1. The heuristics used for both 8 puzzle and 15 puzzle problem are Hamming distance heuristic and Manhattan distance.
   - Hamming distance: This heuristic calculates the number of misplaced tiles.
   - Manhattan distance: Manhattan distance of tile is the number of tiles or slides it is away from the goal state position. The Manhattan distance of state is addition of Manhattan distance of each tile.

   A heuristic is consistent if for every node n, every successor n' of n generated by action a

   $h(n) \leq c(n.a,n') + h(n')$ --------(1)

   we have $f(n') = g(n')+ h(n')$

   => $f(n')= g(n) + c(n,a,n') +h(n')$

   $f(n) = g(n) + h(n)$

   from 1, $f(n) \leq g(n) + c(n,a,n') + h(n')$

   => $f(n) \leq f(n')$

   Therefore, if h is consistent f is non-decreasing along any path.
   In 8 puzzle and 15 puzzle problem the f(n) value is non-decreasing in whole path finding for both the heuristic.

   How h1 dominates h2?
   Consider h1 as Manhattan distance heuristic and h2 as Hamming distance heuristic. For a particular problem.
   for 15 puzzle problem-
   when A* uses Manhattan heuristic, it explores 39 states.
   And with hamming distance, it explore 222 states.
   therefore,
   h1 dominates h2.

   If the heuristic value from h1 for particular state is greater than heuristic value from other function h2. Then h1 dominates h2.

2. A* algorithm stores all explored state in the memory. The number of explored states are much greater than the actual path length.
   For particular 15-puzzle problem with hamming distance heuristics, it stores almost 222 states, where the actual path length of the problem is 16. For better heuristic the number of stored states reduces, but still storing unwanted states in memory causes space issues for large scale problems.
   Throughout the execution of A* algorithm, we have to store all the explored states. So the memory required for 15-puzzle problem depends on the states explored.

3. **IDA* description:**
   I used Iterative deepening A*, the memory bound algorithm which overcomes the memory issue in A* algorithm. Iterative deepening is a shortest path search algorithm which uses iterative deepening algorithm with heuristic function from A* to reach the goal node. Instead of reaching same depth every time, the depth of IDA* iteration depends on the threshold value. If the node

f value is above this threshold. The algorithm starts over again. It explores the most promising nodes with lower heuristic values. If the minimum f value for the nodes on particular level is greater than bound value then bound values gets updated to their minimum value.
Iterative deepening A* algorithm does not store explored states, unlike A* which stores all visited nodes. So, IDA* is memory efficient than A*. Because IDA* does not store explored states, algorithm explores previously visited nodes also number of times.

completeness:

IDA* is complete. IDA* returns a solution for a problem, if it exists.

Optimality:

IDA* finds the optimum solution for the problem. In 8 or 15 puzzle problem it finds the smallest number of paths to the goal state. But if we consider efficiency, IDA* explore visited nodes again and again so it explores more states than A* algorithm. In that case, we can not consider it as optimum.

complexity analysis:

As IDA* only stores the nodes in the path and also during its execution pops some nodes from the path. So, the maximum number of nodes that algorithm stores is equal to final path length. Therefore space complexity for IDA* is O(pathlength).
The time complexity depends on the depth at which goal state is present and the branching factor of each node. Because at particular level algorithm can visit each node.
So the time complexity is $O(m^d)$ where m= branching factor and d= depth of the solution.

Referred to Wikipedia link for A* algorithm.
https://en.wikipedia.org/wiki/Iterative_deepening_A*

4. A* performance

SR No	Puzzle	Number of states explored		Time(milliseconds)		Depth	
		H1	H2	H1	H2	H1	H2
1	1 3 5 4 8 2 0 7 6	9	15	2.016	3.974	8	8
2	1,,3 4,2,5 7,8,6	3	3	1.005	1.004	3	3
3	1,2,3 5,,6 4,7,8	4	4	1.004	1.003	4	4
4	1,3,6 5,2, 4,7,8	7	8	2.005	1.003	7	7

5	1,2,3 4,,5 7,8,6	2	2	1.002	1.002	2	2
6	1,2,3,4 5,6,7,8 9,10,15,11 13,14,12,	4	5	2.007	2.006	4	4
7	1 2 3 4 5 0 6 8 9 10 7 11 13 14 15 12	4	4	1.002	2.004	4	4
8	0 2 3 4 1 6 7 8 5 9 10 11	6	6	2.0012	2.002	6	6
9	1 2 3 4 9 5 7 8 13 6 10 11 14 15 0 12	9	10	1.0160	15.626	9	9
10	1 6 2 3 5 10 7 4 9 11 0 8 13 14 15 12	8	8	1.062	15.62	8	8
IDA* algorithm							
1	1 5 2 0 4 3 7 8 6	7	7	0.962	1.962	5	5
2	1 2 3 4 5 0 7 8 6	3	3	1.003	1.003	1	1
3	1 6 2 4 0 3 7 5 8	9	18	2.006	3.019	6	6
4	1 2 3 0 4 5 7 8 6	7	7	2.006	1.003	3	3
5	1 2 3 8 0 5 4 7 6	16	29	2.018	3.08	6	6
6	1 2 3 4 5 6 7 8 9 14 0 12	14	36	1.002	2.96	6	6

	13 11 10 15						
7	1 2 4 8 5 6 3 11 0 10 7 12 9 13 14 15	40	106	10.027	16.9	12	12
8	1,2,3,4 5,6,11,7 9,10,15,8 13,14,,12	8	8	1.003	1.003	5	5
9	1 2 3 4 5 7 11 8 9 6 12 0 13 10 14 15	15	16	2.02	2.05	7	7
10	1 2 0 4 5 6 3 7 9 10 15 8 13 14 12 11	13	45	2.98	5.08	8	8