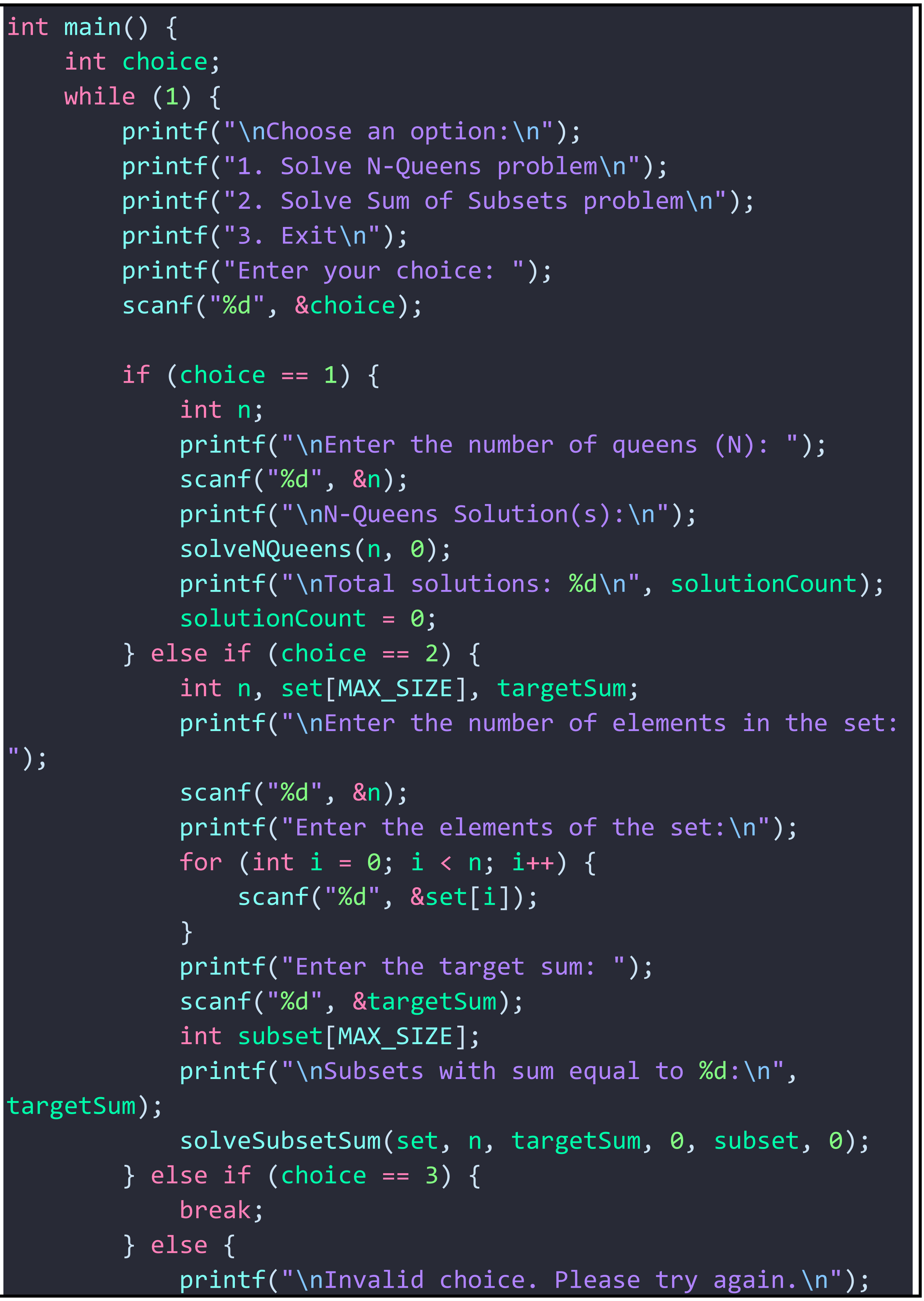
BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Empowered Autonomous Institute Affiliated to Mumbai University)
Department Of Computer Engineering

