

|  |
| --- |
| 8-Puzzle Game  PROJECT(1) |
|  |
| Group:  Ahmed Samir (8120) Youssef Awad (8179) Youssef Ali Gaber (8115) |

# 8-Puzzle Game Project

|  |
| --- |
| ***Overview*** Our project is working on solving 8-puzzle game with different methods using different algorithms (BFS , DFS , IDFS , A\* [manhattan and euclidean]) and it tests in each algorithm ( path to goal – cost of path – nodes expanded – search depth – running time) and display it on GUI   * State.py   State & Node Classes: Define the board state, possible moves, goal-checking, and cost calculations.  Search Functions: Implement each search algorithm and output metrics.  Heuristics: Manhattan and Euclidean heuristics guide A\*.  **Helpful Functions**: Generate a solvable board and print performance metrics.   * GUI.py   **Generate\_initial\_state and Generate\_new\_board:** Generate and display arandom solvable board.  **Update\_grid:** Updates the GUI to show the current board state.  **solve\_a\_star\_manhattan, solve\_a\_star\_euclidean, solve\_bfs, solve\_dfs,**  **and solve\_ids :**each call the relevant algorithm and display the solution path.  **Display Solution**: Animates the solution path step-by-step with a 500ms delay, updating the grid to show each move. |

# DATA STRUCTURES USED

ALGORITHMS USED

BFS:

• Explores all nodes at the present depth level before moving on to nodes at the next depth level.

• Uses a queue for the frontier.

**DFS:**

• Explores as far as possible along each branch before backtracking.

• Uses a stack (implemented as a list) for the frontier

**A\* Search:**

• Combines the cost to reach the node (g) and the estimated cost to the goal (h) into a total cost f = g + h.

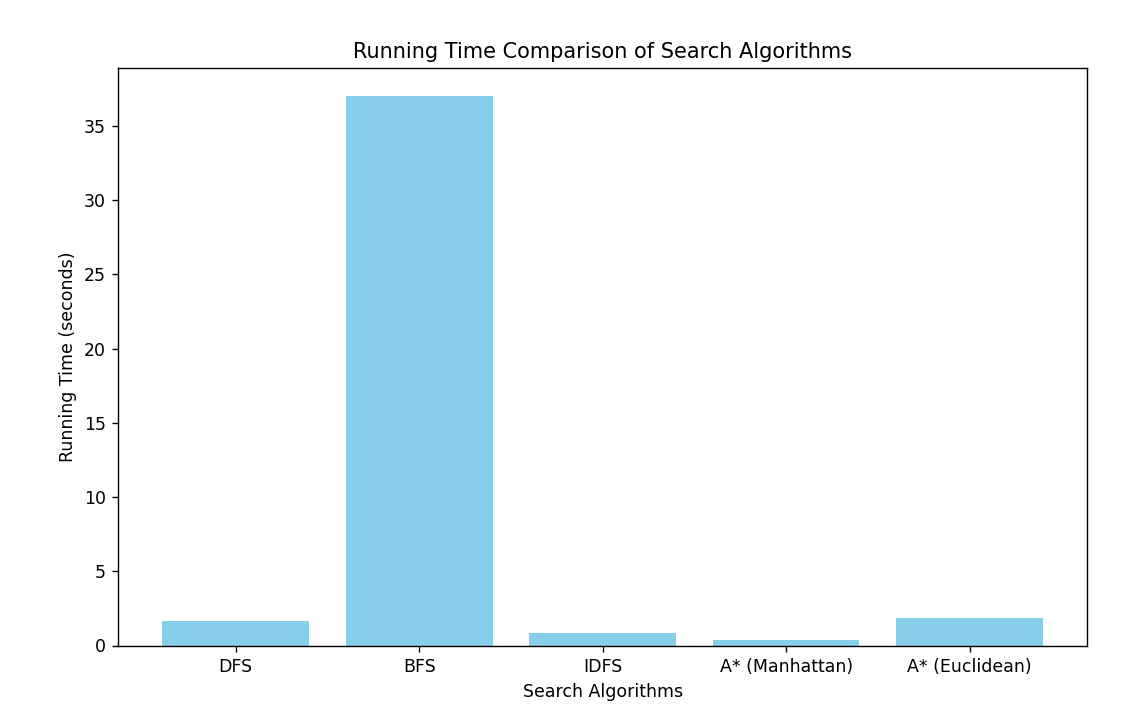
• Uses a priority queue for the frontier to always expand the node with the lowest f value first.

