Artificial Intelligence (CSE 537) Homework 1
Name: Rohan Vaish
SBU ID: 111447435

Section 1:
Heuristics – Describe the two heuristics you used for A*. Show why they are consistent and why h1 dominates h2?

->Manhattan Distance Heuristic:
Manhattan Distance Heuristic is the sum of minimum number of moves it will take for each tile to reach its goal position given there are no constraints.

Consistency: This is the definite minimum number of moves for each tile to reach its goal position and also each tile can only move to another position by moving through the tile board. This performs better than the *Displaced Tiles Heuristic*

Displaced Tiles Heuristic:
Displaced Tiles Heuristic is the number of tiles which are not in their goal position.
Consistency: Each of the displaced tile has to be moved at least once so this heuristic takes in account the number of displaced tiles to estimate the distance to the goal state. This heuristic behaves very poor due to it largely underestimating the distance to the goal state.

Why h1 dominates h2:
Manhattan Distance Heuristic dominates *Displaced Tiles Heuristic* because it takes in account the estimated minimum number of moves needed to be taken to reach the goal state and not just if a tile is misplaced or not (sort of binary decision). So, this h1 estimates a probable distance to the goal state, on the other hand h2 will treat a tile one move away from its goal position the same as a tile three moves away from its goal position

Section 2:

Memory issue with A* -- Describe the memory issue you ran into when running A*. Why does this happen? How much memory do you need to solve the 15-puzzle?

->My code did not run into any memory issue while running A* algorithm. This might be because of having enough memory on my machine.

Memory issues are pretty common with A* due it keeping large sized Explored and Frontier in memory. The numbers of nodes visited (expanded) will be b^d where:

b= Number of children a state can have which at worst will be 4 (3 if you ignore 1 child which takes you back to the parent)
d= Depth at which solution is found or solution length

The measure of b^d will not be reduced much by a better heuristic as the solution depth is remain same.

*"For the 15-puzzle, lengths of optimal solutions range from 0 to 80 single-tile moves"–**Source Wikipedia***

At the worst the solution length or depth will be 80. So, the amount memory needed to solve the 15-puzzle will be the much to store 3^{80} = 1.4780883e+38 nodes. This happens because we end up exploring each node at every depth while there is only one solution.

Section 3:

Memory-bounded algorithm – Describe your memory bounded search algorithm. How does this address the memory issue with A* graph search. Is this algorithm complete? Is it optimal? Give a brief complexity analysis.

->I used a variation of Depth First Search which is known as *Iterative Deepening A* (IDA*)* algorithm. However, it does not do the whole DFS, it makes the use of F(n)= G(n) + H(n) as a threshold. The threshold initially is set to the F(n) of the start node, and first iteration of A* will stop as soon as we explore a node with an F(n) greater than this threshold or it finds a goal state. If it's the former case, the algorithm saves the first higher value found for the next iteration. For the next iteration, the algorithm selects the minimum of the saved values as the new threshold.

How does it address the memory issue with A*?
While A* does bad with memory, IDA* does not remember any nodes except the ones coming in the current path of its algorithm. Thus, it only requires an amount of memory which is linear to the solution length it produces **(Source – Wikipedia)**

Is it complete?
Yes, this algorithm is bound to find the goal state if it exists so it is complete

Is it optimal?
Yes, IDA* is optimal. At the start the threshold is set to F(n) which is IDA*'s guess toward the goal state. It starts on that path until F(n) is decreasing or remains constant and stops when it finds a node with higher F(n) (signifying non-optimality). The next iteration goes with the threshold set as the minimum of the first higher F(n) value found in previous iterations and look for goal state. This ensures that we will never visit a non-optimal node and reach the goal state for sure.

Complexity Analysis:
Space:
As described before, it will only keep in memory the number of nodes linear to the actual solution length (depth) which is O(d) where
d= Depth at which solution is found or solution length
hence space complexity will be O(d)

Time:
The time complexity will be similar to A* algorithm (but more) that is $O(b^d)$ since at the solution depth, the time taken will be of the order b^d. It will however, take lesser time on lowers depths but the time taken on the final depth dominates therefore,
Time complexity will be $O(b^d)$

Section 4:

Table describing the performance of your A* and memory-bounded implementations on a randomly drawn set of 20 solvable puzzles. You should tabulate the number of states explored, time (in milliseconds) to solve the problem, and the depth at which the solution was found for both heuristics.

A*

S.No	Number of States Explored- Manhattan Distance	Time- Manhattan Distance (ms)	Number of States Explored- Displaced Tiles	Time- Displaced Tiles (ms)	Depth of Solution
1	9	1.04564	14	0.800132	8
2	29	2.02608	40	1.36899	10
3	14	1.04689	40	1.74880	10
4	26	1.73282	423	27.10199	16
5	17	1.10888	64	3.48401	12
6	8	0.96392	9	1.98221	8
7	49	3.68070	207	6.71625	14
8	52	3.44491	197	5.34677	14
9	10	1.20401	25	2.53582	10
10	20	1.17087	68	2.24399	12
11	14	1.55186	20	2.15315	10
12	24	1.59166	20	2.32115	14
13	13	1.69014	40	1.97196	12
14	8	1.38711	8	2.54106	8
15	8	0.99492	9	1.11079	8
16	20	2.38013	35	2.08711	14
17	12	1.40404	27	3.56507	10
18	16	1.406192	71	3.60488	12
19	10	1.63888	15	1.47509	8
20	17	1.27220	76	4.15802	14

IDA* (Memory bounded)

S.No	Number of States Explored-Manhattan Distance	Time-Manhattan Distance (ms)	Number of States Explored-Displaced Tiles	Time-Displaced Tiles (ms)	Depth of Solution
1	11	0.71620	34	1.00874	8
2	53	2.32982	174	7.58981	10
3	14	0.85210	163	3.06018	10
4	57	15.46382	1750	25.22277	16
5	44	2.00176	257	5.33223	12
6	12	0.62632	15	0.68283	15
7	169	3.10015	1351	18.92995	14
8	127	3.36885	1005	14.04023	14
9	21	1.21307	64	2.25424	10
10	16	0.85401	156	2.82192	12
11	17	1.36494	63	1.66201	10
12	42	1.83126	132	2.37115	14
13	22	1.30009	158	6.33883	12
14	12	1.00111	12	0.69904	8
15	15	1.02591	15	0.86116	8
16	26	6.91890	43	2.03609	14
17	19	1.04236	98	2.59613	10
18	33	1.26004	437	9.70101	12
19	17	1.22714	51	2.51293	8
20	27	1.25622	263	9.93418	14