Question 1.1:

- The maze solver can be seen as a search problem due to the fact that a maze can be represented as a tree or a graph, where each ‘-’ (dash) can be represented as a node and each ‘#’ (hashtag) can be represented as a wall of a maze that cannot be entered or passed through. Each node can have neighbors. These neighbors are equivalent to branches, which represent different possible paths in the maze that could potentially lead to the finish node. Graph traversal algorithms, such as depth-first search, breadth-first search and Dijkstra’s shortest path algorithm can all be used to solve the search problem. A search algorithm will start at the starting node, traverse the graph and search through different branches as necessary (how it traverses these branches are different, depending on the search algorithm used) until it reaches the ending node.

Question 1.2:

1) The depth-first search algorithm works by starting at a node and traversing as far as possible down each branch before backtracking. Depth-first search differs from other search algorithms as it uses a stack to keep track of what nodes have been visited. The algorithm works by adding the starting node to the stack, ‘popping’ the node off of the stack and visiting its neighbors (adding these neighbors to the stack if not visited), checking if any of these nodes are the finish node. If not, it will repeat the previous steps until the node that is popped off of the stack is the finish node. Backtracking works by ‘popping’ nodes off of the stack until it is able to traverse a new branch again.

2) (0,1)
(1,1)
(1,2)
(1,3)
(1,4)
(1,5)
(1,6)
(1,7)
(1,8)
(1,9)
(1,10)
(1,11)
(1,12)
(1,13)
(1,14)

(1,15)
(2,15)
(3,15)
(4,15)
(5,15)
(6,15)
(6,16)
(6,17)
(7,17)
(8,17)
(8,18)
(9,18)

3) Resultant path of DFS is of length: 27
Number of nodes explored by using DFS: 35
Total execution time in milliseconds: 48

   The above performance statistics have been calculated for the DFS search on maze-Easy.txt. The resultant path is the final path taken from the starting node to the finishing node (and the length is how many nodes are included in this path). The number of nodes explored is the number of nodes explored by DFS to get to the final node; this will include the nodes in the final path as well as nodes that were not (such as nodes down a branch that does not lead to the final node). The total execution time is the time taken to complete the DFS search. As shown in the above performance statistics, the algorithm visited a total of 8 nodes that were not a part of the final path. This can be calculated by subtracting the number of nodes explored by the number of nodes in the resultant path.

4) Due to the large number of steps taken to reach the final node, I have decided not to include the final paths generated from the DFS algorithm executed on these mazes in the PDF, but rather include them in my submission's zip folder. The names of the output files have been named accordingly (dfsmazeeasy_output is the output file containing the nodes in the resulting path for maze-Easy.txt, dfsmazemedium_output for maze-Medium.txt, dfsmazelarge_output for maze-Large.txt and dfsmazevlarge_output for maze-VLarge.txt).

Statistics obtained after executing DFS search on maze-Medium.txt:
Resultant path of DFS is of length: 469
Number of nodes explored by using DFS: 881
Total execution time in milliseconds: 294

The above performance statistics have been calculated for the DFS search on maze-Medium.txt. The resultant path that my DFS search took to go from the starting node to the end node has a length of 469 nodes. The DFS search on this maze explored a total of 881 nodes. The algorithm visited a total of 412 nodes that were not a part of the final path. The search took 294 milliseconds to complete.

Statistics obtained after executing DFS search on maze-Large.txt:
Resultant path of DFS is of length: 1050
Number of nodes explored by using DFS: 73979
Total execution time in milliseconds: 1867

The above performance statistics have been calculated for the DFS search on maze-Large.txt. The resultant path that my DFS search took to go from the starting node to the end node has a length of 1050 nodes. The DFS search on this maze explored a total of 73979 nodes. The algorithm visited a total of 72929 nodes that were not a part of the final path. The search took 1867 milliseconds to complete.

Statistics obtained after executing DFS search on maze-VLarge.txt:
Resultant path of DFS is of length: 3737
Number of nodes explored by using DFS: 389070
Total execution time in milliseconds: 10107

The above performance statistics have been calculated for the DFS search on maze-VLarge.txt. The resultant path that my DFS search took to go from the starting node to the end node has a length of 3737 nodes. The DFS search on this maze explored a total of 389070 nodes. The algorithm visited a total of 385333 nodes that were not a part of the final path. The search took 10107 milliseconds to complete.

Question 1.3:

1) Another algorithm that I used to solve the maze is A* search. A* search works by using heuristics to find the shortest path from the start node to the end node. Unlike depth-first search, it uses a priority queue to keep track of nodes and examines the f value of these nodes to determine the priority. The f value is calculated by adding the actual cost from the start node to the current node, along with the estimated cost from the current node to the end node. I decided to use A* search because I believe that, due to the added intelligence of being able to discern between nodes and decide whether one node will be a better choice that will lead to the final node, it should be more efficient by exploring less nodes and therefore be faster. Over depth-first search, I expect the time taken to decrease, the number of nodes explored to decrease and the number of nodes in the resultant path to decrease too. I expect the number of nodes explored to decrease due to the added intelligence of being able to determine whether a neighboring node will be a better choice than another neighbor, unlike depth-first search, where a neighbor to traverse through is picked arbitrarily. If the number of nodes explored were to decrease, then I believe that the time taken should decrease too. If a good heuristic algorithm is utilised, then the A* search algorithm should generate the shortest path, therefore resulting in the number of nodes in the resultant path to decrease too.

