Analysis:

The problem. The 8-puzzle problem is a puzzle invented and popularized by Noyes Palmer Chapman in the 1870s. It is played on a 3-by-3 grid with 8 square blocks labeled 1 through 8 and a blank square. Your goal is to rearrange the blocks so that they are in order, using as few moves as possible. You are permitted to slide blocks horizontally or vertically into the blank square. The following shows a sequence of legal moves from an initial board (left) to the goal board (right).

Breadth First Traversal (or Search):
For a graph is similar to Breadth First Traversal of a tree (See method 2 of this post). The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array. For simplicity, it is assumed that all vertices are reachable from the starting vertex.

Depth-first search (DFS):
Is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

Iterative Deepening Serach(IDS):
IDDFS calls DFS for different depths starting from an initial value. In every call, DFS is restricted from going beyond given depth. So basically we do DFS in a BFS fashion.

Depth Limited Search:
The unbounded tree problem appeared in DFS can be fixed by imposing a limit on the depth that DFS can reach, this limit we will call depth limit *l*, this solves the infinite path problem.

Bidirectional Search:
As the name suggests, bidirectional search suggests to run 2 simultaneous searches, one from the initial state and the other from the goal state, those 2 searches stop when they meet each other at some point in the middle of the graph

Uniform Cost Search:
Uniform-cost is guided by path cost rather than path length like in BFS, the algorithms starts by expanding the root, then expanding the node with the lowest cost from the root, the search continues in this manner for all nodes.

A Star Algorithm:
Informally speaking, A* Search algorithms, unlike other traversal techniques, it has “brains”. What it means is that it is really a smart algorithm which separates it from the other conventional algorithms.

Using the concept called heuristic we use it to our advantage and design A star algorithm which strategically solves the puzzle. For this case we used the heuristic called Manhattan Distance

Manhattan Distance :
The sum of the Manhattan distances (sum of the vertical and horizontal distance) from the blocks to their goal positions, plus the number of moves made so far to get to the search node.

We have implemented 8 search techniques on different test cases and detailed report of them is in the “output.txt” file.
The “output.txt“ file has implementations of various algorithms on various range of inputs and also noting the number of nodes explored and time taken for it to complete the search, number of steps needed to go to the goal state and also the path it followed to reach the goal state.

The goal state is the picture referred here.

Alogirthm	BFS	DFS	IDS	DLS	A star	UCS	Greedy	Bi Dir
Avg Time	17.323	~1000	8.32	6.45	0.20983	17.323	0.30872	0.40812
Avg Nodes	3702	<10,000	2873	2634	143	3702	208	249

Analysing the algorithms we can conclude that Breadth First Search and Depth First Search Algorithms are the most naivë and brute force.
So taking advantages of both BFS and DFS we have interative deepening search and depth limit search techniques.

For depth limited search we have calculated only for the outputs which worked because in depth limit search for few limit we couldn’t reach goal state same goes with the case of depth first search.

But if we give a short example where at most 10-15 times the puzzle is changed the BFS,UCS,DLS,IDS are of the order 0.00_ seconds and Astar, Bidirectional, Greedy were of order 0.000_.

These are my inferences and observations.

Rohith Gilla,
16XJ1A0515
Mahindra Ecolé Centrale.