→ **Next Permutation (Arrays)**

Approach

① longest prefix match (break pt. when a[i] < a[i+1])

② find > break point but the smallest so you stay close

③ sort the rest

* **longest prefix match**

↓
the next permutation will have longest possible prefix match

- eg: arr = [2 1 5 4 3 0 0]

↓

- $\underbrace{2\ 1\ 5\ 4\ 3\ 0}_{\text{itna match?}}\ 0$ — No, can we get to next?

- $\underbrace{2\ 1\ 5\ 4\ 3}_{\text{match}}\ 0\ 0$ — No
  ∵ ye should be > 0 to get next & we dont have anyone

- $\underbrace{2\ 1\ 5\ 4}_{\text{match}}\ \underbrace{3\ 0\ 0}$ — No
  coz ye should be > 3 but we only have 3, 0, 0

- $\underbrace{2\ 1\ 5}\ 4\ 3\ 0\ 0$ ×

- $\underbrace{2\ 1}\ 5\ 4\ 3\ 0\ 0$ ×

→ - $\underbrace{2}\ 1\ 5\ 4\ 3\ 0\ 0$

to match 2 & get next, we need someone > but smallest possible so that we dont go too

∴ no no = 5

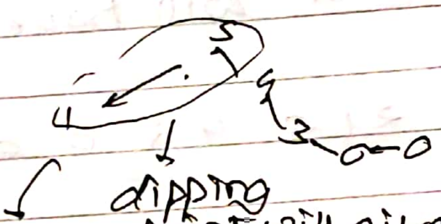
rest??

∴ 2 3 ______

Because of 2 3 it will always be the greater permutation but we want the next

∴ sort the rest to not go too far

- how to find kaha se prefix match ho.?



dipping point will give prefix match

coz this means 1 has someone on the right >1 so we ca swap to get the next permutation.

1x. ∴ replace 1 with someone on the right >1 but smallest possible so its the right next permutation.

* <u>Pseudocode</u>

```
                                    Break point
eg: arr[] = [2  1 | 5  4  3  0  0]
                ↑
               ind
```

```
ind = -1
for(i = n-2; i >= 0; i--)
{   if(a[i] < a[i+1])
        {   ind = i;
            break;
        }
}
if(ind == -1)
{ // no dip => the largest one we have, ~~sort~~ reverse it get the smallest
    reverse(arr)
    return;
}

// now find the next greater element than a[ind]
for( i = n-1; i >= ind; i++)
{   if( arr[i] > arr[ind])
        {   swap(_)
            break;
        }
}

// sort the rest. (although we know its in increasing order so reverse it to get sorted)
~~sort~~(
reverse(rest of array)

}
```

edge case