Elena Beatrice Hadinata
z5159267
COMP3411 Assignment 1

Question 1: Search Algorithms for the 15-Puzzle

a)

	start10	**start12**	**start20**	**start30**	**start40**
UCS	2565	Mem	Mem	Mem	Mem
IDS	2407	13812	5297410	Time	Time
A*	33	26	915	Mem	Mem
IDA*	29	21	952	17297	112571

b) UCS has the worst time and space complexity. It has a worst-case time complexity of O(b^[C*/e]), where C* is the cost of the optimal solution and assuming every transaction costs at least e. It has an exponential space complexity of O(b^[C*/e]), b^[C*/e]=b^d if all steps costs are equal. Therefore it runs out of memory during most of the searches due to having to keep track of and generate all the notes to obtain the solution.

IDS is faster compared to UCS, only timing out for the last two searches. It has a time complexity of O(b^d) and linear space complexity of O(bd). It is, however, still slower than A* and IDA*.

A* performs better than UCS and IDS, but it still takes a lot of memory as it keeps all notes in memory. As we can see, it runs out of memory when performing searches for larger cases such as start30 and start40. It has a worst-case time complexity of O(b^d) and space complexity of O(b^d).

IDA* is the best performing algorithm compared to the other three, as it does not run out of time or memory for larger test cases. It only takes linear O(d) space even though it expands similar number of nodes as A*, meaning it is more efficient. It is a low-memory variant of A*.

Question 2: Heuristic Path Search for 15-Puzzle

a) & c)

	start50		start60		start64	
IDA*	G=50	N=14642512	60	321252368	164	1209086782
1.2	52	191438	62	230861	66	431033
1.4	66	116342	82	4432	94	190278
1.6	100	33504	148	55626	162	235848
Greedy	164	5447	166	1617	184	2174

b) Line 43 "`F1 is G1 + H1`" to "`F1 is (2-w)*G1 + w*H1`" , substituting w for 1.2, 1.4 and 1.6 each.

d) For IDA*, we can see that increasing w's value leads to decreasing the optimality of solution (increasing G) but faster performance (decreasing N). This is because as we increase w, g(n) decreases and h(n) increases. This algorithm cuts off search when f exceeds current threshold. This algorithm reduces to Greedy Search when w = 2, so we can also see it behaves more similar to Greedy the higher the value of w is.

Greedy search minimises the estimated cost to goal, so it expands nodes that are estimated to be closest to the goal. It has a time complexity of O(b^m), where m is the maximum depth in search space and takes O(bm) space as it retains all nodes in memory. This search minimises h(n). We can see above that the greedier (the more weight on h(n)) the faster the search is (decreasing N), even though this also results in the decrease in optimality of solution (increasing G).