HP-71B Sudoku Solver's Sublime Sequel

Valentín Albillo (#1075, PPC #4747)

After my original article ***HP-71B Short & Sweet Sudoku Solver*** appeared in Datafile V24N2 p22, I went to try it with every Sudoku puzzle I could lay my hands on. As part of that extensive testing, I finally concocted a comprehensive 15-puzzle **Test Suite** which featured puzzles of all grades of difficulty, from the very simplest, that my program could solve in a straightforward manner by iteratively applying its forced-digit fill-in algorithm with *no* recursion whatsoever, to the very hardest which needed deep recursion that would take an actual **HP-71B** [1] *days*, if not *weeks*, to complete.

Using this Test Suite (featured at the end of this article!), I then went on to try and improve my Solver Version 1.0 incrementally, using the TS as a performance meter to quantitatively ascertain whether some modification was an improvement or not, and by how much, while adamantly meeting the criteria of *still* resulting in a *very short* program, and as "sweet" as possible. After three beta 'internal releases', here you are, the ultimate SUDOKU71 Version 2.0, a.k.a. **SUDOKUV2** or **V2** for short.

V2 is still a *very small program*, just 45 lines (i.e.: slightly over a *half* of a typical 80-line page), well under 2 Kbytes of code (1898 bytes). How much of an improvement is it ? In a word: ***tremendous !*** Just consider these facts:

- **V2** is faster than **V1** in nearly *every* TS puzzle, with ratios from 0.93x in the worst case to more than 1000x in the best cases. In general, the harder the puzzle the faster **V2** over **V1**. As an extreme case, TS#8 took Emu71 *more than 3 hours* to solve running **V1** while **V2** solves it in just 10 seconds ! Even a real HP-71B takes under 5 minutes to solve it (**V1** would take well over 4 days).
- **V2** needs to use recursion in just 3 out of the 15 TS puzzles. In contrast, **V1** uses recursion in 12 of the 15 puzzles. Which is more, when using recursion **V2** typically stays at shallow depths, never going below depth 3, and even that just for two of the puzzles. **V1**, on the contrary, frequently goes to depths 5 and 6, with the corresponding high running times and memory consumption those depths entail.
- **V2** implements more sophisticated fill-in algorithms which, while still being fairly simple and requiring very little code, nevertheless afford a *substantial* increase in forced-digit detection, resulting in recursion being needed much less frequently, as the grid is usually solved or nearly so by them.

The rest of this article includes the listing, a description of the new sections (refer to the original article in V24N2 for important details), usage instructions, a couple of examples, and last but not least, my **Test Suite**. Enjoy !

[1] No HP-71B, HP-IL ROM or Math ROM ? No problem. Search the Web for **Emu71**, a free emulator for Windows (>40X faster), or **HP-71X**, an excellent emulator (3X) for your HP48/49.

SUDOKUV2 Program Listing

Here's the full listing. For a thorough explanation of the *new* portions, see **Program details** below (refer to the original article in V24N2 for the rest):

