Valid Sudoku - brute force

```
class Solution:
    def is_valid_sudoku(self, board: List[List[str]]) -> bool:

        for row in range(9):
            seen = set()
            for i in range(9):
                if board[row][i] == ".":
                    continue
                if board[row][i] in seen:
                    return False
                seen.add(board[row][i])

        for col in range(9):
            seen = set()
            for i in range(9):
                if board[i][col] == ".":
                    continue
                if board[i][col] in seen:
                    return False
```

```
seen.add(board[i][col])

for square in range(9):
    seen = set()
    for i in range(3):
        for j in range(3):
            row = (square // 3) * 3 + i
            col = (square % 3) * 3 + j
            if board[row][col] == ".":
                continue
            if board[row][col] in seen:
                return False
            seen.add(board[row][col])

return True
```

```
print(Solution().is_valid_sudoku(board =
[["1","2",".",".","3",".",".",".","."],
 ["4",".",".","5",".",".",".",".","."],
 [".","9","8",".",".",".",".",".","3"],
 ["5",".",".",".","6",".",".",".","4"],
 [".",".",".","8",".","3",".",".","5"],
 ["7",".",".",".",".","2",".",".","6"],
```

This exercise is using brute force algorithm, and is running more than 1000 times (as I call them 1000 steps).

BIG Step 1:

```
for row in range(9):
    seen = set()
    for i in range(9):
        if board[row][i] == ".":
            continue
        if board[row][i] in seen:
            return False
        seen.add(board[row][i])
```

This block of code checks if each row contains the digits 1-9 without duplicates. 1st iteration goes through row 0 (1st row) and check each value from the list, if it finds any duplicate, then the code breaks and is returning False.

The second step, checks for each column in the same manner as we did in the first step when we checked for our rows.

BIG Step 3:
```
for square in range(9):
    seen = set()
    for i in range(3):
        for j in range(3):
            row = (square // 3) * 3 + i
            col = (square % 3) * 3 + j
            if board[row][col] == '.':
                continue
            if board[row][col] in seen:
                return False
            seen.add(board[row][col])
```

Info: `row = (square // 3) * 3 + i`

the floor division // rounds the result to the nearest whole number. exp: x = 15, y = 2 } 15//2 = 7

`col = (square % 3) * 3 + j`

the modulo operator % calculates the remainder of

a division operation. exp: $x = 4$ $y = 3$ $4 \% 3 = 1$

In this iteration we check every 3 squares if we got any duplicates, first square