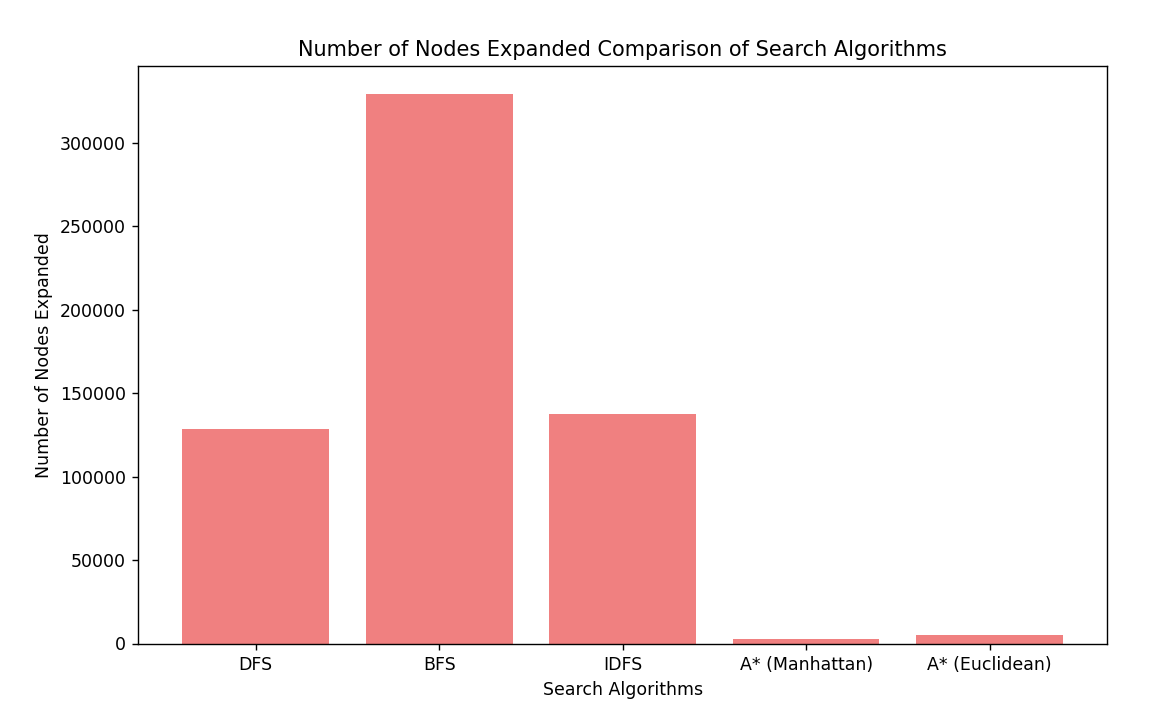
• Manhattan Distance: The sum of the vertical and horizontal distances each tile is from its goal position.