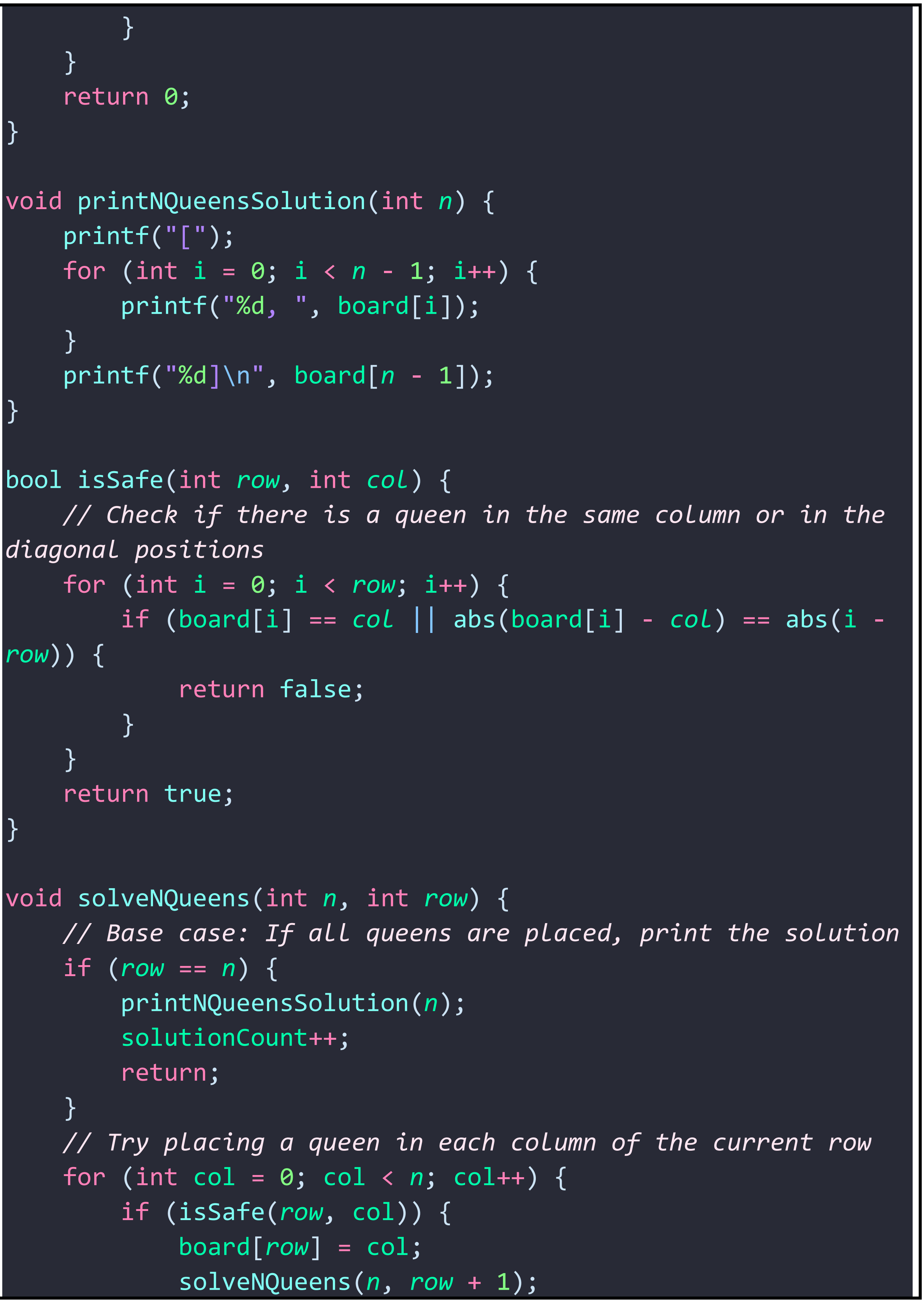
Name	Manish Shashikant Jadhav
UID	2023301005
Subject	Design and Analysis of Algorithms (DAA)
Experiment No.	7
Aim	To implement Backtracking (N-Queen's problem and sum of subsets).