2) Due to the large number of steps taken to reach the final node, I have decided not to include the final paths generated from the A* algorithm executed on these mazes in the PDF, but rather include them in my submission's zip folder. The names of the output files have been named accordingly (astarmazeeasy_output is the output file containing the nodes in the resulting path for maze-Easy.txt, astarmazemedium_output for maze-Medium.txt, astarmazelarge_output for maze-Large.txt and astarmazevlarge_output for maze-VLarge.txt).

3) Statistics obtained after executing A* search on maze-Easy.txt:
   Resultant path after using A* algorithm is of length: 27
   Number of nodes explored by using A* algorithm: 30
   Total execution time in milliseconds: 53

The above performance statistics have been calculated for the A* search on maze-Easy.txt. The resultant path that my A* search took to go from the starting node to the end node has a length of 27 nodes. The A* search on this maze explored a total of 30 nodes. The algorithm visited a total of 3 nodes that were not a part of the final path. The search took 53 milliseconds to complete.

Statistics obtained after executing A* search on maze-Medium.txt:
Resultant path after using A* algorithm is of length: 339
Number of nodes explored by using A* algorithm: 535
Total execution time in milliseconds: 248

The above performance statistics have been calculated for the A* search on maze-Medium.txt. The resultant path that my A* search took to go from the starting node to the end node has a length of 339 nodes. The A* search on this maze explored a total of 535 nodes. The algorithm visited a total of 196 nodes that were not a part of the final path. The search took 248 milliseconds to complete.

Statistics obtained after executing A* search on maze-Large.txt:
Resultant path after using A* algorithm is of length: 1092
Number of nodes explored by using A* algorithm: 11754
Total execution time in milliseconds: 1156

The above performance statistics have been calculated for the A* search on maze-Large.txt. The resultant path that my A* search took to go from the starting node to the end node has a length of 1092 nodes. The A* search on this maze explored a total of 11754 nodes. The algorithm visited a total of 10662 nodes that were not a part of the final path. The search took 1156 milliseconds to complete.

Statistics obtained after executing A* search on maze-VLarge.txt:
Resultant path after using A* algorithm is of length: 3691
Number of nodes explored by using A* algorithm: 45114
Total execution time in milliseconds: 8062

The above performance statistics have been calculated for the A* search on maze-VLarge.txt. The resultant path that my A* search took to go from the starting node to the end node has a length of 3691 nodes. The A* search on this maze explored a total of 45114 nodes. The algorithm visited a total of 41423 nodes that were not a part of the final path. The search took 8062 milliseconds to complete.

4) We can compare the performance of each search algorithm on each maze to determine which algorithm performed better than the other.

   For maze-Easy.txt, both algorithms got a final path with a length of 27 nodes, however depth-first search explored 5 more nodes than A* search. Depth-first search was slightly faster by 5 milliseconds than A* search. Both algorithms performed equally as well for maze-Easy.txt.

   For maze-Medium.txt, depth-first search obtained a final path with a length of 469 nodes, which is significantly more than the final path obtained by A* search, which has a length of 339 nodes. Depth-first search explored more nodes than A* search to get to the final node, at 881 nodes compared to A* search's 535 nodes. Depth-first search took 294 milliseconds to complete, which is slightly longer than A* search's 248 milliseconds to complete. A* search is the unanimous winner for this maze, beating depth-first search in every performance statistic for searching maze-Medium.txt.

   For maze-Large.txt, depth-first search obtained a final path with a length of 1050 nodes, which is slightly better than the path obtained by A* search, which has a length of 1092 nodes. Depth-first search explored notably more nodes than A* search to get to the final node, at 73979 nodes compared to A* search's 11754 nodes. Depth-first search took 1867 milliseconds to complete, which is significantly longer than A* search's 1156 milliseconds to complete. A* search performed better on maze-Large.txt than depth-first search, by scoring better performance statistics in nodes explored and execution time, however depth-first search had the more optimal path.

   For maze-VLarge, depth first search obtained a final path with a length of 3737 nodes, which is slightly worse than the path obtained by A* search, which has a length of 3691 nodes. Depth-first search explored notably more nodes than A* search to get to the final node, at 389070 nodes compared to A* search's 45114 nodes. Depth-first search took 10107 milliseconds to complete, which is significantly longer than A* search's 8062 milliseconds to complete. A* search performed better in all performance metrics on maze-VLarge.txt, beating depth-first search by having a more optimal path, exploring less nodes and taking less time to execute

Over all four mazes, A* search wins, performing equally as good in maze-Easy.txt, however performing better than depth-first search in maze-Medium.txt, maze-Large.txt and maze-VLarge.txt. Overall, A* search explored less nodes than depth-first search due to the added intelligence via a heuristic function and a priority queue. My implementation of a heuristic function also led to faster execution times and shorter paths on some of the mazes compared to depth-first search.