SUDOKUV2 *(1,898 Bytes)*

```
10 DESTROY ALL @ OPTION BASE 1 @ DIM P(9,9),S(9,9),E(9),R(9),C(9),B(9)
12 DISP "Initializing ..." @ STD @ FOR I=1 TO 9 @ K=3*((I-1) DIV 3)+1
14 FOR J=1 TO 9 @ S(I,J)=K+(J-1) DIV 3 @ NEXT J @ E(I)=2^I @ NEXT I
16 DISP "Enter puzzle: "; @ MAT INPUT P
18 INPUT "Max. Depth (1-15) = ","1";X @ X=INT(MAX(MIN(X,15),1))
20 INPUT "Max. Width (2-9) = ","2";Y @ Y=INT(MAX(MIN(Y,15),2))
22 INPUT "Verbose (Y/N) ? ","N";R$ @ Z=R$#"N" @ MAT DISP USING "DX";P
24 CALL TRY(P,F,1,X,Y,Z,S,E,R,C,B) @ IF F=2 THEN DISP "SOLVED!"
26 IF F=3 THEN DISP"Illegal?" ELSE IF F=4 THEN DISP"No solution found"
28 MAT DISP USING "DX";P @ END
30 SUB TRY(P(,),F,W,X,Y,Z,S(,),E(),R(),C(),B())
32 INTEGER A(81,4),N1(9,9),N2(9,9),N3(9,9),G(9,9),T(9,9)
34 DIM R2(9),C2(9),B2(9),D,I,J,U,V,K,M,N,H,L @ M=0 @ IF W#1 THEN 40
36 FOR I=1 TO 9 @ FOR J=1 TO 9 @ IF P(I,J) THEN D=E(P(I,J)) @ GOSUB 98
38 NEXT J @ NEXT I
40 F=2 @ K=0 @ MAT T=P @ MAT R2=R @ MAT C2=C @ MAT B2=B
42 FOR I=1 TO 9 @ U=R(I) @ FOR J=1 TO 9 @ V=C(J) @ IF P(I,J) THEN 56
44 K=K+1 @ H=S(I,J) @ F=F*(F#2) @ L=BINIOR(BINIOR(U,V),B(H))
46 IF L=1022 THEN F=3 @ MAT P=T @ MAT R=R2 @ MAT C=C2 @ MAT B=B2 @ END
48 D=BINAND(BINCMP(L),1023)-1 @ N=BIT(D,1)+BIT(D,2)+BIT(D,3)
50 N=N+BIT(D,4)+BIT(D,5)+BIT(D,6)+BIT(D,7)+BIT(D,8)+BIT(D,9)
52 IF N#1 THEN A(K,1)=I @ A(K,2)=J @ A(K,3)=N @ A(K,4)=L @ GOTO 56
54 P(I,J)=LOG2(D) @F=1 @M=M+1 @R(I)=R(I)+D @ C(J)=C(J)+D @ B(H)=B(H)+D
56 NEXT J @ NEXT I @ IF F=1 THEN 40 ELSE IF F=2 THEN END
58 MAT N1=ZER @MAT N2=ZER @ MAT N3=ZER @ N=0 @ FOR U=1 TO K @ I=A(U,1)
60 J=A(U,2) @ H=S(I,J) @ L=A(U,4) @ FOR V=1 TO 9 @ IF BIT(L,V) THEN 68
62 IF N1(I,V) THEN N1(I,V)=-1 ELSE N1(I,V)=J
64 IF N2(J,V) THEN N2(J,V)=-1 ELSE N2(J,V)=I
66 IF N3(H,V) THEN N3(H,V)=-1 ELSE N3(H,V)=I @ G(H,V)=J
68 NEXT V @ NEXT U @ FOR U=1 TO 9 @ FOR V=1 TO 9
70 J=N1(U,V) @ IF J>0 THEN I=U @ GOSUB 96
72 I=N2(U,V) @ IF I>0 THEN J=U @ GOSUB 96
74 I=N3(U,V) @ IF I>0 THEN J=G(U,V) @ GOSUB 96
76 NEXT V @ NEXT U @IF N THEN 40 ELSE IF Z THEN DISP W;": ";M;"forced"
78 IF W=X THEN F=4 @ END ELSE MAT T=P @ MAT R2=R @ MAT C2=C @ MAT B2=B
80 FOR U=1 TO K @ IF A(U,3)>Y THEN 94
82 I=A(U,1) @ J=A(U,2) @ L=A(U,4) @ FOR V=1 TO 9 @ IF BIT(L,V) THEN 92
84 P(I,J)=V @ H=S(I,J) @D=E(V) @R(I)=R(I)+D @C(J)=C(J)+D @ B(H)=B(H)+D
86 DISP W;":>Try";I+J/10;"=";V @ CALL TRY(P,F,W+1,X,Y,Z,S,E,R,C,B)
88 IF F=2 THEN END ELSE IF F=3 AND Z THEN DISP W;":    (dead end)"
90 MAT P=T @ MAT R=R2 @ MAT C=C2 @ MAT B=B2
92 NEXT V
94 NEXT U @ F=4 @ END
96 IF P(I,J) THEN RETURN ELSE M=M+1 @ N=1 @ P(I,J)=V @ D=E(V)
98 H=S(I,J) @ R(I)=R(I)+D @ C(J)=C(J)+D @ B(H)=B(H)+D @ RETURN
```

Notes: The program requires approximately **1725 + 2791*MD** bytes of free RAM available, where **MD** is the maximum depth of the search. It also makes use of keywords from the **Math ROM** and **HP-IL ROM**. **Emu71** executes the program *50-100X faster* on a typical 2.4 Ghz PC.

Programming details & techniques

> **Note**: A brief explanation of ***sudoku*** puzzles: you're given a 9x9 grid, each cell to be occupied by a *single* digit 1-9, such that *each* of the 9 columns, rows, and 3x3 non-overlapping blocks contain all digits without repetition. Initially some cells are already filled-in and you must fill the rest.

SUDOKUV2 continues to be a no-frills, didactic program, all its improvements being related to performance, i.e., reducing solving time. Only the *new* algorithms that have been added and other important changes will be discussed here, you should refer to the original article in V24N2 for important functional details as well as to get an overall view and thorough description of all remaining program parts.

