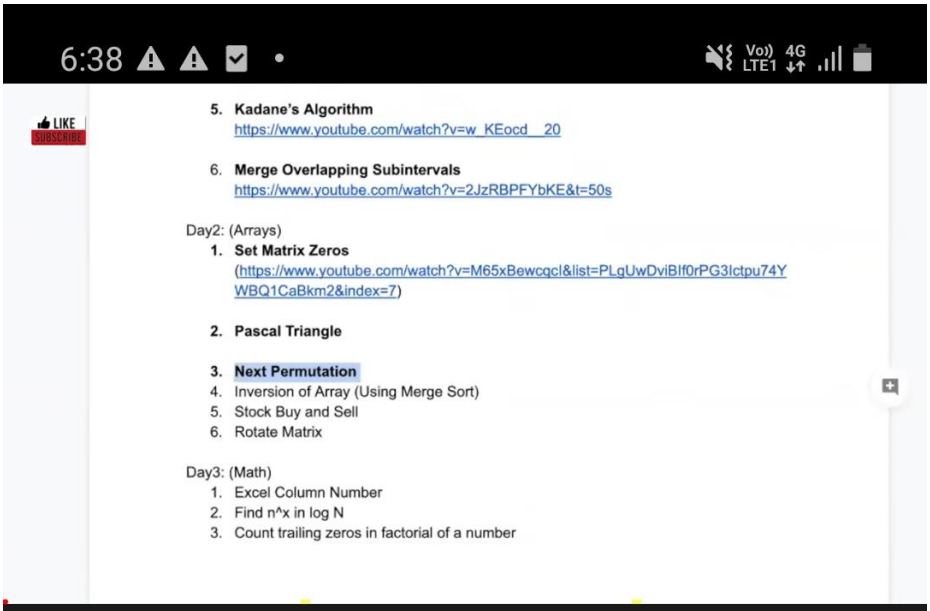


5. **Kadane's Algorithm**
   https://www.youtube.com/watch?v=w_KEocd__20

6. **Merge Overlapping Subintervals**
   https://www.youtube.com/watch?v=2JzRBPFYbKE&t=50s

Day2: (Arrays)

1. **Set Matrix Zeros**
   (https://www.youtube.com/watch?v=M65xBewcqcI&list=PLgUwDviBIf0rPG3Ictpu74YWBQ1CaBkm2&index=7)

2. **Pascal Triangle**

3. **Next Permutation**
4. Inversion of Array (Using Merge Sort)
5. Stock Buy and Sell
6. Rotate Matrix

Day3: (Math)

1. Excel Column Number
2. Find n^x in log N
3. Count trailing zeros in factorial of a number

Placement Series

84 videos · take U forward

- C++ | Java | 3 problems asked ... (9:42)
- NEXT PERMUTATION | Leetcode | Know the Intuition ... (11:33)
- COUNT INVERSIONS in an ARRAY | Leetcode | C++ | Java ... (14:02)
- Best Time to BUY and SELL STOCK | Leetcode | C++ | Java ... (6:49)
- ROTATE IMAGE | Leetcode | C++ | Java | Brute-Optimal (4:53)
- Search in 2D-MATRIX | Leetcode | GFG | C++ | Java | B... (14:58)
- POW(x,n) | Binary Exponentiation | Leetcode (8:10)
- Majority Element | Leetcode | C++ | Java | Brute-Better-Optimal ... (11:19)

31. Next Permutation

Medium 3640 1299 Add to List Share

Implement **next permutation**, which rearranges numbers into the lexicographically next greater permutation of numbers.

If such arrangement is not possible, it must rearrange it as the lowest possible order (ie, sorted in ascending order).

The replacement must be **in-place** and use only constant extra memory.

Here are some examples. Inputs are in the left-hand column and its corresponding outputs are in the right-hand column.

```
1,2,3 → 1,3,2
3,2,1 → 1,2,3
1,1,5 → 1,5,1
```

Accepted 387,310 Submissions 1,187,159

TUF

1 3 2.

