Question 1.1)

The 8-puzzle problem can be seen as a search problem, as each unique configuration of tiles may be seen as a state, creating a state space composed of all possible tile configurations. As only tiles adjacent to the blank space can move into the blank space, the movement of a tile may be represented as a transition from one state into another. Because of this, the goal state is reachable after a sequence of specific transitions. This is a search problem, as there will always be multiple transitions from a state, but commonly these transitions will likely lead to the goal state in a sub-optimal way, or not at all. Therefore, a range of paths must be searched for the optimal route to the goal.

Question 1.2)

1) The A* algorithm is an algorithm that can be used to find the optimal route to a goal state through a state transition diagram. It functions by looking at the cumulative cost of each path from a given node to its neighbours, choosing and taking the path with the lowest cumulative cost. The total cost of the journey to each node is recorded until the goal state is reached, resulting in the most optimal path to the goal state from the start state.

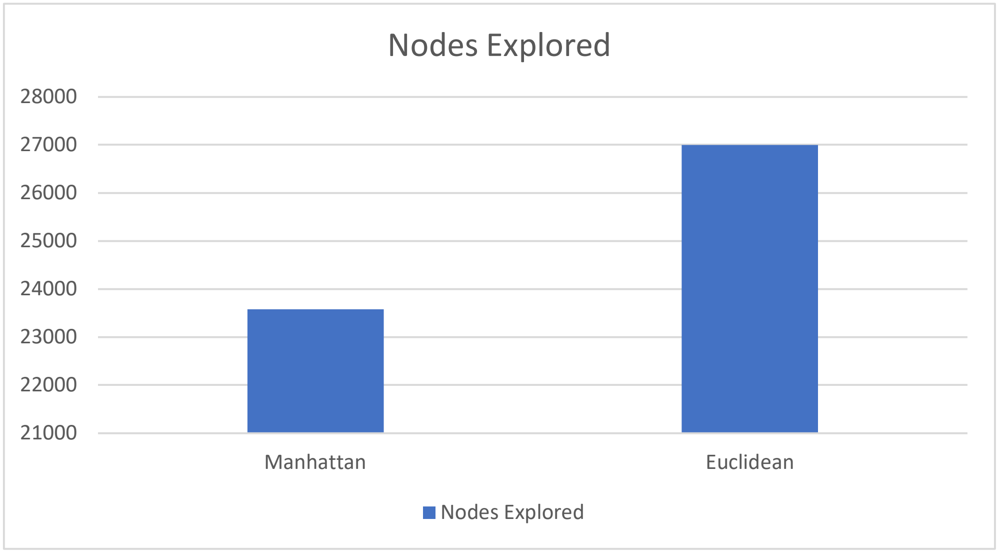
2) Manhattan distance is the measure of the absolute difference between two sets of co-ordinates. It is an admissible heuristic in this scenario, as the difference in co-ordinates will always be equal to or greater than the number of moves required. A tile will always need to move the distance of at least 1 tile if it does not start within its goal state, which the Manhattan distance will always measure.
Mathematically, it is given by the formula $h(n) = (|n_x| + |n_y|)$, where n is a tile and n_x n_y are the difference in x and y co-ordinates of that tile in the goal state.
When the tile starts in its goal, no moves are required, which is included in the heuristic $(|0| + |0|) = 0$. More moves may be required, despite it remaining in place, meaning it will never overestimate the distance required. In all other states, $|n_x| + |n_y|$ will have either value as one or more, which is equal to or lower than the required number of moves.

Euclidean distance is a measure of the total length between two given points. It is given by the formula $h(n) = \sqrt{n_x^2 + n_y^2}$. Where n is a tile, n_x and n_y are the difference in locations between the tiles current state and the goal state.
It is an admissible heuristic in this scenario, as $h(n)$ will give 0 in the event the tile starts on its space within the goal state. This will be equal to or less than the required number of moves to reach the goal state of the puzzle. In any other case, the function will be at least $\sqrt{1^2}$, which is 1, which is equal to or less than the minimum number of moves required to reach the goal state from another tile.

These two heuristics were chosen over Hamming distance, as Hamming distance only measures the number of different tiles from the start and end state. Although it is admissible, the change in distance between two given states can be negligible, meaning that not minimal information would be gathered from certain transitions.

