A* Algorithm:

```
using System.Diagnostics;

namespace N_PUZZLE
{
    class NPuzzleSolution
    {
        public static Stopwatch CalculateTime =  new Stopwatch(); //θ(1)
        public static HashSet<int> HashMatrix; //θ(1)
        public static PriorityQueue<Node, int> priorityQueue; //θ(1)
        public static int z_index_x; //θ(1)
        public static int z_index_y; //θ(1)
        public static Node n1; //θ(1)
        public static int AStarAlgorithm(int[,] Array2D, int MatrixSize, int[] Array1D)
//E log(V)      E: Number of steps

//              V: priorityQueue.Count
        {
            priorityQueue = new PriorityQueue<Node, int>(); //θ(1)
            HashMatrix = new HashSet<int>(); //θ(1)
            Node n = new Node(Array2D, MatrixSize, null); //θ(n^2)      n: matrix_size

            while (priorityQueue.Count > 0)  //E: number of steps
            {
                CalculateTime.Start();
                n1 = priorityQueue.Dequeue(); //θ(log(V))     V: priorityQueue.Count
                HashMatrix.Add(n1.Hash_Matrix); //θ(1)
                Node tmp = n1; //θ(1)
                if (n1.Cost - n1.level == 0) //θ(1)
                {
                    CalculateTime.Stop();
                    return tmp.level; //θ(1)
                }
                n1.Get_adjs(n1); //θ(n^2)      n: matrix_size
            }
            return 0;
        }
        /// Hamming Function
        public static int HammingFun(int[,] Mat, int MatrixSize) //θ(MatrixSize^2)
        {
            int Hamming_value = 0; //θ(1)
            for (int i = 0; i < MatrixSize; i++) //θ(MatrixSize^2)
            {
                for (int j = 0; j < MatrixSize; j++) //θ(MatrixSize)
                {
                    if (Mat[i, j] == 0) //θ(1)
                        continue; //θ(1)
                    else if ((i * MatrixSize + j + 1) != Mat[i, j]) //θ(1)
                        Hamming_value += 1; //θ(1)
                }
            }
            return Hamming_value;
        }
        /// Manhatan Function
        public static int ManhatanFun(int[,] Matrix2D, int MatrixSize) //θ(n^2)      n:
MatrixSize
        {
            int Col = 0; //θ(1)
```

```
            int Row = 0; //θ(1)
            int Manhatan_value = 0; //θ(1)
            for (int i = 0; i < MatrixSize; i++) //θ(n^2)       n: MatrixSize
            {
                for (int j = 0; j < MatrixSize; j++) //θ(n)     n: MatrixSize
                {
                    if (Matrix2D[i, j] == 0) //θ(1)
                        continue; //θ(1)
                    else if (Matrix2D[i, j] != (i * MatrixSize + j) + 1) //θ(1)
                    {
                        Col = ((Matrix2D[i, j] - 1) % MatrixSize); //θ(1)
                        Row = ((Matrix2D[i, j] - 1) / MatrixSize); //θ(1)
                        Manhatan_value += (Math.Abs(Row - i) + Math.Abs(Col - j));
//θ(1)
                    }
                }
            }
            return Manhatan_value;
        }
        /// Check Solvable or not
        public static bool CheckSolvability(int[] Matrix1D, int MatrixSize)  //θ(n^4) =
θ(s^2)      n: MatrixSize, s: Matrix Length
        {
            int inv = 0; //θ(1)
            //compare from first cell to the pre last
            for (int i = 0; i < (MatrixSize * MatrixSize); i++)
            {
                if (Matrix1D[i] == 0) //θ(1)
                {
                    // Get Zero Index
                    z_index_x = i / MatrixSize; //θ(1)
                    z_index_y = i % MatrixSize; //θ(1)
                    continue;
                }
                //compare with the cell after i cell till the last cell
                for (int j = (i + 1); j < (MatrixSize * MatrixSize); j++) //θ(n^2)
n: MatrixSize
                {
                    if (Matrix1D[j] == 0) //θ(1)
                        continue; //θ(1)
                    else if (Matrix1D[i] > Matrix1D[j]) //θ(1)
                        inv++; //θ(1)
                }
            }
            if (MatrixSize % 2 != 0 && inv % 2 == 0) //θ(1)
                return true; //θ(1)
