**Memory usage:**

In any puzzle where the path lengths were of a decent length or no solution existed the algorithms without visited or expanded lists ran out of memory. This isn’t a surprise as these algorithms don’t avoid local loops allowing them to repeatedly add new paths to the same state to the queue even if no solution exists. They should, however, find a best solution if given enough time and memory.

Surprisingly, the progressive deepening search with non-strict visited list algorithm ran out of memory in puzzles with no solution. This can be explained by the global loops caused by the depth first order of each depth limited sub-search allocating the same puzzle state multiple times which combined with the fact that allocator used to allocate the puzzle states only releases the memory when it is destructed, means that large amounts of memory are used. Fixing this would require either storing the path in a puzzle state and deleting a puzzle state after it has been expanded, using reference counting e.g. std::shared\_ptr, or using a garbage collector.

**Running time:**

A\* with expanded list and sum of Manhattan distance as heuristic was the fastest at solving the puzzles finding a solution in under 1ms over 10 times faster than the fastest other algorithm. Its heuristic was very effective at reducing the number of state expansions it had to do compared to using number of misplaced tiles. However, in puzzles with no solution, using a heuristic doesn’t help but just adds overhead making A\* 2-3 times slower than breadth first search which was the fastest.

**Hashing performance:**

Given that hashmaps have average O(1) find and insert time and using an unsorted array with linear searching has average O(n) find and O(1) insert time it isn’t a surprise that linear searching gave unacceptably slow running times for algorithms when their visited/expanded list become long and slower but by a smaller factor running times when the visited/expanded was relatively small.