**Q // JHONSON TROTTER**

#include <stdio.h>

#include <stdbool.h>

#define MAX\_SIZE 10

// Function to print the current permutation

void printPermutation(int arr[], int dir[], int n) {

for (int i = 0; i < n; i++) {

printf("%d ", arr[i]);

if (dir[i] == -1)

printf("< ");

else

printf("> ");

}

printf("\n");

}

// Find the largest mobile element in the given permutation

int findLargestMobileElement(int arr[], int dir[], int n) {

int mobileElement = -1;

int mobileIndex = -1;

for (int i = 0; i < n; i++) {

if (dir[i] == -1 && i > 0 && arr[i] > arr[i - 1] && arr[i] > mobileElement) {

mobileElement = arr[i];

mobileIndex = i;

}

if (dir[i] == 1 && i < n - 1 && arr[i] > arr[i + 1] && arr[i] > mobileElement) {

mobileElement = arr[i];

mobileIndex = i;

}

}

return mobileIndex;

}

// Function to perform one step of the Johnson-Trotter algorithm

void performOneStep(int arr[], int dir[], int n, int mobileIndex) {

int mobileElement = arr[mobileIndex];

int temp;

// Swap the mobile element with its adjacent element in the direction of the mobile element

if (dir[mobileIndex] == -1) {

temp = arr[mobileIndex - 1];

arr[mobileIndex - 1] = arr[mobileIndex];

arr[mobileIndex] = temp;

temp = dir[mobileIndex - 1];

dir[mobileIndex - 1] = dir[mobileIndex];

dir[mobileIndex] = temp;

} else {

temp = arr[mobileIndex + 1];

arr[mobileIndex + 1] = arr[mobileIndex];

arr[mobileIndex] = temp;

temp = dir[mobileIndex + 1];

dir[mobileIndex + 1] = dir[mobileIndex];

dir[mobileIndex] = temp;

}

// Reverse the direction of all elements greater than the mobile element

for (int i = 0; i < n; i++) {

if (arr[i] > mobileElement)

dir[i] = -dir[i];

}

}

// Function to generate all permutations using the Johnson-Trotter algorithm

void generatePermutations(int n) {

int arr[MAX\_SIZE];

int dir[MAX\_SIZE];

// Initialize the array and direction

for (int i = 0; i < n; i++) {

arr[i] = i + 1;

dir[i] = -1;

}

// Print the first permutation

printPermutation(arr, dir, n);

// Generate other permutations

bool mobileElementExists = true;

while (mobileElementExists) {

int mobileIndex = findLargestMobileElement(arr, dir, n);

if (mobileIndex == -1)

mobileElementExists = false;

else

performOneStep(arr, dir, n, mobileIndex);

printPermutation(arr, dir, n);

}

}

int main() {

int n;

printf("Enter the number of elements (maximum %d): ", MAX\_SIZE);

scanf("%d", &n);

if (n < 1 || n > MAX\_SIZE) {

printf("Invalid number of elements.\n");

return 0;

}

printf("Permutations:\n");

generatePermutations(n);

return 0;

}

