 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Design and Analysis of Algorithms (01CT0512)	Aim: Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution	
Experiment No: 14	Date:	Enrollment No: 92200133030

Aim: Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution

IDE: Visual Studio Code

Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution

- Given a partially filled 9×9 2D array 'grid[9][9]', the goal is to assign digits (from 1 to 9) to the empty cells so that every row, column, and subgrid of size 3×3 contains exactly one instance of the digits from 1 to 9.

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

1. Naive Approach:


- The naive approach is to generate all possible configurations of numbers from 1 to 9 to fill the empty cells. Try every configuration one by one until the correct configuration is found, i.e. for every unassigned position fill the position with a number from 1 to 9. After filling all the unassigned positions check if the matrix is safe or not. If safe print else recurs for other cases.
- Follow the steps below to solve the problem:
 - Create a function that checks if the given matrix is valid sudoku or not. Keep Hashmap for the row, column and boxes. If any number has a frequency greater than 1 in the hashMap return false else return true;



Aim: Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution

Enrollment No: 92200133030

Algorithm: -

 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Design and Analysis of Algorithms (01CT0512)	Aim: Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution	
Experiment No: 14	Date:	Enrollment No: 92200133030

Code :-

```
#include <iostream>
using namespace std;
#define N 9

void print(int arr[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
            cout << arr[i][j] << " ";
        cout << endl;
    }
}

bool isSafe(int grid[N][N], int row, int col, int num) {

    for (int x = 0; x <= 8; x++)
        if (grid[row][x] == num)
            return false;

    for (int x = 0; x <= 8; x++)
        if (grid[x][col] == num)
            return false;

    int startRow = row - row % 3, startCol = col - col % 3;

    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            if (grid[i + startRow][j + startCol] == num)
                return false;


    return true;
}

bool solveSudoku(int grid[N][N], int row, int col) {
    if (row == N - 1 && col == N)
        return true;

    if (col == N) {
        row++;
        col = 0;
    }

    if (grid[row][col] > 0)
        return solveSudoku(grid, row, col + 1);

    for (int num = 1; num <= N; num++) {
```

 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Design and Analysis of Algorithms (01CT0512)	Aim: Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution	
Experiment No: 14	Date:	Enrollment No: 92200133030

```

        if (isSafe(grid, row, col, num)) {
            grid[row][col] = num;

            if (solveSudoku(grid, row, col + 1))
                return true;
        }
        grid[row][col] = 0;
    }
    return false;
}

int main() {
    int grid[N][N] = { { 3, 0, 6, 5, 0, 8, 4, 0, 0 },
                        { 5, 2, 0, 0, 0, 0, 0, 0, 0 },
                        { 0, 8, 7, 0, 0, 0, 0, 3, 1 },
                        { 0, 0, 3, 0, 1, 0, 0, 8, 0 },
                        { 9, 0, 0, 8, 6, 3, 0, 0, 5 },
                        { 0, 5, 0, 0, 9, 0, 6, 0, 0 },
                        { 1, 3, 0, 0, 0, 0, 2, 5, 0 },
                        { 0, 0, 0, 0, 0, 0, 0, 7, 4 },
                        { 0, 0, 5, 2, 0, 6, 3, 0, 0 } };

    if (solveSudoku(grid, 0, 0))
        print(grid);
    else
        cout << "no solution exists " << endl;
    return 0;
}


```

Output :-

```

PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 14> cd "d:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 14\" ; if ($?) { g++ Sudoku_Solver_Naive.cpp -o Sudoku_Solver_Naive } ; if ($?) { .\Sudoku_Solver_Naive }
3 1 6 5 7 8 4 9 2
5 2 9 1 3 4 7 6 8
4 8 7 6 2 9 5 3 1
2 6 3 4 1 5 9 8 7
9 7 4 8 6 3 1 2 5
8 5 1 7 9 2 6 4 3
1 3 8 9 4 7 2 5 6
6 9 2 3 5 1 8 7 4
7 4 5 2 8 6 3 1 9
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 14>

```

 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Design and Analysis of Algorithms (01CT0512)	Aim: Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution	
Experiment No: 14	Date:	Enrollment No: 92200133030

Space Complexity:- _____

Justification: -

Time Complexity:

Best Case Time Complexity: _____

Justification: -

Worst Case Time Complexity:- _____

Justification: -

2. **Sudoku Using Backtracking :-**

- Like all other Backtracking problems, Sudoku can be solved by assigning numbers one by one to empty cells. Before assigning a number, check whether it is safe to assign.
- Check that the same number is not present in the current row, current column and current 3X3 subgrid. After checking for safety, assign the number, and recursively check whether this assignment leads to a solution or not. If the assignment doesn't lead to a solution, then try the next number for the current empty cell. And if none of the number (1 to 9) leads to a solution, return false and print no solution exists.
- Follow the steps below to solve the problem:
 - Create a function that checks after assigning the current index the grid becomes unsafe or not. Keep Hashmap for a row, column and boxes. If any number has a frequency greater than 1 in the hashMap return false else return true; hashMap can be avoided by using loops.



