Agenda

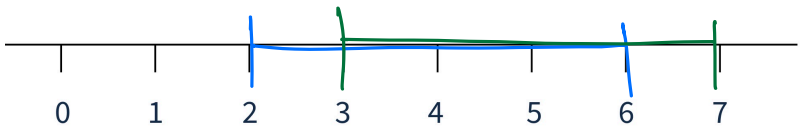
1. Merge Intervals
2. Missing Integer
3. Next Permutation

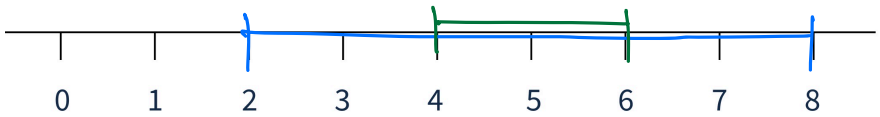
Merge Overlapping Intervals

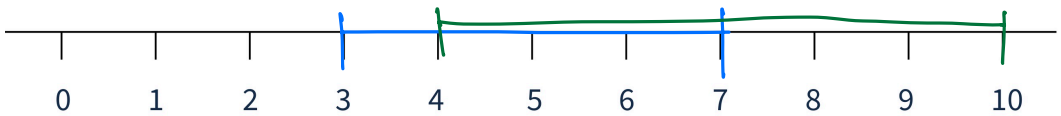
defined by start-time and end-time.

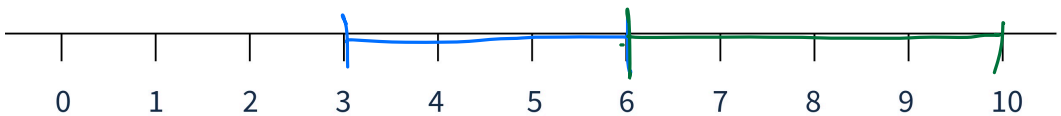
merged interval

I_1 I_2

(2, 6) (3, 7) →  [2, 7]

(2, 8) (4, 6) →  [2, 8]

(3, 7) (4, 10) →  [3, 10]

(3, 6) (6, 10) →  [3, 10]

(2, 5) (8, 10) → Non-overlapping intervals.

(5, 8) (1, 3) → Non-overlapping intervals.

(6, 10) (8, 15) → [6, 15]

</> Code

$[a_1, b_1]$, $[a_2, b_2]$

```
if ( a2 > b1 || a1 > b2) {
    //non-overlapping intervals
}
else{
    a3 = Min(a1, a2)
    b3 = Max(b1, b2)
}
```

merged interval

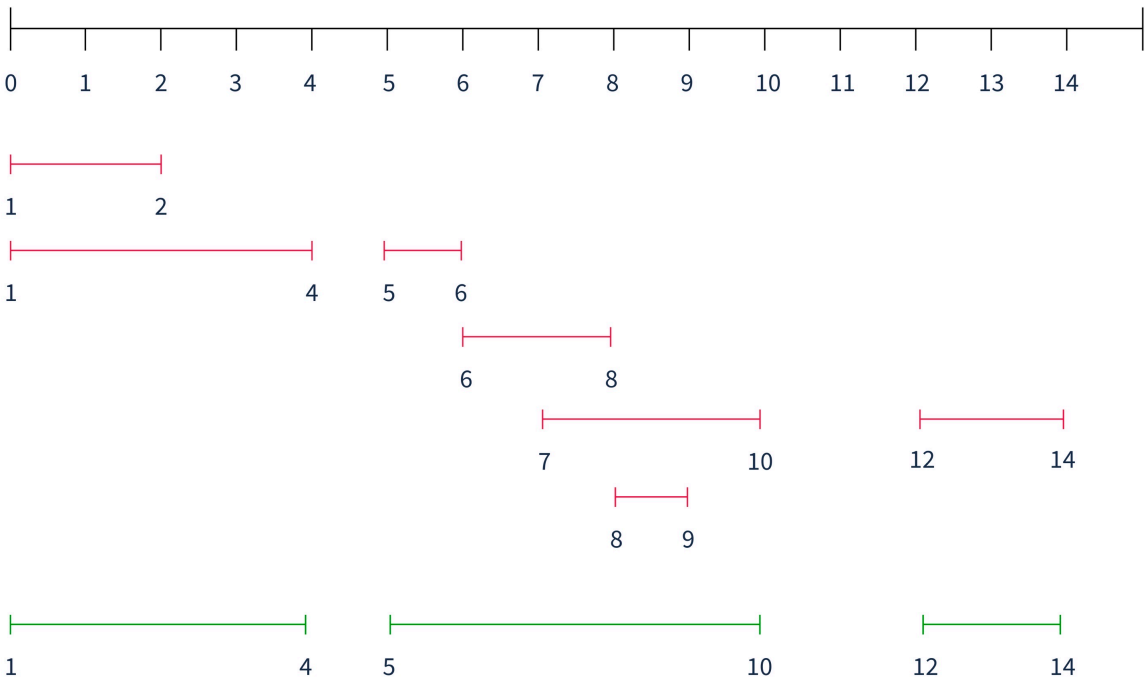
< *Question* > : Given a sorted list of overlapping intervals, sorted based on start-time, Merge all overlapping intervals and return the sorted list of non-overlapping intervals.