Updating the bitboards more efficienly

For efficiency, **V1** updated all bitboards dynamically each time a cell was assigned a value. However, this was done only *within* the iterative processes at each particular depth. Everytime the **TRY** subprogram was called, it would first of all *regenerate* the bitboards from the *current* grid, *regardless* of the depth.

The new **V2**, on the other hand, only generates the bitboards ***once***, from the *initial* grid, at depth **1**, as seen in this code fragment:

```
30 SUB TRY(P(,),F,W,X,Y,Z,S(,),E(),R(),C(),B())
32 ...
34 ... IF W#1 THEN 40
36 FOR I=1 TO 9 @ FOR J=1 TO 9 @ IF P(I,J) THEN D=E(P(I,J)) @ GOSUB 98
38 NEXT J @ NEXT I
```

TRY checks the current depth at line 34, and generates the bitboards *only* if at depth **1**. Once the bitboards are generated, they're updated whenever a cell changes status, and are then made available to the *next* depth by passing them by reference as parameters of **TRY**. Notice that now **TRY's** parameter list *does* include the three bitboard vectors **R**(), **C**(), and **B**().

This results in significant savings in processing time as generating the bitboards is a time-consuming process. Now this is done only once per puzzle.

Filling-in additional forced digits: the Conjugate Criterium

In **V1**, at each depth level and before resorting to expensive recursion, **TRY** attempted to correctly fill in as many cells as possible by determining which cells could admit only a *single*, *forced* value, repeting the process iteratively until no more cells could be filled-in forcibly.

V1 essentially determined those cells that could hold only one value because all other possible values were already used up in cells belonging to their row, column, or block. Though these weren't the only forced digits out there, they could be found *very fast* by using <u>bitboards</u> (see V24N2), and the idea was to make findings as quickly as possible, then let the recursive search do the rest.

While this works quite well, the extensive tests I made with the help of the **TS** conclusively demonstrated that it was preferable to proceed otherwise, namely to spend *extra* time searching for *other* kinds of forced digits, hoping to either avoid recursion altogether or at the very least let it handle a less empty grid. So **V2** follows this strategy and, just after filling-in as many cells as possible with the former forced-digit criterium, it goes on to search *additional* forced digits by using the conjugate criterium, namely to look for those digits which can go in just one particular cell because all other empty cells in its row, column, or block also belong to a row, column, or block that already includes that digit. Note the conjugation: this is the equivalent of the first criterium after exchanging *cell* and *digit*. This sample puzzle from V24N2 will make it clear:

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

We saw in the original article (V24N2) how the first criterium uses *bitboards* to discover that, for instance, cell (1,6) is forced to have the value **7**.

Now, the conjugate criterium allows us to discover that, for instance, cell (1,8) must forcibly hold the value **4**, and cell (1,9) must hold a **5**. This is accomplished using this table for Row 1 which the program has previously generated from information already gathered at the time:

		Legal Digits								
Row	**Column**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	1	0	0	1	0	0	0	0	1	0
1	2	0	0	1	0	0	1	1	1	0
1	4	0	0	1	0	0	1	1	0	0
1	6	0	0	0	0	0	0	1	0	0
1	8	0	0	0	1	0	1	0	1	0
1	9	0	0	0	0	1	1	0	0	0

Notice that the bit representing the digit 5 is set *only* for column 9, so cell (1,9) is forced to hold a 5. Likewise , the bit representing the digit 4 is set *only* for column 8, so cell (1,8) is forced to hold a 4.

These tables are generated on the fly and stored in three 9x9 matrices, one for each of Rows, Columns, and Blocks. Then the *solitary* bits which denote *forced* digits are detected and extracted. These ten lines of code do it all:

```
58 MAT N1=ZER @ MAT N2=ZER @ MAT N3=ZER @ N=0 @ FOR U=1 TO K @ I=A(U,1)
60 J=A(U,2) @ H=S(I,J) @ L=A(U,4) @ FOR V=1 TO 9 @ IF BIT(L,V) THEN 68
62 IF N1(I,V) THEN N1(I,V)=-1 ELSE N1(I,V)=J
64 IF N2(J,V) THEN N2(J,V)=-1 ELSE N2(J,V)=I
66 IF N3(H,V) THEN N3(H,V)=-1 ELSE N3(H,V)=I @ G(H,V)=J
68 NEXT V @ NEXT U @ FOR U=1 TO 9 @ FOR V=1 TO 9
70 J=N1(U,V) @ IF J>0 THEN I=U @ GOSUB 96
72 I=N2(U,V) @ IF I>0 THEN J=U @ GOSUB 96
74 I=N3(U,V) @ IF I>0 THEN J=G(U,V) @ GOSUB 96
76 NEXT V @ NEXT U @ IF N THEN 40 ELSE IF Z THEN DISP W;": ";M;"forced"
```

These are essentially all the main changes to **V1**, the rest having to do with the dimensioning and update of the existing bitboards and new tables. Have a look at the original article in V24N2, which explains very thoroughly all sections not dealt with here, bitboards in particular.

