Description:

Sudoku is a puzzle played on an n^2 x n^2 grid, with numbers from 1 to n^2. There are 3 different groups of n^2 numbers, where each needs to have each number in it exactly once. The 3 groups are rows, columns, and boxes.

In any given row in the grid, there needs to be n^2 numbers with no duplicates (all numbers from 1 to n^2).

In any given column, the same rule as above applies.

A box is a sub grid of size n x n. In this sub grid, the same rule applies; there must be all numbers from 1 to n^2 with no duplicates.

This is generally played on a 9x9 grid with numbers 1-9. So each row, column, and box needs all numbers 1-9. Here is an example of a completed 9x9 Sudoku board. The boxes are the nine 3x3 sub-grids within the 9x9 grid.

4	3	5	2	6	9	7	8	1
6	8	2	5	7	1	4	9	3
1	9	7	8	3	4	5	6	2
8	2	6	1	9	5	3	4	7
3	7	4	6	8	2	9	1	5
9	5	1	7	4	3	6	2	8
5	1	9	3	2	6	8	7	4
2	4	8	9	5	7	1	3	6
7	6	3	4	1	8	2	5	9

The algorithm implemented to check a solution to determine if it is valid or not iterates over each space of the board, using nested for loops. Because we are checking indexes for a row and column, we will iterate n times, inside of another loop which iterates n times. As we iterate through the inner loop, we will start with empty lists representing the values in a given row, column, and a box (all separate lists). We will add n numbers to each list. Each time we add a new number to the list, we check to see if it is already in that respective list. If it is, then the solution does not work because we cannot have duplicates in a row/column/box, and we return False. If the number isn't already in it, we continue checking the rest of the values. After each list fills up (n values), then we empty the lists in the outer loop and check the next n values. If we make it through the whole board without any duplicates, the solution is valid.

Complexity:

The time complexity used in my algorithm is O(n^4). In the 9x9 board, n would be equal to 3.

If you are considering n as the length of the grid (number of rows/columns/boxes), then the complexity would be considered O(n^2). In the 9x9 example, n=9. These runtimes are the same, but the definition of n is different, so I just wanted to clarify the definition of n for this problem.

Proof:

Step 1: As stated above, we can verify the solution of a Sudoku puzzle in O(n^2) time, which is polynomial time. Therefore, the Sudoku checker solution is NP. We do this by iterating n x n times to check all spaces on the board and the respective rows, columns, and boxes.

Step 2:

1. I'll select the SAT problem because it is one of the simpler NP-Complete problems to comprehend, and it is known to be NP-complete.
2. Transformation from SAT to an instance of Sudoku can be done in polynomial time by creating n^3 variables to represent each possibility (n) for each cell in the n x n grid, then applying the following constraints:

Create n x n x n different boolean variables $X_{i,j,k}$ where i, j, and k take on the values 1 to n.

Each row must have the values from 1 to n.

Each column must have the values from 1 to n.

Each box (sqrt(n) x sqrt(n)) must have the values from 1 to n.

Each cell must have only one value from 1 to n.

$X_{i,j,k}$; i represents the row, j represents the column, k represents the value at (i,j)

Example, for when n = 9:

Row i:

(if $X_{i,1,1}$, then NOT ($X_{i,2,1}$ OR $X_{i,3,1}$ OR $X_{i,4,1}$ OR $X_{i,5,1}$ OR $X_{i,6,1}$ OR $X_{i,7,1}$ OR $X_{i,8,1}$ OR $X_{i,9,1}$))

&

(if $X_{i,1,2}$, the NOT ($X_{i,2,2}$ OR $X_{i,3,2}$ OR $X_{i,4,2}$ OR $X_{i,5,2}$ OR $X_{i,6,2}$ OR $X_{i,7,2}$ OR $X_{i,8,2}$ OR $X_{i,9,2}$))

& … &

(if $X_{i,1,9}$, then NOT ($X_{i,2,9}$ OR $X_{i,3,9}$ OR $X_{i,4,9}$ OR $X_{i,5,9}$ OR $X_{i,6,9}$ OR $X_{i,7,9}$ OR $X_{i,8,9}$ OR $X_{i,9,9}$))

For the above constraints, in row i, column 1, there must be one number between 1 and 9 (n), and that number must not be in any other column in row i. Now, we must continue to check this condition for all columns from 1 to 9 (n).

