Configuration	d	A*(h1)	A*(h2)
125348670	4	10	10
125304678	4	12	12
142635078	4	12	12
140352678	4	12	12
142358670	4	12	12
142375068	4	12	12
142375680	4	12	12
032415678	4	15	12
320415678	4	12	12
310452678	4	12	12
042135678	4	15	12
142635780	6	17	17
125408367	8	22	22
235104678	8	70	32
142603785	8	43	29
154632078	8	42	26
120348657	8	44	32
472105368	8	73	34
174302685	8	48	26
325408617	8	43	29
431502678	8	73	34
174302685	8	48	26
031642785	8	23	20
134602785	10	86	31
025168437	12	179	72
035274168	12	209	45
420163785	12	206	39
154603782	12	196	52
128307645	12	323	145
472158036	12	207	44
174602835	12	195	50
358402617	12	307	102
541632078	12	290	128
350128467	12	162	36
154608723	14	466	109
174862350	14	472	55
134608752	14	753	146
127804365	14	738	109
452308617	14	914	346
425187063	14	426	90
634201785	14	700	106
328647015	14	561	135
265187430	16	1249	109
375248160	16	978	160
423186750	16	1354	116
187302645	16	1710	462
472586130	16	910	73
358602147	16	1550	212
541603782	16	1530	254
140852376	16	1004	317
267105384	16	1410	147
632751840	16	958	195
458126370	16	939	208
732108564	16	1752	250
327861450	16	970	134
542613780	16	1467	247
371625084	16	1409	373
340612587	16	1575	365
250163478	16	1028	205
350681247	16	1360	144
560132748	16	1491	213
548103627	16	1584	158
564108723	18	3569	203
014876352	18	3019	160
612805743	18	3729	542
273401685	18	3603	488
430871562	18	2013	159
380215467	18	3475	472
532708164	18	4656	679
612804357	18	4203	980
012763854	18	3530	485
165408372	18	4476	732
231685470	18	3758	790
435108672	18	3938	839
063821475	18	2819	336
163702845	18	3415	399
634851720	18	2785	247
532807146	18	3262	273
257138046	18	3013	182
147305268	18	3204	655
714325860	18	2908	439
160342578	18	3458	664
082437615	18	3547	447
814602357	18	3714	452
425801673	18	4310	500
460187352	18	2130	183
180372546	18	2890	452
265807143	20	9557	425
375128046	20	7795	845
043128756	20	7569	794
870132645	20	7182	1026
472806513	20	8102	475
580362147	20	6733	694
541763820	20	6762	315
613758042	20	7278	950
120863745	20	7199	1294
058264371	20	7713	769
135768024	20	7269	1062
821367045	20	8412	720
847123065	20	7164	321
432718650	20	8706	1742
561207348	20	11176	1333
715608243	20	10552	827
361745820	20	7148	1656
321746850	20	6947	1030
230647815	20	6815	772
630281745	20	8124	954
420138765	20	7593	1802
783402651	20	10133	755
632847051	20	7184	839
047135268	20	8377	1239
742608135	20	11552	1102
754321680	20	7390	1045
435817062	20	7539	872
043815627	20	7358	767
435267018	20	7169	836
127835064	20	7525	1009
743625180	20	7325	500
380216754	20	6822	741
631547820	20	6858	732
051342786	20	9240	1480
450831762	20	7709	934
037254618	20	7228	1216
615834270	20	6781	761
380265471	20	7838	692
274305168	20	8068	1504
124508763	20	10865	1205
140673528	20	7779	931
126854370	20	7600	911
021756438	20	7664	1122
230614758	20	7801	1810
042836715	20	6754	859
487312560	20	7895	520
028365174	20	7195	835
814265037	20	6888	917
018724563	20	6807	498
321487056	20	7152	1041
357602184	20	11480	1345
035124768	20	10014	1821
126805374	22	24243	3152
028456731	22	15770	1012
413568270	22	16716	1032
472613580	22	17226	1860
172536480	22	21138	1894
613205874	22	17478	1953
510274836	22	15636	1395
184256370	22	15385	1854
726145830	22	15541	1052
031564287	22	16053	1111
413687052	22	15613	1851
871365042	22	16784	1822
256783140	22	17935	540
570483126	22	14531	306
823471065	22	15620	1778
457621380	22	17698	2828
058261734	22	15727	1504
843527610	22	20060	931
048361257	22	15461	1135
184365270	22	21060	2200
548367012	22	16555	1280
078415326	22	15834	1490
830612547	22	16169	1089
135476082	22	20807	2588
085162734	22	15693	1802
541807236	22	22630	1328
741365082	22	15700	3315
021645378	22	16955	3709
643258170	22	14921	1386
274863150	22	15258	990
716528043	22	16353	761
564318072	22	21174	1662
364802751	22	26266	3548
562314780	22	21387	3525
561372084	22	17824	2752
075813624	22	20461	1536
654183270	22	17798	627
720168435	22	17306	1878
083165724	22	16928	1763
124637580	22	22037	3196
261374085	22	17057	3122
023675148	22	16151	2264
617325840	22	20991	1956
830642157	22	15880	1114
473561028	22	14768	1285
271603854	22	25165	2676
017564328	22	18157	2105
541306872	22	23740	2594
420136587	22	15534	2017
157684032	22	16464	2714
071836425	22	15720	949
037618452	22	20762	2067
340578126	22	14579	1507
074836152	22	15503	465
763245018	22	15859	647
685274310	24	46748	3194
657403812	24	55631	2130
538264071	24	37286	1987
073246581	24	37328	645
768514023	24	35959	1070
430127586	24	45946	1934
076485132	24	45728	2256
735802146	24	56444	3991
781645023	24	45604	3591
032546187	24	45874	4044
270513486	24	31368	1781
470561238	24	35304	3599
374815026	24	34419	2085
741385260	24	34972	3782
214703658	24	59846	7192
763258140	24	34342	1718
580261734	24	48056	3916
752601348	24	59124	8483
024738516	24	34220	3537
016723584	24	34603	3218
724508361	24	58205	4456
520738641	24	47530	4255
675401832	24	53321	4238
821534067	24	47950	5267
247865013	24	37542	2274
260873154	24	33045	509
820617354	24	47260	4471
426805317	24	55075	4072
275481036	24	35191	3492
520143867	24	49385	6292
726384150	24	35061	3103
528641730	24	35077	3748
350746218	24	37273	4389
267534018	24	35838	1980
075148263	24	46372	4986
870514326	24	31025	1071
280571346	24	35472	4304
850126743	24	45089	2805
658307142	24	54608	3511
568703214	24	50785	910
861425073	24	38424	4770
851674320	24	48587	4435
260541783	24	36125	3277
621538047	24	38015	4155
450817623	24	37390	5277
780316542	24	44130	2618
274836510	24	33294	1571
421706538	24	55305	7059
820613574	24	46146	1997
046158237	24	45485	3145
061243578	24	36800	5338
724356180	24	38610	3845
130547268	24	37503	3100
630827145	24	36048	2171
873502164	24	51065	1956
753426180	24	48077	2600