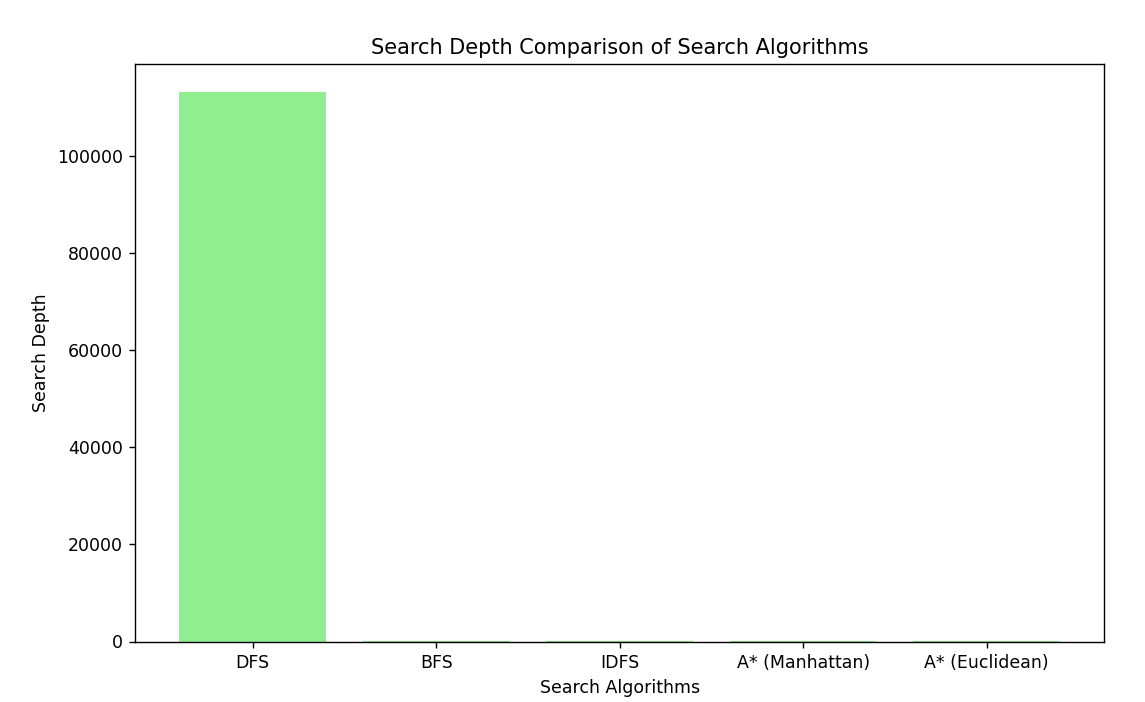
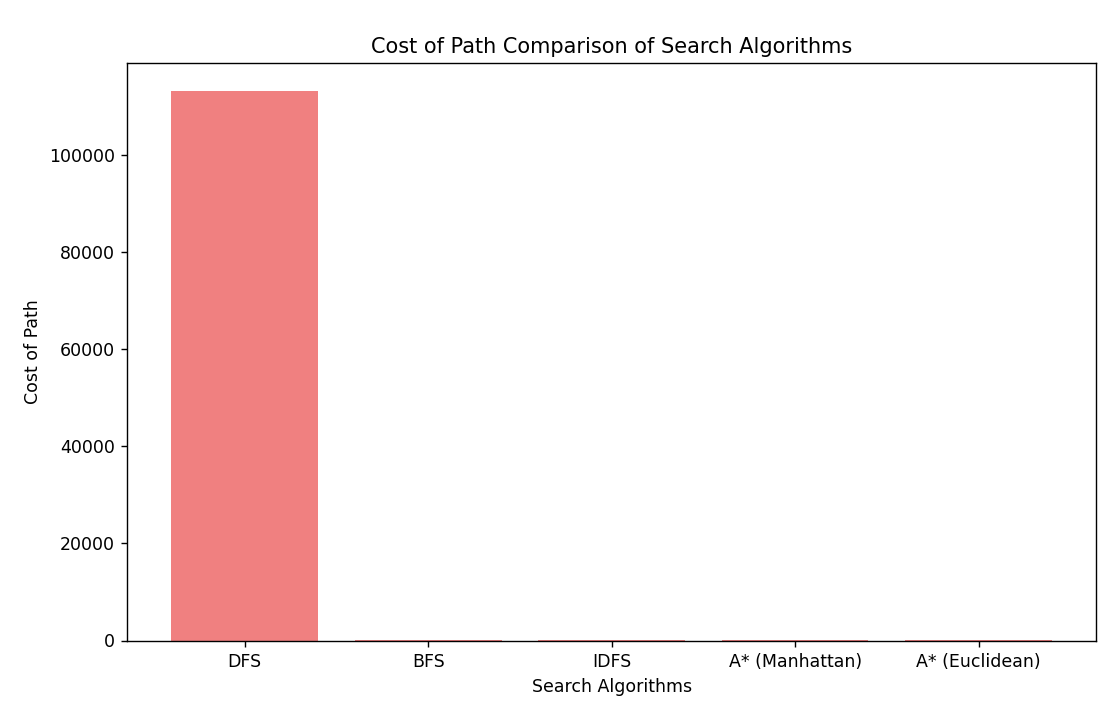
• Euclidean Distance: The straight-line distance for each tile to its goal position

