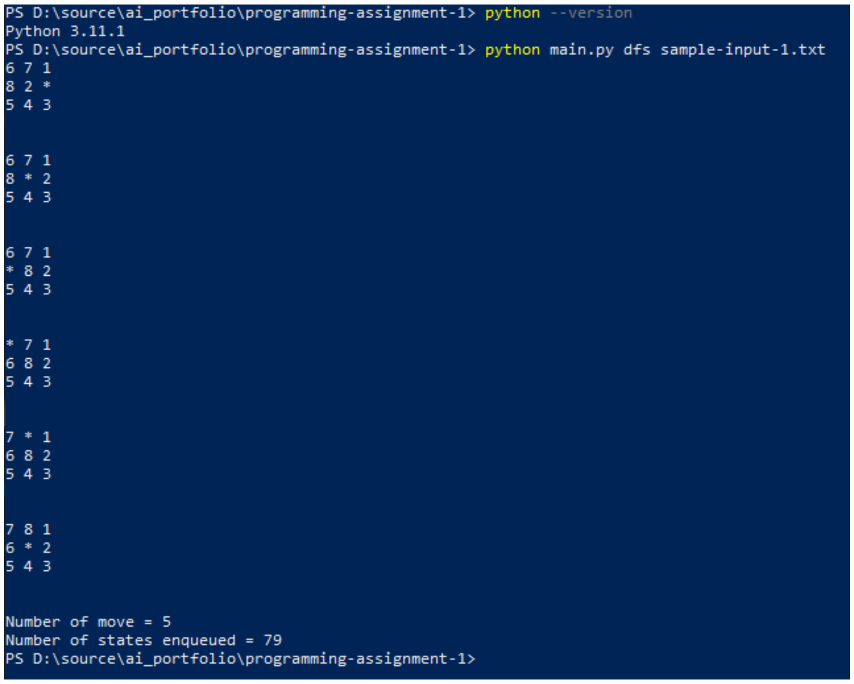
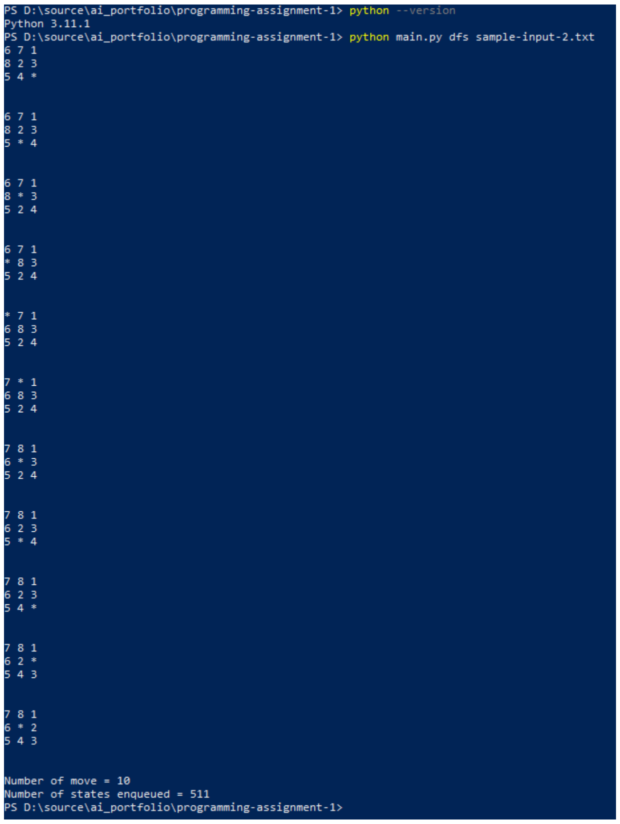
# Sample Input and Corresponding Output

