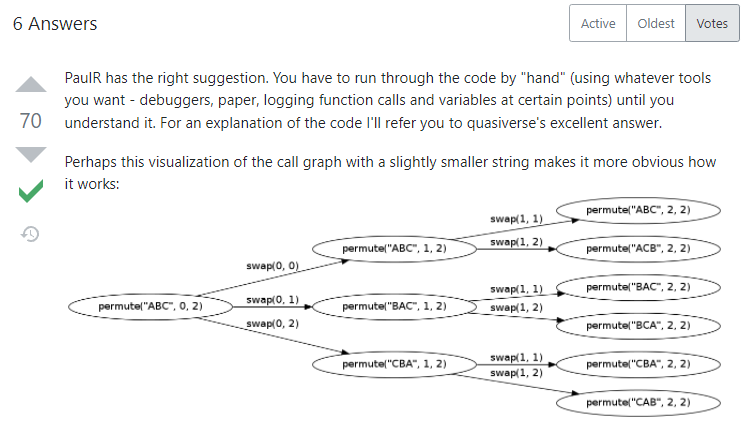
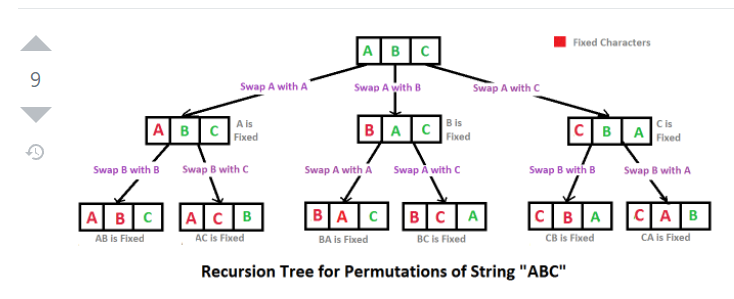
#==============🡺

<https://stackoverflow.com/questions/7537791/understanding-recursion-to-generate-permutations>