Usage instructions

- The following are sample inputs and outputs. To start the program, press:

```
[RUN]     -> Initializing ...
          -> Enter puzzle:P(1,1)?
```

- Enter the **contents of all cells** (**0** if empty), left to right, top to bottom, separated by commas. You can enter up to 48 cells at a time, but entering just a *single row* per prompt will make it probably easier for you to keep track. For example:

```
          4,5,0,0,0,0,0,0,6  [ENTER]
P(2,1)?   0,0,3,0,0,1,0,0,7  [ENTER]
               ...           (enter rows 3 to 8)
P(9,1)?   7,0,0,0,0,0,0,6,4  [ENTER]
```

- Now enter the **Maximum Depth** and **Maximum Width** of the search (or accept the default values) and specify whether you want **Verbose** output or not:

```
Max. Depth (1-15) =  2  [ENTER]
Max. Width (2-9) =   2  [ENTER]
Verbose (Y/N) ?      N  [ENTER]
```

- The search will proceed unattended until a solution is found *or* it exhausts Max. Depth/Width without finding one (see **Examples** for more sample outputs):

```
Solving ...
(depth level#) :>Try (cell)=(digit)
...
```

then eventually, the search results are reported, which will be one of these:

`SOLVED` !	The subsequent grid is a *full solution* to the puzzle.
`Illegal`?	The puzzle is *inconsistent* and has *no solution*.
`No solution found`	The grid is output with *as many correct digits filled in as found*. Increase **Max.Depth/Width**, no need to re-enter the puzzle, just: `CONT 18[ENTER]`

Notes:

- Running time increases exponentially with **Maximum Depth/Width**, so it's advisable to start with the *lowest* values (Max. Depth = 1, Max. Width=2) and increase them if no solution is found. Usually (but not always), it's best to *restrict* **Width** to the range 2-4 and *increase* **Depth** instead. See V24N2.

- **Verbose** output includes the number of *digits forced* at each node of the search as well as indicating when it has encountered a *dead end* and it's *backtracking*.

- The program doesn't alter the `DELAY` setting, use the one that suits you best.

- Should you *miss* the final grid, you can *re-output* it by executing this right from the keyboard:

```
RUN 28  [ENTER]
```

Examples (directly taken from the Test Suite below)

Test case #1: 30 cells

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

```
>RUN
Initializing ...
Enter puzzle:
P(1,1)? 0,5,0,0,0,1,4,0,0  [ENTER]
P(2,1)? 2,0,3,0,0,0,7,0,0  [ENTER]
P(3,1)? 0,7,0,3,0,0,1,8,2  [ENTER]
P(4,1)? 0,0,4,0,5,0,0,0,7  [ENTER]
P(5,1)? 0,0,0,1,0,3,0,0,0  [ENTER]
P(6,1)? 8,0,0,0,2,0,6,0,0  [ENTER]
P(7,1)? 1,8,5,0,0,6,0,9,0  [ENTER]
P(8,1)? 0,0,2,0,0,0,8,0,3  [ENTER]
P(9,1)? 0,0,6,4,0,0,0,7,0  [ENTER]
```

```
Max. Depth (1-15) = 1   [ENTER]     6 5 8 2 7 1 4 3 9
Max. Width (2-9) =  2   [ENTER]     2 1 3 8 9 4 7 5 6
Verbose (Y/N) ?     N   [ENTER]     4 7 9 3 6 5 1 8 2
                                    9 2 4 6 5 8 3 1 7
Solving ...                         5 6 7 1 4 3 9 2 8
                                    8 3 1 9 2 7 6 4 5
SOLVED!      { HP71B: 59 seconds }  1 8 5 7 3 6 2 9 4
             { Emu71: 1 second   }  7 4 2 5 1 9 8 6 3
                                    3 9 6 4 8 2 5 7 1
```

Notes: This one is very easy, and no recursion is needed at all (**Depth =1**)