$1 \leq N \leq 10^5$

N = 7

Intervals

1	2
1	4
5	6
6	8
7	10
8	9
12	14



start time of B ≤ end time of A

Quiz : $[[1,10], [2,3], [4,5], [9,12]]$

S → 1
E → 10 12

ans → $[[1,12]]$

idea → Iterate on all intervals & try to merge if they are overlapping.

Example :

Intervals[N] → [(0, 2) , (1, 4) , (5, 6) , (6, 8) , (7, 10) , (8, 9) , (12, 14)]

s = 12
e = 14

ans → [[0,4], [5,10], [12,14]]

code →

```
s = Intervals[0][0],   e = Intervals[0][1];
list < int[] > ans ;
for( i = 1 ; i < N ; i++){
    if( Intervals[i][0] ≤ e) {  //overlapping.
        e = Max(e, Intervals[i][1]);
    }
    else{
        ans.add ( {s,e} );
        s = Intervals[i][0];
        e = Intervals[i][1];
    }
}
ans.add ( {s,e} );
return ans;
```

T.C → O(N)
S.C → O(1)

Gemini

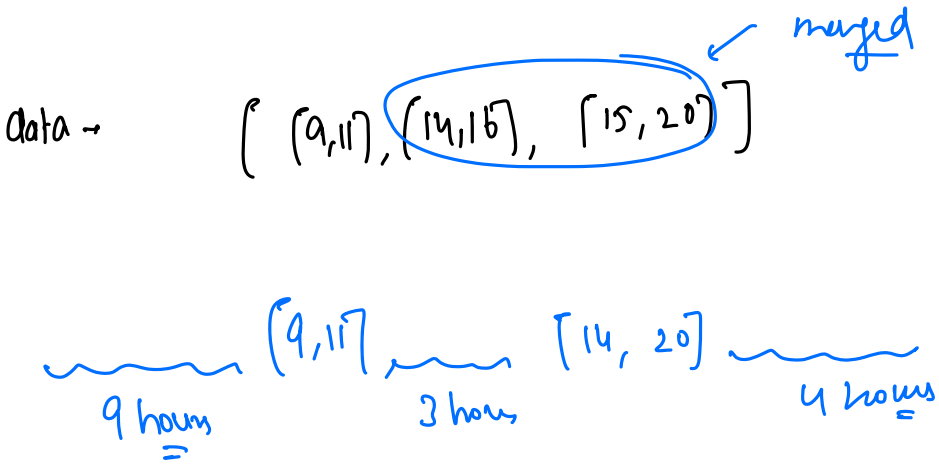
Background

Scaler Academy, a leading ed-tech platform known for its comprehensive learning programs, is planning to conduct maintenance on its website to enhance user experience and introduce new features.

To ensure the maintenance work does not disrupt the learning process for its students, Scaler Academy aims to schedule this maintenance during the period of no user activity.

