**RECURSION:**

* Recursion is a fundamental in computer science
* It involves a function repeatedly calling itself until a base case is reached
* Recursion is used to solve problems by breaking them down into smaller , similar to sub-problems.
* It can significantly reduce code length and complexity compared to iteration.

**Q.WHAT IS RECURSION ?**

**Ans:** Recursion is a programming technique in computer science where a function solves a problem by breaking it down into smaller or sub- problem and calling itself to solve problem.

**BACKTRACKING:**

* Backtracking is technique for finding solution to problems through recursive exploration.
* It involves undoing recursive changes if certain conditions are not met and discarding less optimal solutions.
* Backtracking is more efficient than brute force ( Check all the possibility) in solving complex problems, although it can have exponential time complexity.
* It excels in solving challenging problems like the N-Queens problem and the Traveling Salesman Problem (TSP).

**Q. What is backtracking ?**

Ans: Backtracking is a problem-solving technique where we explore different path to find solutions to a problem. It’s like trying to solve a maze. You start by taking a path, and if it leads to a dead end, you backtrack( go back where you made the last choice ) and try a different path.

How’s how it works:

1. You break the problem into smaller parts or sub-problems.
2. You try to solve each sub-problem step by step.
3. If a sub-problem doesn’t work out or doesn’t meet the condition you need, you backtrack ( go back) to the previous step and try a different approach.
4. You keep doing this until you find a solution that satisfies all the problem’s requirements.

**N-Queen’s Problem:**

1. Start from the leftmost column of the chessboard.
2. Place queens in different rows, one by one.
3. If a safe row is found for the queen in the current column, place the queen.

PROGRAM………..

def is\_safe(board, row, col):

for i in range(row):

if board[i][col] == 1:

return False

for i, j in zip(range(row, -1, -1), range(col, -1, -1)):

if board[i][j] == 1:

return False

for i, j in zip(range(row, -1, -1), range(col, len(board))):

if board[i][j] == 1:

return False

return True

def solve\_n\_queens(board, row):

if row == len(board):

return True

for col in range(len(board)):

if is\_safe(board, row, col):

board[row][col] = 1

if solve\_n\_queens(board, row + 1):

return True

board[row][col] = 0

return False

def print\_solution(board):

for row in board:

print(' | '.join(['Q' if x == 1 else '\_' for x in row]))

def n\_queens\_solver(n):

n = int(input("Enter your board size: "))

board = [[0 for \_ in range(n)] for \_ in range(n)]

if solve\_n\_queens(board, 0):

print\_solution(board)

else:

print("No solution exists.")

n\_queens\_solver(4)

**Q. Find the target valu from the given array with the help of combination of multiple value ?**

from itertools import combinations

arr = [6, 8, 9, 5, 4, 3, 26, 2]

target = int(input("Enter target sum: "))

combinations\_list = []

for i in range(0, len(arr)):

for combo in combinations(arr, i):

if sum(combo) == target:

combinations\_list.append(combo)

print("Combinations that sum to", target, ":")

for combo in combinations\_list:

print(combo)

**2ND METHOD:**

def find\_combinations(n, target):

total\_sum = sum(n)

if target > total\_sum:

return "Your target value is out of bounds."

def combinations(n, target, curr\_sum, sum=[]):

if curr\_sum == target:

result\_list.append(sum)

return

if not n or curr\_sum > target:

return

combinations(n[1:], target, curr\_sum, sum)

combinations(n[1:], target, curr\_sum + n[0], sum + [n[0]])

result\_list = []

combinations(n, target, 0)

return result\_list

n = input("Enter elements of the array: ").split()

n = [int(x) for x in n]

target = int(input("Enter the sum value you want to find: "))

if target > sum(n):

print("Your target value is out of bounds.")

else:

combinations\_list = find\_combinations(n, target)

print("Combinations that sum to", target, ":")

for combo in combinations\_list:

**print(combo)**