**Depth-First Search algorithm:**

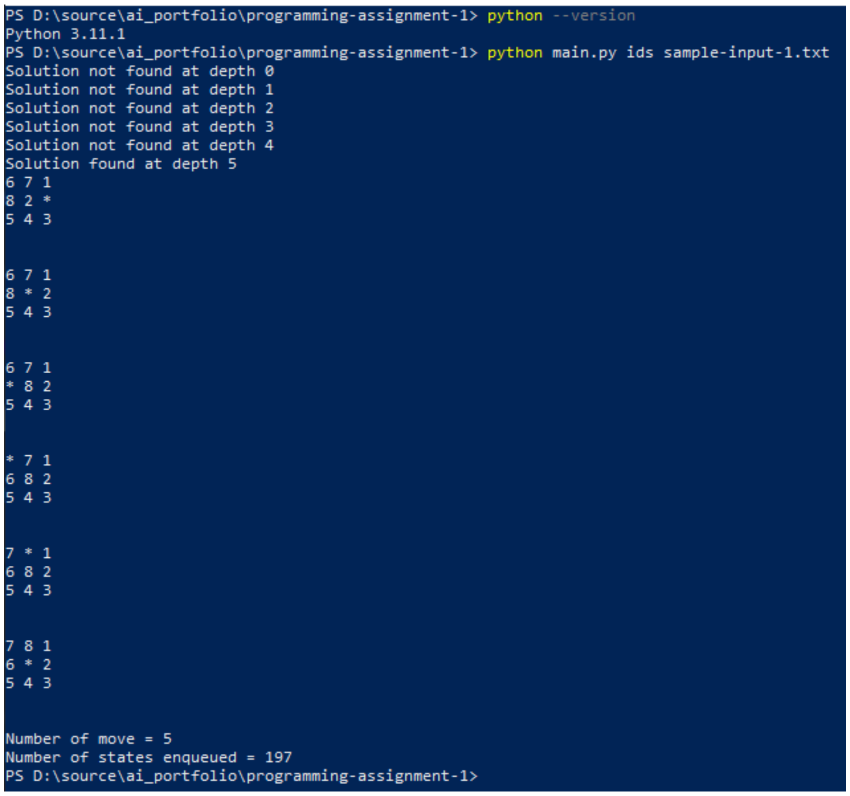


```
PS D:\source\ai_portfolio\programming-assignment-1> python --version
Python 3.11.1
PS D:\source\ai_portfolio\programming-assignment-1> python main.py dfs sample-input-1.txt
6 7 1
8 2 *
5 4 3


6 7 1
8 * 2
5 4 3


6 7 1
* 8 2
5 4 3


* 7 1
6 8 2
5 4 3


7 * 1
6 8 2
5 4 3


7 8 1
6 * 2
5 4 3


Number of move = 5
Number of states enqueued = 79
PS D:\source\ai_portfolio\programming-assignment-1>
```