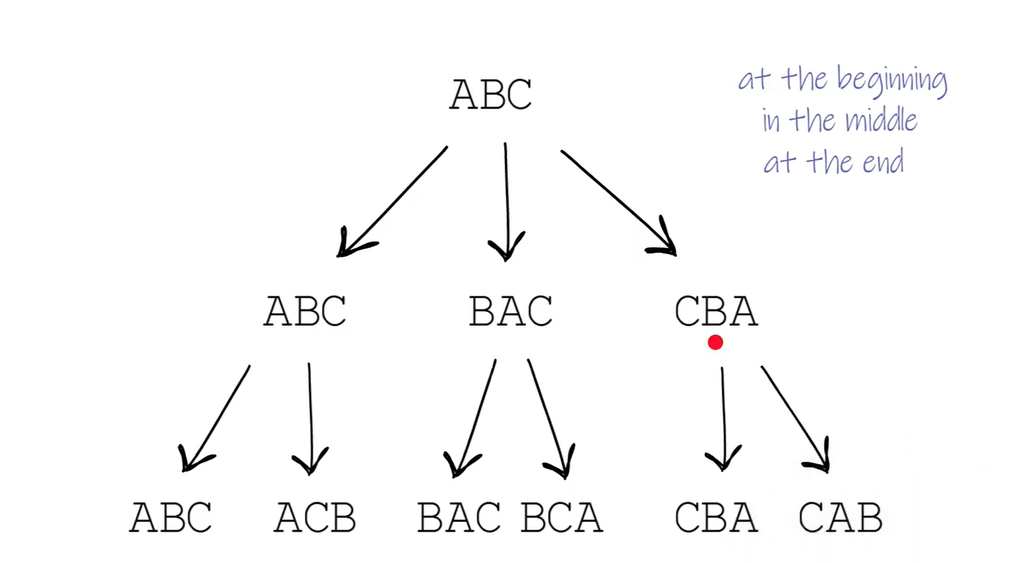


#==============🡺



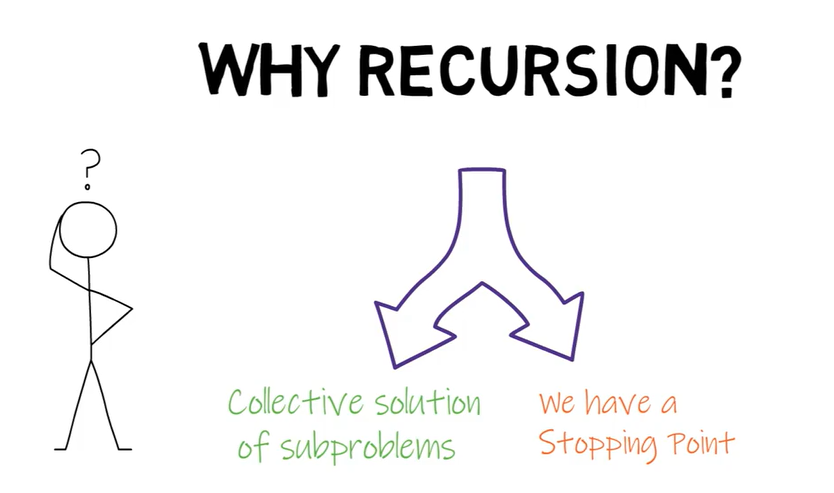
#======================🡺

<https://www.youtube.com/watch?v=TnZHaH9i6-0&ab_channel=DevBox>



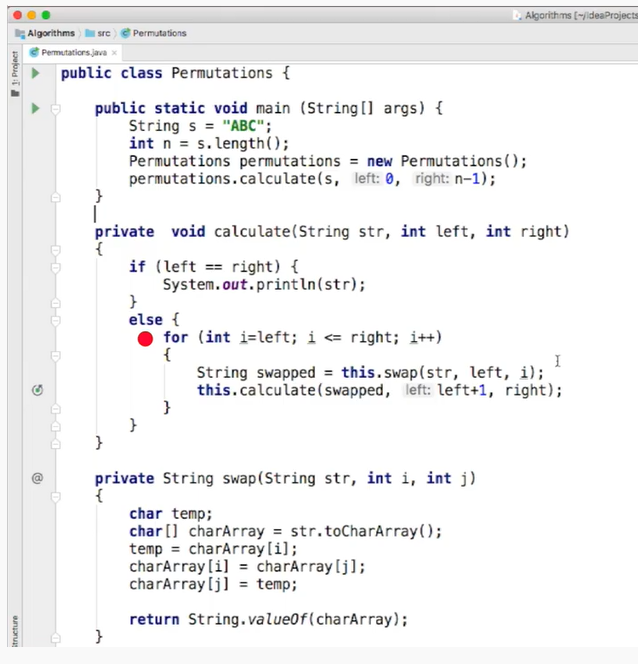
Solution is basically the last level here.

Why is this problem eligible for recursive solution. 🡺 we do same thing everytime, with shifting the character once.



#-======🡺

Lets dive deeper into this solution.