            else if (MatrixSize % 2 == 0 && inv % 2 != 0 && z_index_x % 2 == 0) //θ(1)
                return true; //θ(1)
            else if (MatrixSize % 2 == 0 && inv % 2 == 0 && z_index_x % 2 != 0) //θ(1)
                return true; //θ(1)
            return false; //θ(1)
        }
        public static void PrintPath(Node n) //θ(n^2 * c)      n: MatrixSize      c:
number of nodes in path
        {
            List<Node> list = new List<Node>(); //θ(1)
            while (n.parent != null) //Best: θ(1),      Worst: θ(v)
            {
                list.Add(n); //θ(1)
                n = n.parent; //θ(1)
            }
            int number_of_movements = 1; //θ(1)
```

```
            for (int i = list.Count - 1; i >= 0; i--) //θ(n^2 * c)      n: MatrixSize
c: number of nodes in path
            {
                Console.WriteLine("----------> #" + number_of_movements); //θ(1)
                for (int j = 0; j < list[i].matrix_size; j++) //θ(n^2)      n:
MatrixSize
                {
                    for (int k = 0; k < list[i].matrix_size; k++) //θ(n)      n:
MatrixSize
                        Console.Write(list[i].matrix[j, k] + " "); //θ(1)
                    Console.WriteLine(); //θ(1)
                }
                number_of_movements++; //θ(1)
                Console.WriteLine(); //θ(1)
            }
        }
    }
}
```

Node Class:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using static N_PUZZLE.NPuzzleSolution;
namespace N_PUZZLE
{
    public class Node
    {
        public static int ManOrHam = 0; //θ(1)
        public int[,] matrix; //θ(1)
        public int matrix_size; //θ(1)
        public Node parent; //θ(1)
        public int z_x; //θ(1)
        public int z_y; //θ(1)
        public int Cost; //θ(1)
        public int level; //θ(1)
        public int Hash_Matrix; //θ(1)

        public Node(int[,] Matrix2D, int Size, Node par) //θ(n^2)      n: matrix_size
        {
            matrix = Matrix2D; //θ(1)
            matrix_size = Size; //θ(1)
            Hash_Matrix = GetMatrixHash(this.matrix); //θ(n^2)      n: matrix_size
            if (par == null) //θ(n^2)      n: matrix_size
            {
                level = 0; //θ(1)
                parent = null; //θ(1)
                z_x = z_index_x; //θ(1)
                z_y = z_index_y; //θ(1)
                Console.WriteLine("----------------------\n[1] Hamming Distance\n[2]
Manhatan Distance"); //θ(1)
                Console.Write("\nEnter your choice [1-2]: "); //θ(1)
                char Ch = (char)Console.ReadLine()[0]; //θ(1)
                if (Ch == '1') //θ(n^2)      n: MatrixSize
                {
```

```
                    Cost = HammingFun(matrix, matrix_size); //θ(n^2)      n: matrix_size
                    ManOrHam = 1; //θ(1)
                }
                else //θ(n^2)     n: MatrixSize
                {
                    Cost = ManhatanFun(matrix, matrix_size); //θ(n^2)     n: matrix_size
                }
                priorityQueue.Enqueue(this, Cost); //θ(log(n))      n:
priorityQueue.Count
            }
            else //θ(n^2)      n: matrix_size
            {
                parent = par; //θ(1)
                level = parent.level + 1; //θ(1)
                if (ManOrHam == 1) //θ(n^2)     n: MatrixSize
                {
                    Cost = HammingFun(matrix, matrix_size) + level; //θ(n^2)      n:
matrix_size
                }
                else //θ(n^2)    n: matrix_size
                {
                    Cost = ManhatanFun(matrix, matrix_size) + level; //θ(n^2)      n:
matrix_size
                }
            }
        }
        public void Get_adjs(Node x)  //θ(n^2)      n: matrix_size
        {
            if (x.z_x + 1 < x.matrix_size) //θ(n^2)      n: matrix_size
            {
                int[,] mat = new int[x.matrix_size, x.matrix_size]; //θ(1)
                Array.Copy(x.matrix, mat, x.matrix_size * x.matrix_size); //θ(n^2)
n: matrix_size
                mat[x.z_x, x.z_y] = mat[x.z_x + 1, x.z_y]; //θ(1)
                mat[x.z_x + 1, x.z_y] = 0; //θ(1)