Test case #13: 19 cells

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

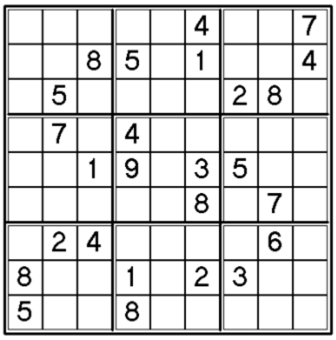
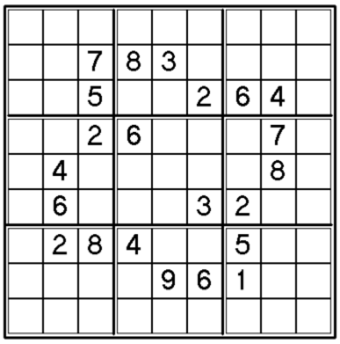
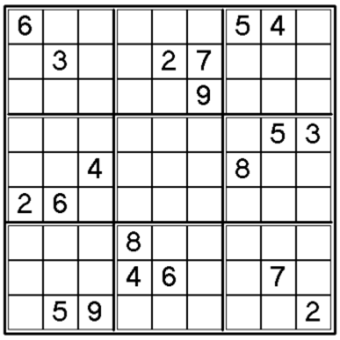
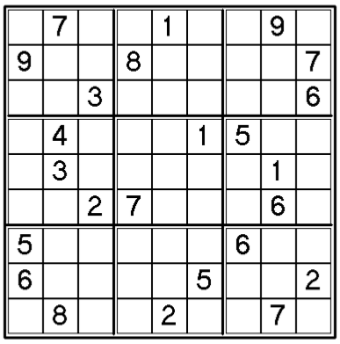
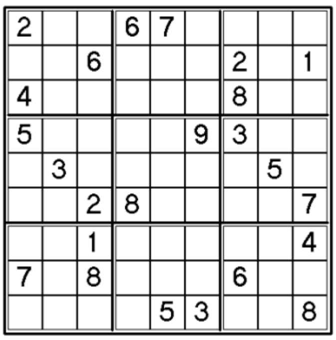
```
>RUN
Initializing ...
Enter puzzle:
P(1,1)? 0,0,0,0,0,9,0,0,0  [ENTER]
P(2,1)? 0,0,0,0,1,4,7,0,0  [ENTER]
P(3,1)? 0,0,2,0,0,0,0,0,0  [ENTER]
P(4,1)? 7,0,0,0,0,0,0,8,6  [ENTER]
P(5,1)? 5,0,0,0,3,0,0,0,2  [ENTER]
P(6,1)? 9,4,0,0,0,0,0,0,1  [ENTER]
P(7,1)? 0,0,0,0,0,0,4,0,0  [ENTER]
P(8,1)? 0,0,6,2,5,0,0,0,0  [ENTER]
P(9,1)? 0,0,0,8,0,0,0,0,0  [ENTER]
```

```
Max. Depth (1-15) = 1   [ENTER]
Max. Width (2-9) =  2   [ENTER]           8 1 4 7 2 9 6 3 5
Verbose (Y/N) ?     N   [ENTER]           6 5 9 3 1 4 7 2 8
                                          3 7 2 5 6 8 1 9 4
Solving ...                               7 2 1 4 9 5 3 8 6
                                          5 6 8 1 3 7 9 4 2
SOLVED!      { HP71B: 4 min 45 sec }      9 4 3 6 8 2 5 7 1
             { Emu71: 10 sec }            2 8 5 9 7 1 4 6 3
                                          4 9 6 2 5 3 8 1 7
                                          1 3 7 8 4 6 2 5 9
```

Notes: A lot harder, but recursion is *still not needed* ! **V1** took nearly <u>8 hours</u> (in an *actual* HP-71B) and had to go all the way to <u>depth 5</u> in order to solve it.

Test Suite: 15 choice puzzles of various difficulties, from very easy to very hard.