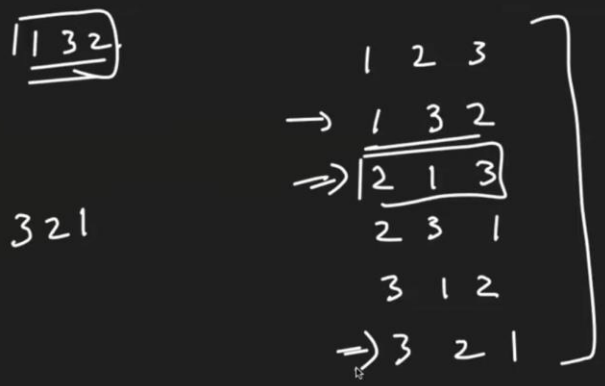
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1

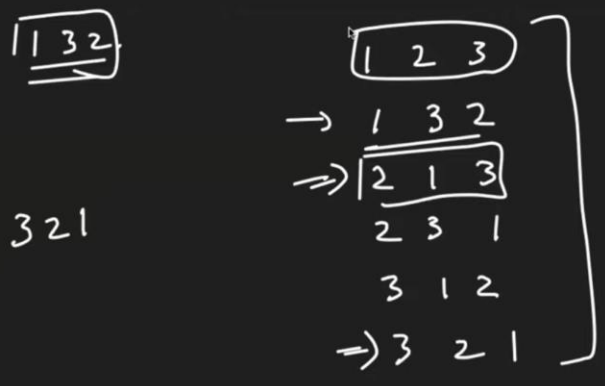
TUF

1 3 2

1 2 3
→ 1 3 2
2 1 3
2 3 1
3 1 2
3 2 1

TUF





1 3 5 4 2 → 1 4 2 3 5

(i) a[i] (a[i+1]

1 3 5 4 2 → 1 4 2 3 5

(i) a[i] < a[i+1] ind1 = 1

(ii) a[ind2] > a[ind1] ind2 = 3

1 3 5 4 2 → 1 4 2 3 5

(i) a[i] < a[i+1] ind1 = 1

(ii) a[ind2] > a[ind1] ind2 = 3

(iii) swap (a[ind1], a[ind2])

(iv) reverse (ind1+1 → last)

1 4 5 3 2

TUF

1 3 5 4 2 → 1 4 2 3 5

(i) a[i] < a[i+1] ind1 = 1

(ii) a[ind2] > a[ind1] ind2 = 3

(iii) swap (a[ind1], a[ind2])

(iv) reverse (ind1+1 → last)

