R-4.7

The solution consists of storing the sequence $(1, 2, \ldots, n)$ in an n-element array A and then calling the recursive algorithm shown in the code fragment below, with the call Permute(A, n, n).

This recursive algorithm works as follows. The second argument m indicates that the m elements of A form the active portion of the array, and may be permuted in future calls. The last $n-m$ elements of the array are the inactive elements and do not change. They are kept for purposes of printing. Consider the active subarray $A[0..m-1]$ at some entry to the recursive function. Each element of $A[m-1]$ is swapped into the last position of the array. After the swap, this last element now becomes part of the inactive portion of the array, and the recursive call Permute$(A, m-1, n)$ is then made to compute all the permutations involving the first $m-1$ elements. On return, we return the last element to its original location, and continue the process with the next element. When all elements have had an opportunity to be the last element of the active subarray, we return to the next higher level of the recursive process.

This algorithm does not print the permutations in lexicographical order. Can you see how to modify it so that the permutations are printed in lexicographically increasing order?

Algorithm Permute(F, P):

Input: An array A and two integers m and n, $m \leq n$
Output: Prints all the permutations of the elements of A

```
if m = 1 then
  output A[0..n-1]
else
  for i = 0 to m-1 do
    swap A[i] with A[m-1]
    Permute(A, m-1, n)
    swap A[m-1] with A[i]
```