Test cases shown: 250

d	A*(h1)	A*(h2)
4	12	12
6	17	17
8	48	28
10	86	31
12	227	71
14	629	137
16	1311	217
18	3417	470
20	7957	974
22	17819	1798
24	42873	3421

Numbers are rounded

(Program can generate random puzzles with odd-numbered depths and depths greater than 24 as well, they were just omitted to keep the tables concise)

This project was my first time implementing an AI algorithm, and it allowed me to apply my prior experience in object-oriented programming towards what I have learned in this class. My first step was to make a Node class containing variables for f, g, and h so my algorithm could iterate through the puzzles in the correct order (lowest f first). My next step was to implement functions returning h1 (number of misplaced tiles) and h2 (sum of Manhattan distances from correct position) for a given puzzle, as well as a function to evaluate whether a puzzle was solvable (even number of inversions). It was not too difficult designing each helper function but combining them all into one search function (A*) was a tedious process. It was rewarding in the end, and I was delighted to see how efficiently the program could solve the puzzles.

The solutions that A*(h1) and A*(h2) generated contained the same number of steps for each random puzzle, and both algorithms returned an optimal solution (i.e. when testing the puzzles in Length20.txt, no solution generated took more than 20 steps). While both algorithms were successful, it was clear that A*(h1) took longer than A*(h2) by a huge margin. This is because h2 as a heuristic was more informational than h1. Counting misplaced tiles did not give any insight on how far each tile was from the correct spot, but summing up the Manhattan distances considered the horizontal and vertical movement of each tile.

The biggest lesson I learned from this project was to make sure I understand an algorithm before I try to implement it. Writing the Python code was the easy part, while understanding A* took a few days. Furthermore, I learned that if I ever try to implement A* elsewhere, it is good to experiment with several heuristics and not just one. In the case of this project, it made a huge difference in search cost.