Permutation Generation using swap from first position (left to right)

Input : [a,b,c]

-- At p1 : we can place all 3 chars at p1

-- At p2 : we can place remaining 2 chars at p2

-- At p3 : we can place last remaining char at p3.

Swap-operation fixes the char at any given position.

Example fixing the position P1 in [a,b,c]

--swap a with a

--swap a with b

--swap a with c

Same strategy need to be called for fixing the position P1 and P2.

Time Complexity :

--Depth of the tree is equal to the size of the input i.e. n.

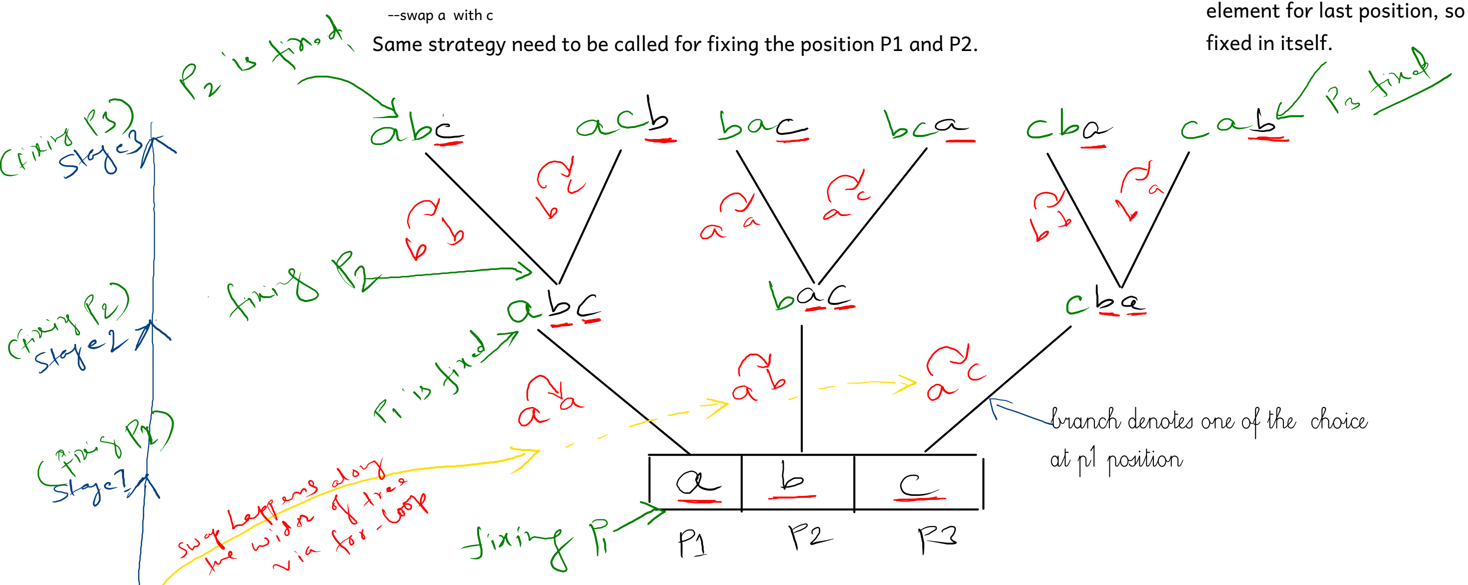
--Thus, to reach each leaf node we need to traverse n steps.

--There are factorial-n leaf nodes (as permuation of n elements is n!)

Time Complexity = O(n*n!)

Space Complexity = O(1)

since we are just left with single element for last position, so it is fixed in itself.



```
for (int i = position; i < input.length; i++) {
    swap(input, position, i);
    permuteArray1(input, position + 1);
    swap(input, position, i);
}
```

Permutation Generation using swap from last position (right to left)

Algorithm is similar to swap from first postion(left to right)

Input : [a,b,c]

-- At p3 : we can place all 3 chars at p3

-- At p2 : we can place remaining 2 chars at p2

-- At p1 : we can place last remaining char at p1.

Swap-operation fixes the char at any given position.

Example fixing the position P3 in [a,b,c]

--swap c with c

--swap c with b

--swap c with a

Same strategy need to be called for fixing the position P1 and P2.