```
#include <iostream>
#include <vector>
using namespace std;

int totalBacktracking = 0;
int totalBranchAndBound = 0;

// Print 2D board
void printBoard(vector<vector<char>>& board) {
  for (const auto& row : board) {
    for (char cell : row)
      cout << cell << " ";
    cout << endl;
  }
  cout << endl;
}

// -------- BACKTRACKING (Row-wise) --------
bool isSafeBacktrack(vector<vector<char>>& board, int row, int col, int n) {
  // Check column
  for (int i = 0; i < row; i++)
    if (board[i][col] == 'Q') return false;

  // Check upper-left diagonal
  for (int i = row - 1, j = col - 1; i >= 0 && j >= 0; i--, j--)
    if (board[i][j] == 'Q') return false;

  // Check upper-right diagonal
  for (int i = row - 1, j = col + 1; i >= 0 && j < n; i--, j++)
    if (board[i][j] == 'Q') return false;

  return true;
}

void solveBacktracking(vector<vector<char>>& board, int row, int n) {
  if(row == n) {
    totalBacktracking++;
    printBoard(board);
    return;
  }

  for(int col = 0; col < n; col++) {
    if(isSafeBacktrack(board, row, col, n)) {
      board[row][col] = 'Q';
      solveBacktracking(board, row + 1, n);
      board[row][col] = '.'; // backtrack
    }
```

```
    }
}

// -------- BRANCH AND BOUND (Row-wise) --------
void solveBranchAndBound(vector<vector<char>>& board, vector<bool>& cols,
vector<bool>& diag1, vector<bool>& diag2, int row, int n) {
    if(row == n) {
        totalBranchAndBound++;
        printBoard(board);
        return;
    }

    for (int col = 0; col < n; col++) {
        if (!cols[col] && !diag1[row + col] && !diag2[row - col + n - 1]) {
            board[row][col] = 'Q';
            cols[col] = diag1[row + col] = diag2[row - col + n - 1] = true;
            solveBranchAndBound(board, cols, diag1, diag2, row + 1, n);
            board[row][col] = '.'; // backtrack
            cols[col] = diag1[row + col] = diag2[row - col + n - 1] = false;
        }
    }
}

// -------- MAIN FUNCTION --------
int main() {
    int n;
    cout << "Enter value of N for N-Queens: ";
    cin >> n;

    vector<vector<char>> board(n, vector<char>(n, '.'));

    cout << "\n--- Solving using Backtracking (Row-wise) ---\n";
    solveBacktracking(board, 0, n);
    cout << "Total Solutions (Backtracking): " << totalBacktracking << "\n";

    board.assign(n, vector<char>(n, '.'));
    vector<bool> cols(n, false), diag1(2 * n - 1, false), diag2(2 * n - 1, false);

    cout << "\n--- Solving using Branch and Bound (Row-wise) ---\n";
    solveBranchAndBound(board, cols, diag1, diag2, 0, n);
    cout << "Total Solutions (Branch and Bound): " << totalBranchAndBound << "\n";

    return 0;
}
```