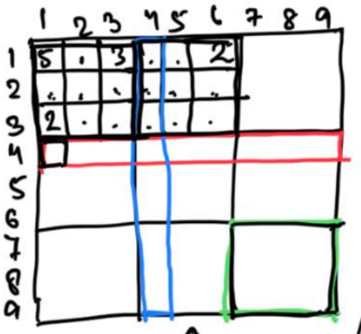
Valid Sudoku

No. should not repeat
- same row
- same col
- same square

In all cases, same logic has to be applied.



→ Validate if the numbers which are filled satisfy the criteria for this grid to be valid sudoku

→ HashSet → subset of cells → values
→ if any is repeating return false

In total - n^2 cells.

→ As soon as, any subset is invalid return false.

→ Equal elements in all → n

→ It's a rectangle/sub-matrix (Top-Left & Bottom-right)

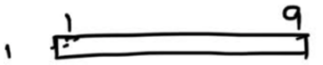
(1, 4) — (9,4)

(7,7) — (9,9)

→ boolean isValidSubMatrix(grid, tlx, tly, brx, bry)

// HashSet →

bool validate

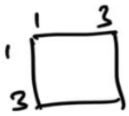


TL (1,1) , BR(1,9)

```
9 { for ( i = 1 - 9)
      if(! isValidSM(i, 1, i, 9))
         return false
1
```

TL - (1,1)
BR - (9,1)

```
9 { for (i = 1 - 9)
      if (! isVSM(1, i, 9, i))
         return false
1
```



TL (1, 1)
BR (3,3)

```
9 { for (i = 1 - 9; i+=3)
      for (j = 1 - 9; j+=3)
         if (! isVSM(i, j, i+3, j+3))
            return false
1
```

return true

(i,j) → (1,1) (1,4) (1,7)
→ (4,1) (4,4) (4,7)
(7,1) (7,4) (7,7)

```
→ isV.SM (tlx, tly, brx, bry)
  Set
  for i = tlx - brx
     for j = tly - bry
        if (set.cont(g[i][j]))
           return false.
```

Overall T.C: $O(n^2)$
S.C: $O(n^2)$

Don't include empty cells in the grid. →

Solving a valid Sudoku.

→ Backtracking by checking every all nos. for every empty cell till we find the grid.

Solving a valid Sudoku.

→ Backtracking by checking every all nos. for every empty cell till we find a way to fill all empty cells.

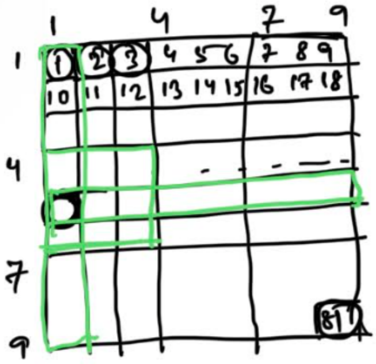
.	2	.	1	.	.

6 cells. → Peumutations

1–9

→ 4 2 5 1 8 7

→ 8 2 9 1 3 6



2D matrix — array address in 1D array store

grid[indx] = '.'

1 – 9 → Try a possibility

Check if that possibility is correct   grid[indx] = '1' 2 3 revert

if correct
move to next index

else
try next possibity

if no possibility worked
backtrack → prev.   grid[indx] = '.'

This backtracking method needs to check something

- return type - boolean.

Base Cases.
if indx = 81
return true.

Method Signature: start → 1

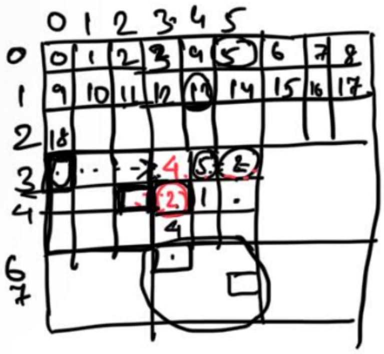
```
boolean isValidSudoku(grid, indx)
    if indx = 81 {
        return true.
    }
    if (grid[indx] != '.') {  ←
        return isValidSudoku(grid, indx+1)
    for (i = 1 to 9) {  ←
```

1 1

```
if indx = 81 {
    return true.
}
if (grid[indx] != '.') {
    return isValidSudoku(grid, indx+1)
}
for (i = 1 to 9) {
    grid[indx] = i
    if (isValidCell(indx))
        return isValidSudoku(grid, indx+1)
}
grid[indx] = '.'      → To signify empty cells after indx
return false;
```

All submatrices with the cell at indx must be Valid. →

3 — isVSM



$13 \% 9 \rightarrow 4$ (1,4)

$13 / 9 \rightarrow 1$

indx → (indx/9, indx % 9)

(x, y)

grid[indx] = grid[x][y].

(0,5) → (0,6)
(0,8) → (1,0) } Not the same change every time.

isValidSudoku

↘ recursive stack → maintaining backtracking

isValidCell.(indx)

[x = indx/9
  y = indx % 9

isVSM(x,0,x,8) ← x^{th} row

isVSM(0,y,8,y) y^{th} col

Integer Division → (x/3, y/3)
= (x1, y1)

(3x1, 3y1)
= (x2, y2)

isVSM(x2,y2,x2+3,y2+3) ← (x2, y2)

(4,2)

(1,0)

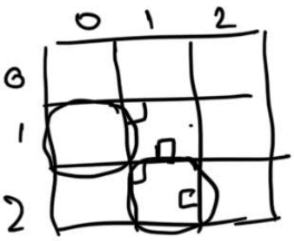
(

(1,0)

(1,0)

(3,0)

(3,0)



(7,5)

(2,1)

(6,3)