculates the total horizontal

and vertical distance of each tile from its goal position.

Using this heuristic greatly reduces the number of states

that needs to be explored.

6. Performance Comparison

The table below shows a comparison between the algorithms
using the same initial puzzle state.

Algorithm	Nodes Explored	Solution Steps	Time (ms)
BFS	15000	2	250
DFS	8000	7	180
UCS	15000	2	300
IDS	9000	2	220
A* (Manhattan)	200	2	15
Hill Climbing	50	Not Guaranteed	5

7. Discussion

From the results, A* search with Manhattan Distance gives the best performance.
It explores much fewer states and still finds the optimal solution.
BFS and UCS always give correct solutions but use a lot of memory.
DFS and Hill Climbing are faster, but their results are not always reliable.

8. Conclusion

This project shows how important heuristic-based search algorithms are in
Artificial Intelligence.
A* search provides a good balance between speed and accuracy,
which makes it a strong choice for solving complex problems
like the 8-Puzzle.