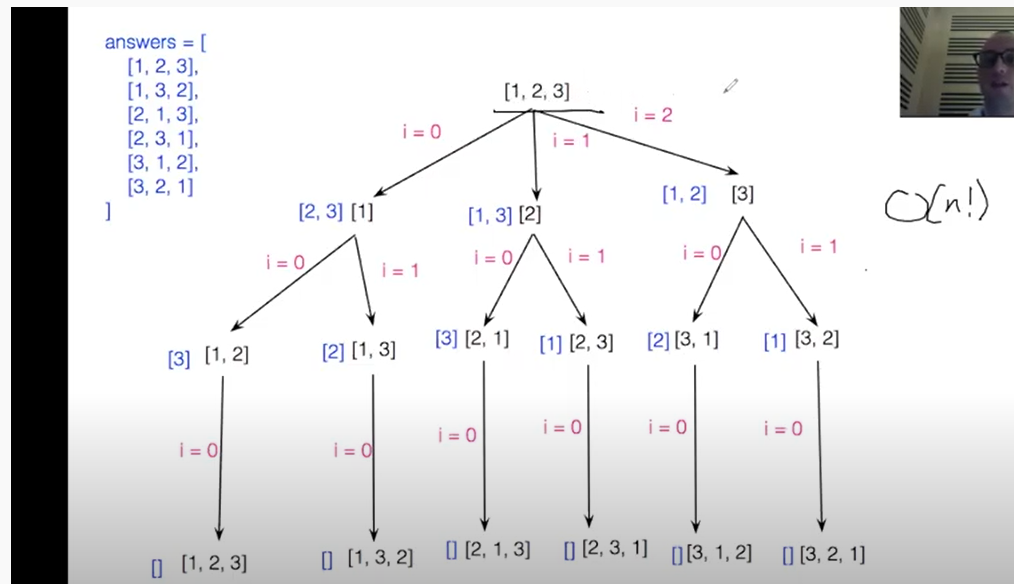


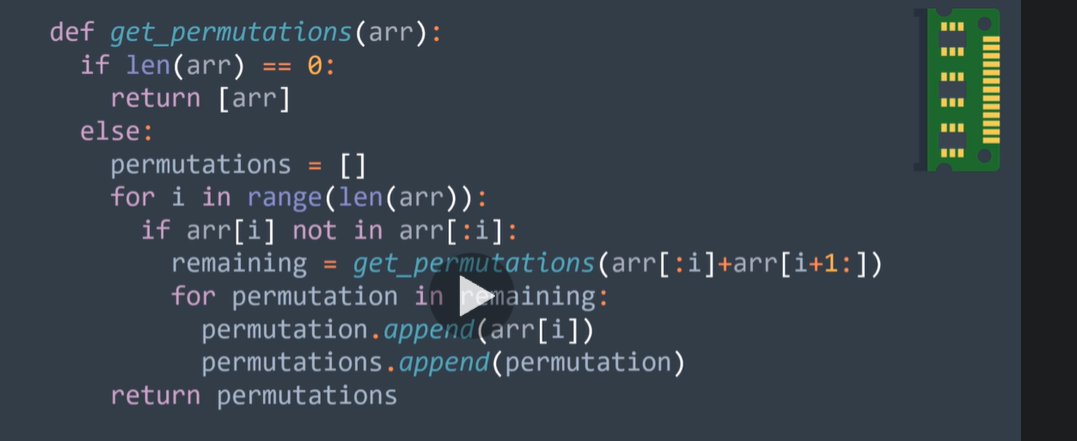
#=======🡺

#=========🡺

<https://www.youtube.com/watch?v=KukNnoN-SoY&ab_channel=TimeComplexityInfinity>

Leetcode 46 - Permutations





Consider remaining as a stack above.

[[]+[3]; []+[2,3]; []+[1,2,3]] where the first element of the stack is the left onne.. and last is the right one.

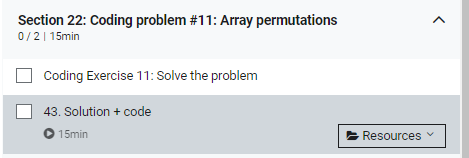
The first element of each of the list components get appended and so on..