1 4 5 3 2

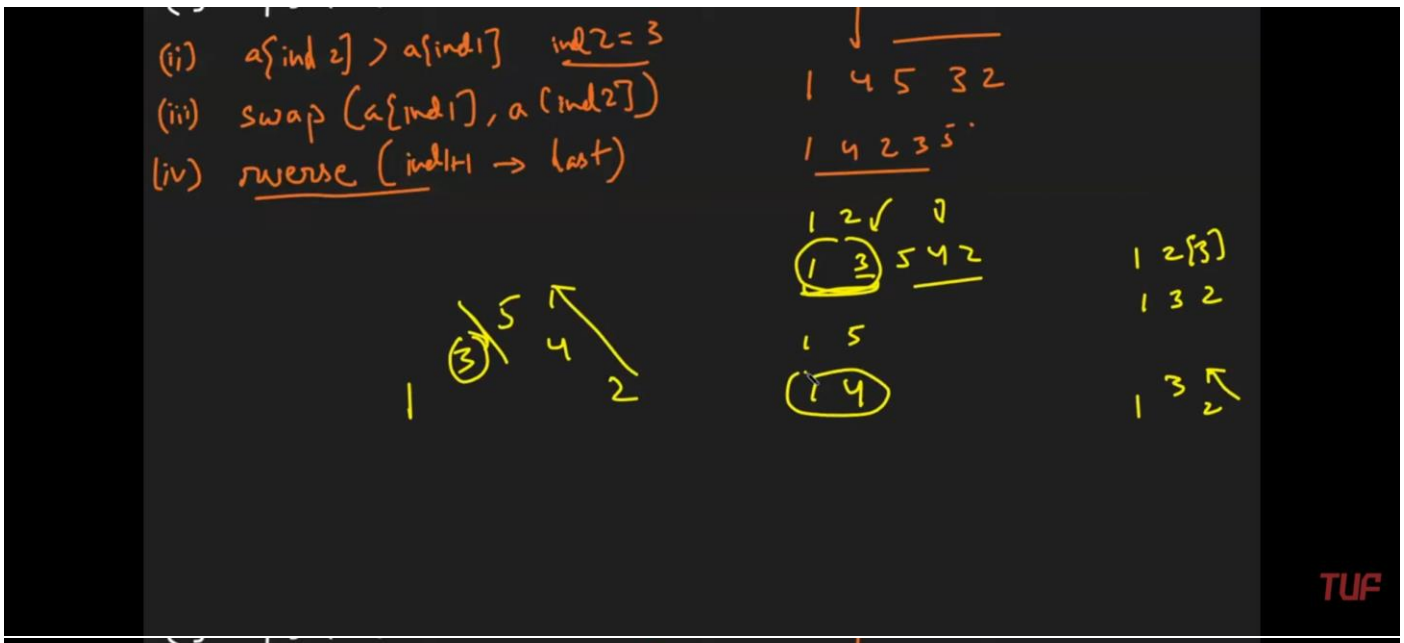
1 4 2 3 5

TUF

(ii) a[ind 2] > a[ind1] ind2 = 3

(iii) swap (a[ind1], a[ind2])

(iv) reverse (ind1+1 → last)