                Node n1 = new Node(mat, matrix_size, x); //θ(n^2)      n: matrix_size
                n1.z_x = x.z_x + 1; //θ(1)
                n1.z_y = x.z_y; //θ(1)
                bool visited = HashMatrix.Contains(n1.Hash_Matrix); //θ(n)      n:
HashMatrix.Count
                if (!visited)  //θ(log(n))      n: priorityQueue.Count
                {
                    priorityQueue.Enqueue(n1, n1.Cost); //θ(log(n))      n:
priorityQueue.Count
                }
            }
            if (x.z_x - 1 >= 0) //θ(n^2)      n: matrix_size
            {
                int[,] mat = new int[x.matrix_size, x.matrix_size]; //θ(1)
                Array.Copy(x.matrix, mat, x.matrix_size * x.matrix_size); //θ(n^2)
n: matrix_size
                mat[x.z_x, x.z_y] = mat[x.z_x - 1, x.z_y]; //θ(1)
                mat[x.z_x - 1, x.z_y] = 0; //θ(1)
                Node n2 = new Node(mat, matrix_size, x); //θ(n^2)      n: matrix_size
                n2.z_x = x.z_x - 1; //θ(1)
                n2.z_y = x.z_y; //θ(1)
                bool visited = HashMatrix.Contains(n2.Hash_Matrix); //θ(n)      n:
HashMatrix.Count
                if (!visited)  //θ(log(n))      n: priorityQueue.Count
                {
                    priorityQueue.Enqueue(n2, n2.Cost); //θ(log(n))      n:
priorityQueue.Count
                }
```

```
            }
            if (x.z_y + 1 < x.matrix_size) //θ(n^2)      n: matrix_size
            {
                int[,] mat = new int[x.matrix_size, x.matrix_size]; //θ(1)
                Array.Copy(x.matrix, mat, x.matrix_size * x.matrix_size); //θ(n^2)
n: matrix_size
                mat[x.z_x, x.z_y] = mat[x.z_x, x.z_y + 1]; //θ(1)
                mat[x.z_x, x.z_y + 1] = 0; //θ(1)
                Node n3 = new Node(mat, matrix_size, x); //θ(n^2)      n: matrix_size
                n3.z_x = x.z_x; //θ(1)
                n3.z_y = x.z_y + 1; //θ(1)
                bool visited = HashMatrix.Contains(n3.Hash_Matrix); //θ(n)      n:
HashMatrix.Count
                if (!visited)  //θ(log(n))      n: priorityQueue.Count
                {
                    priorityQueue.Enqueue(n3, n3.Cost); //θ(log(n))      n:
priorityQueue.Count
                }
            }
            if (x.z_y - 1 >= 0) //θ(n^2)      n: matrix_size
            {
                int[,] mat = new int[x.matrix_size, x.matrix_size]; //θ(1)
                Array.Copy(x.matrix, mat, x.matrix_size * x.matrix_size); //θ(n^2)
n: matrix_size
                mat[x.z_x, x.z_y] = mat[x.z_x, x.z_y - 1]; //θ(1)
                mat[x.z_x, x.z_y - 1] = 0; //θ(1)
                Node n4 = new Node(mat, matrix_size, x); //θ(n^2)      n: matrix_size
                n4.z_x = x.z_x; //θ(1)
                n4.z_y = x.z_y - 1; //θ(1)
                bool visited = HashMatrix.Contains(n4.Hash_Matrix); //θ(n)      n:
HashMatrix.Count
                if (!visited)  //θ(log(n))      n: priorityQueue.Count
                {
                    priorityQueue.Enqueue(n4, n4.Cost); //θ(log(n))      n:
priorityQueue.Count
                }
            }
        }
        public int GetMatrixHash(int[,] matrix_hash) //θ(n^2)      n: matrix_size
        {
            int RandomHash = 193; //θ(1)
            for (int row = 0; row < matrix_size; row++) //θ(n^2)      n: matrix_size
            {
                for (int col = 0; col < matrix_size; col++) //θ(n)      n: matrix_size
                {
                    RandomHash = RandomHash * 59 + (matrix_hash[row, col]); //θ(1)
                }
            }
            return RandomHash; //θ(1)
        }
    }
}
```

Program.Cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Diagnostics;
using static N_PUZZLE.NPuzzleSolution;

namespace N_Puzz
{
    class Program
    {
        static void Main(string[] args) //θ(n^2)      n: MatrixSize
        {
            Console.WriteLine("N Puzzle Problem:\n[1] Sample test cases\n[2] Complete
testing"); //θ(1)
            Console.Write("\nEnter your choice [1-2]: "); //θ(1)
            char choice = (char)Console.ReadLine()[0]; //θ(1)
            bool succeed = false; //θ(1)
