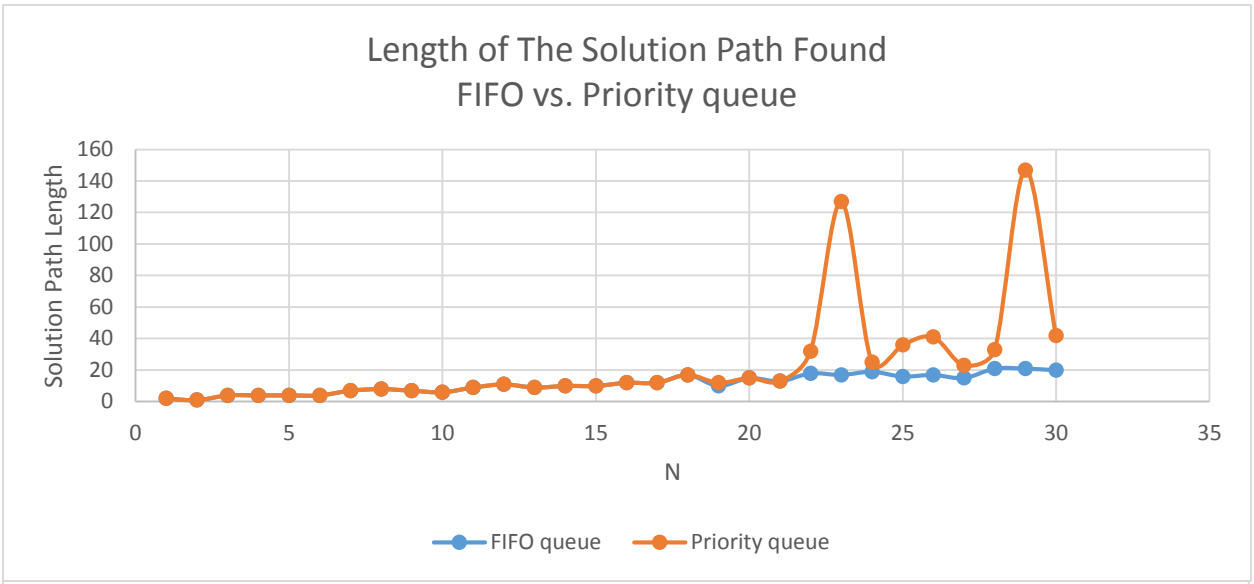
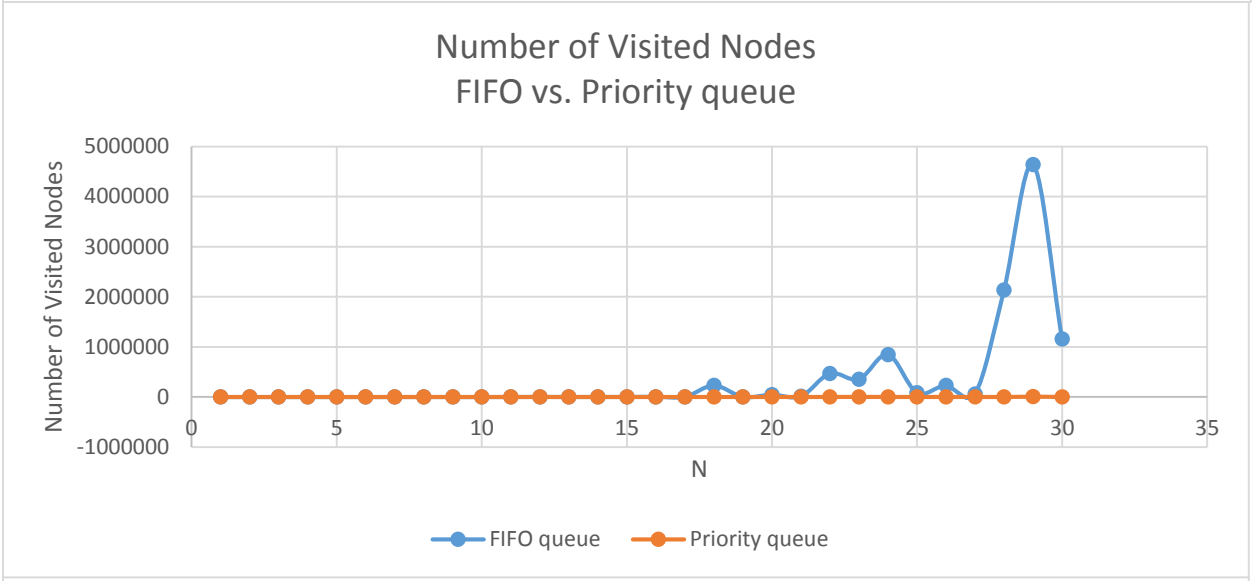
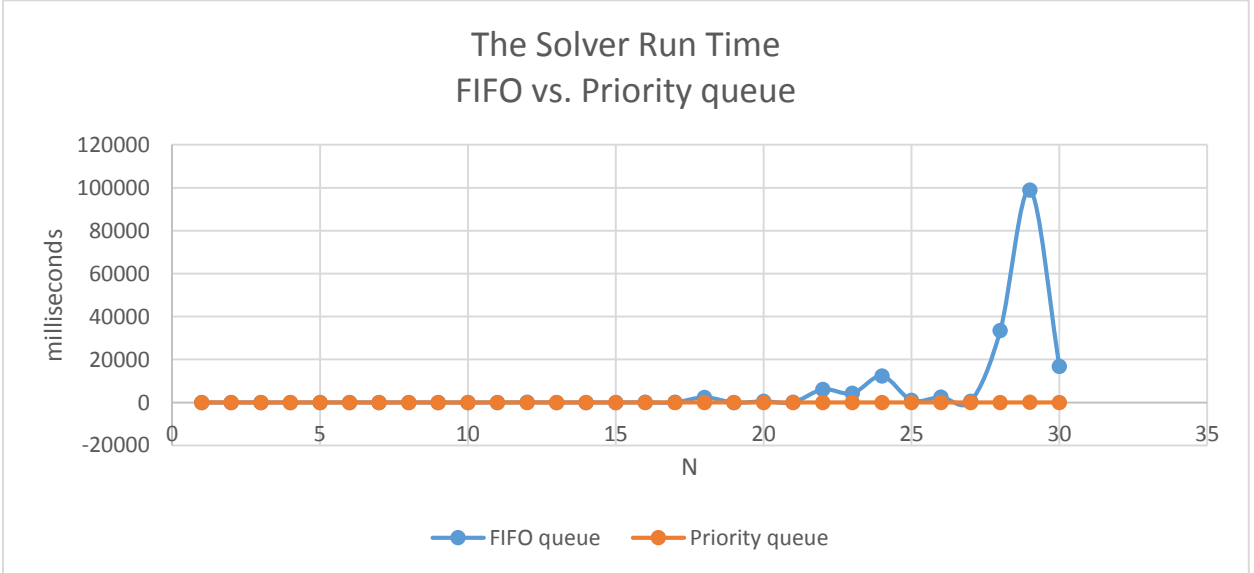