Aim: Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution


Enrollment No: 92200133030

- Algorithm: -**

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Code :-

```
#include <bits/stdc++.h>
using namespace std;
#define UNASSIGNED 0
#define N 9
```

 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Design and Analysis of Algorithms (01CT0512)	Aim: Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution	
Experiment No: 14	Date:	Enrollment No: 92200133030

```
bool FindUnassignedLocation(int grid[N][N],int& row, int& col);
bool isSafe(int grid[N][N], int row,int col, int num);
```

```
bool SolveSudoku(int grid[N][N]) {
    int row, col;

    if (!FindUnassignedLocation(grid, row, col))
        return true;


    for (int num = 1; num <= 9; num++)
    {
        if (isSafe(grid, row, col, num))
        {
            grid[row][col] = num;
            if (SolveSudoku(grid))
                return true;
            grid[row][col] = UNASSIGNED;
        }
    }
    return false;
}
```

```
bool FindUnassignedLocation(int grid[N][N],int& row, int& col) {
    for (row = 0; row < N; row++)
        for (col = 0; col < N; col++)
            if (grid[row][col] == UNASSIGNED)
                return true;
    return false;
}
```

```
bool UsedInRow(int grid[N][N], int row, int num) {
    for (int col = 0; col < N; col++)
        if (grid[row][col] == num)
            return true;
    return false;
}
```

```
bool UsedInCol(int grid[N][N], int col, int num) {
    for (int row = 0; row < N; row++)
        if (grid[row][col] == num)
            return true;
    return false;
}
```

```
bool UsedInBox(int grid[N][N], int boxStartRow, int boxStartCol, int num) {
    for (int row = 0; row < 3; row++)
        for (int col = 0; col < 3; col++)
            if (grid[row + boxStartRow][col + boxStartCol] == num)
                return true;
}
```

 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Design and Analysis of Algorithms (01CT0512)	Aim: Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution	
Experiment No: 14	Date:	Enrollment No: 92200133030

```

    return false;
}

bool isSafe(int grid[N][N], int row, int col, int num) {
    return !UsedInRow(grid, row, num) && !UsedInCol(grid, col, num) && !UsedInBox(grid,
row - row % 3, col - col % 3, num) && grid[row][col] == UNASSIGNED;
}

void printGrid(int grid[N][N]) {
    for (int row = 0; row < N; row++)
    {
        for (int col = 0; col < N; col++)
            cout << grid[row][col] << " ";
        cout << endl;
    }
}

int main()
{
    int grid[N][N] = { { 3, 0, 6, 5, 0, 8, 4, 0, 0 },
                        { 5, 2, 0, 0, 0, 0, 0, 0, 0 },
                        { 0, 8, 7, 0, 0, 0, 0, 3, 1 },
                        { 0, 0, 3, 0, 1, 0, 0, 8, 0 },
                        { 9, 0, 0, 8, 6, 3, 0, 0, 5 },
                        { 0, 5, 0, 0, 9, 0, 6, 0, 0 },
                        { 1, 3, 0, 0, 0, 0, 2, 5, 0 },
                        { 0, 0, 0, 0, 0, 0, 0, 7, 4 },
                        { 0, 0, 5, 2, 0, 6, 3, 0, 0 } };

    if (SolveSudoku(grid) == true)
        printGrid(grid);
    else
        cout << "No solution exists";

    return 0;
}


```

Output :-

```

PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 14> cd "d:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 14\" ; if ($?) { g++ Sudoku_Solver_Naive.cpp -o Sudoku_Solver_Naive } ; if ($?) { .\Sudoku_Solver_Naive }
3 1 6 5 7 8 4 9 2
5 2 9 1 3 4 7 6 8
4 8 7 6 2 9 5 3 1
2 6 3 4 1 5 9 8 7
9 7 4 8 6 3 1 2 5
8 5 1 7 9 2 6 4 3
1 3 8 9 4 7 2 5 6
6 9 2 3 5 1 8 7 4
7 4 5 2 8 6 3 1 9
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 14>

```


 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Design and Analysis of Algorithms (01CT0512)	Aim: Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution	
Experiment No: 14	Date:	Enrollment No: 92200133030

Space Complexity:- _____

Justification: -

Time Complexity:

Best Case Time Complexity: _____

Justification: -

Worst Case Time Complexity:- _____

Justification: -

3. Sudoku Using Bit Masks :-


- This method is a slight optimization to the above 2 methods. For each row/column/box create a bitmask and for each element in the grid set the bit at position 'value' to 1 in the corresponding bitmasks, for O(1) checks.
- **Follow the steps below to solve the problem:**
 - Create 3 arrays of size N (one for rows, columns, and boxes).
 - The boxes are indexed from 0 to 8. (in order to find the box index of an element we use the following formula: $\text{row} / 3 * 3 + \text{column} / 3$).
 - Map the initial values of the grid first.
 - Each time we add/remove an element to/from the grid set the bit to 1/0 to the corresponding bitmasks.

```
#include <bits/stdc++.h>
using namespace std;
#define N 9

int row[N], col[N], box[N];
bool seted = false;

int getBox(int i, int j) {
    return i / 3 * 3 + j / 3;
}

bool isSafe(int i, int j, int number) {
    return !((row[i] >> number) & 1) && !((col[j] >> number) & 1) && !((box[getBox(i, j)] >> number) & 1);
}
```

 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Design and Analysis of Algorithms (01CT0512)	Aim: Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution	
Experiment No: 14	Date:	Enrollment No: 92200133030

```
void setInitialValues(int grid[N][N]) {
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            row[i] |= 1 << grid[i][j],
            col[j] |= 1 << grid[i][j],
            box[getBox(i, j)] |= 1 << grid[i][j];
}
```

```
bool SolveSudoku(int grid[N][N], int i, int j) {
    if (!seted)
        seted = true, setInitialValues(grid);

    if (i == N - 1 && j == N)
        return true;
    if (j == N)
        j = 0, i++;

    if (grid[i][j])
        return SolveSudoku(grid, i, j + 1);

    for (int nr = 1; nr <= N; nr++) {
        if (isSafe(i, j, nr)) {
            grid[i][j] = nr;
            row[i] |= 1 << nr;
            col[j] |= 1 << nr;
            box[getBox(i, j)] |= 1 << nr;


            if (SolveSudoku(grid, i, j + 1))
                return true;

            row[i] &= ~(1 << nr);
            col[j] &= ~(1 << nr);
            box[getBox(i, j)] &= ~(1 << nr);
        }

        grid[i][j] = 0;
    }

    return false;
}
```

```
void print(int grid[N][N])
{
    for (int i = 0; i < N; i++, cout << '\n')
        for (int j = 0; j < N; j++)
            cout << grid[i][j] << ' ';
}
```

 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Design and Analysis of Algorithms (01CT0512)	Aim: Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution	
Experiment No: 14	Date:	Enrollment No: 92200133030

```
int main()
{
    int grid[N][N] = { { 3, 0, 6, 5, 0, 8, 4, 0, 0 },
                        { 5, 2, 0, 0, 0, 0, 0, 0, 0 },
                        { 0, 8, 7, 0, 0, 0, 0, 3, 1 },
                        { 0, 0, 3, 0, 1, 0, 0, 8, 0 },
                        { 9, 0, 0, 8, 6, 3, 0, 0, 5 },
                        { 0, 5, 0, 0, 9, 0, 6, 0, 0 },
                        { 1, 3, 0, 0, 0, 0, 2, 5, 0 },
                        { 0, 0, 0, 0, 0, 0, 0, 7, 4 },
                        { 0, 0, 5, 2, 0, 6, 3, 0, 0 } };


    if (SolveSudoku(grid, 0, 0))
        print(grid);
    else
        cout << "No solution exists\n";

    return 0;
}
```

Output :-

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 14> cd "d:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 14\" ; if ($?) { g++ Sudoku_Solver_Bit_Masks.cpp -o Sudoku_Solver_Bit_Masks } ; if ($?) { .\Sudoku_Solver_Bit_Masks }
3 1 6 5 7 8 4 9 2
5 2 9 1 3 4 7 6 8
4 8 7 6 2 9 5 3 1
2 6 3 4 1 5 9 8 7
9 7 4 8 6 3 1 2 5
8 5 1 7 9 2 6 4 3
1 3 8 9 4 7 2 5 6
6 9 2 3 5 1 8 7 4
7 4 5 2 8 6 3 1 9
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 14> |
```

Space Complexity:- _____
Justification: -

 Marwadi University	Marwari University Faculty of Technology Department of Information and Communication Technology	
Subject: Design and Analysis of Algorithms (01CT0512)	Aim: Implementing the Backtracking, Branch and Bound algorithm for Sudoku Solution	
Experiment No: 14	Date:	Enrollment No: 92200133030

Time Complexity:

Best Case Time Complexity: _____

Justification: -

Worst Case Time Complexity:- _____

Justification: -

Conclusion:-