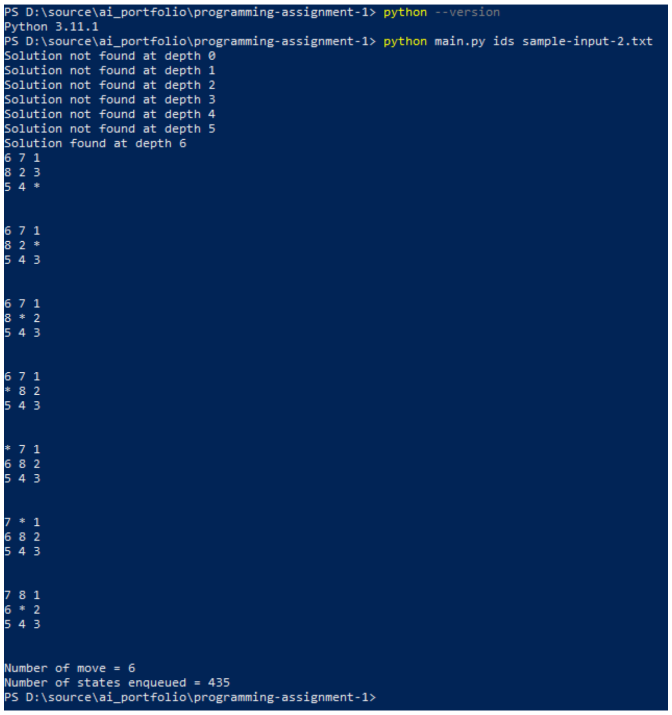


```
PS D:\source\ai_portfolio\programming-assignment-1> python --version
Python 3.11.1
PS D:\source\ai_portfolio\programming-assignment-1> python main.py dfs sample-input-2.txt
6 7 1
8 2 3
5 4 *


6 7 1
8 2 3
5 * 4


6 7 1
8 * 3
5 2 4


6 7 1
* 8 3
5 2 4


* 7 1
6 8 3
5 2 4


7 * 1
6 8 3
5 2 4


7 8 1
6 * 3
5 2 4


7 8 1
6 2 3
5 * 4


7 8 1
6 2 3
5 4 *


7 8 1
6 2 *
5 4 3


7 8 1
6 * 2
5 4 3


Number of move = 10
Number of states enqueued = 511
PS D:\source\ai_portfolio\programming-assignment-1>
```

**Iterative deepening search algorithm:**



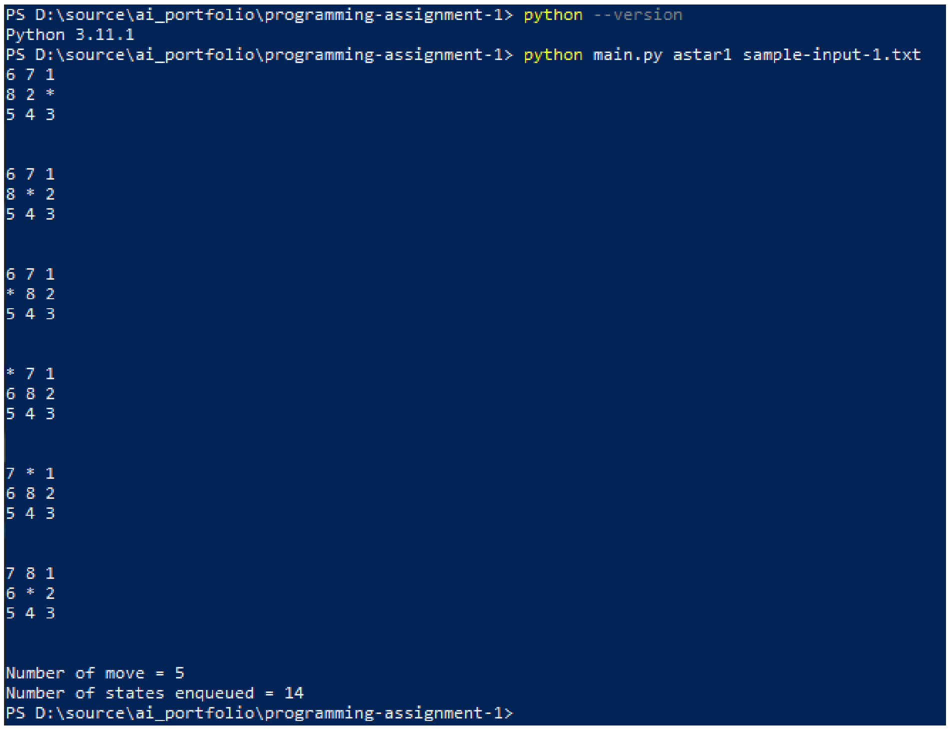
```
PS D:\source\ai_portfolio\programming-assignment-1> python --version
Python 3.11.1
PS D:\source\ai_portfolio\programming-assignment-1> python main.py ids sample-input-1.txt
Solution not found at depth 0
Solution not found at depth 1
Solution not found at depth 2
Solution not found at depth 3
Solution not found at depth 4
Solution found at depth 5
6 7 1
8 2 *
5 4 3


6 7 1
8 * 2
5 4 3


6 7 1
* 8 2
5 4 3


* 7 1
6 8 2
5 4 3


7 * 1
6 8 2
5 4 3


7 8 1
6 * 2
5 4 3


Number of move = 5
Number of states enqueued = 197
PS D:\source\ai_portfolio\programming-assignment-1>
```



