Valid Sudoku

Approach :

Find in the matrix for empty space. When an empty space is found try out all possible numbers that can be present at that cell one by one and call it recursively to find whether the sudoku is possible or not if all the numbers are tried and no recursive calls can be made because none of the number can be inserted at that cell then we return a false starting the sudoku is not possible with that recursive call and backtrack removing the number last inserted in the cell to 0.

If the whole matrix has been filled succesfully and the loops are terminated we return a true stating that matrix is a part of the ans.

```
bool isValid(int i,int j,int c,int matrix[9][9])
{
    for(int row=0;row<9;row++)
    {
        if(matrix[row][j]==c)
            return false;
    }
    for(int col=0;col<9;col++)
    {
        if(matrix[i][col]==c)
            return false;
    }
    int row=i/3;
    int col=j/3;
    for(int p=0;p<9;p++)
    {
        if(matrix[(3*row)+p/3][(3*col)+p%3]==c)
            return false;
    }
    return true;
}
bool isItSudoku(int matrix[9][9]) {
    // Write your code here.
    for(int i=0;i<9;i++)
    {
        for(int j=0;j<9;j++)
        {
            if(matrix[i][j]==0)
            {
              for (int t = 1; t <= 9; t++) {
                  if(isValid(i,j,t,matrix))
                  {
```

```
                        matrix[i][j]=t;
                        if(isItSudoku(matrix))
                        {
                            return true;
                        } else {
                          matrix[i][j] = 0;
                        }
                    }
                }
                return false;
            }
        }
    }
    return true;
}
```

- Time Complexity : O(9^(n^2))

9 possible numbers so 9 recursive calls can be made.

- Space Complexity : O(1)