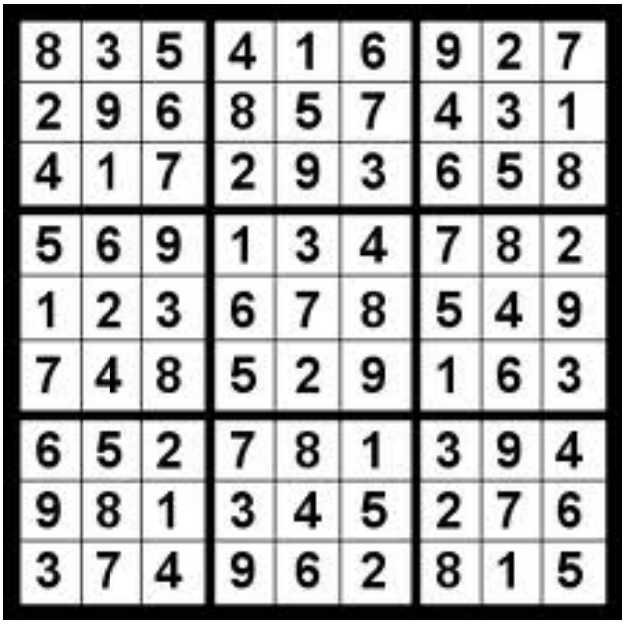
Sudoku Example



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

In Sudoku, the goal is to create a 9x9 matrix like the one shown above where each row contains every digit 1-9 exactly once, and so does each column and each 3-3 region. Our goal is to develop an algorithm to test whether a given 2-dimensional matrix represents a valid Sodoku solution or not.

Algorithm Design

The first thing we look for is whether there are basic operations that can be reused. If we notice that every one of the validity checks (rows, columns, regions) is about making sure that all digits 1-9 are present exactly once in a certain set of 9 elements, we can use a method for performing this check as the basis of the algorithm.

Step 1) Write a method that determines if a 1-d array of 9 elements contains each digit 1-9 exactly once (hint: modify the “count” example from lecture)

Now, we break down the problem and look at each of the 3 types of checks separately, and create a method for each.

Step 2) Create a method to check each row. Since these are already 1d arrays we can call the method from step 1 directly.

Step 3) Create a method to check each column. To do so we create a temporary 1d array to hold the elements and fill it with all of the elements from a single column, using a for loop. This 1d array can then be checked using the method from step 1.

Step 3) Create a method to check each region. For each region, we can use the same trick to create a 1d array with all of the elements in the region and test it with the method from step 1. To keep things clean, I have created a separate method to do this for a single region, given the "starting point" representing the upper left corner of the region. Then, a separate method checks all 9 regions by using for loops to find each of the 9 starting points individually and calling the method to check a single region with the starting point. This is the trickiest part of the program!

Step 4) We also need to check the basic validity of the matrix (ie, that it is 9x9), create a method for this.

Step 5) Combine all of the checks – if the matrix passes every check then it is a valid matrix!

Code

```
public class Sudoku {

  public boolean testBoard(int[][] board) {
    if (!testSize(board)) return false;
    if (!testRows(board)) return false;
    if (!testCols(board)) return false;
    if (!testRegions(board)) return false;
    return true;
  }

  // check that the board is 9 x 9
  boolean testSize(int[][] board) {
    if (board.length != 9) return false;
    for (int i = 0; i < board.length; i++) {
      if (board[i].length != 9) return false;
    }
    return true;
  }









  // check that the digits 1-9 each appear
```

```
// exactly once in the given array
boolean checkDigits(int[] array) {
  if (array.length != 9) return false;

  int[] counts = new int[10];
  for (int i = 0; i < array.length; i++) {
    // invalid number
    if (array[i] < 1 || array[i] > 9) return false;

    // we have already seen this number
    if (counts[array[i]] > 0) return false;
    counts[array[i]]++;
  }
  return true;
}

// return true if all rows are correct
boolean testRows(int[][] board) {
  for (int i = 0; i < board.length; i++) {
    if (!checkDigits(board[i])) {
      return false;
    }
  }
  return true;
}

// return true if all columns are correct
```

```
boolean testCols(int[][] board) {
  int[] tmp = new int[board.length];
  for (int col = 0; col < board.length; col++) {

    // fill a temp array with every element of the column
    for (int row = 0; row < board.length; row++) {
      tmp[row] = board[row][col];
    }

    // check to make sure it has all the right digits
    if (!checkDigits(tmp)) {
      return false;
    }
  }
  return true;
}
```

```
// return true if every region is correct
boolean testRegions(int[][] board) {
```

```
 // loop through each region, passing the indices
 // of the upper-left corner to the next method
 // note that we increment row and column counters by 3 here
 for (int row = 0; row < board.length; row += 3) {
   for (int col = 0; col < board.length; col += 3) {
     if (!testRegion(board, row, col)) {
       return false;
     }
   }
 }
 return true;
}

// test a specific region, given the upper left corner
boolean testRegion(int[][] board, int startRow, int startCol) {
  int[] tmp = new int[board.length];

  // fill a temporary array with every element of the region
  int index = 0;
  for (int row = startRow; row < startRow+3; row++) {
    for (int col = startCol; col < startCol+3; col++) {
      tmp[index] = board[row][col];
      index++;
    }
  }

  // check if we have all of the right digits in the region
```

```
    return checkDigits(tmp);
  }
}

public class Test {

  public static void main(String[] args) {
    testSudoku();
  }

  static void testSudoku() {
    // correct solution
    int[][] test = {{8,3,5,4,1,6,9,2,7},
                    {2,9,6,8,5,7,4,3,1},
                    {4,1,7,2,9,3,6,5,8},
                    {5,6,9,1,3,4,7,8,2},
                    {1,2,3,6,7,8,5,4,9},
                    {7,4,8,5,2,9,1,6,3},
                    {6,5,2,7,8,1,3,9,4},
                    {9,8,1,3,4,5,2,7,6},
                    {3,7,4,9,6,2,8,1,5}};

    // too large
    int[][] test2 = {{8,3,5,4,1,6,9,2,7},
                    {2,9,6,8,5,7,4,3,1},
                    {4,1,7,2,9,3,6,5,8},
                    {5,6,9,1,3,4,7,8,2,1},
```

```
                    {1,2,3,6,7,8,5,4,9},
                    {7,4,8,5,2,9,1,6,3},
                    {6,5,2,7,8,1,3,9,4},
                    {9,8,1,3,4,5,2,7,6},
                    {3,7,4,9,6,2,8,1,5}};

    // bad matrix
    int[][] test3 = {{8,3,5,4,1,6,9,2,7},
                    {2,9,6,8,5,7,4,3,1},
                    {4,1,7,2,9,3,6,5,8},
                    {5,6,9,1,3,4,2,8,2},
                    {1,2,3,6,7,8,5,4,9},
                    {7,4,8,4,2,9,1,6,3},
                    {6,5,2,7,8,1,3,9,4},
                    {9,8,1,3,4,5,2,7,6},
                    {3,7,4,9,6,2,8,1,5}};




    Sudoku sudoku = new Sudoku();
    boolean result = sudoku.testBoard(test3);
    System.out.println("Result: " + result);
  }
}
```