            switch (choice)
            {
                case '1':
                    Console.WriteLine("---------------------\n[1] Solvable\n[2]
UnSolvable"); //θ(1)
                    Console.Write("\nEnter your choice [1-2]: "); //θ(1)
                    char ch = (char)Console.ReadLine()[0]; //θ(1)
                    if (ch == '1')
                    {
                        Console.WriteLine("---------------------\n[1] 8 Puzzle (1)\n[2]
8 Puzzle (2)"); //θ(1)
                        Console.WriteLine("[3] 8 Puzzle (3)\n[4] 15 Puzzle - 1");
//θ(1)
                        Console.WriteLine("[5] 24 Puzzle 1\n[6] 24 Puzzle 2"); //θ(1)
                        Console.Write("\nEnter your choice [1-2-3-4-5-6]: "); //θ(1)
                        char ch1 = (char)Console.ReadLine()[0]; //θ(1)
                        if (ch1 == '1') //θ(n^2)      n: MatrixSize
                            succeed =
Check("..//net6.0//Sample//Solvable//8Puzzle.txt"); //θ(n^2)      n: MatrixSize
                        else if(ch1 == '2') //θ(n^2)      n: MatrixSize
                            succeed =
Check("..//net6.0//Sample//Solvable//20Puzzle.txt"); //θ(n^2)      n: MatrixSize
                        else if (ch1 == '3') //θ(n^2)      n: MatrixSize
                            succeed =
Check("..//net6.0//Sample//Solvable//14Puzzle.txt"); //θ(n^2)      n: MatrixSize
                        else if (ch1 == '4') //θ(n^2)      n: MatrixSize
                            succeed =
Check("..//net6.0//Sample//Solvable//15Puzzle.txt"); //θ(n^2)      n: MatrixSize
                        else if (ch1 == '5') //θ(n^2)      n: MatrixSize
                            succeed =
Check("..//net6.0//Sample//Solvable//11Puzzle.txt"); //θ(n^2)      n: MatrixSize
                        else if (ch1 == '6') //θ(n^2)      n: MatrixSize
                            succeed =
Check("..//net6.0//Sample//Solvable//24Puzzle.txt"); //θ(n^2)      n: MatrixSize
                    }
                    else if(ch == '2')
                    {
```

```
                        Console.WriteLine("--------------------\n[1] 8 Puzzle - Case 1
(1)\n[2] 8 Puzzle(2) - Case 1"); //θ(1)
                        Console.WriteLine("[3] 8 Puzzle(3) - Case 1\n[4] 15 Puzzle -
Case 2\n[5] 15 Puzzle - Case 3"); //θ(1)
                        Console.Write("\nEnter your choice [1-2-3-4-5]: "); //θ(1)
                        char ch1 = (char)Console.ReadLine()[0]; //θ(1)
                        if (ch1 == '1') //θ(n^2)      n: MatrixSize
                            succeed = Check("..//net6.0//Sample//Unsolvable//8 Puzzle -
Case 1.txt");
                        else if (ch1 == '2') //θ(n^2)      n: MatrixSize
                            succeed = Check("..//net6.0//Sample//Unsolvable//8
Puzzle(2) - Case 1.txt");
                        else if (ch1 == '3') //θ(n^2)      n: MatrixSize
                            succeed = Check("..//net6.0//Sample//Unsolvable//8
Puzzle(3) - Case 1.txt");
                        else if (ch1 == '4') //θ(n^2)      n: MatrixSize
                            succeed = Check("..//net6.0//Sample//Unsolvable//15 Puzzle
- Case 2.txt");
                        else if (ch1 == '5') //θ(n^2)      n: MatrixSize
                            succeed = Check("..//net6.0//Sample//Unsolvable//15 Puzzle
- Case 3.txt");
                    }
                    break;
                case '2':
                    Console.WriteLine("--------------------\n[1] Solvable\n[2]
UnSolvable\n[3] Very Larg"); //θ(1)
                    Console.Write("\nEnter your choice [1-2-3]: "); //θ(1)
                    char ch2 = (char)Console.ReadLine()[0]; //θ(1)
                    if (ch2 == '1')
                    {
                        Console.WriteLine("--------------------\n[1] Manhattan &
Hamming\n[2] Manhattan Only"); //θ(1)
                        Console.Write("\nEnter your choice [1-2]: "); //θ(1)
                        char ch3 = (char)Console.ReadLine()[0]; //θ(1)
                        if (ch3 == '1')
                        {
                            Console.WriteLine("--------------------\n[1] 50 Puzzle
