Explanation:

Both algorithm uses A* algorithm which consist of $g(t) + h(t)$. Where $g(t)$ is the distance from initial state to state t. And $h(t)$ is the estimate cost to reach the goal from state t. The difference between 2 of the algorithms only exist in the estimate cost to reach the goal. Because, the prediction seems logical that if more misplace tile will have larger $h(t)$. And also logical as well if the further a tile to its place will have higher $h(t)$.

- The first method, $h(t) =$ the number of misplaced pieces in the puzzle.
- The second method, $h(t) =$ the total manhattan distance from every pieces to their own goal location.

Statistics:

Method	Test case 1	Test case 2	Test case 3 (Unsolvable)
1	(6, 9)	(23477, 89004)	(31042, 241920)
2	(6, 9)	(3547, 10746)	(24145, 241920)

(a, b) denotes this algorithm at most has a nodes in its frontier, and created b nodes in the process

Time spent:

Method	Test case 1	Test case 2	Test case 3 (Unsolvable)
1	0.01s	1.20s	4.80s
2	0.01s	0.16s	5.67s

Analysis:

- Both algorithm is complete. The test case 3 shows that if there is no solution at all. Both method have the same number of nodes created in the process have explore all possibilities of structure.
- Both algorithm is not optimal. Since there's a possible way that it actually have a better solution.
- Prove that both algorithm is admissible
  1. $h(t)$ of the first method is the number of misplaced pieces. Which never overestimate the cost since every misplaced pieces need to be moved at least 1 time.
  2. $h(t)$ of the second method is the total distance of every pieces. Which also never overestimate. Since the fastest way to move every pieces is the total distance of every pieces. So $h(t) \leq$ actual total distance to reach goal.
- Both algorithm is consistent.

$= h(n) \leq c(n, n') + h(n')$
$= h(n) \leq 1 + h(n')$ (the cost will always be 1, since it only move 1 piece)

$= (h(n) - 1) \leq h(n') \leq (h(n) + 1)$
Because in both algorithm, the changes will at most be 1.
This happen in method 1 because only 1 piece move.
Either move to their goal, still not in their goal, or move out from their goal
This happen in method 2 because only 1 piece move and it's only moving for 1 distance
Either closer to their goal, or further to their goal.

$= h(n) \leq 1 + h(n) - 1$
$= h(n) \leq h(n)$
$\therefore$ Both method is consistent.

- The performance and memory consumption of the second method is better than the first one. This occurs because the prediction of second method is closer to the actual answer rather than the first method. So it will go through the better path.

- The time complexity of both method is $O(1)$ since both of the method has a constant size of puzzle. But even though the Big-O complexity is the same, the time spent on each test cases can be seen above.

  If the solution exist, the 2^{nd} method is faster because it has better heuristic of predicting the remaining cost ot reach the goal.
  But if the solution doesn't exist, the 2^{nd} method will take a longer time. Even though it will create the same amount of node, it will take a longer time to compute the $h(t)$ since it needs to compute the distance and sum it all.