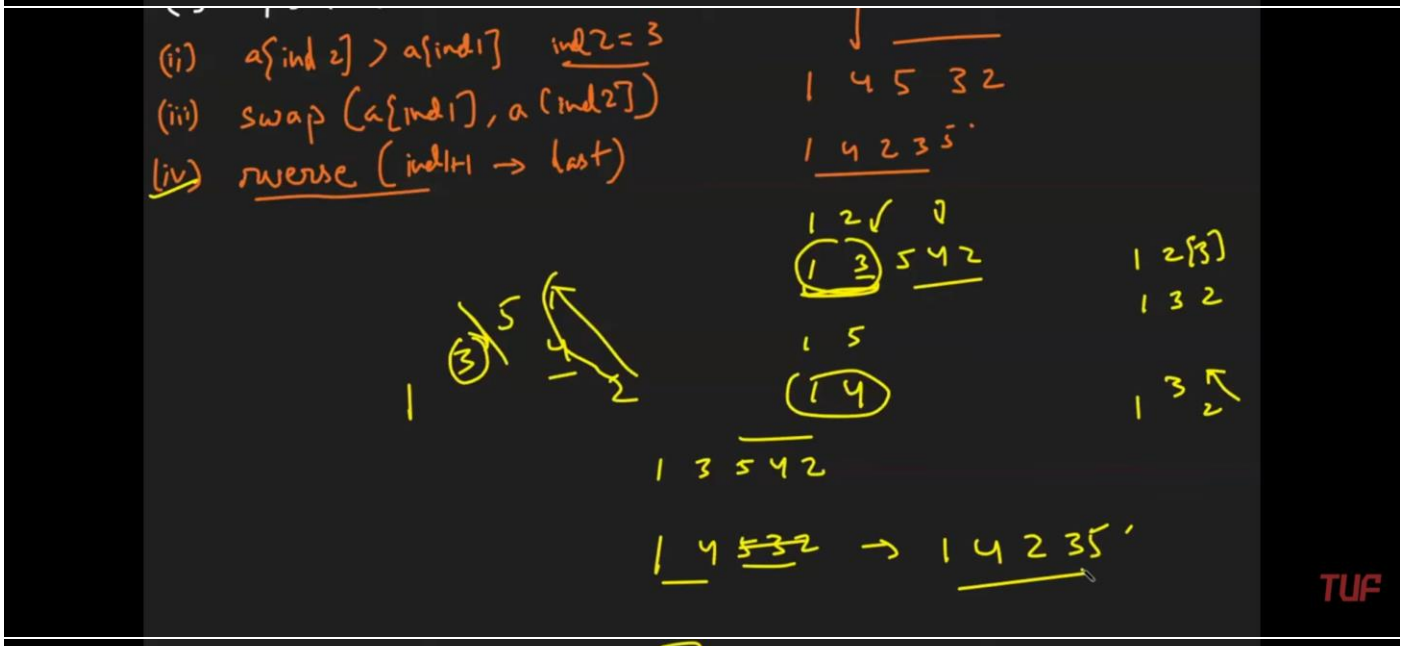


TUF

(ii) a[ind 2] > a[ind1] ind2 = 3

(iii) swap (a[ind1], a[ind2])

(iv) reverse (ind1+1 → last)



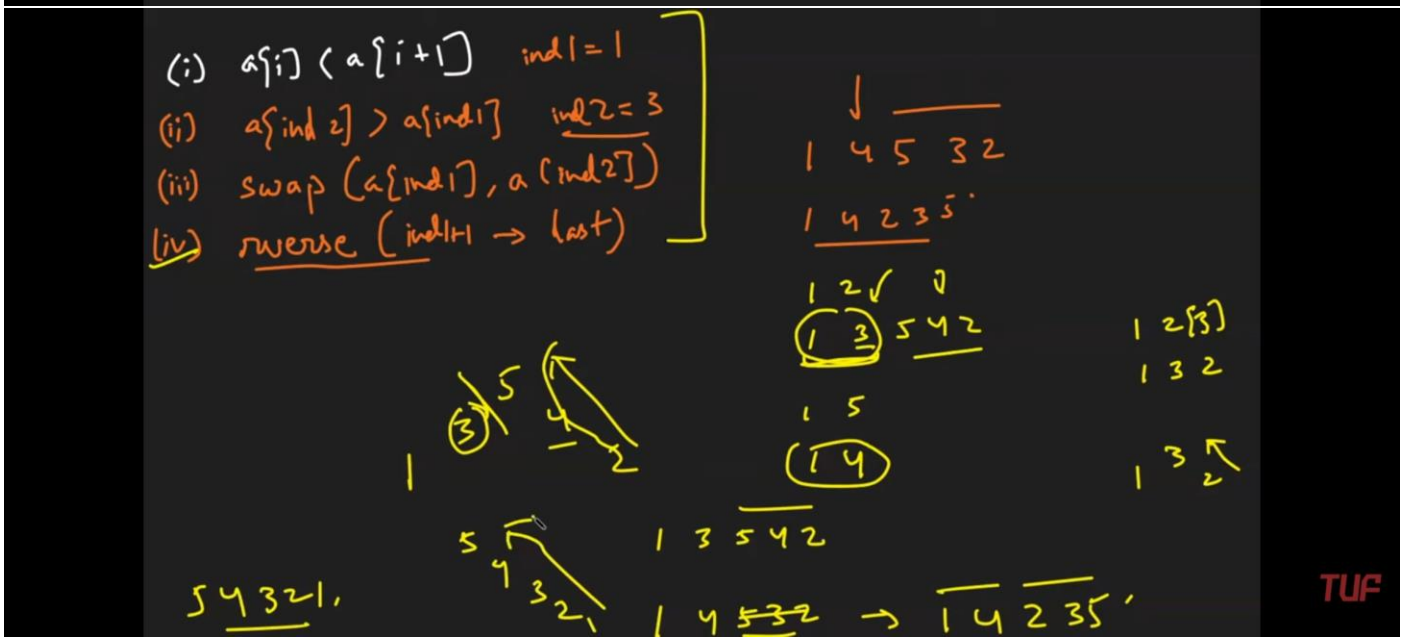
TUF

(i) a[i] < a[i+1] ind1 = 1

(ii) a[ind 2] > a[ind1] ind2 = 3

(iii) swap (a[ind1], a[ind2])