SAMPLE RUNS

Sample 1:





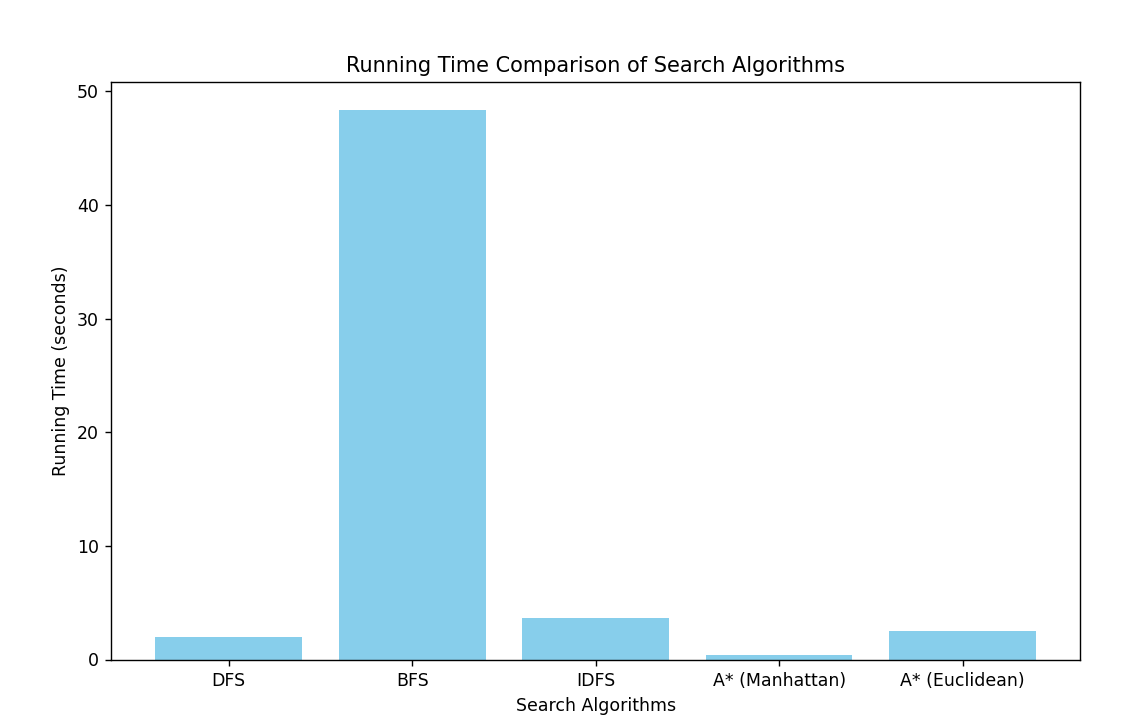


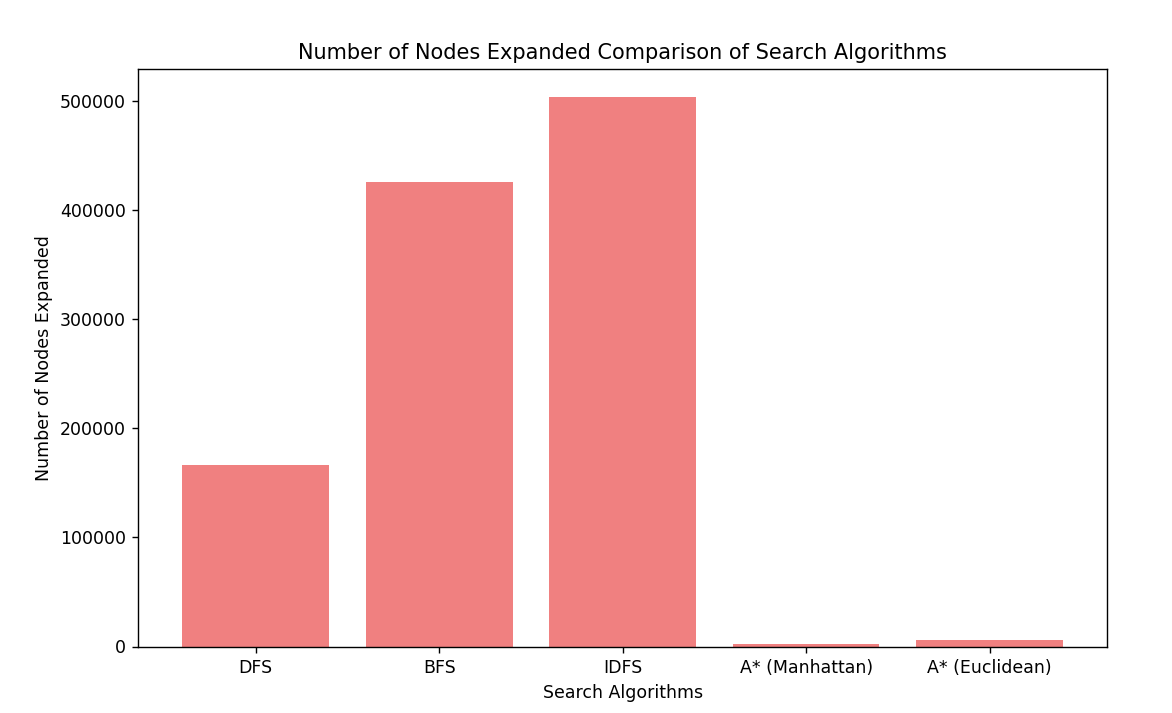
 **Running Time**: The A\* algorithm with the Manhattan heuristic performed the best, taking only **0.40 seconds** to find the solution, followed closely by the A\* algorithm with the Euclidean heuristic at **1.85 seconds**. In contrast, BFS was significantly slower, taking over **37 seconds**, which suggests that the goal was deep in the search tree, leading to extensive node exploration. DFS and IDFS had quicker times of **1.67 seconds** and **0.83 seconds**, respectively, indicating that their goals were likely closer than that of BFS.