#------------------🡪

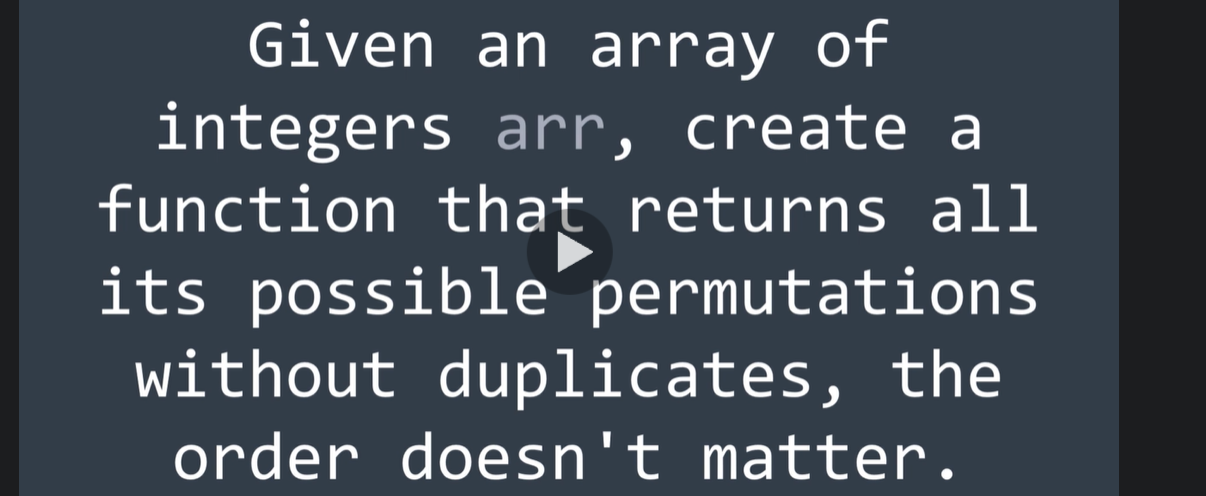
#=======🡺

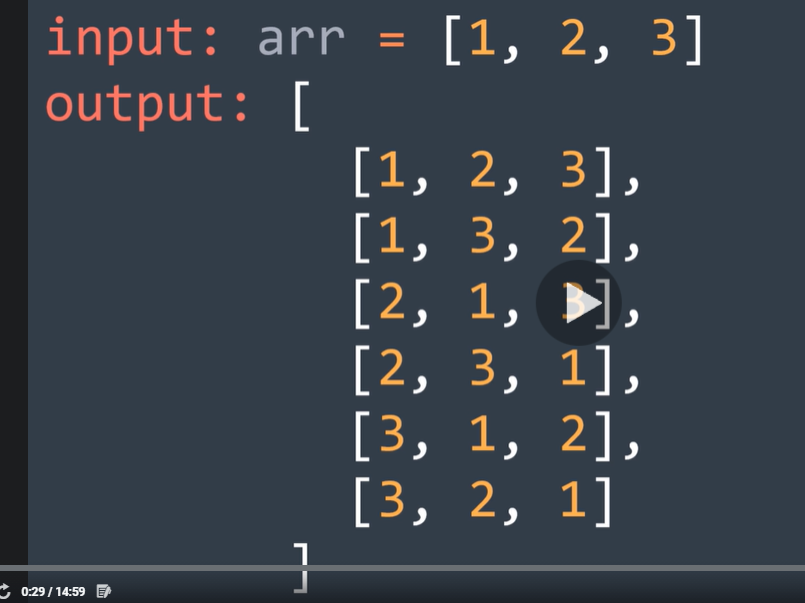
#=========🡺

From Recursion course: Udemy:



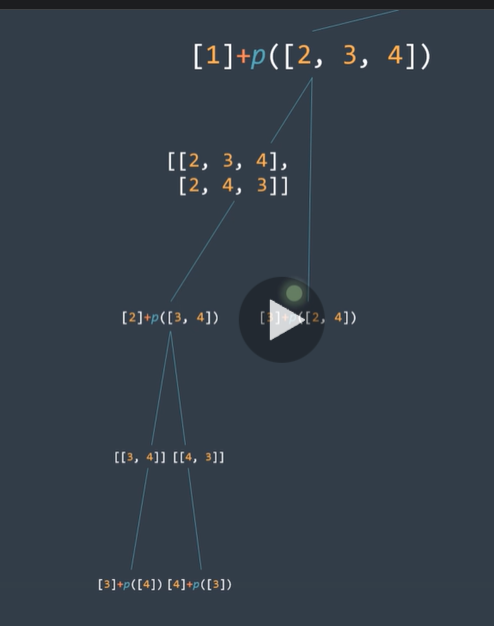


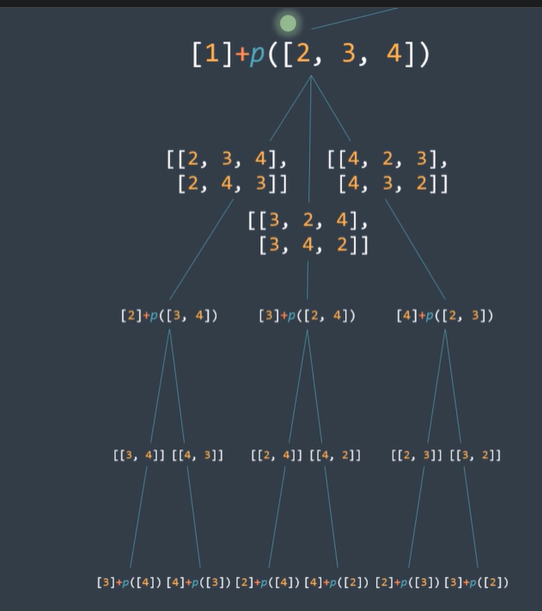




#=-================🡺

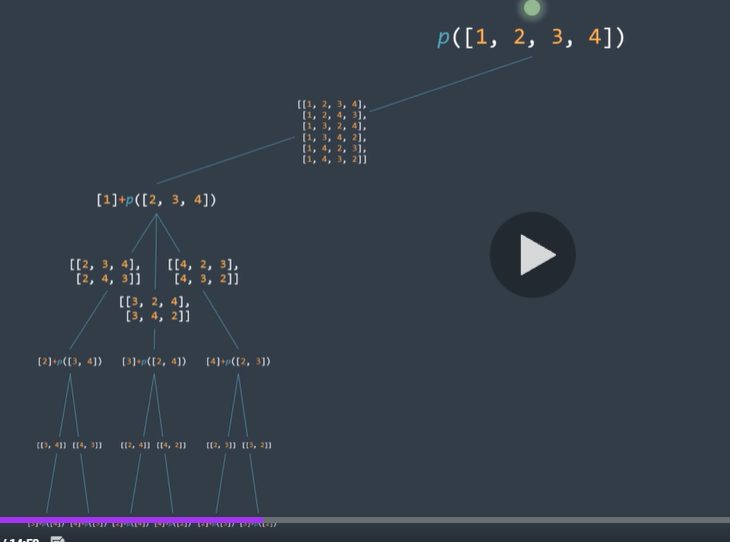
#=-================🡺





#=============🡺

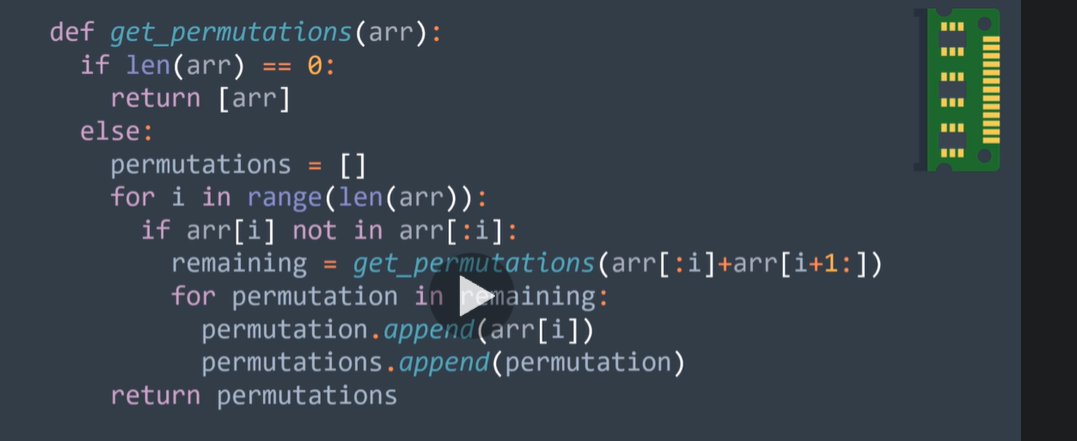
#-==s===============🡺

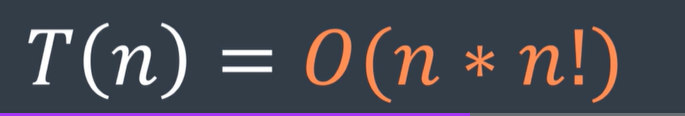


#==================🡺

So complete recursion tree can look like..