(iv) reverse (ind1+1 → last)



TUF

(i) a[i] < a[i+1] ind1 = 1

(ii) a[ind2] > a[ind1] ind2 = 3

(iii) swap (a[ind1], a[ind2])

(iv) reverse (ind1+1 → last)

1 4 5 3 2

1 4 2 3 5

1 2 0

1 3 5 4 2

1 2[3]

1 3 2

1 5

1 4

1 3

2

1 3 5 4 2

5 4 3 2 1.

1 4 5 3 2 → 1 4 2 3 5.

TUF

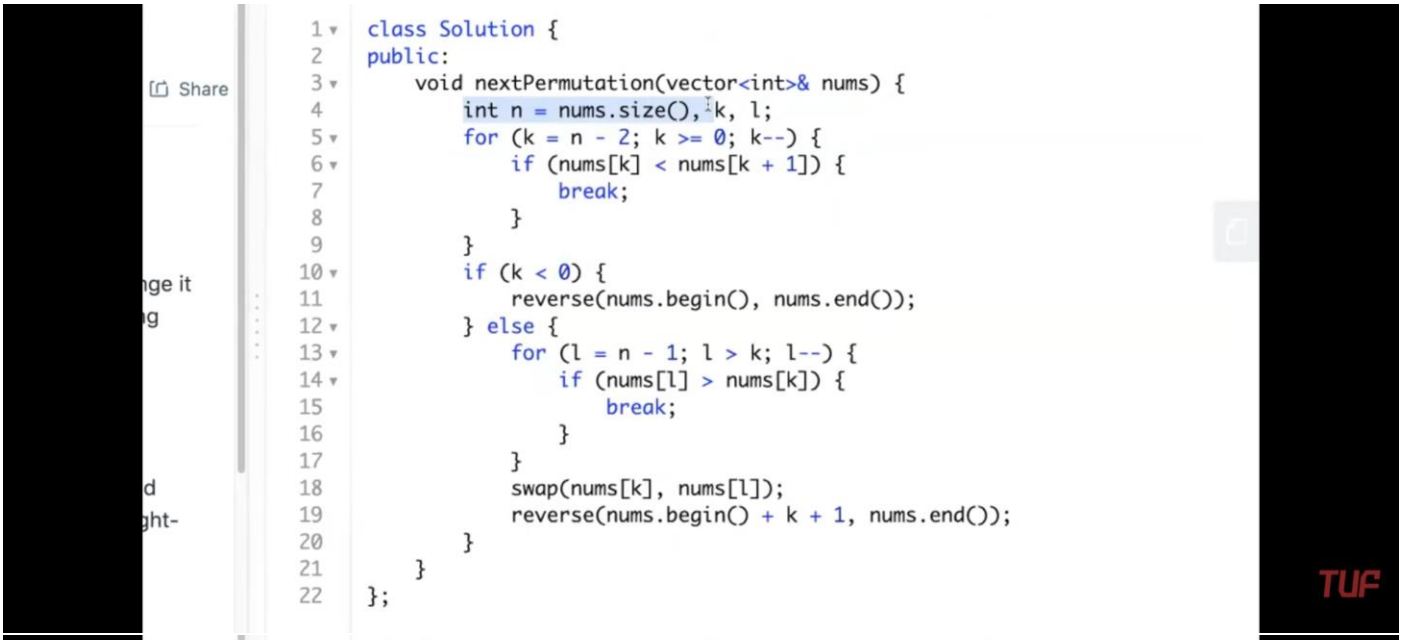
$O(n) + O(n) + O(n)$ ...