Code:

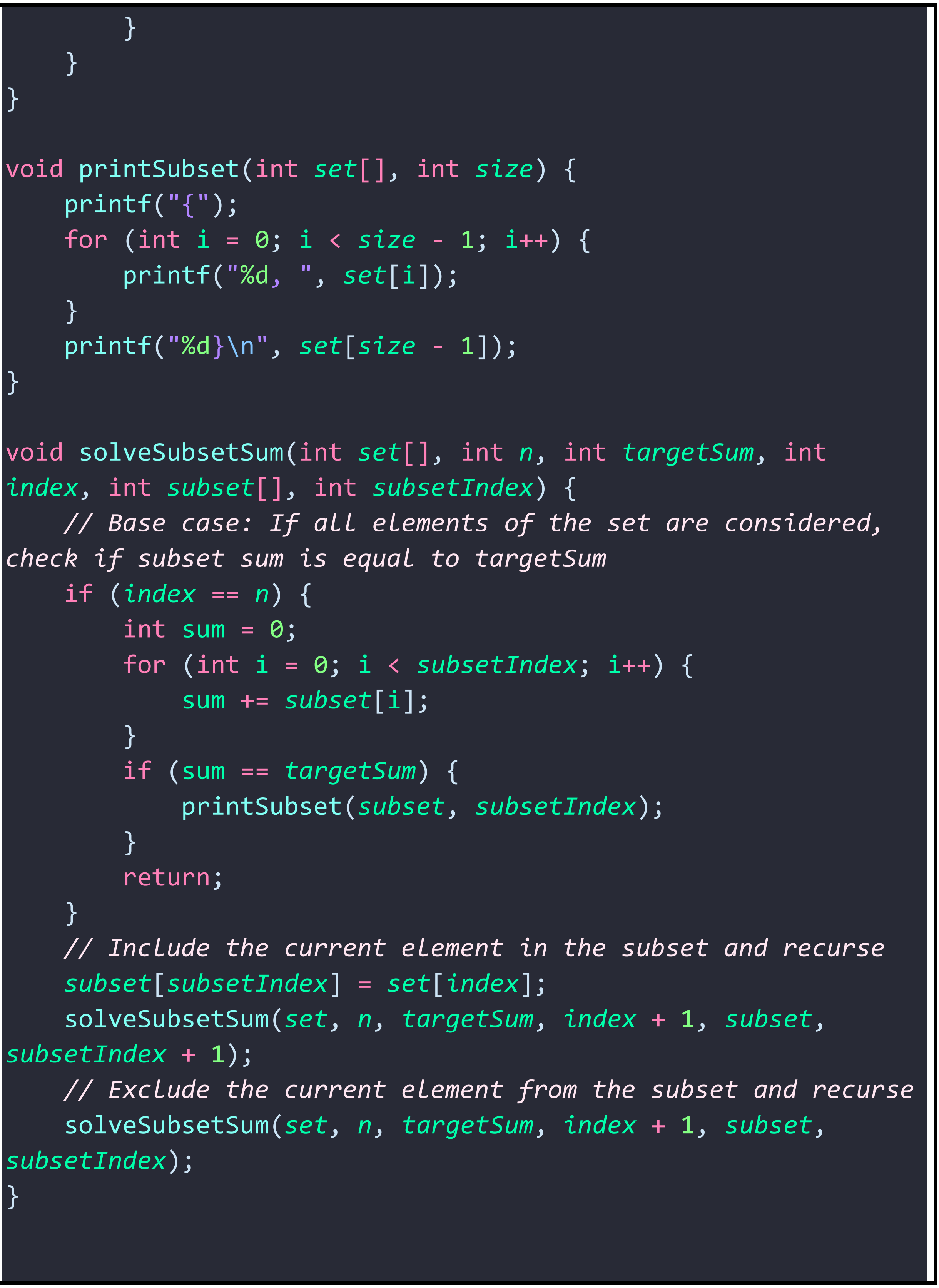
```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define MAX_SIZE 20

int board[MAX_SIZE]; // Array to store the positions of queens
in N-Queens problem
int solutionCount = 0; // Counter to keep track of the number
of solutions found

// Function to print a solution to the N-Queens problem
void printNQueensSolution(int n);

// Function to check if placing a queen at position (row, col)
is safe
bool isSafe(int row, int col);

// Recursive function to solve the N-Queens problem
void solveNQueens(int n, int row);

// Function to print a subset of a set
void printSubset(int set[], int size);

// Recursive function to solve the Sum of Subsets problem
void solveSubsetSum(int set[], int n, int targetSum, int
index, int subset[], int subsetIndex);
```



```
int main() {
    int choice;
    while (1) {
        printf("\nChoose an option:\n");
        printf("1. Solve N-Queens problem\n");
        printf("2. Solve Sum of Subsets problem\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        if (choice == 1) {
            int n;
            printf("\nEnter the number of queens (N): ");
            scanf("%d", &n);
            printf("\nN-Queens Solution(s):\n");
            solveNQueens(n, 0);
            printf("\nTotal solutions: %d\n", solutionCount);
            solutionCount = 0;
        } else if (choice == 2) {
            int n, set[MAX_SIZE], targetSum;
            printf("\nEnter the number of elements in the set:
");

            scanf("%d", &n);
            printf("Enter the elements of the set:\n");
            for (int i = 0; i < n; i++) {
                scanf("%d", &set[i]);
            }
            printf("Enter the target sum: ");
            scanf("%d", &targetSum);
            int subset[MAX_SIZE];
            printf("\nSubsets with sum equal to %d:\n",
targetSum);
            solveSubsetSum(set, n, targetSum, 0, subset, 0);
        } else if (choice == 3) {
            break;
        } else {
            printf("\nInvalid choice. Please try again.\n");
```

