Breath First Search :

Assumptions: As in BFS all nodes are expanded level wise.If we take N=4 and N=5 the solution is optimal by BFS but the number of moves will depend upon our input and it will take more time.For N=3 it will take comparatively less time.

Methodology:

1.Taking input from user for given value of n.

2.We have goal state and if our current state is equal to goal state then it will give result.

3.For this there is need to compare the neighbours of each element and swap it and add it to the list.

4.If perticular node is already visited then we will not put it into list.

5.The current state is popped from list using list.pop(0) that is we are using queue and it will maintain the BFS property that is all nodes will be traversed level wise.

Results:

Enter the order of puzzle3

0 1 3

4 2 5

7 8 6

Goal Found No of visited nodes 50 Time in ms -1535308811.914412

Observation: The time complexity in best case will O(1) and in worst case it is O(n). The Space Complexity will be O(n) as all nodes that are expanded will be store in list.

Depth First Search :

Assumptions: As in DFS all nodes are expanded depth wise and if solution is not found then it again traverse from root to another depth so in DFS it does not provide optinmal solution.If we take N=4 and N=5 then it may take more time to give solution.For N=3 it will take comparatively less time.

Methodology:

1.Taking input from user for given value of n.

2.We have goal state and if our current state is equal to goal state then it will give result.

3.For this there is need to compare the neighbours of each element and swap it and add it to the list.

4.If perticular node is already visited then we will not put it into list.

5.The current state is popped from list using list.pop() that is we are using stack and it will maintain the DFS property that is all nodes will be traversed depth wise.

Results:

Enter the order of puzzle3

0 1 3

4 2 5

7 8 6

Goal Found No of visited nodes 578 Time in ms -1535309001.6286063

Observation: The time complexity in best case will O(1) and in worst case it is total number nodes in tree. The Space Complexity will be O(n) as all nodes that are expanded will be store in list.

A* :

Assumptions: As in A* nodes are expanded by calculating manhatton distance(it is distance of element in array which is far away from its original position).It provides optimal solution.For N=3 or N=4 or N=5 it will provide optimal solution.

Methodology:

1.Taking input from user for given value of n.

2.We have goal state and if our current state is equal to goal state then it will give result.

3.For this there is need to compare the neighbours of each element and swap it and we are calculating manhatton distance of each element.We are using heap to maintain the priority queue.

4.If perticular node is already visited then we will not put it into list.

5.The current state is popped from list using list.pop() and check for the condtions.

Results:

Enter the order of puzzle3

0 1 3

4 2 5

7 8 6

Goal Found Number of iterations : 14 Time required : -1535309340.4198399

Enter the order of puzzle 4

1 2 3 4

0 5 6 7

8 9 10 11

12 13 14 15

Goal Found Number of iterations : 7848 Time required : -1535309076.2596025

Observation: The time complexity in best case will O(1) and in worst case it depth of the tree.If we compared it with BFS and DFS then number of iteration required will be less and time is also less.