<table>
<tr><th>Sudoku Test Puzzle #</th><th>Cells</th><th>71B (Emu71)</th><th>Version 1.0</th><th>Computed Solution</th></tr>
<tr><td>01.
<table>
<tr><td></td><td>5</td><td></td><td></td><td></td><td>1</td><td>4</td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>3</td><td></td><td></td><td></td><td>7</td><td></td><td></td></tr>
<tr><td></td><td>7</td><td></td><td>3</td><td></td><td></td><td>1</td><td>8</td><td>2</td></tr>
<tr><td></td><td></td><td>4</td><td></td><td>5</td><td></td><td></td><td></td><td>7</td></tr>
<tr><td></td><td></td><td></td><td>1</td><td></td><td>3</td><td></td><td></td><td></td></tr>
<tr><td>8</td><td></td><td></td><td></td><td>2</td><td></td><td>6</td><td></td><td></td></tr>
<tr><td>1</td><td>8</td><td>5</td><td></td><td></td><td>6</td><td></td><td>9</td><td></td></tr>
<tr><td></td><td></td><td>2</td><td></td><td></td><td></td><td>8</td><td></td><td>3</td></tr>
<tr><td></td><td></td><td>6</td><td>4</td><td></td><td></td><td></td><td>7</td><td></td></tr>
</table></td><td>30</td><td>00:00:59
(00:00:01)
using 1-2
Recursion
NOT needed</td><td>00:00:55
(00:00:01)
using 1-2
Recursion
NOT needed</td><td>6 5 8 2 7 1 4 3 9
2 1 3 8 9 4 7 5 6
4 7 9 3 6 5 1 8 2
9 2 4 6 5 8 3 1 7
5 6 7 1 4 3 9 2 8
8 3 1 9 2 7 6 4 5
1 8 5 7 3 6 2 9 4
7 4 2 5 1 9 8 6 3
3 9 6 4 8 2 5 7 1</td></tr>
<tr><td>02.
<table>
<tr><td>6</td><td></td><td></td><td>1</td><td></td><td></td><td></td><td>5</td><td></td></tr>
<tr><td></td><td>4</td><td></td><td></td><td></td><td>2</td><td></td><td></td><td>3</td></tr>
<tr><td></td><td></td><td>3</td><td></td><td>5</td><td></td><td>1</td><td></td><td></td></tr>
<tr><td></td><td>1</td><td></td><td></td><td></td><td>3</td><td></td><td></td><td>4</td></tr>
<tr><td></td><td></td><td>5</td><td></td><td>6</td><td></td><td>9</td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td>9</td><td></td><td></td><td></td><td>1</td><td></td></tr>
<tr><td></td><td></td><td>6</td><td></td><td>8</td><td></td><td>7</td><td></td><td></td></tr>
<tr><td>3</td><td></td><td></td><td>7</td><td></td><td></td><td></td><td>6</td><td></td></tr>
<tr><td></td><td>2</td><td></td><td></td><td></td><td>6</td><td></td><td></td><td>1</td></tr>
</table></td><td>27</td><td>00:02:53
(00:00:06)
using 1-2
Recursion
NOT needed</td><td>Untested
(00:00:30)
using 3-2
Recursion
needed</td><td>6 8 2 1 3 9 4 5 7
5 4 1 6 7 2 8 9 3
9 7 3 4 5 8 1 2 6
7 1 9 5 2 3 6 8 4
4 3 5 8 6 1 9 7 2
2 6 8 9 4 7 3 1 5
1 5 6 2 8 4 7 3 9
3 9 4 7 1 5 2 6 8
8 2 7 3 9 6 5 4 1</td></tr>
<tr><td>03.
<table>
<tr><td>3</td><td>2</td><td></td><td></td><td></td><td></td><td>7</td><td></td><td></td></tr>
<tr><td></td><td></td><td>9</td><td>8</td><td></td><td></td><td>3</td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td>6</td><td></td><td></td><td>2</td></tr>
<tr><td></td><td>5</td><td></td><td></td><td></td><td>9</td><td></td><td></td><td>3</td></tr>
<tr><td></td><td>4</td><td></td><td></td><td></td><td></td><td></td><td>2</td><td></td></tr>
<tr><td>7</td><td></td><td></td><td>3</td><td></td><td></td><td></td><td>4</td><td></td></tr>
<tr><td>1</td><td></td><td></td><td>7</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>6</td><td></td><td></td><td>2</td><td>5</td><td></td><td></td></tr>
<tr><td></td><td></td><td>4</td><td></td><td></td><td></td><td></td><td>8</td><td>6</td></tr>
</table></td><td>24</td><td>00:03:58
(00:00:08)
using 1-2
Recursion
NOT needed</td><td>00:06:55
(00:00:11)
using 3-3
Recursion
needed</td><td>3 2 1 9 5 4 7 6 8
4 6 9 8 2 7 3 5 1
5 8 7 1 3 6 4 9 2
6 5 8 2 4 9 1 7 3
9 4 3 6 7 1 8 2 5
7 1 2 3 8 5 6 4 9
1 9 5 7 6 8 2 3 4
8 3 6 4 9 2 5 1 7
2 7 4 5 1 3 9 8 6</td></tr>
<tr><td>04.
<table>
<tr><td>4</td><td>5</td><td></td><td></td><td></td><td></td><td></td><td></td><td>6</td></tr>
<tr><td></td><td></td><td>3</td><td></td><td></td><td>1</td><td></td><td></td><td>7</td></tr>
<tr><td></td><td></td><td></td><td></td><td>2</td><td>3</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td>4</td><td></td><td>2</td><td>5</td><td></td></tr>
<tr><td></td><td></td><td>9</td><td>3</td><td></td><td>2</td><td>1</td><td></td><td></td></tr>
<tr><td></td><td>8</td><td>1</td><td></td><td>7</td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td>5</td><td>8</td><td></td><td></td><td></td><td></td></tr>
<tr><td>9</td><td></td><td></td><td>7</td><td></td><td></td><td>8</td><td></td><td></td></tr>
<tr><td>7</td><td></td><td></td><td></td><td></td><td></td><td></td><td>6</td><td>4</td></tr>
</table></td><td>26</td><td>00:02:32
(00:00:05)
using 1-2
Recursion
NOT needed</td><td>00:02:29
(00:00:04)
using 2-2
Recursion
needed</td><td>4 5 2 8 9 7 3 1 6
8 9 3 6 5 1 4 2 7
1 7 6 4 2 3 9 8 5
6 3 7 1 4 8 2 5 9
5 4 9 3 6 2 1 7 8
2 8 1 9 7 5 6 4 3
3 2 4 5 8 6 7 9 1
9 6 5 7 1 4 8 3 2
7 1 8 2 3 9 5 6 4</td></tr>
<tr><td>05.
<table>
<tr><td></td><td></td><td>4</td><td></td><td>5</td><td></td><td></td><td>6</td><td></td></tr>
<tr><td></td><td>6</td><td></td><td>1</td><td></td><td></td><td>8</td><td></td><td>9</td></tr>
<tr><td>3</td><td></td><td></td><td></td><td></td><td>7</td><td></td><td></td><td></td></tr>
<tr><td></td><td>8</td><td></td><td></td><td></td><td></td><td>5</td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td>4</td><td></td><td>3</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>6</td><td></td><td></td><td></td><td></td><td>7</td><td></td></tr>
<tr><td></td><td></td><td></td><td>2</td><td></td><td></td><td></td><td></td><td>6</td></tr>
<tr><td>1</td><td></td><td>5</td><td></td><td></td><td>4</td><td></td><td>3</td><td></td></tr>
<tr><td></td><td>2</td><td></td><td></td><td>7</td><td></td><td>1</td><td></td><td></td></tr>
</table></td><td>24</td><td>00:03:50
(00:00:08)
using 1-2
Recursion
NOT needed</td><td>Untested
(00:00:08)
using 2-2
Recursion
needed</td><td>2 1 4 9 5 8 3 6 7
5 6 7 1 3 2 8 4 9
3 9 8 6 4 7 2 1 5
4 8 1 7 9 6 5 2 3
7 5 2 4 8 3 6 9 1
9 3 6 5 2 1 4 7 8
8 4 3 2 1 9 7 5 6
1 7 5 8 6 4 9 3 2
6 2 9 3 7 5 1 8 4</td></tr>
</table>

