

3. Implement Johnson Trotter algorithm to generate permutations.

Code:

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

#define left_to_right true
#define right_to_left false

int getPosOfMobile(int a[], int n, int mobile) {
    for (int i = 0; i < n; i++) {
        if (a[i] == mobile)
            return i + 1;
    }
    return 0;
}

int getMobile(int a[], bool dir[], int n) {
    int mobile_prev = 0, mobile = 0;
    for (int i = 0; i < n; i++) {
        // direction 0 represents RIGHT TO LEFT.
        if (dir[a[i] - 1] == right_to_left && i != 0) {
            if (a[i] > a[i - 1] && a[i] > mobile_prev) {
                mobile = a[i];
                mobile_prev = mobile;
            }
        }

        // direction 1 represents LEFT TO RIGHT.
        if (dir[a[i] - 1] == left_to_right && i != n - 1) {
            if (a[i] > a[i + 1] && a[i] > mobile_prev) {
                mobile = a[i];
                mobile_prev = mobile;
            }
        }
    }

    if (mobile == 0 && mobile_prev == 0)
        return 0;
    else
        return mobile;
}
```

```

void produceOnePermutation(int a[], bool dir[], int n) {
    int mobile = getMobile(a, dir, n);
    int pos = getPosOfMobile(a, n, mobile);

```

```

    if (dir[a[pos] - 1] == right_to_left) {
        int temp = a[pos - 1];
        a[pos - 1] = a[pos - 2];
        a[pos - 2] = temp;
    } else if (dir[a[pos] - 1] == left_to_right) {
        int temp = a[pos];
        a[pos] = a[pos - 1];
        a[pos - 1] = temp;
    }

```

```

    // changing the directions for elements
    // greater than largest mobile integer.

```

```

    for (int i = 0; i < n; i++) {
        if (a[i] > mobile) {
            if (dir[a[i] - 1] == left_to_right)
                dir[a[i] - 1] = right_to_left;
            else if (dir[a[i] - 1] == right_to_left)
                dir[a[i] - 1] = left_to_right;
        }
    }

```

```

    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");
}

```

```

int fact(int n)
{
    int result=1;
    for(int i=1;i<=n;i++)
    {
        result*=i;
    }
    return result;
}

```

```

void producePermutation(int n) {

```

```

// To store the current permutation
int a[n];

// To store the current directions
bool dir[n];

// Storing the elements from 1 to n and
// printing the first permutation.
for (int i = 0; i < n; i++) {
    a[i] = i + 1;
    printf("%d ", a[i]);
}
printf("\n");

// Initially all directions are set
// to RIGHT TO LEFT i.e. 0.
for (int i = 0; i < n; i++)
    dir[i] = right_to_left;

// For generating permutations in order.
for (int i = 1; i < fact(n); i++)
    produceOnePermutation(a, dir, n);
}

