Implement Johnson Trotter algorithm to generate permutations

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

int LEFT_TO_RIGHT = 1;
int RIGHT_TO_LEFT = 0;


// Utility functions for
// finding the position
// of largest mobile
// integer in a[].

int searchArr(int a[], int n, int mobile)
{
   for (int i = 0; i < n; i++)

      if (a[i] == mobile)
         return i + 1;

   return 0;
}



// To carry out step 1
// of the algorithm i.e.
// to find the largest
// mobile integer.
int getMobile(int a[], int 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;
}

// Prints a single
// permutation

int printOnePerm(int a[], int dir[], int n)
{
    int mobile = getMobile(a, dir, n);
    int pos = searchArr(a, n, mobile);

    // swapping the elements
    // according to the
    // direction i.e. dir[].
    if (dir[a[pos - 1] - 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] - 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(" ");

    return 0;
}



  // To end the algorithm
    // for efficiency it ends
    // at the factorial of n
    // because number of
```

```
// permutations possible
// is just n!.
 int fact(int n)
{
   int res = 1;

   for (int i = 1; i <= n; i++)
      res = res * i;
   return res;
}

 // This function mainly
// calls printOnePerm()
// one by one to print
// all permutations.
void printPermutation(int n)
{
   // To store current
   // permutation
  int a[20];

   // To store current
   // directions
  int dir[20];

   // storing the elements
   // from 1 to n and
   // printing 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 the order.
        for (int i = 1; i < fact(n); i++)
            printOnePerm(a, dir, n);
    }

int main()
{
    int n;
    printf("enter the number of numbers to permutate\n");
    scanf("%d",&n);
    printPermutation(n);
    return 0;
}
```

```
enter the number of numbers to permutate
4
1234
1243 1423 4123 4132 1432 1342 1324 3124 3142 3412 4312 4321 3421 3241 3214
Process returned 0 (0x0)   execution time : 2.731 s
Press any key to continue.
```