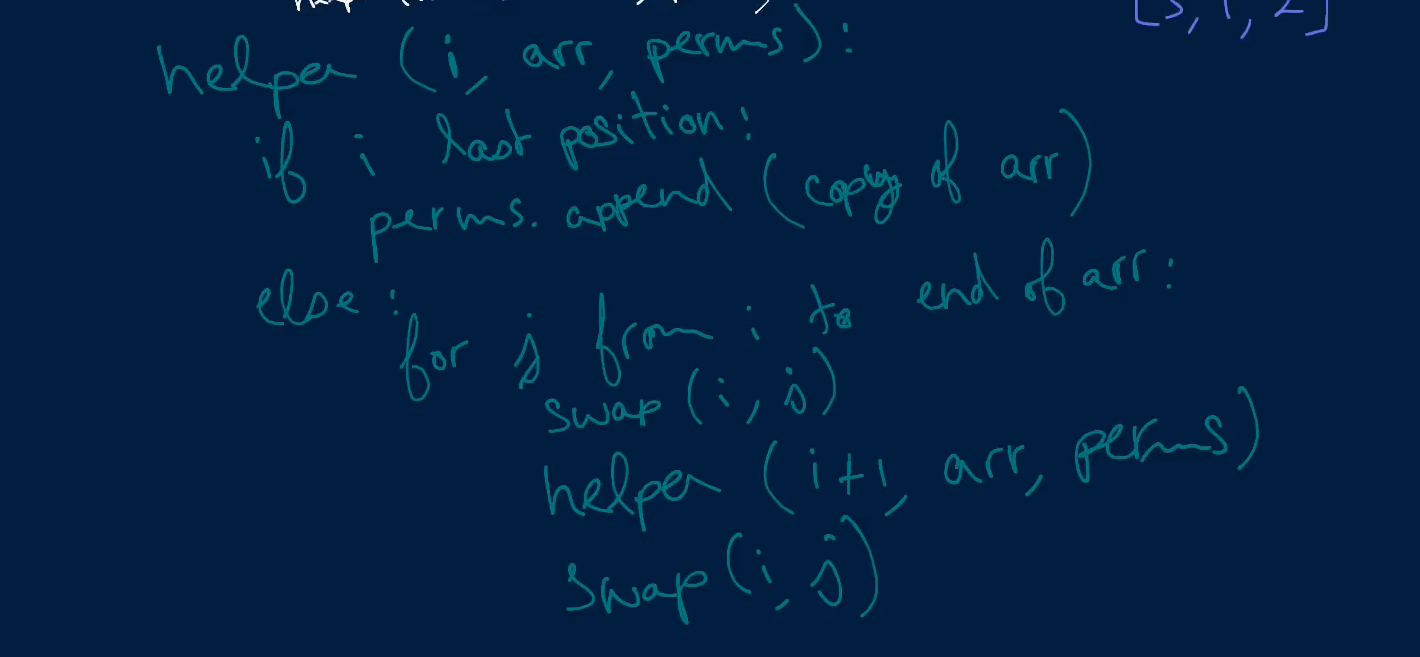


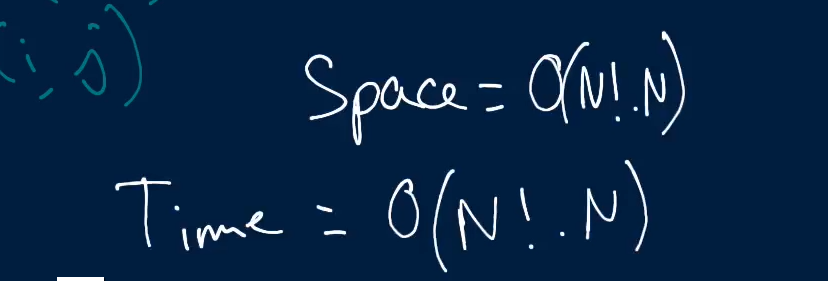
#=🡺==🡺==🡺==🡺==🡺===🡺==🡺

In code: how would this look:

#=🡺=🡺==🡺==🡺

2nd Way: Not very clear : From AlgoExpert.





2nd Way:

The steps of working:

Initially start with [1,2,3].

Add it in permutations list.

Keep first number intact and swap 2 and 3. to get [1,3,2] 🡺 put it in permutation list.

From [1,3,2] go back to [1,2,3] (reswap)

Swap 1 with 2 to get [2,1,3] 🡺 add it in permutations list.

Keep 2 intact and swap 1 and 3 to get [2,3,1] 🡺 add in permutations list.

First all is done with 2 as first number, swap 1 and 3 first to get [2,1,3]..

Swap 2 and 1 to get [1,2,3]//

Then swap 3 and 1 to get [3,2,1]// add it in permutations list.

Swap 2 and 1 to get [3,1,2] 🡺 add it in permutations list.

#===========🡺

First check if I is the final index? By comparing with len(array)..

Then lets append to our permutation, whatever we have.

Otherwise if we are not the end of array..

We start at our current position and iterate through all the numbers. And swap all the numbers with the number at our current position

Then give a call to permutations helper

Reswap

#=======🡺

Roughly written steps.

Confusing way of solving this problem.. but a better way to solve this.

We are going to build all the permutations for the array (say [1,2,3]) in this array itself.

In the first case say array is [1,2,3]//.. 🡺 put thee pointers, 1 on each number.

In the second case, grab 2 and swap it with 3. [1,3,2].. we have covered all the permutations that starts with 1.

Then go back to the first pointer i.e 1.. and swap it with the any other number (say number 2).

So now 2 is at the first position. And leaving it as such we have 2,1,3..

Swap 1 and 3.. now.. to get 2,3,1..

At the end when all permutations with number 2 at first place is done.. swap 1 and 3 again.. to reach to initial stage of 2,1,3.

