Part 1

My h3 algorithm functions in a simple way. It checks the position of the tiles similar to the Misplaced-Tiles heuristic, but also includes the order of the tiles.

For Information about running for building and running the program, please read the README.md file.

Part 2

S1 - 352617084

BFS:
Total nodes generated: 468380
Total time: 0.47156286239624023
Solution found in 24 moves:
DDLUULDRRULLDRRDLLURRULL

IDS:
Total nodes generated: 604664
Total time: 0.30230283737182617
Solution found in 28 moves:
DDLURULLDRRDLLURRULLDRRULDLU

H1:
Total nodes generated: 140621
Total time: 0.8037943840026855
Solution found in 24 moves:
DDLUULDRRULLDRRDLLURRULL

H2:
Total nodes generated: 31025
Total time: 0.10871195793151855
Solution found in 24 moves:
DDLUULDRRULLDRRDLLURRULL

H3:
Total nodes generated: 45094
Total time: 0.2088155746459961
Solution found in 24 moves:
DDLUULDRRULLDRRDLLURRULL

S2 - 651482037

BFS:
Total nodes generated: 164348
Total time: 0.1541588306427002
Solution found in 20 moves:
DDLLURULDRRDLLURRULL

IDS:
Total nodes generated: 264792
Total time: 0.12874412536621094
Solution found in 20 moves:
DDLLURULDRRDLLURRULL

H1:
Total nodes generated: 42701
Total time: 0.23002338409423828
Solution found in 20 moves:
DDLLURULDRRDLLURRULL

H2:
Total nodes generated: 5017
Total time: 0.01599860191345215
Solution found in 20 moves:
DDLLURULDRRDLLURRULL

H3:
Total nodes generated: 8288
Total time: 0.03679633140563965
Solution found in 20 moves:
DDLLURULDRRDLLURRULL

S3 - 752631480

All of them:
The puzzle is not solvable

7 5 2
6 3 1
4 8 _

S4 - 867254301

BFS:
Total nodes generated: 725756
Total time: 0.7214357852935791
Solution found in 31 moves:
DDLURRULLDRRDLLURURDDLUURDDLLUU

IDS:
Total nodes generated: 813500
Total time: 0.4030475616455078
Solution found in 31 moves:
LDRRULLDDRRULDRUULLDRULDDRRUULL

H1:
Total nodes generated: 241804
Total time: 1.6019611358642578
Solution found in 31 moves:
DDLURRULLDRRDLLURURDDLUURDDLLUU

H2:
Total nodes generated: 207997
Total time: 0.8724639415740967
Solution found in 31 moves:
LDDRRUULLDRRULLDRDLURDRUULLDRUL

H3:
Total nodes generated: 224759
Total time: 1.2797377109527588
Solution found in 31 moves:
DDLURRULLDRRDLLURURDDLUURDDLLUU

S5 - 413206758

BFS:
Total nodes generated: 488
Total time: 0.0010001659393310547
Solution found in 6 moves:
RDLUUL

IDS:
Total nodes generated: 348
Total time: 0.0
Solution found in 6 moves:
RDLUUL

H1:
Total nodes generated: 135
Total time: 0.001999378204345703
Solution found in 6 moves:
RDLUUL

H2:
Total nodes generated: 28
Total time: 0.0
Solution found in 6 moves:
RDLUUL

H3:
Total nodes generated: 30
Total time: 0.0
Solution found in 6 moves:
RDLUUL

Part 3

```
Processing Part3\L8\256.txt
Processing Part3\L8\258.txt
```

depth	bfs-nodes	bfs-time	ids-nodes	ids-time	A*h1-nodes	A*h1-time	A*h2-nodes	A*h2-time	A*h3-nodes	A*h3-time
15	26878.4	0.0267565	60574.2	0.0309334	6589.05	0.0352716	787.15	0.00235792	1073.4	0.00489249
24	512120	0.522903	1.44305e+06	0.736528	143222	0.856261	31845.2	0.112931	46223.3	0.220157
8	917.2	0.000743544	1119.8	0.000548673	247.25	0.00110357	45.2	0.000198948	51.3	0.000206411

```
Process finished with exit code 0
```

The required table can be achieved by running the program in automatic mode. Running information on how to do that, can be found in the README.md file.

With a memory queue of already visited states, BFS seems to be performing pretty well. But with the exponential growth of the generated nodes, it is possible to run into memory issues with bigger depth issues and bigger puzzles.

As for IDS, it is generating more nodes, due to the constant re-iteration of a DFS algorithm in ever increasing depth and therefore the loss of the memory queue. However, this in turn decreases the memory usage making it more efficient memory wise for bigger and more difficult puzzles. It also seems to be on the slower side, compared to BFS which could be attributed to it needing to get to a depth, where it is able to find a solution, first.

A* with the Misplaced-States Heuristics seems to perform pretty poorly compared to the other algorithms, especially in the average time. It is the slowest algorithm in all depths, which could be attributed to it needing to calculate the heuristics often. Misplaced tiles is a simple calculation, but also not very accurate for finding solutions. Therefore it needs to run the calculations a lot of times (the most out of the implemented heuristics as can be seen by the generated nodes) which in turn slows the algorithm down.

A* with the Manhattan-Distance Heuristics is performing the best out of all the implemented solutions. Due to a very efficient way of calculating the cost/value of a state, it seems to find the shortest solution in the least amount of time, with the least generated nodes.

I was personally very surprised by the performance of my Order-Heuristics as I “designed” it on a fly while reading through the assignment. Including the order of tiles into the Misplaced-Tiles Heuristics has improved the accuracy of calculating the value of a state greatly, which in turn improved the time and memory efficiency greatly. It outperformed every algorithm except A* with Manhattan heuristics which makes it the second best algorithm implemented.

In summary the experiment showed how different algorithms perform at different problem depths and made it easy to understand what algorithm might be suitable for what environment.
As an easy algorithm with no calculations, BFS can definitely be used for shallower problems. With increasing depth, IDS should be considered for that case. If memory and computation are not a problem, A* can be chosen. The experiment also showed the importance of using a accurate heuristics function when using an A* algorithm.