Length of The Solution Path Found
FIFO vs. Priority queue
Solution Path Length
160
140
120
100
80
60
40
20
0
0
5
10
15
20
25
30
35
N
FIFO queue
Priority queue


Number of Visited Nodes
FIFO vs. Priority queue
Number of Visited Nodes
5000000
4000000
3000000
2000000
1000000
0
-1000000
0
5
10
15
20
25
30
35
N
FIFO queue
Priority queue


The Solver Run Time
FIFO vs. Priority queue
milliseconds
120000
100000
80000
60000
40000
20000
0
-20000
0
5
10
15
20
25
30
35
N
FIFO queue
Priority queue

Analysis

Looking at the graphs above, we can tell that the 15-puzzle problem is solved by using Breadth First Search (BFS) Algorithm + a Priority queue sometimes does not give out the shortest path from the initial state to the goal state as the BFS + a FIFO does. However, BFS + a Priority queue is way better than BFS + a FIFO queue in terms of performance, assuming with a big enough value of N. The number of states that are generated by BFS + a Priority queue is much less than that by BFS + a FIFO queue. Also, the amount of time it takes for BFS + a Priority queue to finish is a lot less than for BFS + a FIFO queue.

Overall object-oriented design

1. Solver: where to process all the commands from inputs

2. State: represents the state of a specific puzzle configuration

3. BFS: where the Breadth First Search Algorithm is performed to solve the 15-puzzle problem

4. DistMetricComp: a comparator for the PriorityQueue data structure that is used for the BFS Algorithm. The comparator is used for selecting the element with the least priority in the PriorityQueue

5. NodeStore: a linear-probing hash table with step size of 1 which stores the set of puzzle states that have already been visited so far by the Breadth First Search Algorithm.