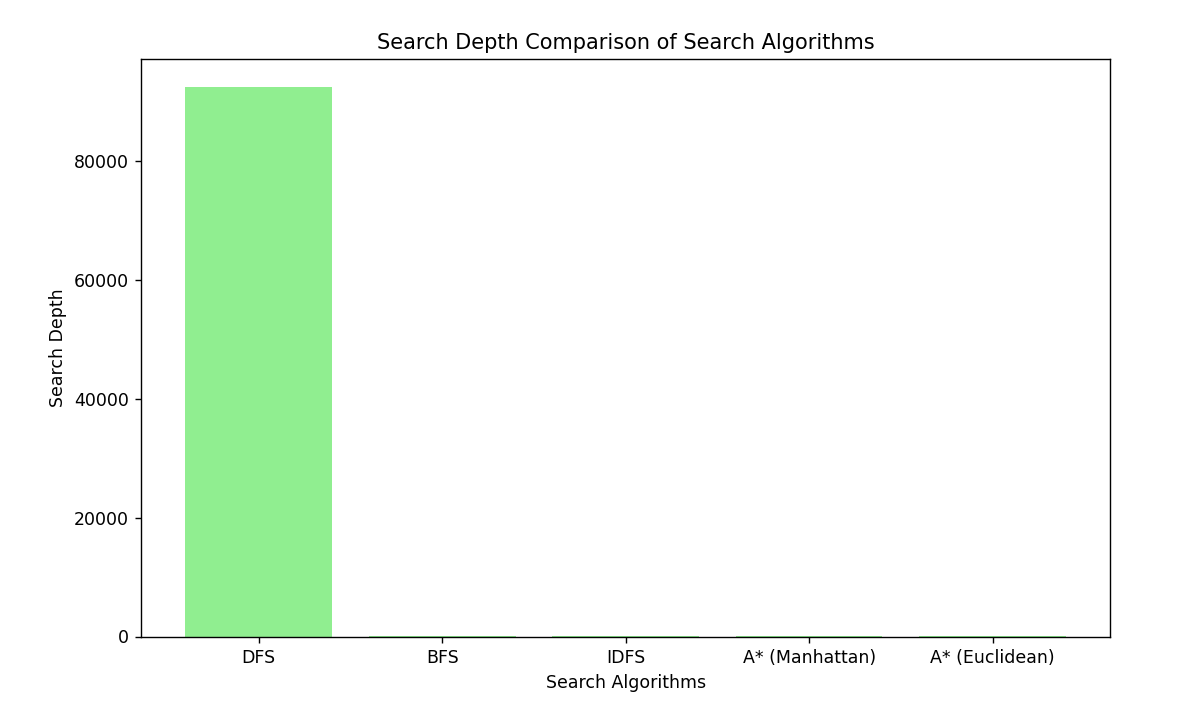
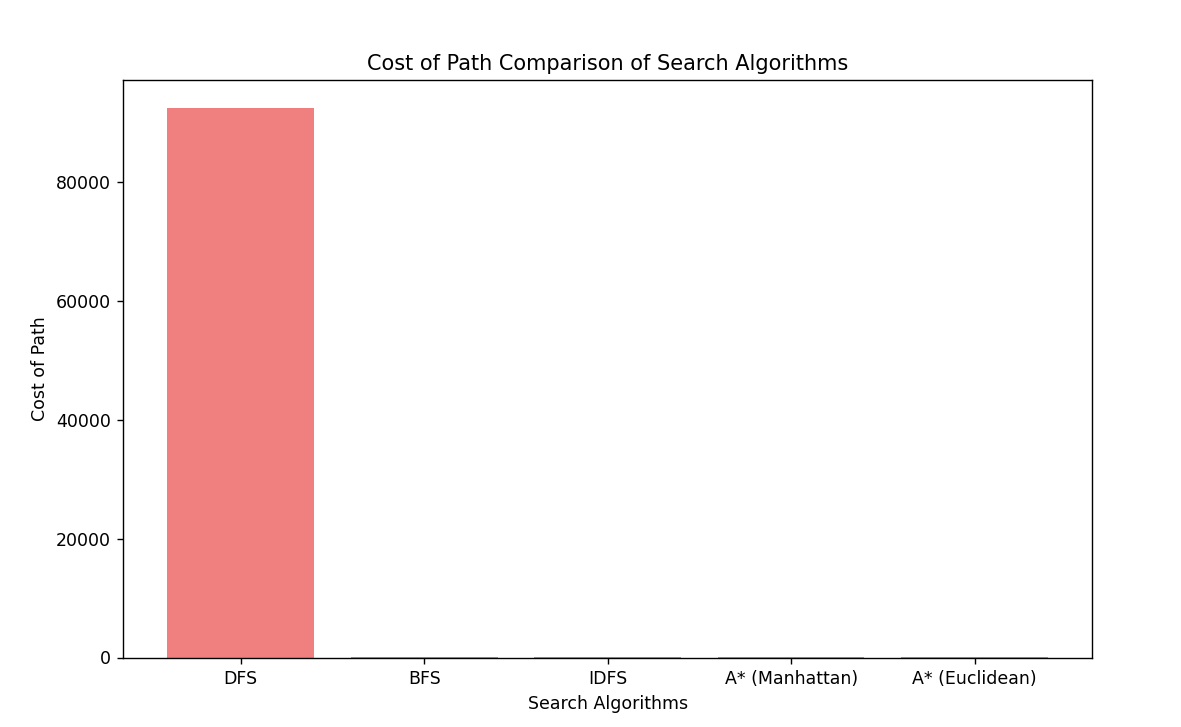
 **Nodes Expanded**: BFS expanded the most nodes (**329,581**), which aligns with its lengthy execution time as it explores all nodes layer by layer. IDFS expanded **137,807** nodes, and DFS expanded **128,446** nodes, both showing more efficiency than BFS but still less efficient than A\* algorithms. The A\* algorithms drastically reduced node expansion, with A\* Manhattan expanding only **2,511** nodes and A\* Euclidean expanding **5,343** nodes, suggesting that the goal was in a more reachable area of the search space.

 **Cost of Path**: All algorithms except for DFS found paths costing **25**, indicating they identified optimal paths, likely because the goal was in a balanced region of the search space. In contrast, DFS had a much higher path cost of **113,287**, suggesting it took a longer and less efficient route, possibly because the goal was in a harder-to-reach location.