06. 	26	**00:02:51 (00:00:06) using 1-2 Recursion NOT needed**	**Untested (00:01:30) using 3-2 Recursion needed**	**3 1 2 6 8 4 9 5 7 7 9 8 5 2 1 6 3 4 4 5 6 7 3 9 2 8 1 9 7 5 4 1 6 8 2 3 2 8 1 9 7 3 5 4 6 6 4 3 2 5 8 1 7 9 1 2 4 3 9 5 7 6 8 8 6 7 1 4 2 3 9 5 5 3 9 8 6 7 4 1 2**
07. 	22	**00:04:52 (00:00:10) using 1-2 Recursion NOT needed**	**00:19:58 (00:00:21) using 2-3 Recursion needed**	**2 9 4 1 6 5 8 3 7 6 1 7 8 3 4 9 5 2 3 8 5 9 7 2 6 4 1 5 3 2 6 8 1 4 7 9 7 4 1 2 5 9 3 8 6 8 6 9 7 4 3 2 1 5 9 2 8 4 1 7 5 6 3 4 7 3 5 9 6 1 2 8 1 5 6 3 2 8 7 9 4**
08. 	20	**00:04:44 (00:00:10) using 1-2 Recursion NOT needed**	**Untested (03:10:30) using 5-3 Recursion needed**	**6 9 2 3 8 1 5 4 7 4 3 5 6 2 7 1 8 9 1 8 7 5 4 9 2 3 6 9 1 8 2 7 4 6 5 3 5 7 4 9 3 6 8 2 1 2 6 3 1 5 8 7 9 4 7 4 6 8 9 2 3 1 5 3 2 1 4 6 5 9 7 8 8 5 9 7 1 3 4 6 2**
09. 	24	**00:06:54 (00:00:15) using 1-2 Recursion NOT needed**	**Untested (00:14:26) using 6-2 Recursion needed**	**4 7 8 5 1 6 2 9 3 9 6 5 8 3 2 1 4 7 2 1 3 4 9 7 8 5 6 7 4 9 3 6 1 5 2 8 8 3 6 2 5 9 7 1 4 1 5 2 7 4 8 3 6 9 5 2 4 9 7 3 6 8 1 6 9 7 1 8 5 4 3 2 3 8 1 6 2 4 9 7 5**
10. 	24	**00:06:26 (00:00:14) using 2-2 Recursion needed**	**Untested (01:00:33) using 5-2 Recursion needed**	**2 8 3 6 7 1 9 4 5 9 7 6 5 4 8 2 3 1 4 1 5 3 9 2 8 7 6 5 6 7 4 1 9 3 8 2 8 3 4 2 6 7 1 5 9 1 9 2 8 3 5 4 6 7 3 2 1 7 8 6 5 9 4 7 5 8 9 2 4 6 1 3 6 4 9 1 5 3 7 2 8**

