**Part B**

**Solution for B.1.1 Detailed Explanation of Approach**

We use Backtracking Algorithm to Solve the Problem. In backtracking algorithms, we try to build a solution one step at a time. If at some step it becomes clear that the current path that we are on cannot lead to a solution we go back to the previous step (backtrack) and choose a different path. Briefly, once we exhaust all our options at a certain step we go back. In Short A backtracking algorithm is a problem-solving algorithm that uses a brute force approach for finding the desired output.

1. Like all other Backtracking problems, we solve Sudoku by one-by-one assigning numbers to empty cells.
2. Before assigning a number, we need to confirm that the same number is not present in current row, current column and current 3X3 subgrid.
3. If number is not present in respective row, column or subgrid, we can assign the number, and recursively check if this assignment leads to a solution or not. If the assignment doesn’t lead to a solution, then we try next number for current empty cell. And if none of number (1 to 9) lead to solution, we return false.

**Solution for B.1.2 Algorithm**

**Algorithm**

**Step 1**: Start

**Step 2 :** Define a method called isPresentInCol(), this will take call and num.

**Step 3:**for each row r in the grid, do

if grid[r, col] = num, then return true.

**Step 4:**return false otherwise

**Step 5:**Define a method called isPresentInRow(), this will take row and num.

**Step 6:**for each column c in the grid, do.

if grid[row, c] = num, then return true

**Step 7:**return false otherwise

**Step 8:**Define a method called isPresentInBox() this will take boxStartRow, boxStartCol, num.

**Step 9:**for each row r in boxStartRow to next 3 rows, do

for each col r in boxStartCol to next 3 columns, do

if grid[r, c] = num, then return true

**Step 10:**return false otherwise.

**Step 11:**Define a method called findEmptyPlace(), this will take row and col.

**Step 12:**for each row r in the grid, do.

for each column c in the grid, do.

if grid[r, c] = 0, then return true.

**Step 13:**return false.

**Step 14:**Define a method called isValidPlace(), this will take row, col, num.

**Step 15:**if isPresentInRow(row, num) and isPresentInCol(col, num) and isPresntInBox(row – row mod 3, col – col mod 3, num) all are false, then return true.

**Step 16:**Define a method called solveSudoku(), this will take the grid.

**Step 17:**if no place in the grid is empty, then return true.

**Step 18:**for number 1 to 9, do.

if isValidPlace(row, col, number), then.

grid[row, col] := number.

if solveSudoku = true, then return true.

grid[row, col] := 0.

**Step 19:**return false.

**Step 20 :** Stop