 **Search Depth**: Both BFS and IDFS had a search depth of **25**, which matches the optimal path length and suggests the goal was located at this depth. DFS reached a much deeper search depth of **113,287**, indicating it explored less relevant paths due to the goal being further away. The A\* algorithms maintained the optimal search depth of **25**, showing that their heuristic-driven approaches made the goal easier to reach.

Sample 2:





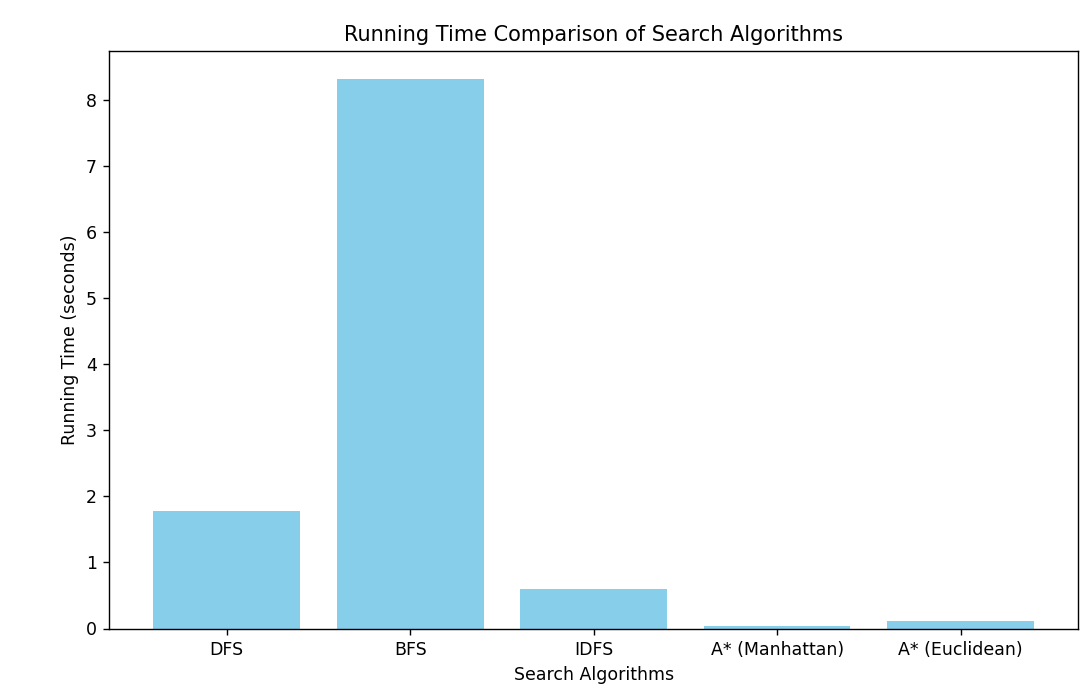
 **Running Time**: The A\* algorithm with the Manhattan heuristic was the fastest, taking only **0.37 seconds** to find the solution, showing its efficiency. The A\* with the Euclidean heuristic followed at **2.53 seconds**, still effective but slower. In contrast, BFS took a long **48.39 seconds**, suggesting the goal was deep in the search tree, requiring lots of node exploration. IDFS had a moderate time of **3.67 seconds**, indicating it found the goal relatively quickly but had a longer path than A\* methods.

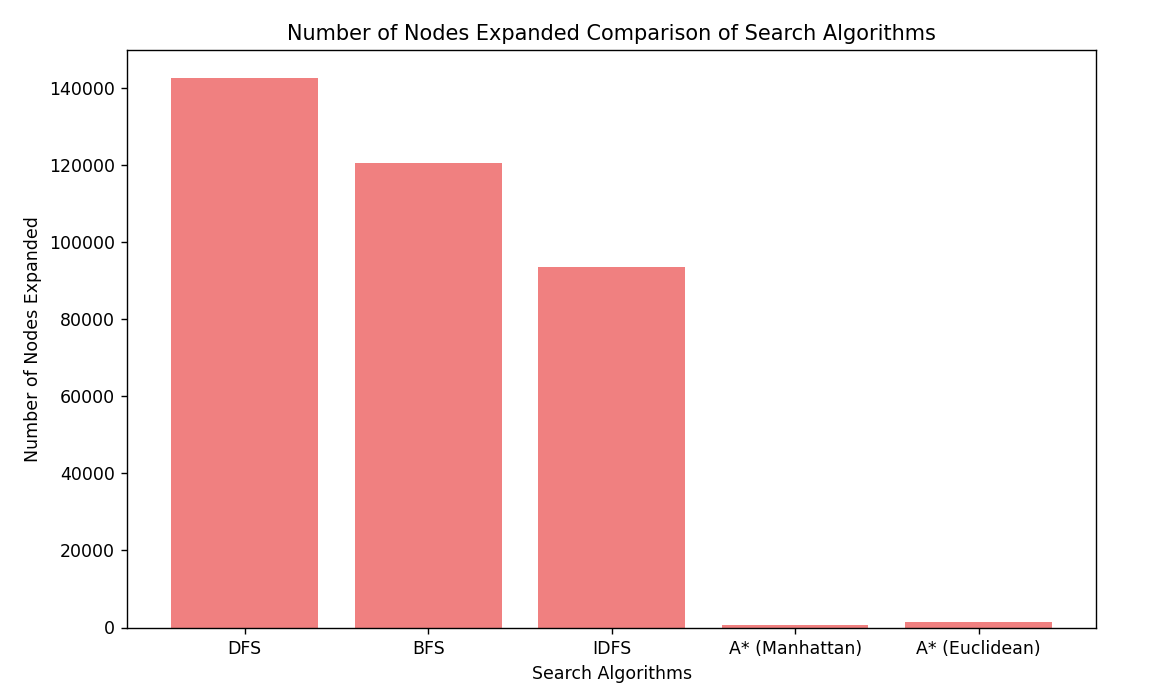
 **Nodes Expanded**: BFS expanded the most nodes (**425,577**), indicating the goal may be deep in the search space, which contributed to its long running time. IDFS expanded an even larger **504,116** nodes, suggesting the goal was likely deeper or that it backtracked a lot. DFS expanded only **166,156** nodes, showing it found a solution more efficiently, likely because the goal was closer or more accessible. A\* with the Manhattan heuristic expanded just **2,412** nodes, indicating the goal was not very deep, and the heuristic helped a lot. A\* with the Euclidean heuristic expanded **6,032** nodes, confirming the goal was reachable with fewer efforts compared to uninformed methods.