3) Python scripts may be found within Zip. The script is 8PuzzleGame.py, and uses the configuration from figure 1 by default. Custom inputs may be entered for reaching the goal at a faster rate for testing purposes.

4) Manhattan distance as a heuristic explored 23754 nodes before reaching its goal.
Euclidean distance as a heuristic explored 26999 nodes before reaching its goal.
Manhattan distance therefore performs better as it explores less nodes. Additionally, Euclidean distance requires more complex operations, such as square rooting, which may also affect performance.



Question 1.3)

The code has the option of receiving custom inputs to create different starting ending tile configurations. Provided that each input passes the requirement to have the chosen numbers without duplicates, then any input should be possible with any output. This is due to this algorithm and implementation being a traversal of all possible states that the 8 puzzle could be. As a result, provided the inputs are within the state space, then it should be possible to construct an optimal path from the start state to the end state.

Question 2.1)

1) a) The solution space is represented as a list called “population”. Each solution is a list of rows. Each row is created to contain only one instance of each number, so each row is guaranteed to be suitable for a solution. Each row has fixed values/indices to ensure that it remains part of the valid solution.
Genetically, each row represents a chromosome, and each solution is made up of 9 individual chromosomes.

b) The fitness function used will be based on the number of errors within a given solution. The formula I have used is *81 – (number of errors within a column + number of errors within a 3X3 square)*. This is suitable, as a completed solution will have a fitness score of 81 (the maximum), and any other value will have a smaller value, with worse solutions having a lower fitness than those more suitable.

c) The crossover function I have used constructs a random solution based on two parents. An empty solution is created, and each row is chosen from either parent based on a random selection. This represents an exchanging of chromosomes, as a child of two parents can have a combination of chromosomes from each parent.

d) The mutation operation I have used chooses to mutate each member of the population based on a probability given by *1/population size*. When a solution mutates, a chromosome mutates to a new form by randomising the organisation of numbers within row. This ensures new chromosomes are within the gene pool, allowing new possible beneficial mutations.

e) The population is initialised as a list of sudoku puzzles assembled from random rows. This represents a random spread of organisms, with individual combinations of different chromosomes resulting in different levels of fitness.

f) Truncation selection is used to select which solutions are used within the next generation. The average fitness of each solution is taken, and the solutions with greater than average fitness are used. The spaces below average are instead replaced with the children of parents of the above average fitness.

g) The algorithm terminates once a solution with a fitness score of *81* is reached, or a set number of iterations chosen by the user is reached. This is due to the fact that a solution to the problem has been found, as represented by a fitness score of 81, or that the efficiency of the program is not performing to a desired standard.

2) Each grid was tested 5 times, and the average results of each are being displayed in the graphs and tables below. Unfortunately, the program was not capable of reaching a solution within the set number of iterations for each population count.
Populations of 10 were iterated over 1000 times.
Populations of 100 were iterated over 100 times.
Populations of 1000 and 10000 were iterated over 50 times.
The average fitness for the first generation, the highest average fitness for a given generation and value of highest scoring grid are recorded for each test. There are some cases where the highest average fitness is greater than the fitness of the best puzzle, this may be a result of floating point calculations.
These are the results for grid 1:

						Grid 1								
10	I = 1000						100	I = 100						
Seed	1	2	3	4	5	Average	Seed	1	2	3	4	5	Average	
Highest Avg	57	56	56	54	57	56	Highest Avg	55.91	59.51	57.28	59.55	61.26	58.702	
Starting Avg	43.8	43.9	41.9	42	44.9	43.3	Starting Avg	45.24	41	43.44	44.67	45.09	43.888	
Best Puzzle	57	56	56	54	57	56	Best Puzzle	59	61	58	61	63	60.4	
1000	I = 50						10000	I = 50						
Seed	1	2	3	4	5	Average	Seed	1	2	3	4	5	Avg	Total Averages
Highest Avg	64.88	66.1	70.31	69.21	67.11	67.522	Highest Avg	74.22	71.16	73.05	73.06	69.92	72.282	63.6265
Starting Avg	44.7	44.6	44.33	44.51	44.39	44.506	Starting Avg	44.56	44.62	44.58	44.64	44.57	44.594	44.072
Best Puzzle	66	67	72	72	68	69	Best Puzzle	74	73	73	72	69	72.2	64.4