(1)\n[2] 99 Puzzle - 1"); //θ(1)
                            Console.WriteLine("[3] 99 Puzzle - 2\n[4] 9999 Puzzle");
//θ(1)
                            Console.Write("\nEnter your choice [1-2-3-4]: ");  //θ(1)
                            char ch4 = (char)Console.ReadLine()[0]; //θ(1)
                            if (ch4 == '1') //θ(n^2)      n: MatrixSize
                                succeed = Check("..//net6.0//Complete//Solvable
puzzles//Manhattan & Hamming//18Puzzle.txt");
                            else if (ch4 == '2') //θ(n^2)      n: MatrixSize
                                succeed = Check("..//net6.0//Complete//Solvable
puzzles//Manhattan & Hamming//18Puzzle2.txt");
                            else if (ch4 == '3') //θ(n^2)      n: MatrixSize
                                succeed = Check("..//net6.0//Complete//Solvable
puzzles//Manhattan & Hamming//38Puzzle.txt");
                            else if (ch4 == '4') //θ(n^2)      n: MatrixSize
                                succeed = Check("..//net6.0//Complete//Solvable
puzzles//Manhattan & Hamming//4Puzzle.txt");
                        }
                        if (ch3 == '2')
                        {
                            Console.WriteLine("--------------------\n[1] 15 Puzzle 1
(1)\n[2] 15 Puzzle 3"); //θ(1)
                            Console.WriteLine("[3] 15 Puzzle 4\n[4] 15 Puzzle 5");
//θ(1)
                            Console.Write("\nEnter your choice [1-2-3-4]: "); //θ(1)
                            char ch4 = (char)Console.ReadLine()[0]; //θ(1)
```

```
                            if (ch4 == '1') //θ(n^2)      n: MatrixSize
                                succeed = Check("..//net6.0//Complete//Solvable
puzzles//Manhattan Only//46Puzzle.txt");
                            else if (ch4 == '2') //θ(n^2)      n: MatrixSize
                                succeed = Check("..//net6.0//Complete//Solvable
puzzles//Manhattan Only//38Puzzle.txt");
                            else if (ch4 == '3') //θ(n^2)      n: MatrixSize
                                succeed = Check("..//net6.0//Complete//Solvable
puzzles//Manhattan Only//44Puzzle.txt");
                            else if (ch4 == '4') //θ(n^2)      n: MatrixSize
                                succeed = Check("..//net6.0//Complete//Solvable
puzzles//Manhattan Only//45Puzzle.txt");
                        }
                    }
                    else if(ch2 == '2')
                    {
                        Console.WriteLine("---------------------\n[1] 15 Puzzle 1 -
Unsolvable\n[2] 99 Puzzle - Unsolvable Case 1"); //θ(1)
                        Console.WriteLine("[3] 99 Puzzle - Unsolvable Case 2\n[4] 9999
Puzzle"); //θ(1)
                        Console.Write("\nEnter your choice [1-2-3-4]: "); //θ(1)
                        char ch4 = (char)Console.ReadLine()[0]; //θ(1)
                        if (ch4 == '1') //θ(n^2)      n: MatrixSize
                            succeed = Check("..//net6.0//Complete//Unsolvable
puzzles//15 Puzzle 1 - Unsolvable.txt");
                        else if (ch4 == '2') //θ(n^2)      n: MatrixSize
                            succeed = Check("..//net6.0//Complete//Unsolvable
puzzles//99 Puzzle - Unsolvable Case 1.txt");
                        else if (ch4 == '3') //θ(n^2)      n: MatrixSize
                            succeed = Check("..//net6.0//Complete//Unsolvable
puzzles//99 Puzzle - Unsolvable Case 2.txt");
                        else if (ch4 == '4') //θ(n^2)      n: MatrixSize
                            succeed = Check("..//net6.0//Complete//Unsolvable
puzzles//9999 Puzzle.txt");
                    }
                    else if(ch2 == '3') //θ(n^2)      n: MatrixSize
                        succeed = Check("..//net6.0//Complete//V. Large test
case//TEST.txt");
                    if (succeed)
                        Console.WriteLine("\nCongratulations... your program runs
successfully"); //θ(1)
                    break;
                }
            }
            ///  terun 1Darry to 2Darray.......