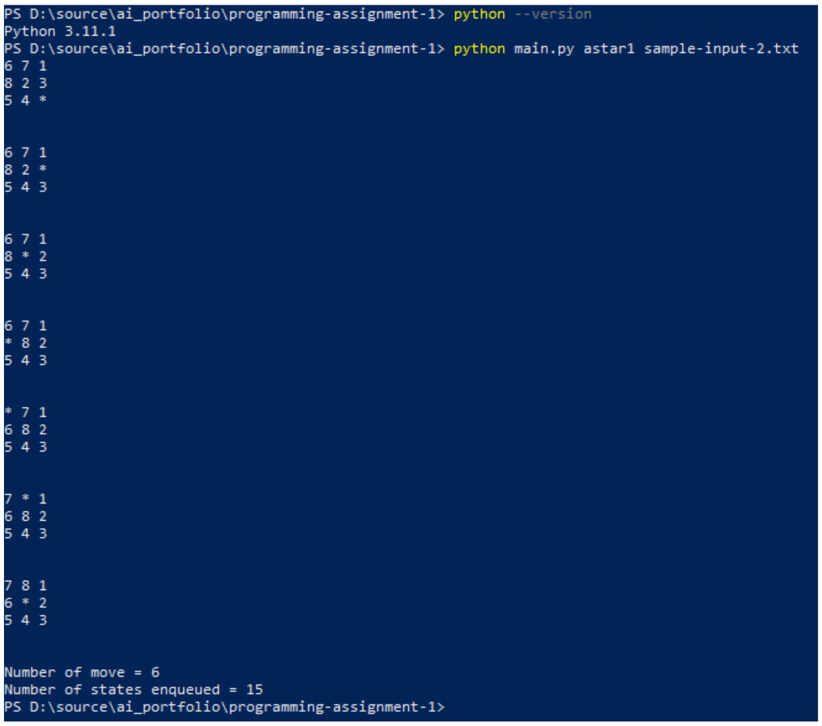
```
PS D:\source\ai_portfolio\programming-assignment-1> python --version
Python 3.11.1
PS D:\source\ai_portfolio\programming-assignment-1> python main.py ids sample-input-2.txt
Solution not found at depth 0
Solution not found at depth 1
Solution not found at depth 2
Solution not found at depth 3
Solution not found at depth 4
Solution not found at depth 5
Solution found at depth 6
6 7 1
8 2 3
5 4 *


6 7 1
8 2 *
5 4 3


6 7 1
8 * 2
5 4 3


6 7 1
* 8 2
5 4 3


* 7 1
6 8 2
5 4 3


7 * 1
6 8 2
5 4 3


7 8 1
6 * 2
5 4 3


Number of move = 6
Number of states enqueued = 435
PS D:\source\ai_portfolio\programming-assignment-1>
```

Here it is interesting that, on the second sample input,  iterative deepening search found a solution that was better than the solution found by depth-first search

