**EXPERIMENT:-**

Q1) Implement a backtracking algorithm for solving N Queen problem. Compute all possible solution for N Queen and also compute the number of backtracks. Perform the

experiment from N = 2 to 9.

**Pseudo Code(N Queen Problem)**

IS\_ATTACK(a, b, board, N)

for c in 1 to a-1

if board[c][b]==1

return TRUE

c = a-1

l = b+1

while c>=1 and l<=N

if board[c][l] == 1

return TRUE

c=c+1

l=l+1

c = a-1

l = b-1

while c>=1 and l>=1

if board[c][l] == 1

return TRUE

c=c-1

l=l-1

return FALSE

N-QUEEN(row, n, N, board)

if n==0

return TRUE

for b in 1 to N

if !IS-ATTACK(row, b, board, N)

board[row][b] = 1

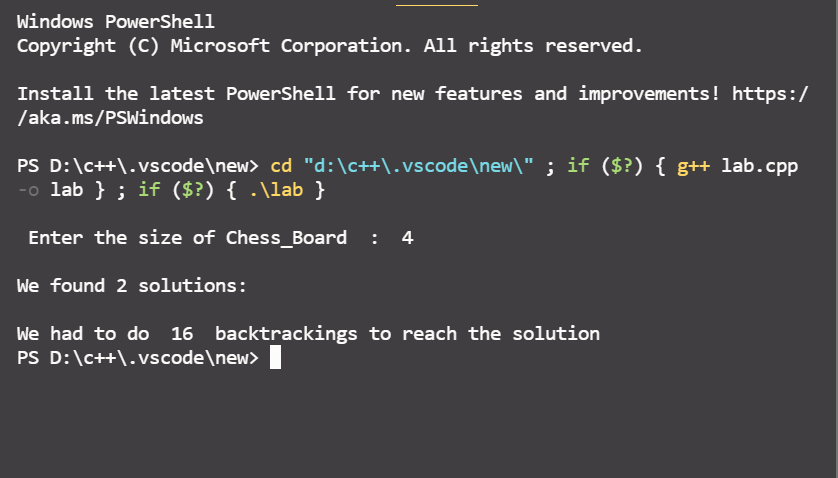
if N-QUEEN(row+1, n-1, N, board)

return TRUE

board[row][b] = 0

return FALSE

**Output**



Q2) Given a set of candidate numbers (candidates) (without duplicates) and a target number

(target), find all unique combinations in candidates where the candidate numbers sums

to target. eg. W : [5, 10, 12, 13, 15, 18] and target = 30.

**Pseudo Code( Combination sum)**

vector<vector<int>>(vector<int>nums, int target) {

vector<vector<int>> result

sort(nums)

vector<int> curr

backtrack(result, curr, nums, target, 0)

return result

}

void backtrack(vector<vector<int>> res, vector<int> candidate, vector<int>nums, int target, int start) {

if(target < 0)

return

else

if(target == 0)

res.push\_back(candidate)

else{

for(int i = start; i < nums.size(); i++) {

candidate.push\_back(nums[i])

backtrack(list, candidate, nums, target - nums[i], i)

candidate.pop\_back(candidate.size() - 1)

}

}

}

**Output**

