K-th Permutation.

Brute force $(O(\cancel{?}))(n! \times n)$

we can generate all the permutations then can store them in an array and at [K-1] index we will have answer.

Optimal Solution:

$n=4$ $K=17$

if $n=4$ $n!=24$

so there will be 24 sequences.

if we start permutation from

$\rightarrow 1 + \overset{3!}{(2,3,4)}$] 6 total

$\rightarrow 2 + \overset{3!}{(1,3,4)}$] 6

$\rightarrow 3 + \overset{3!}{(1,2,4)}$] 6

$\rightarrow 4 + \overset{3!}{(1,2,3)}$] 6

$= 24$

As

$n=4$, $K=17$

make an array $(1,2,3,4)$
0 1 2 3

first
$1,2,3,4 \rightarrow 0$

⋮

$4,3,2,1 \rightarrow 23^{th}$
last

$1 + \overset{3!}{(2,3,4)} \Big] 6 \quad (0-5)$

$2 + \overset{3!}{(1,3,4)} \Big] 6 \quad (6-11)$

$3 + \overset{3!}{(1,2,4)} \Big] 6 \quad (12-17)$

$4 + \overset{3!}{(1,2,3)} \Big] 6 \quad (18-23)$

24

If the question is saying we have to find 17 permutation (16) so we are looking that pair
if 0 based indexing

as we can see first number in 16^{th} permutation is 3 which is $\frac{16}{6}$ wants $= 2$ / total no in a pair

index which is 3

So our sequence is lying in range 12 - 17 and $\frac{16}{6} = 4^{th}$ sequence

it is

So we select 3

then find the 4 permutation

- in (1,2,4)

again

1 + [2,4] } 2 (0-1)

2 + [1,4] } 2 (2-3)

→ 4 + [1,2] } 2 (4-5)

Now 6 are total

So

$\underline{3}$ $\underline{4}$ _ _

Now again

$K=4$

$\frac{4}{2} = 2$ and $4\%2$

last is selected then

→ Now 0^{th} permutation as $4\%2=0$

the 0^{th} permutation among all of these.

1 + [2] } 1 (0-0)

2 + [1] } $\frac{1}{2}$ (1-1)

So $K = \frac{0}{1} = 0$

$K\%1 = 0$

$\underline{3}\ \underline{4}\ \underline{1}\ \underline{2} \to 0^{th}$.

cuz after selectn
2 [_]
nothing left

So the whole algorithm is.

initially we are at 4

~~16/4 = 4~~

24 / 4 = 6 → a block of 6

(4-1)! = 3! = 6

] 6
] 6
] 6 ← 16 % 6 = 4
] 6

at each point
total perm = (n-1)!

and idx = K // total perm.
= 16 / 6 = 2
so num.pop (2)
and res.append (nums[idx])

2 (3-1)!
] 2 (0-1)
] 2 (2-3)
] 2 (4-5)

] 1
] 1

n = 4 k = 13 nums = [1, 2, 3, 4]

~~i = 1~~

fact = (n-1) !
idx = k // fact
res.append(num[idx])
nums.pop(idx)
k %= fact

loop from back

i = 4
fact = 3! = 6
idx = 13/6 = 2 k//fact
res = [3]
num = [1, 2, 4]
k = 1

i = 3
f = 2
idx = 1/2 = 0
res = [3, 1]
num = [2, 4]
k = 1%2 = 1

i=2

f= 1

idx= 1//1 = 1

res=[3,1,4]

num=[2]

K=0

i=1

f=1

idx=0

res=[3,1,4,2]

num=[]

K=0

What we are doing it
making chunks

then idx to see in which chunks
it falls.
and grap that number first
element then moves on.