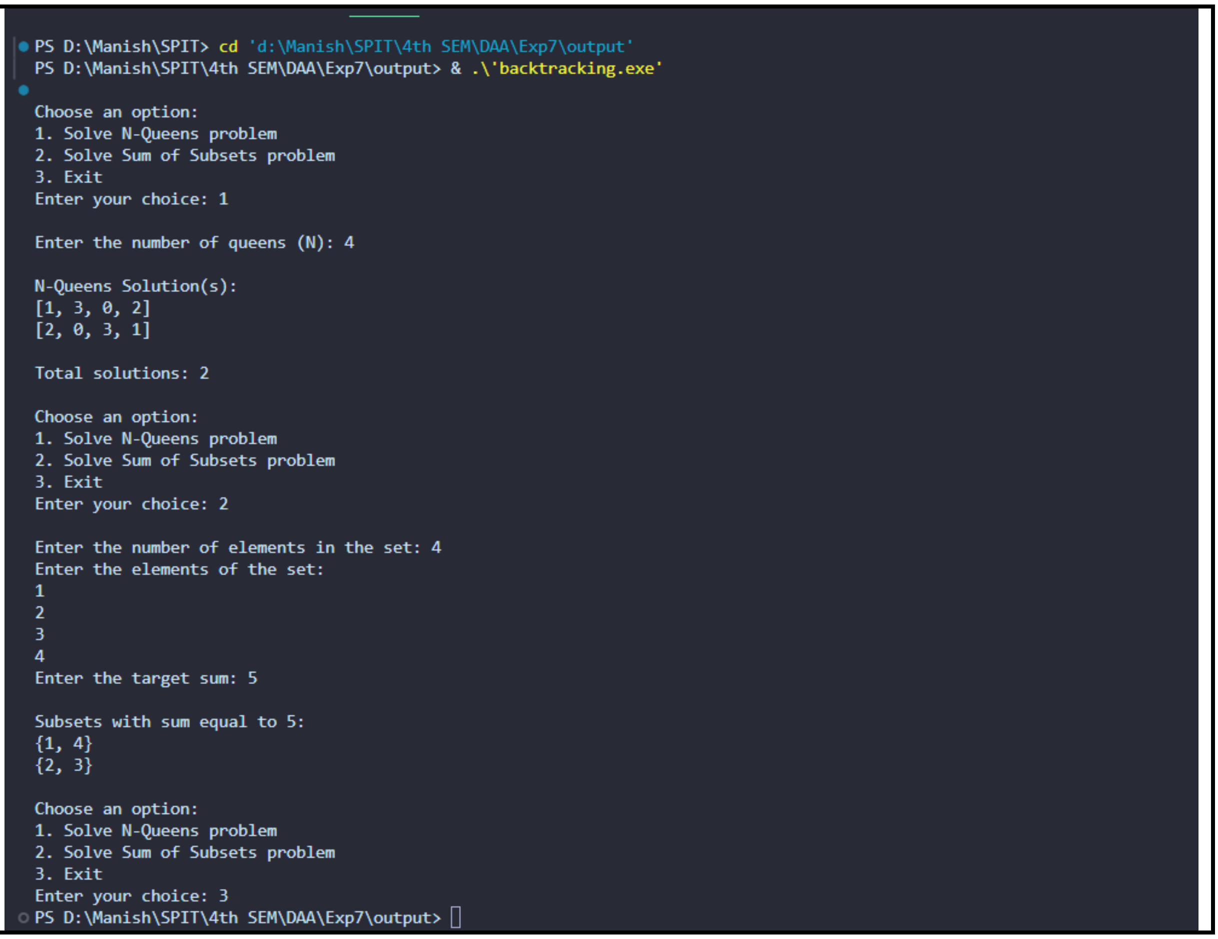


```
        }
    }
    return 0;
}

void printNQueensSolution(int n) {
    printf("[");
    for (int i = 0; i < n - 1; i++) {
        printf("%d, ", board[i]);
    }
    printf("%d]\n", board[n - 1]);
}

bool isSafe(int row, int col) {
    // Check if there is a queen in the same column or in the
diagonal positions
    for (int i = 0; i < row; i++) {
        if (board[i] == col || abs(board[i] - col) == abs(i -
row)) {
            return false;
        }
    }
    return true;
}

void solveNQueens(int n, int row) {
    // Base case: If all queens are placed, print the solution
    if (row == n) {
        printNQueensSolution(n);
        solutionCount++;
        return;
    }
    // Try placing a queen in each column of the current row
    for (int col = 0; col < n; col++) {
        if (isSafe(row, col)) {
            board[row] = col;
            solveNQueens(n, row + 1);
```