TUF

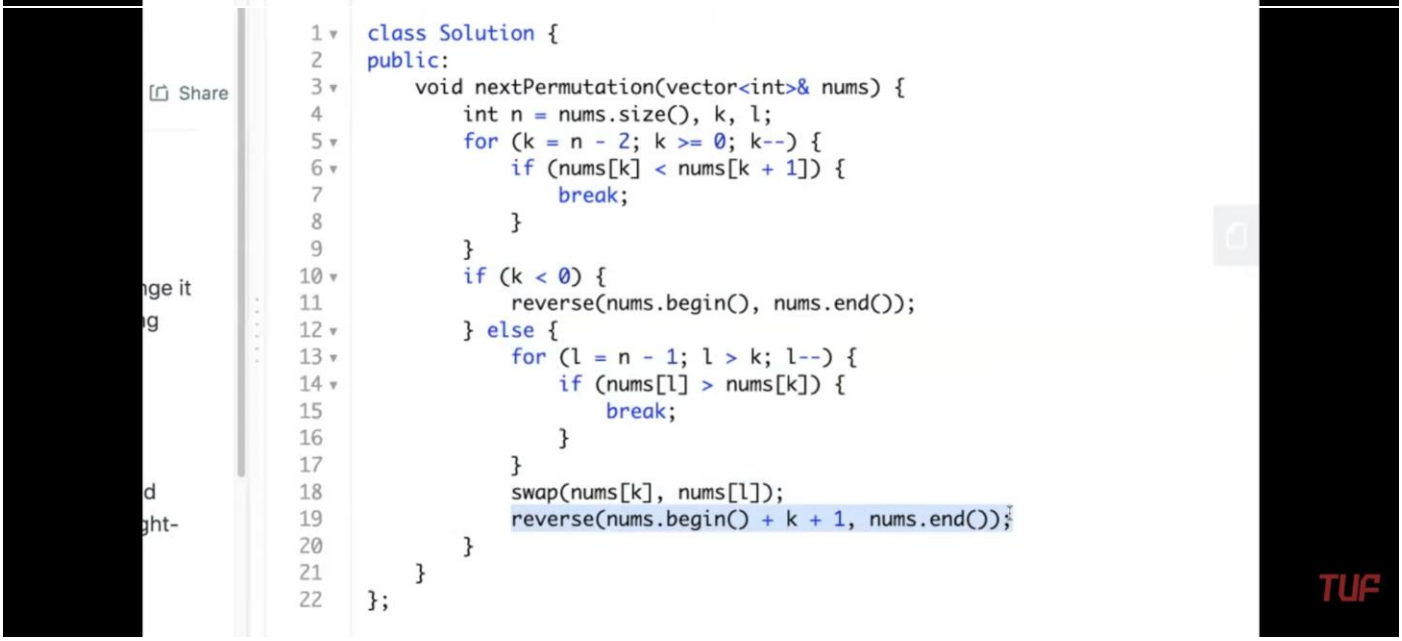
$O(n) + O(n) + O(n) \simeq O(n)$

$= O(1)$

TUF



```
class Solution {
public:
    void nextPermutation(vector<int>& nums) {
        int n = nums.size(), k, l;
        for (k = n - 2; k >= 0; k--) {
            if (nums[k] < nums[k + 1]) {
                break;
            }
        }
        if (k < 0) {
            reverse(nums.begin(), nums.end());
        } else {
            for (l = n - 1; l > k; l--) {
                if (nums[l] > nums[k]) {
                    break;
                }
            }
            swap(nums[k], nums[l]);
            reverse(nums.begin() + k + 1, nums.end());
        }
    }
};
```



```
class Solution {
public:
    void nextPermutation(vector<int>& nums) {
        int n = nums.size(), k, l;
        for (k = n - 2; k >= 0; k--) {
            if (nums[k] < nums[k + 1]) {
                break;
            }
        }
        if (k < 0) {
            reverse(nums.begin(), nums.end());
        } else {
            for (l = n - 1; l > k; l--) {
                if (nums[l] > nums[k]) {
                    break;
                }
            }
            swap(nums[k], nums[l]);
            reverse(nums.begin() + k + 1, nums.end());
        }
    }
};
```