Performance Metrics:

Starting State	*Total blocks*	*Total Stacks*	*Total Iterations*	*Max Queue Size*	*Solution Path*
0 | B 1 | CE 2 | AD	5	3	12	24	10
0 | BC 1 | AF 2 | DG 3 | E	7	4	16	89	12
0 | D 1 | EFIJ 2 | BG 3 | CH 4 | A	10	5	80	1009	19
0 | JL 1 | F 2 | DI 3 | A 4 | BCEK 5 | GH 6 |	12	7	26	684	20
0 | GHIO 1 | ABK 2 | F 3 | JLM 4 | DN 5 | 6 | CE	15	7	134	4206	30
0 | GI 1 | AHJ 2 | BM 3 |	20	8	3061	125234	41

4 | DNP 5 | FR 6 | EKT 7 | CLOQS					

Heuristic Function:

*3.5 * current_state_move + future_moves + 0.1 * total_estimated_moves*

where

current_state_move : Calculates the number of moves needed in current state to move the next in order alphabet to the stack number 0 to target index

total_estimated_moves: Calculates the total blocks on top of each alphabet which are not in stack 0 (target stack). This gives a very naive idea about the total moves needed to reach the goal state.

future_moves: Calculates the total blocks above each alphabet (which are not in order) irrespective of the stack number. This gives a little more relevant but still naive count of total moves needed to reach target state.

Logic:

I tried using various heuristics functions and in the end the combination of these heuristic functions end up working really well. So, the heuristic function gives a balanced count based on the moves needed to move the next unordered alphabet to its position and total moves needed for the given state to reach the final state. And these values are given priorities or weights based on the parameters they are multiplied with.

The *current_state_move* is given more priority here, by multiplying with parameter 3.5, over the other overall count moves.

The *total_estimated_moves* is given lowest priority here by multiplying with 0.1. I tried not to use this value, but found that the small amount of consideration of this naive move count gives better result.

Admissible or not?

The heuristic defined above is not admissible. We can see that from the heuristic value calculated for the given state:

iter:4, queue=7, **f=g+h=14.1,** depth=2

0 |

1 | CEBD

2 | A

For the given problem the optimal solution length is 10 moves, but heuristic ends up giving the value 12.1 for the state under consideration. Hence, it overestimates the total number of moves needed.

But, as the number of iterations increase the heuristic function becomes more and more admissible.

Interesting Note on Scalability:
1.Same blocks arrangement and expanding total stacks

0 | D
1 | EFIJ
2 | BG

Total blocks	*Total Stacks*	*Total Iterations*	*Max Queue Size*	*Solution Path*
10	5	**80**	1009	19
10	6	**26**	498	18
10	7	**22**	575	16
10	8	**19**	638	16
10	9	**17**	663	16
10	10	**17**	756	16

The heuristic function ***scales really well*** as the number of stacks are increased and the problem gets solved in much less number of iterations with much smaller solution path length.

2. Constant stack size with increasing block numbers

Total blocks	*Total Stacks*	*Total Iterations*	*Max Queue Size*	*Solution Path*
5	3	**14**	28	11
6	3	**15**	27	13
7	3	**37**	67	21
8	3	**103**	175	20
9	3	**2216**	3235	28
10	3	**7942**	12672	24

We can see that with increase in total number of blocks for constant stack size, the problem gets ***more harder*** to solve.