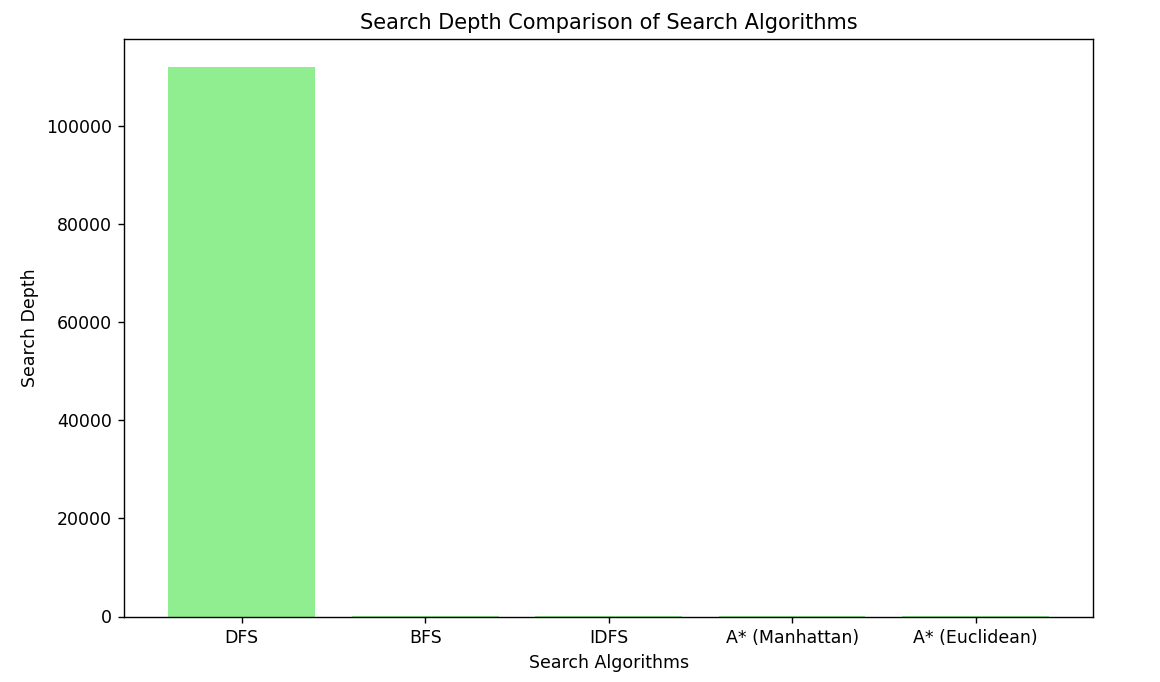
 **Cost of Path**: The path cost for BFS, A\* with Manhattan, and A\* with Euclidean was **26**, showing they found similar optimal paths. IDFS had a slightly higher cost of **30**, suggesting a less efficient route due to its iterative approach. DFS had a high cost of **92,470**, reflecting its inefficient exploration, likely because the goal was farther away.