(if $X_{i,2,1}$, then NOT ($X_{i,1,1}$ OR $X_{i,3,1}$ OR $X_{i,4,1}$ OR $X_{i,5,1}$ OR $X_{i,6,1}$ OR $X_{i,7,1}$ OR $X_{i,8,1}$ OR $X_{i,9,1}$))

& … &

(if $X_{i,2,9}$, then NOT ($X_{i,1,9}$ OR $X_{i,3,9}$ OR $X_{i,4,9}$ OR $X_{i,5,9}$ OR $X_{i,6,9}$ OR $X_{i,7,9}$ OR $X_{i,8,9}$ OR $X_{i,9,9}$))

This checks the same constraints for column 2. If (i, 2) contains a value k, then k cannot be in any other cell in row i. We repeat these steps for all remaining columns.

We follow this exact same pattern to find column constraints:

(if $X_{1,j,1}$, then NOT ($X_{2,j,1}$ OR $X_{3,j,1}$ OR $X_{4,j,1}$ OR $X_{5,j,1}$ OR $X_{6,j,1}$ OR $X_{7,j,1}$ OR $X_{8,j,1}$ OR $X_{9,j,1}$))

& … &

(if $X_{1,j,9}$ then NOT ($X_{2,j,9}$ OR $X_{3,j,9}$ OR $X_{4,j,9}$ OR $X_{5,j,9}$ OR $X_{6,j,9}$ OR $X_{7,j,9}$ OR $X_{8,j,9}$ OR $X_{9,j,9}$))

This checks (1,j). We repeat these steps for all rows.

We follow a similar pattern for checking box constraints:

(if $X_{1,1,1}$, then NOT ($X_{1,2,1}$ OR $X_{1,3,1}$ OR $X_{2,1,1}$ OR $X_{2,2,1}$ OR $X_{2,3,1}$ OR $X_{3,1,1}$ OR $X_{3,2,1}$ OR $X_{3,3,1}$))

& … &

(if $X_{1,1,9}$ then NOT ($X_{1,2,9}$ OR $X_{1,3,9}$ OR $X_{2,1,9}$ OR $X_{2,2,9}$ OR $X_{2,3,9}$ OR $X_{3,1,9}$ OR $X_{3,2,9}$ OR $X_{3,3,9}$))

& … &

(if $X_{3,3,1}$, then NOT ($X_{1,1,1}$ OR $X1_{,2,1}$ OR $X_{1,3,1}$ OR $X_{2,1,1}$ OR $X_{2,2,1}$ OR $X_{2,3,1}$ OR $X_{3,1,1}$ OR $X_{3,2,1}$))

& … &

(if $X_{3,3,9}$ then NOT ($X_{1,1,9}$ OR $X_{1,2,9}$ OR $X_{1,3,9}$ OR $X_{2,1,9}$ OR $X_{2,2,9}$ OR $X_{2,3,9}$ OR $X_{3,1,9}$ OR $X_{3,2,9}$))

This checks the upper left cell to make sure 1-9 occur once and only once inside the 3 x 3 sub-grid. We repeat this step for all other sub-grids, incrementing i + 3 each time (instead of starting at i=1, we start at i=4 and check through i=6), until it reaches 9. Then we drop i back to 1, then increment j by 3 in the same fashion we did with i, and continue the pattern until all 9 sub-grids have been accounted for.

We also need to verify that k is 1-9

($X_{i,j,1}$ OR $X_{i,j,2}$ OR $X_{i,j,3}$ OR $X_{i,j,4}$ OR $X_{i,j,5}$ OR $X_{i,j,6}$ OR $X_{i,j,7}$ OR $X_{i,j,8}$ OR $X_{i,j,9}$)

3. If all of these conditions are satisfied, then that means there will only be one instance of a number in any given row, only one instance of a number in any given column, and only one instance of a number in each of the sqrt(n) x sqrt(n) sub-grids. If at any point one of these conditions is not satisfied, the entire formula becomes false. This is an equivalent description of the rules for Sudoku.
4. Since SAT is NP-complete, then Sudoku must be NP-Hard.

Because we proved that Sudoku is both NP and NP-Hard, we conclude that Sudoku is NP-Complete.