Batsirai Malcolm Dzimati
Assignment 1

Used manhattan heuristic:

The manhattan heuristic, looks at each of the numbers current position and compares it to the position of the same number in the goal state and it calculates the distance between the two. It does this for every number and sums the total and that is the total heuristic of the entire puzzle. The Manhattan distance/heuristic never overestimates the optimal heuristic as every tile will have to be moved at least the number of spots in between itself and its correct position.

Children were generated in left, right, up down. And for the assignment customer data structures were used with each implementation having their own variation. With most combining hash sets for O(1) lookup and queues, priority queues or dequeue for ordered insertion and deletion each with custom hash functions and customer comparative functions to allow for object comparative based on heuristics. The data puzzles were stored in a one dimensional array of integers. Source code provided for further inspection if required.

A Star Algorithm:

```
[momo@VivoBookMomo A* Search Algorithm]$ make make run clean
g++ -c *.cpp
g++ *.o -o ASA
./ASA
It took 10 Number of iterations, with 5 Optimum
It took 26 Number of iterations, with 9 Optimum
It took 42 Number of iterations, with 12 Optimum
It took 12 Number of iterations, with 6 Optimum
It took 75 Number of iterations, with 14 Optimum
It took 195 Number of iterations, with 16 Optimum
It took 268 Number of iterations, with 16 Optimum
It took 1453 Number of iterations, with 30 Optimum
It took 5958 Number of iterations, with 28 Optimum
It took 7383 Number of iterations, with 31 Optimum
rm *.o
rm ASA
[momo@VivoBookMomo A* Search Algorithm]$
```

Known Optimum

```
5 Optimum
9 Optimum
12 Optimum
6 Optimum
14 Optimum
h 16 Optimum
h 16 Optimum
th 30 Optimum
th 28 Optimum
th 31 Optimum

m]$
```

A* star algorithm took the shortest path (Known Optimum), with a relatively low number of iterations, as it found the known optimum for all 10 puzzles. Of all the algorithms that found the known optimum, A star has the lowest iterations. Meaning this was the best performing algorithm in terms of number of iterations and known optimum(shortest path).

Best First Search:

```
[momo@VivoBookMomo BestFirstSearch]$ make make run clean
g++ -c *.cpp
g++ *.o -o BFS
./BFS
It took 9 Number of iterations, with 5 Optimum
It took 103 Number of iterations, with 35 Optimum
It took 171 Number of iterations, with 34 Optimum
It took 12 Number of iterations, with 6 Optimum
It took 189 Number of iterations, with 22 Optimum
It took 186 Number of iterations, with 24 Optimum
It took 183 Number of iterations, with 24 Optimum
It took 272 Number of iterations, with 54 Optimum
It took 630 Number of iterations, with 52 Optimum
It took 147 Number of iterations, with 53 Optimum
rm *.o
rm BFS
[momo@VivoBookMomo BestFirstSearch]$
```

Known Optimum

```
5 Optimum
9 Optimum
12 Optimum
6 Optimum
14 Optimum
h 16 Optimum
h 16 Optimum
th 30 Optimum
th 28 Optimum
th 31 Optimum

ml$
```

Best first search failed to find the shortest path(known optimum) for 8/10 of the puzzles, finding an optimum greater than the known optimum for 8 of the puzzles and finding the optimum for only 2. Only finding the known optimum for the first and the fourth. Even though it failed to find the known optimum it did take the lowest number of iterations amongst all the algorithms for the majority of the puzzles. Making it the best performing in terms of the number of steps it takes to find the problem. Minimising the cost to find a solution. For all the algorithms it took the least number of iterations to find the solution

Breadth First Search:

Known Optimum

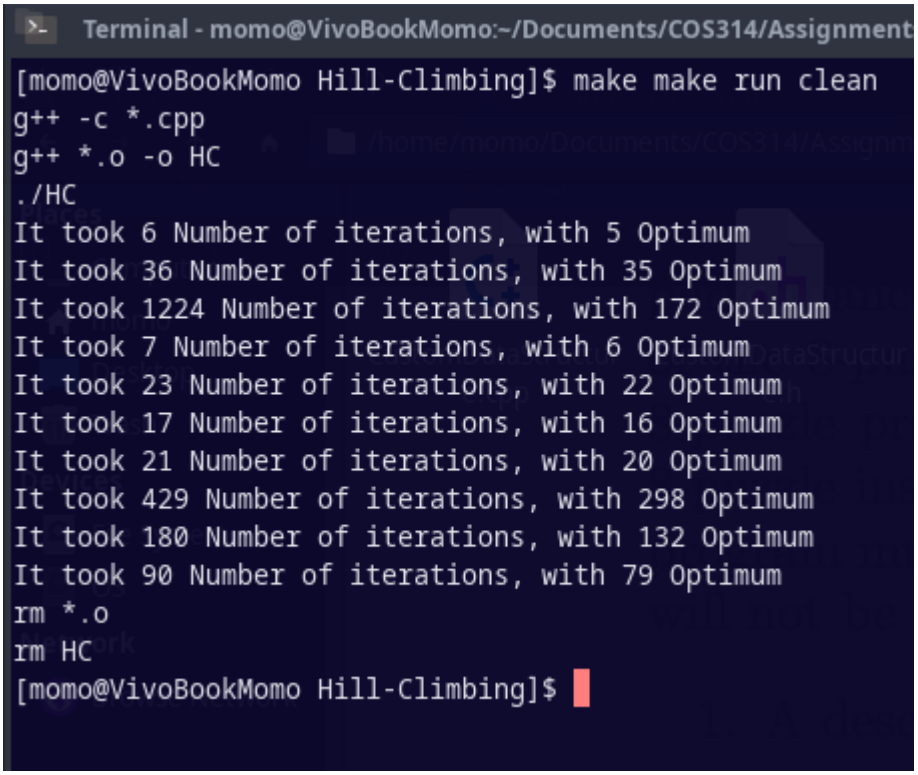
```
Terminal - momo@VivoBookMomo:~/Documents/COS314/Assignments/ArtificialI
[momo@VivoBookMomo BreadthFirstSearch]$ make make run clean
g++ -c *.cpp
g++ *.o -o BFS
./BFS
It took 56 Number of iterations, with 5 Optimum
It took 467 Number of iterations, with 9 Optimum
It took 2043 Number of iterations, with 12 Optimum
It took 122 Number of iterations, with 6 Optimum
It took 3828 Number of iterations, with 14 Optimum
It took 9163 Number of iterations, with 16 Optimum
It took 9222 Number of iterations, with 16 Optimum
It took 181364 Number of iterations, with 30 Optimum
It took 180595 Number of iterations, with 28 Optimum
It took 181439 Number of iterations, with 31 Optimum
rm *.o
rm BFS
[momo@VivoBookMomo BreadthFirstSearch]$
```

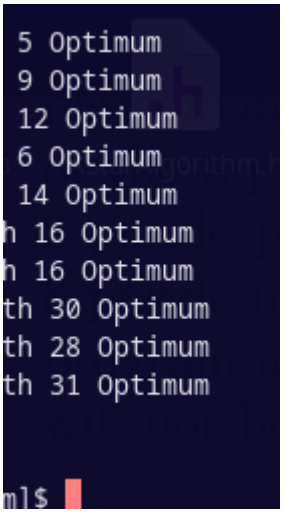
```
5 Optimum
9 Optimum
12 Optimum
6 Optimum
14 Optimum
h 16 Optimum
h 16 Optimum
th 30 Optimum
th 28 Optimum
th 31 Optimum
ml$
```

Breadth's first search took the largest number of iterations on average to find the solution. But it found all the solutions at the known optimum meaning it found the shortest path. But due to the large number of iterations required to find the optimal solution it drops the performance in terms of number of iterations. Breadth first found all the solutions.

Hill-Climbing

Known Optimum





Hill climbing was volatile. It found the known optimum for 3/10 puzzle piece and for the remaining 7 puzzles it found an optimum larger than the known optimum. And for some instances it took some of the lowest iterations to solve the puzzles but for the others it took a really large number of iterations to solve. Hill climbing was the most hardest to analyse as its results are not really inconclusive, as some of the puzzle pieces found the results in the highest optimum of all the other algorithms while for the others it found the solution at the known optimum at the lowest number of iterations of all the algorithms. But it found every solution.

Conclusion:
When looking at performance in terms of number iterations and optimum. The A star Algorithm was the best when considering both followed by best search followed by hill climbing even though it was volatile the heuristic made it perform better than breadth first which was doing a blind search as it took the highest number of iterations to solve the puzzles even though it found the optimum.