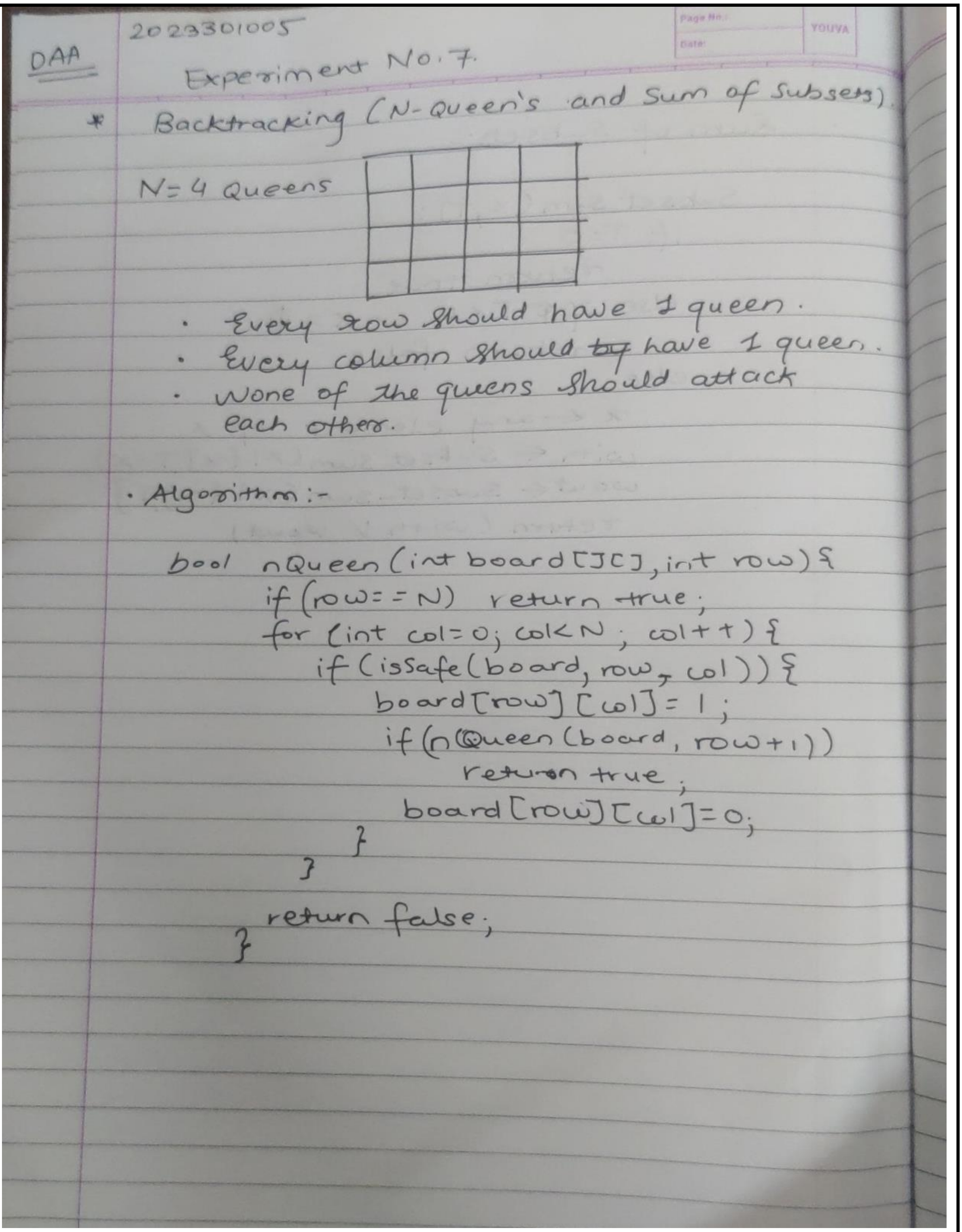
```
        }
    }
}

void printSubset(int set[], int size) {
    printf("{");
    for (int i = 0; i < size - 1; i++) {
        printf("%d, ", set[i]);
    }
    printf("%d}\n", set[size - 1]);
}

void solveSubsetSum(int set[], int n, int targetSum, int
index, int subset[], int subsetIndex) {
    // Base case: If all elements of the set are considered,
check if subset sum is equal to targetSum
    if (index == n) {
        int sum = 0;
        for (int i = 0; i < subsetIndex; i++) {
            sum += subset[i];
        }
        if (sum == targetSum) {
            printSubset(subset, subsetIndex);
        }
        return;
    }
    // Include the current element in the subset and recurse
    subset[subsetIndex] = set[index];
    solveSubsetSum(set, n, targetSum, index + 1, subset,
subsetIndex + 1);
    // Exclude the current element from the subset and recurse
    solveSubsetSum(set, n, targetSum, index + 1, subset,
subsetIndex);
}
```