## A* with the first heuristic:

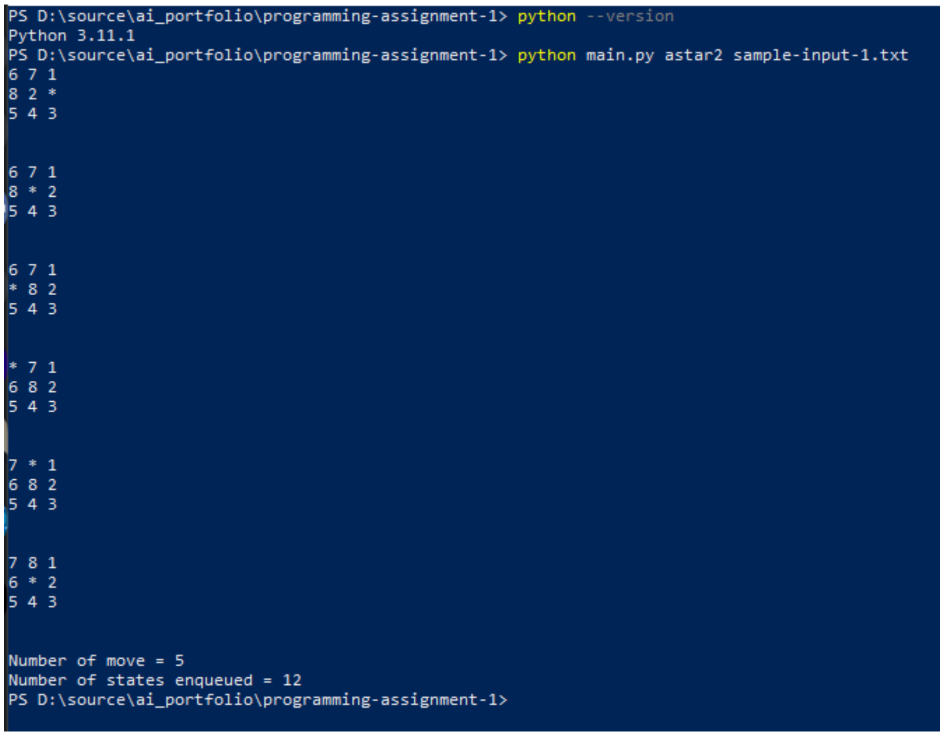


```
PS D:\source\ai_portfolio\programming-assignment-1> python --version
Python 3.11.1
PS D:\source\ai_portfolio\programming-assignment-1> python main.py astar1 sample-input-1.txt
6 7 1
8 2 *
5 4 3


6 7 1
8 * 2
5 4 3


6 7 1
* 8 2
5 4 3


* 7 1
6 8 2
5 4 3


7 * 1
6 8 2
5 4 3


7 8 1
6 * 2
5 4 3


Number of move = 5
Number of states enqueued = 14
PS D:\source\ai_portfolio\programming-assignment-1>
```

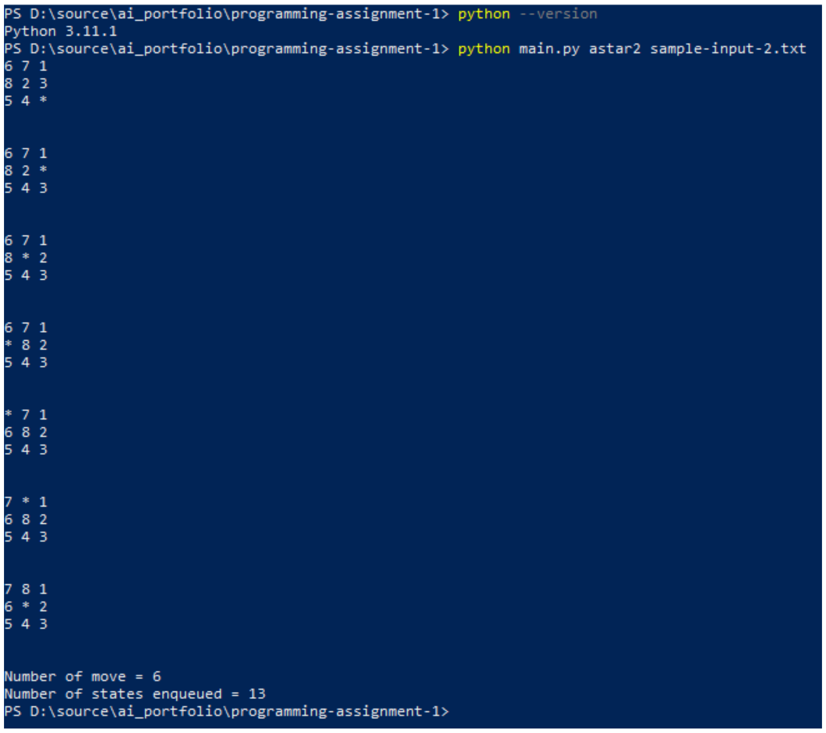


```
PS D:\source\ai_portfolio\programming-assignment-1> python --version
Python 3.11.1
PS D:\source\ai_portfolio\programming-assignment-1> python main.py astar1 sample-input-2.txt
6 7 1
8 2 3
5 4 *


6 7 1
8 2 *
5 4 3


6 7 1
8 * 2
5 4 3


6 7 1
* 8 2
5 4 3


* 7 1
6 8 2
5 4 3


7 * 1
6 8 2
5 4 3


7 8 1
6 * 2
5 4 3


Number of move = 6
Number of states enqueued = 15
PS D:\source\ai_portfolio\programming-assignment-1>
```