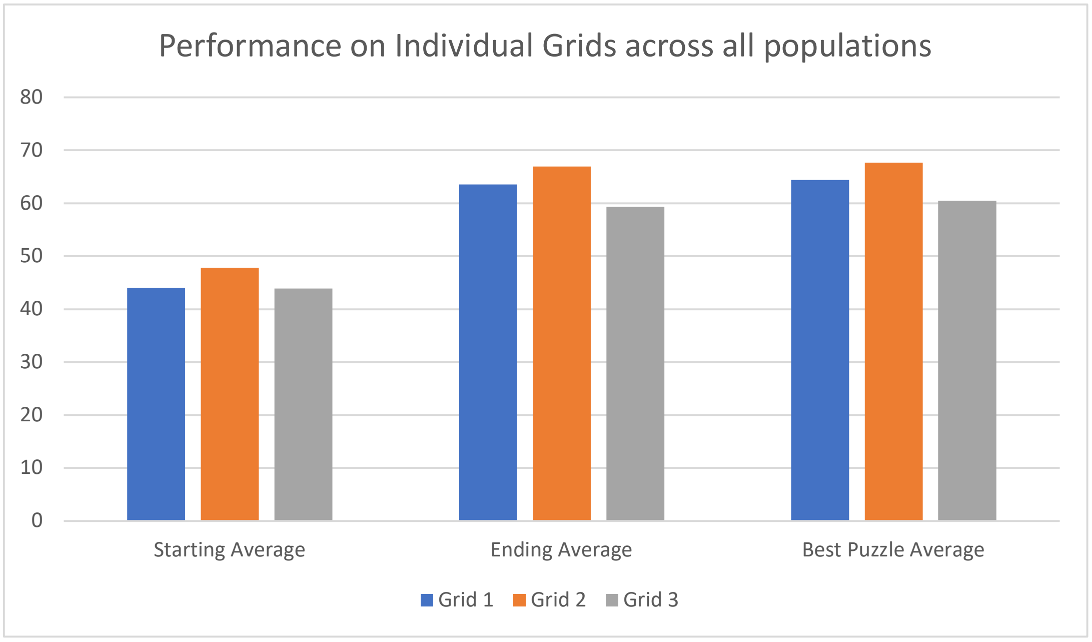
These are the results for grid 2:

						Grid 2								
10	I = 1000						100	I = 100						
Seed	1	2	3	4	5	Average	Seed	1	2	3	4	5	Average	
Highest Avg	61	58	59	58	59	59	Highest Avg	59.71	64.72	64.16	61.13	67.69	63.482	
Starting Avg	47.7	48.3	49.7	48.3	47.8	48.36	Starting Avg	47.81	48.37	47.4	47.61	48.08	47.854	
Best Puzzle	61	58	59	58	59	59	Best Puzzle	62	65	65	61	69	64.4	
1000	I = 50						10000	I = 50						
Seed	1	2	3	4	5	Average	Seed	1	2	3	4	5	Avg	Total Averages
Highest Avg	75.71	69.4	71.14	71.67	67.71	71.126	Highest Avg	74.07	75.79	73.43	75.82	72.15	74.252	66.965
Starting Avg	47.61	47.54	47.58	47.5	47.71	47.588	Starting Avg	47.53	47.54	47.53	47.5	47.61	47.542	47.836
Best Puzzle	76	71	72	73	70	72.4	Best Puzzle	75	76	74	77	73	75	67.7

These are the results for grid 3:

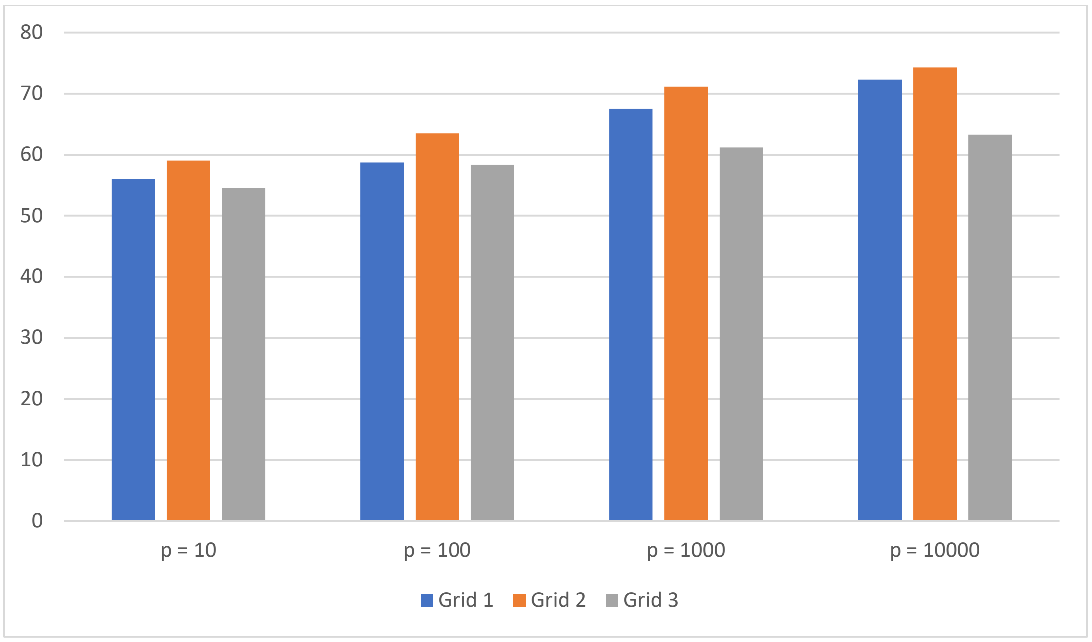
						Grid 3								
10	I = 1000						100	I = 100						
Seed	1	2	3	4	5	Average	Seed	1	2	3	4	5	Average	
Highest Avg	54	54	54.7	53	57	54.54	Highest Avg	60.27	58.14	58	59.1	56.4	58.382	
Starting Avg	41	44	44.1	43.5	43.3	43.18	Starting Avg	44.06	44.12	44.19	44.16	44.31	44.168	
Best Puzzle	54	54	55	53	57	54.6	Best Puzzle	61	58	61	61	57	59.6	
1000	I = 50						10000	I = 50						
Seed	1	2	3	4	5	Average	Seed	1	2	3	4	5	Avg	Total Averages
Highest Avg	61.67	61.03	60.88	61	61.2	61.156	Highest Avg	63.29	63.09	62.91	64.09	63.02	63.28	59.3395
Starting Avg	44.18	44.34	44.14	44.12	44	44.156	Starting Avg	44.16	44.18	44.06	44.15	44.12	44.134	43.9095
Best Puzzle	63	65	64	63	63	63.6	Best Puzzle	66	64	62	64	64	64	60.45

These results are represented on the following graphs:



Question 2.2)

1) The best population size is 10000, resulting in greater heights of average populations.
The following graph shows the comparison of best average populations based on population size:



From the graph, we can see that the average fitness of each best generation increases as the population increases. Therefore, p = 10000 was the best population.

2) This is likely due to the possible variations within the gene pool, allowing greater ranges of unique solutions. It may also be due to the population and iterations not being proportional, so more information is calculated with higher populations.

3) We can also see in the same grid that Grid 3 has the lowest value for the best performing generation across all populations. This means that Grid 3 is the hardest to solve, as the solution is the furthest distance away.

4) This is likely due to the number of positions that have been initially filled out. Grid 3 has a smaller number of positions filled out compared to Grids 2 and 1. We also see that Grid 2 has the best performance in all populations, which is likely due to it having the greatest number of initially filled out positions.

5) It would be useful to conduct further experiments with different mutation levels, selection methods, and fitness algorithms. This would encourage different interactions within populations, resulting in more or less efficient evolutionary algorithms, which could allow a study of calculation speed and efficiency, as well as a study of the initial discoveries.
It may also be beneficial to test more grids with greater and smaller numbers of pre-defined numbers to confirm the hypothesis that, the more starting numbers there are, the easier it is to fill reach a solution.