void main()
{
    int n;
    printf("\nEnter the number of objects whose permutations are to be generated: ");
    scanf("%d",&n);
    producePermutation(n);
}

```

Output:

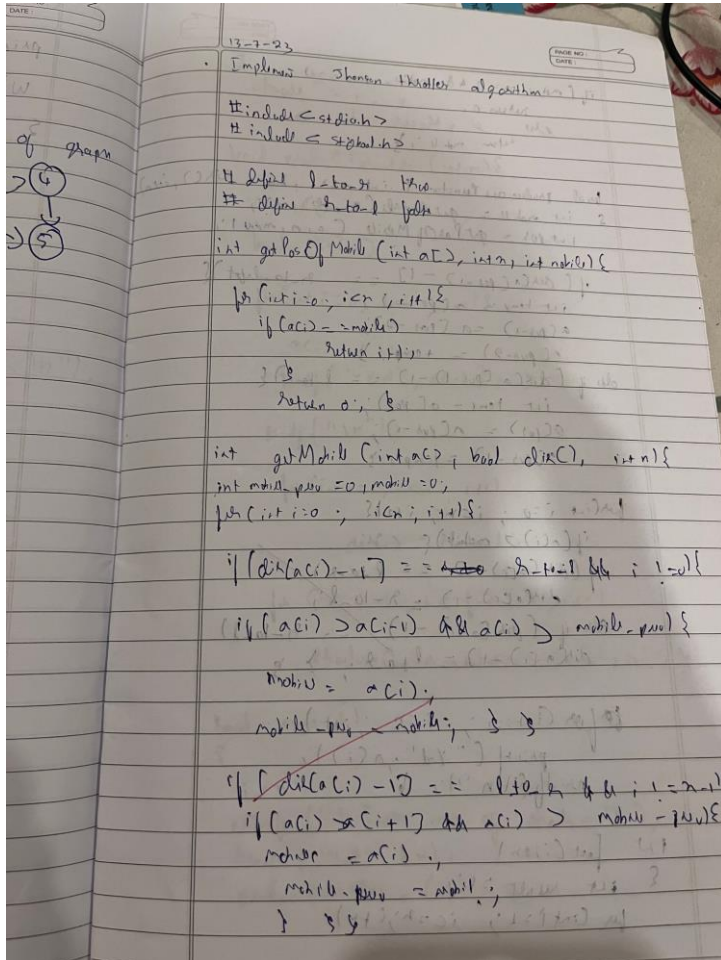
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS D:\VS Code\ADA> cd "d:\VS Code\ADA\" ; if ($?) { gcc JT.c -o JT } ; if ($?) { .\JT }

Enter the number of objects whose permutations are to be generated: 4
1 2 3 4
1 2 4 3
1 4 2 3
4 1 2 3
4 1 3 2
1 4 3 2
1 3 4 2
1 3 2 4
3 1 2 4
3 1 4 2
3 4 1 2
4 3 1 2
4 3 2 1
3 4 2 1
3 2 4 1
3 2 1 4
2 3 1 4
2 3 4 1
2 4 3 1
4 2 3 1
4 2 1 3
2 4 1 3
2 1 4 3
2 1 3 4
PS D:\VS Code\ADA>

```

Observation:



if (mobile == 0 && mobile == 0) {

return 0;

else

return mobile;

void reverseArray (int arr[], int start, int end)

{ int mobile = getPosOfMobile (arr, start, end);

int pos = getPosOfMobile (arr, mobile);

if (arr[pos-1] - 1 == right-left) {

int temp = arr[pos-1];

arr[pos-1] = arr[pos-2];

arr[pos-2] = temp;

else if (arr[pos-1] - 1 == left-right) {

int temp = arr[pos];

arr[pos] = arr[pos-1];

arr[pos-1] = temp;

for (int i = 0; i < arr.length; i++)

if (arr[i] > mobile) {

if (arr[pos] - 1 == right-left) {

arr[pos-1] = arr[pos];

else if (arr[pos] - 1 == left-right) {

arr[pos] = arr[pos-1];

for (int i = 0; i < arr.length; i++)

print (" ", arr[i]);

print ("\n");

int main ()

{ int n = 1;

for (int i = 1; i < n; i++)

```

PAGE NO.
DATE

§
return i;
return x;

void producePermutation(int n) {
    int a[n];
    bool di[n];
    for (int i = 0; i < n; i++) {
        a[i] = i + 1;
        printf("%d ", a[i]);
    }
    printf("\n");
    for (int i = 0; i < n; i++) {
        di[i] = false;
    }
    for (int i = 1; i <= fact(n); i++) {
        producePermutation(a, di, n);
    }
}

int main() {
    int n;
    printf("Enter no of digits: ");
    scanf("%d", &n);
    producePermutation(n);
}

```

Output

Enter the number of objects whose permutations
are to be generated : 3

1 2 3

1 3 2

2 1 3

3 2 1

3 1 2

2 3 1