Output



```
PS D:\Manish\SPIT> cd 'd:\Manish\SPIT\4th SEM\DAA\Exp7\output'
PS D:\Manish\SPIT\4th SEM\DAA\Exp7\output> & .\'backtracking.exe'

Choose an option:
1. Solve N-Queens problem
2. Solve Sum of Subsets problem
3. Exit
Enter your choice: 1

Enter the number of queens (N): 4

N-Queens Solution(s):
[1, 3, 0, 2]
[2, 0, 3, 1]

Total solutions: 2

Choose an option:
1. Solve N-Queens problem
2. Solve Sum of Subsets problem
3. Exit
Enter your choice: 2

Enter the number of elements in the set: 4
Enter the elements of the set:
1
2
3
4
Enter the target sum: 5

Subsets with sum equal to 5:
{1, 4}
{2, 3}

Choose an option:
1. Solve N-Queens problem
2. Solve Sum of Subsets problem
3. Exit
Enter your choice: 3
PS D:\Manish\SPIT\4th SEM\DAA\Exp7\output>
```

Pseudo Code



2023301005

DAA

Experiment No. 7.

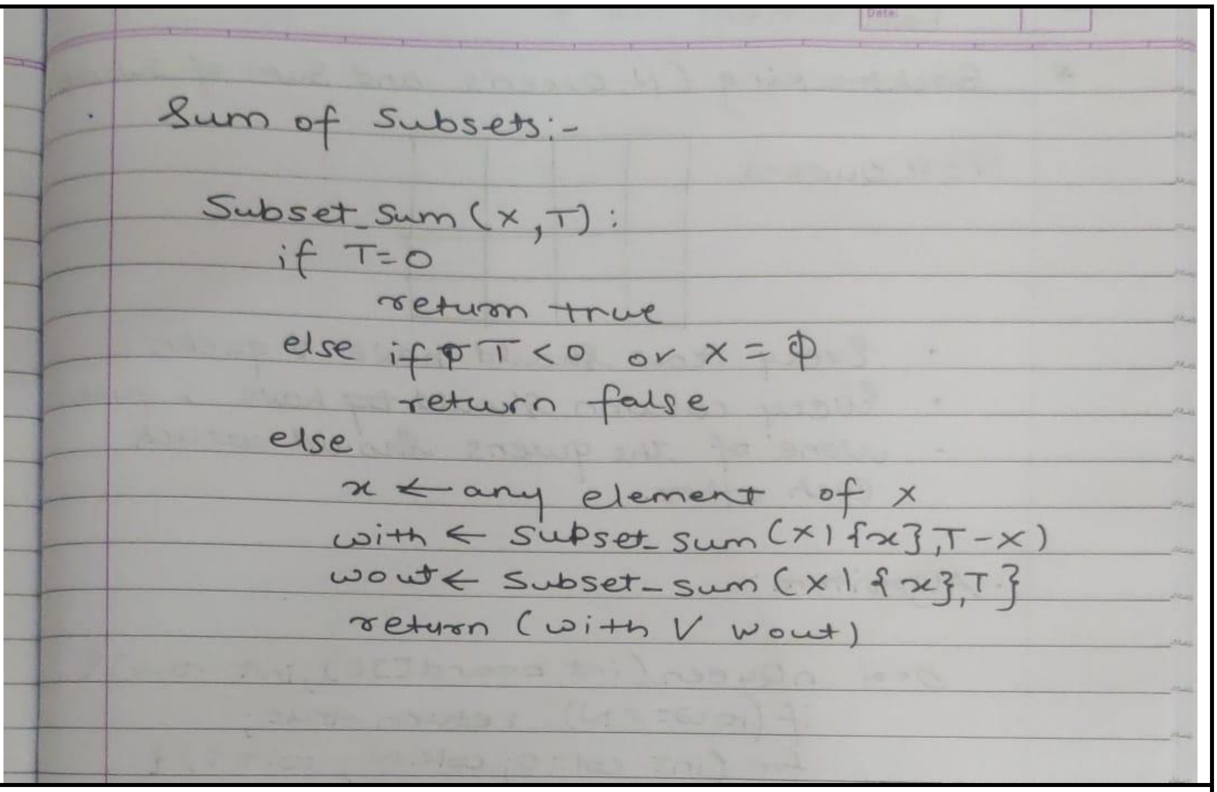
* Backtracking (N-Queen's and Sum of Subsets).

N=4 Queens

- Every row should have 1 queen.
- Every column should ~~by~~ have 1 queen.
- None of the queens should attack each other.

- Algorithm:-

```
bool nQueen(int board[][], int row){
    if(row==N) return true;
    for(int col=0; col<N; col++){
        if(isSafe(board, row, col)){
            board[row][col]=1;
            if(nQueen(board, row+1))
                return true;
            board[row][col]=0;
        }
    }
    return false;
}
```

	
	Sum of Subsets:-
	Subset_Sum(X,T):
	  if T=0
	    return true
	  else if φ T < 0 or X = Φ
	    return false
	  else
	    x ← any element of X
	    with ← Subset_sum(X \ {x}, T − x)
	    wout ← Subset_sum(X \ {x}, T}
	    return (with ∨ wout)

Conclusion	Hence, by completing this experiment I came to know about implementation of Backtracking (N-Queen's problem and sum of subsets).