## A* with the second heuristic:



```
PS D:\source\ai_portfolio\programming-assignment-1> python --version
Python 3.11.1
PS D:\source\ai_portfolio\programming-assignment-1> python main.py astar2 sample-input-1.txt
6 7 1
8 2 *
5 4 3


6 7 1
8 * 2
5 4 3


6 7 1
* 8 2
5 4 3


* 7 1
6 8 2
5 4 3


7 * 1
6 8 2
5 4 3


7 8 1
6 * 2
5 4 3


Number of move = 5
Number of states enqueued = 12
PS D:\source\ai_portfolio\programming-assignment-1>
```



```
PS D:\source\ai_portfolio\programming-assignment-1> python --version
Python 3.11.1
PS D:\source\ai_portfolio\programming-assignment-1> python main.py astar2 sample-input-2.txt
6 7 1
8 2 3
5 4 *


6 7 1
8 2 *
5 4 3


6 7 1
8 * 2
5 4 3


6 7 1
* 8 2
5 4 3


* 7 1
6 8 2
5 4 3


7 * 1
6 8 2
5 4 3


7 8 1
6 * 2
5 4 3


Number of move = 6
Number of states enqueued = 13
PS D:\source\ai_portfolio\programming-assignment-1>
```

# Comparative Analysis of the two Heuristics used for A*

**Heuristic 1**
Heuristic 1 simply counts the number of tiles that are in the wrong spot, excluding the blank spot.

This heuristic is always **guaranteed to be an underestimate**, since for every tile in the wrong spot, we will need at least one action to get to the goal state. This makes heuristic 1 **admissible**, even though it is usually less accurate than heuristic 2.

**Heuristic 2**
Heuristic 2 sums up all of the Manhattan distances of each tile (excluding the blank tile) to their goal positions.

One issue with using this heuristic is that h(n) overpowers g(n) since the Manhattan sum is usually much bigger than the depth of the node. One solution to this could be taking the average of the Manhattan distances, rather than the sum of the Manhattan distances.

Even with the imbalance between g(n) and h(n), heuristic 2 is still **admissible** because it will never overestimate the relative distance to goal that a state has relative to other states.

As observed in the sample test cases, **running the A* algorithm with the second heuristic usually performs better than with the first heuristic.** The metric used to define "better" is the number of states that were enqueued, which is proportional to running time. I believe that this increased performance is because the second heuristic is more accurate in estimating the distance of a state to the goal state. This increased accuracy is because the second heuristic takes into account the distance of tiles to their goal positions, rather than simply if a tile is in the wrong positions. According to heuristic 1, a state that needs only 1 move to achieve the goal will have the same h(n) as a state that needs several moves, but only has 1 tile in the wrong position.