11. .6.1.4.5. ..83.56.. 2.......1 8..4.7..6 ..6...3.. 7..9.1..4 5.......2 ..72.69.. .4.5.8.7.	30	00:02:04 (00:00:04) using 1-2 Recursion NOT needed	Untested (00:00:04) using 2-2 Recursion needed	9 6 3 1 7 4 2 5 8 1 7 8 3 2 5 6 4 9 2 5 4 6 8 9 7 3 1 8 2 1 4 3 7 5 9 6 4 9 6 8 5 2 3 1 7 7 3 5 9 6 1 8 2 4 5 8 9 7 1 3 4 6 2 3 1 7 2 4 6 9 8 5 6 4 2 5 9 8 1 7 3
12. ..2.1.9.. ...5.4... 45..8..37 7..263..8 .... 5..879..2 69..2..54 ...9.6... ..1.4.6..	30	00:01:03 (00:00:01) using 1-2 Recursion NOT needed	Untested (00:00:01) using 1-2 Recursion NOT needed	8 6 2 3 1 7 9 4 5 1 7 3 5 9 4 2 8 6 4 5 9 6 8 2 1 3 7 7 1 4 2 6 3 5 9 8 9 2 8 4 5 1 7 6 3 5 3 6 8 7 9 4 1 2 6 9 7 1 2 8 3 5 4 2 4 5 9 3 6 8 7 1 3 8 1 7 4 5 6 2 9
13. .....9... ....147.. ..2...... 7......86 5...3...2 94......1 ......4.. ..625.... ...8.....	19	00:04:45 (00:00:10) using 1-2 Recursion NOT needed	07:58:03 (00:06:36) using 5-2 Recursion needed	8 1 4 7 2 9 6 3 5 6 5 9 3 1 4 7 2 8 3 7 2 5 6 8 1 9 4 7 2 1 4 9 5 3 8 6 5 6 8 1 3 7 9 4 2 9 4 3 6 8 2 5 7 1 2 8 5 9 7 1 4 6 3 4 9 6 2 5 3 8 1 7 1 3 7 8 4 6 2 5 9
14. ....4..3. 98.6.1... ......2.. ....1 ..4.5.7.. 6........ ..5...... ...9.8.76 .7..3....	19	00:15:21 (00:00:32) using 3-2 Recursion needed	Untested (03:26:48) using 6-2 Recursion needed	2 6 7 5 4 9 1 3 8 9 8 3 6 2 1 4 5 7 5 4 1 7 8 3 2 6 9 7 5 8 3 9 2 6 4 1 3 1 4 8 5 6 7 9 2 6 2 9 1 7 4 3 8 5 1 9 5 4 6 7 8 2 3 4 3 2 9 1 8 5 7 6 8 7 6 2 3 5 9 1 4
15. .2....... ...6....3 .74.8.... .....3..2 .8..4..1. 6..5..... ....1.78. 5....9... ....4.	19	00:09:38 (00:00:19) using 3-4 Recursion needed	Untested (02:42:07) using 5-4 Recursion needed	1 2 6 4 3 7 9 5 8 8 9 5 6 2 1 4 7 3 3 7 4 9 8 5 1 2 6 4 5 7 1 9 3 8 6 2 9 8 3 2 4 6 5 1 7 6 1 2 5 7 8 3 9 4 2 6 9 3 1 4 7 8 5 5 4 8 7 6 9 2 3 1 7 3 1 8 5 2 6 4 9