Q15 Next permutation

i/p → {1, 2, 3}
o/p → {1, 3, 2}

Brute force

1) Generate all the permutations of the given array in sorted order

1	2	3
1	3	2
2	1	3
2	3	1
3	1	2
3	2	1

} 3! ways = 6 ways

2) Linearly traverse the permutations & find the next permutation. If it exists simply return that else simply return the 1st permutation

Better solution

There is an inbuilt STL function namely next_permutation which changes the array to the next permutation

```
next_permutation (arr.begin(), arr.end());
return arr;
```

Optimal solution

For example take 3 words raj, rax and rbx. Now raj comes first in the dictionary, then rax & then rbx. This is because of longer prefix match.

1) We need to find the prefix match.

arr → {2 1 5 4 3 0 0}

→ Leaving behind last zero and taking all remaining digits

2 1 5 4 3 0 0

No permutation possible.

→ Leaving behind last 2 digits and then see the permutation.

2 1 5 4 3 0 0

No permutation possible

→ Leaving behind last 3 digits & then see the permutation.

2 1 5 4 3 0 0

Permutation is possible but not greater & that's what we need.

→ Leaving behind last 4 digits & then see the permutation

2 1 5 4 3 0 0

Permutation possible but not greater.

→ Leaving behind the last 5 digits & then see the permutation.

2 1 5 4 3 0 0

Permutation possible but not greater

→ Now leaving behind last 6 digits & then see the permutations.

2 1 5 4 3 0 0

Now permutations are possible which are greater & this happened as on right of 1 there are numbers greater than 1.

2) 2 will stay but instead of 1 whom will we take.

2 1 | 5 4 3 0 0

>1 <1

Find someone greater than 1 but closest one.

2 3 0 0 1 4 5

>21

3) Try to place rest of digits in the sorted fashion.

Hence next permutation is 2300145.

Algorithm

1) Find the breakpoint or the longest prefix match.
Initially index = -1.
Run a loop from n-2 to 0 & if we find any element such that a[i] < a[i+1], just store it in the index & break.

If index remains -1, This means there is no dip / breakpoint which implies that this is the last permutation & simply reverse the array & return that array.

2) Find someone just greater than the number at breakpoint.
Simply traverse in backward direxn, just swap the 2 numbers once we found at greater element.

2 3 5 4 1 0 0
Simply reverse

2 3 0 0 1 4 5 ⇒ Next permutation

Code

```
void nextPermutation (vector <int> &nums){
    //Step-1 ⇒ Finding longest prefix match
    int n = num.size();
    int index = -1;
    for (int i = n-2; i >= 0; i--) {
            // Breakpoint condition
            if (nums[i] < nums[i+1]) {
                index = i;
                break;
            }
    }
    // Handling the edge case
    if (index == -1) {
        reverse (arr.begin(), arr.end());
        return;
    }
```

```
//Step-2 => Swap number at breakpoint
for (int i=n-1; i>=0; i--) {
        if (nums[i] > nums[index]) {
                swap (nums[i], nums[index]);
                break;
        }
}
// Reverse array after break => (sorted)
 reverse (nums.begin()+index+1, nums.end());
}
```

Time complexity = O(n)
Space complexity = O(1)