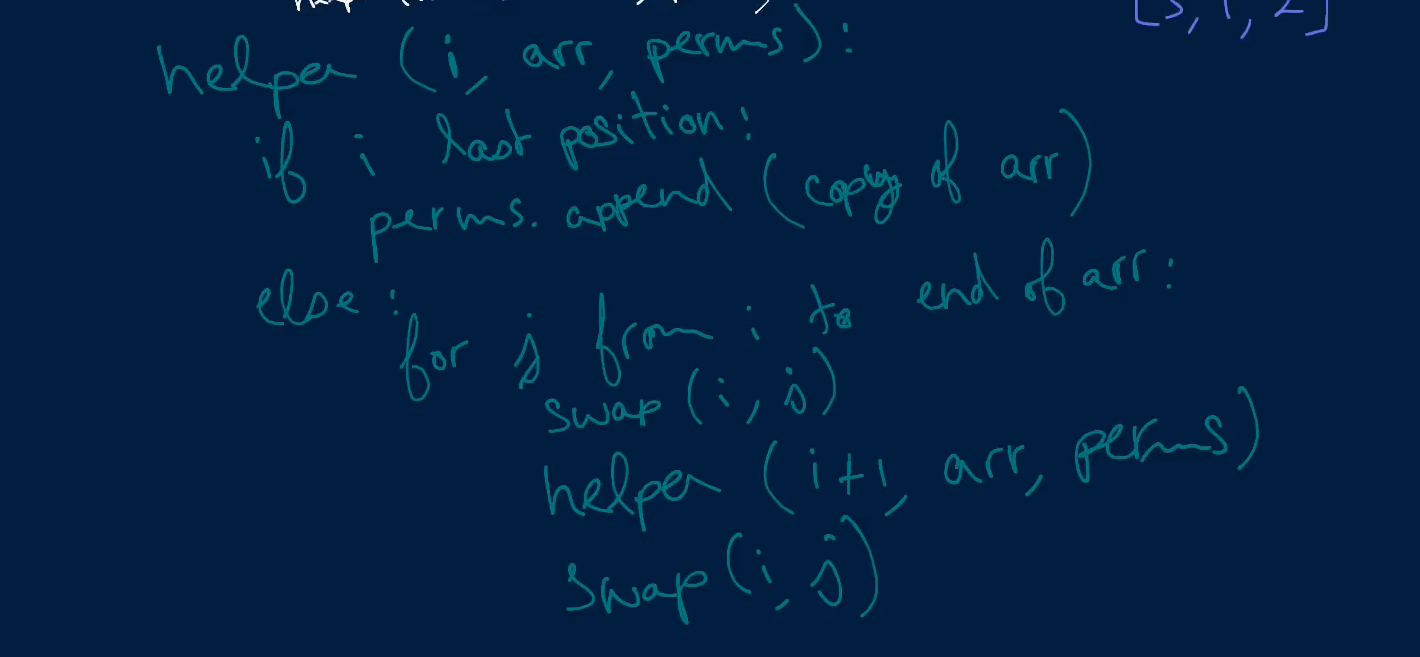
#====🡺============🡺

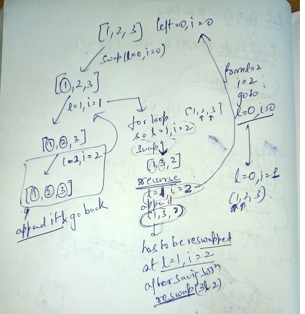
Swap 2 and 1. [2,1,3] to [1,2,3]

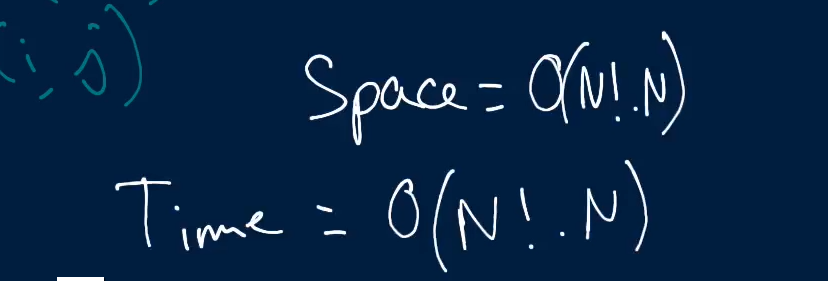
Then swap 1 and 3. [1,2,3] to [3,2,1]/

And after getting another permutation of [3,1,2]//

Go back to the final position of [1,2,3]..







How??

#=========🡺

Time complexity:

Space complexity: O(N!N)

#=========🡺