1) Set of valid states: Every state is valid.
   Successor function: Add piece at all places in NxN board and generate every possible state. This will result in duplicate states being generated.
   Cost: Uniform. Cost for generating every state is the same.
   Goal: No two rooks can share either a row or column.

2) It uses DFS.
   N=4, it takes longer than a minute.
   N=8, it takes longer than a minute.
   In order to change it to BFS, just pop from the front. Change it to pop(0). Please see code submission for the code.
   BFS:
   For N=4, runs in 0.0269597 seconds.
   For N=8, it takes longer than a minute.

3) DFS, N=5, 0.592065913398 seconds
   BFS, N=5, 2.88161393056 seconds

   DFS, N=6, 77.6370550745 seconds
   BFS, N=6, takes longer than 400 seconds

   DFS takes lesser time than BFS. This is because in nrooks, once a particular branch is selected, we are guaranteed a solution in that branch. This ensures that we reach the goal faster as the goal will be at depth N, and a BFS till depth N will take longer time.

4) 58.38 seconds N = 114

   Abstraction:
   Set of valid states: Only states that have <= N rooks and that place an extra rook on the board under the condition that the current column and row has no existing rooks.
   Successor function: Generate a new state that adds a rook and satisfies the valid state condition.
   Cost: Uniform. Cost for generating every state is the same.
   Goal: No two rooks can share either a row or column.

   Extra optimization: For n-rooks, we know that the goal state can be reached from every valid state. Using this, we can only generate one valid successor state and discard other valid states. So simply return successors after the first valid successor is found. Using this, we can run for N= 265 in 58.34 seconds.

5) N-Queens after optimizations runs for N=11 in 2.83 seconds.
   N=12 takes longer than a minute.
   N=13 runs in 61 seconds.