OUTPUT:

The algorithm will ultimately print every possible permutation of the input List.

Each different permutation of the list is going to be printed with a f1() call, and each element of any single permutation is going to be printed on a new line.

Output Example for Input: N=3, List = [0,1,2]

```
0
1
2

0
2
1

1
0
2

1
2
0

2
1
0
```

2

0

1

MECHANISM

f1():

The f1() function has only 1 cycle that prints each element of the list on a new line, so this function complexity is O(N)

f2():

Each cycle iteration within f2() calls another f2() with a smaller distance between “start” and “end”. This distance is the number of cycle iterations for that specific call.

Each call of f2() runs a loop from current index start to end and do:

1. Swap list[i] and list[start].
2. Find all other possible permutations, from f2(list,start + 1,end).
3. Swap list[start] and list[i].

So for the first call of f2, the cycle will iterate N times, calling f2() N times

For the second call of f2, the cycle will iterate N-1 times, calling f2 another N-1 times, and so on.

With this mechanism, the f2() function is going to be called a number of times equal to:

$$\sum_{i=0}^{n} \frac{n!}{(n-i)!}$$

At the end of every f2() mechanism, an f1() call wil be computed.

This way of finding all possible solution is called backracking.

COMPUTATIONAL COST

Therefore we can say thar the f1() function will be called N! times for a total complexity of O(N*N!), as we established the complexity of f1() to be O(N).

We are still leaving out the computational cost of f2() calls, but as we know that:

$$\sum_{i=0}^{n} \frac{n!}{(n-i)!} < n \cdot n!$$

they won't be taken in account for asymptotic complexity evaluation.

Finally, we can say that the algorithm has a asymptotic computational cost of O(N*N!) which is an extrmely high cost, as the O(N!) time class is the most expensive.

BETTER ALGORITHM

An algorithm with a lower asymptotic computational cost for this task is not possible.

The reason is that there are n! permutations of any given number n, and O(N) time is required to print a permutation this way. Also it is impossible to generate all the permutations faster than O(N!), as they are in fact N!.

Thus, printing each element of all permutations from a given list of length N takes O(N * N!) time.