            private static int[,] Conver_To_2D_Array(int[] Matrix1D, int MatrixSize)
//θ(n^2)      n: MatrixSize
            {
                int[,] Array2D = new int[MatrixSize, MatrixSize]; //θ(1)
                for (int i = 0; i < MatrixSize; i++) //θ(n^2)      n: MatrixSize
                {
                    for (int j = 0; j < MatrixSize; j++) //θ(n)        n: MatrixSize
                        Array2D[i, j] = Matrix1D[i * MatrixSize + j]; //θ(1)
                }
                return Array2D; //θ(1)
            }
            public static bool Check(string Name_of_file) //θ(n^2)      n: MatrixSize
            {
                FileStream File = new FileStream(Name_of_file, FileMode.Open,
FileAccess.Read); //θ(n^2)      n: MatrixSize
                StreamReader Stream_Reader = new StreamReader(File);
                int wrongAnswer = 0; //θ(1)
                int MatrixSize = int.Parse(Stream_Reader.ReadLine()); //θ(1)
```

```
            int[] Matrix1D = new int[MatrixSize * MatrixSize]; //θ(1)
            int k = 0; //θ(1)
            for (int i = 0; i < MatrixSize; i++)  //θ(n^2)      n: MatrixSize
            {
                string line = Stream_Reader.ReadLine(); //θ(1)
                string[] ss = line.Split(' '); //θ(1)

                for (int j = 0; j < MatrixSize; j++)  //θ(n)      n: MatrixSize
                {
                    int x = Int32.Parse(ss[j]); //θ(1)
                    Matrix1D[k++] = x; //θ(1)
                }
            }
            int RightAnswer = int.Parse(Stream_Reader.ReadLine()); //θ(1)
            int receivedResult;
            if (CheckSolvability(Matrix1D, MatrixSize))  //θ(n^4) = θ(s^2)      n:
MatrixSize,  s: Matrix Length
            {
                receivedResult =
N_PUZZLE.NPuzzleSolution.AStarAlgorithm(Conver_To_2D_Array(Matrix1D, MatrixSize),
MatrixSize, Matrix1D);
            }
            else
            {
                Console.WriteLine("\n====================="); //θ(1)
                Console.WriteLine("    Not Solvable     "); //θ(1)
                Console.WriteLine("====================="); //θ(1)
                return false; //θ(1)
            }
            if (receivedResult != RightAnswer) //θ(1)
            {
                Console.WriteLine("\nwrong answer at number of movements --> expected =
" + RightAnswer + " received = " + receivedResult + "\n"); //θ(1)
                return false; //θ(1)
            }
            else  //θ(1)
            {
                if (MatrixSize == 3) //θ(n^2 * c)      n: MatrixSize      c: number of
nodes in path
                {
                    PrintPath(n1); //θ(n^2 * c)      n: MatrixSize      c: number of
nodes in path
                }
                Console.WriteLine("\n# of movements = " + n1.level); //θ(1)
                Console.WriteLine("\nime : " + CalculateTime.Elapsed); //θ(1)
                Console.WriteLine("\nCongratulations... :)\n"); //θ(1)
                return true; //θ(1)
            }
            Stream_Reader.Close(); //θ(1)
            File.Close(); //θ(1)
        }
    }
}
```

Comparing Table:

File Name	Manhattan Time	Hamming Time	Num. Of Moves
50 Puzzle(1).txt	00:00:00.0072954	00:00:00.0874090	18
99 Puzzle - 1.txt	00:00:00.0018032	00:00:00.0018385	18
99 Puzzle - 2.txt	00:00:00.0019596	00:00:00.0025287	38
9999 Puzle.txt	00:00:00.0024273	00:00:00.0022700	4