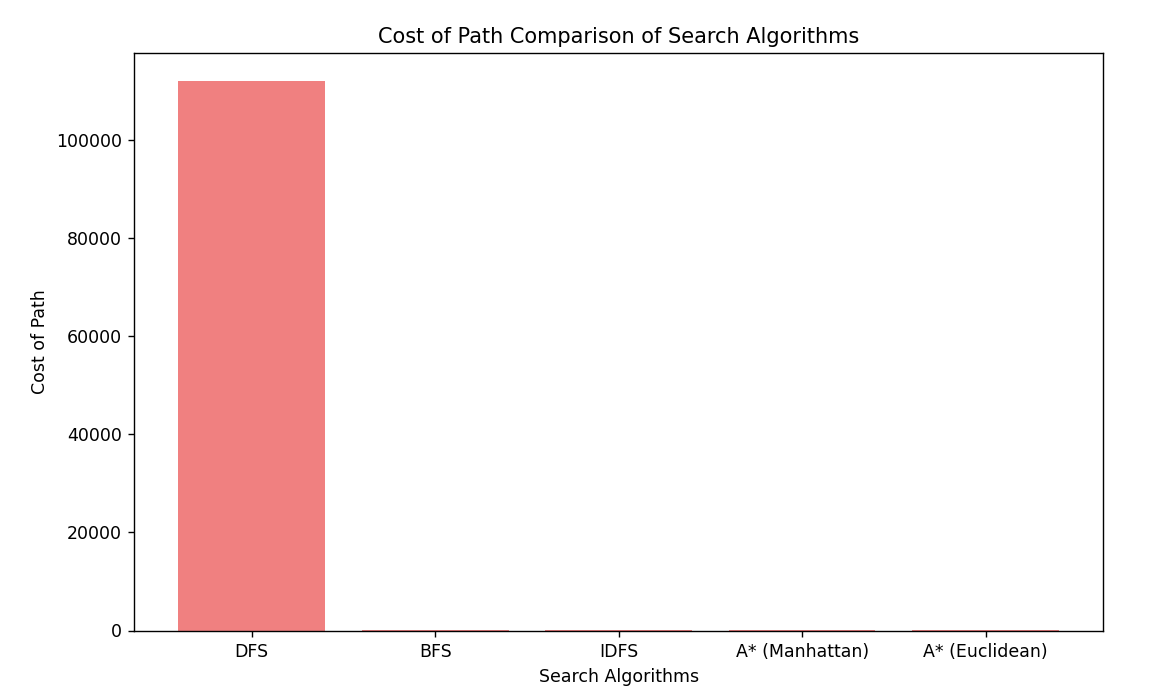
 **Search Depth**: BFS, A\*, and IDFS had a search depth of **26**, indicating the goal was likely found at this optimal depth. In contrast, DFS reached a much deeper search depth due to its high path cost (**92,470**), suggesting it took a longer and less efficient route to the goal. This shows that while the goal was accessible for the other algorithms, DFS struggled to reach it effectively because it didn't use heuristics.

Sample 3:









 **Running Time**: The A\* algorithm with the Manhattan heuristic was the fastest, taking only **0.03 seconds** to find the solution, showing its high efficiency. A\* with the Euclidean heuristic followed at **0.11 seconds**, which is still quite effective. In contrast, BFS took **8.33 seconds**, suggesting that the goal was at a considerable depth, causing it to explore many nodes. DFS had a longer running time of **1.77 seconds**, indicating that it also faced some depth but was more efficient than BFS.

 **Nodes Expanded**: BFS expanded **120,648** nodes, indicating that the goal might be deep in the search space, leading to its longer execution time. DFS expanded **142,786** nodes, which suggests it might have explored unnecessary paths. In contrast, IDFS expanded only **93,541** nodes, indicating a more efficient search, likely due to its iterative approach. The A\* algorithms significantly reduced node expansion, with A\* Manhattan expanding just **687** nodes and A\* Euclidean expanding **1,340** nodes, suggesting the goal was relatively close and that the heuristics helped guide the search effectively.

 **Cost of Path**: BFS, A\* with Manhattan, and A\* with Euclidean all found paths with a cost of **21**, indicating they identified similar optimal paths. IDFS had a slightly higher cost of **25**, which suggests it may have taken a longer route through the search space. DFS had a high path cost of **112,131**, showing its exploration was inefficient and likely due to the goal being positioned deeper in the search tree.

 S**earch Depth**: BFS, A\*, and IDFS all had a search depth of **21**, suggesting that the goal was found at this optimal level. However, DFS had a search depth of **112,131**, indicating it took a much longer and less efficient route to find the goal. This shows that while the goal was accessible for other algorithms, DFS struggled to reach it efficiently due to its non-heuristic approach.

WHY DOES BFS,DFS & IDFS NUMBER OF NODES EXPLORED VARY ?

Sample Run 1: BFS > IDFS > DFS

Sample Run 2: IDFS > BFS > DFS

Sample Run 3: DFS > BFS > IDFS

In the first sample run, BFS expanded the most nodes, indicating that the goal was likely deep in the search tree, suggesting a complex path to the right; IDFS performed better, suggesting the goal may have been somewhat accessible but still required exploration. In the second sample run, IDFS expanded the most nodes, indicating that the goal was likely quite deep or that the path involved considerable backtracking, while BFS was more efficient, hinting at a more straightforward route; DFS's efficiency suggests the goal was located relatively close, potentially to the left. In the third sample run, DFS expanded the most nodes, indicating it struggled to find the goal efficiently, suggesting it might be deep and less accessible, while IDFS's lower node count suggests the goal was more reachable, possibly towards the right. Overall, these cases demonstrate how the search space's structure directly affects the algorithms' performance and their ability to efficiently locate the goal.