Problem Statement

Given sorted data on the active hours of multiple learners on the platform, your task is to analyze this data and identify the longest continuous period when no learners are active. This identified time slot is crucial as it represents the best opportunity to perform website maintenance with the least disruption to learners' activities.



[1, 2, 3, 4, 5, — — ∞]

< Question > : Find the first missing natural number. N = length of array

Example 1: arr [5] : [3 -1 1 2 7] → 4

Example 2: arr [7] : [9 2 6 4 -8 1 3] → 5

Example 3 : arr [6] : [1 0 -5 -6 4 2] → 3

Example 4 : arr [6] : [1 2 5 6 4 3] → 7

Example 5 : arr [4] : [1 2 3 4] → 5

Quiz : [5, 3, 1, -1, -2, -4, 7, 2] → 4

BF Idea

ans → [1, N+1]

iterate from 1 to N, check if this no. is present in the array.

[T.C → O(N²)
S.C → O(1)]

Idea -2 → Sort

[5, 3, 1, -1, -2, -4, 7, 2]

↓ sort

[-4, -2, -1, 1, 2, 3, 5, 7]

[T.C → O(NlogN)
S.C → O(depends on sorting algo)]

Idea -3 → Can we use the fact that our answer is lying in this range [1, N+1]

arr[N] →

-5	-2	-7	-1	-3	4	-8
0	1	2	3	4	5	6

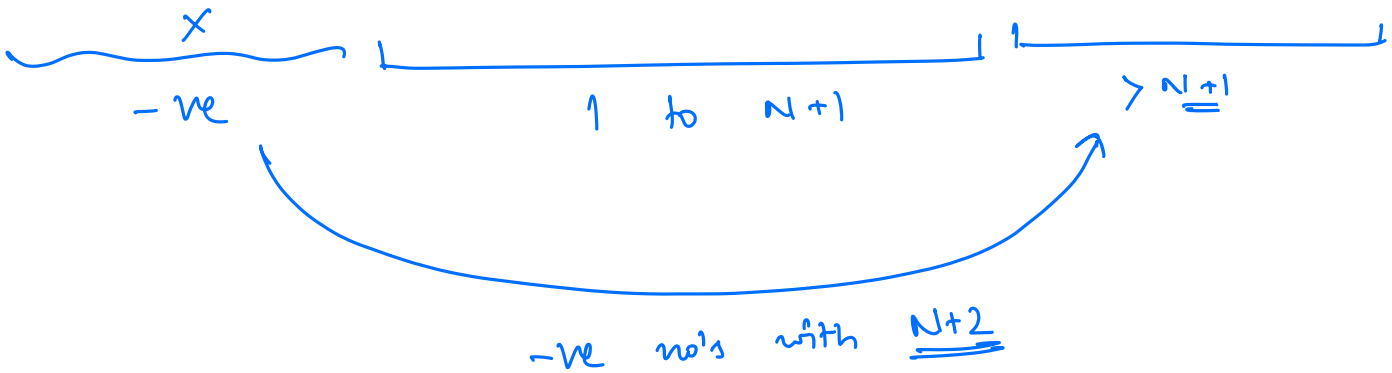
ans = 6

presence of 1 → idx 0
2 → 1
3 → 2
⋮
i → i-1

Arr[] → [-5, -2, -7, -5, -3, -6, -1, 4, 3]

0 1 2 3 4 5 6 7 8

→ How to handle the -ve no's ?



</> Code

```
for( i : 0;  i < N;  i++){
|     if ( arr[i] <= 0){
|     |     arr[i] = N+2 ;
|     }
}

for( i : 0;  i < N;  i++){
|       ele  =  Abs ( arr[i] )
|       if ( ele <= N){
|       |     arr[ ele-1]  =  Abs ( arr[ele-1] ) x -1 ;
|       }
}

for( i = 0;  i < N;  i++){
|     if ( arr[i] > 0){
|     |     return  i+1 ;
|     }
}

return  N+1 ;
```

T.C → O(N)
S.C → O(1)

Break - 8:38 → 8:45

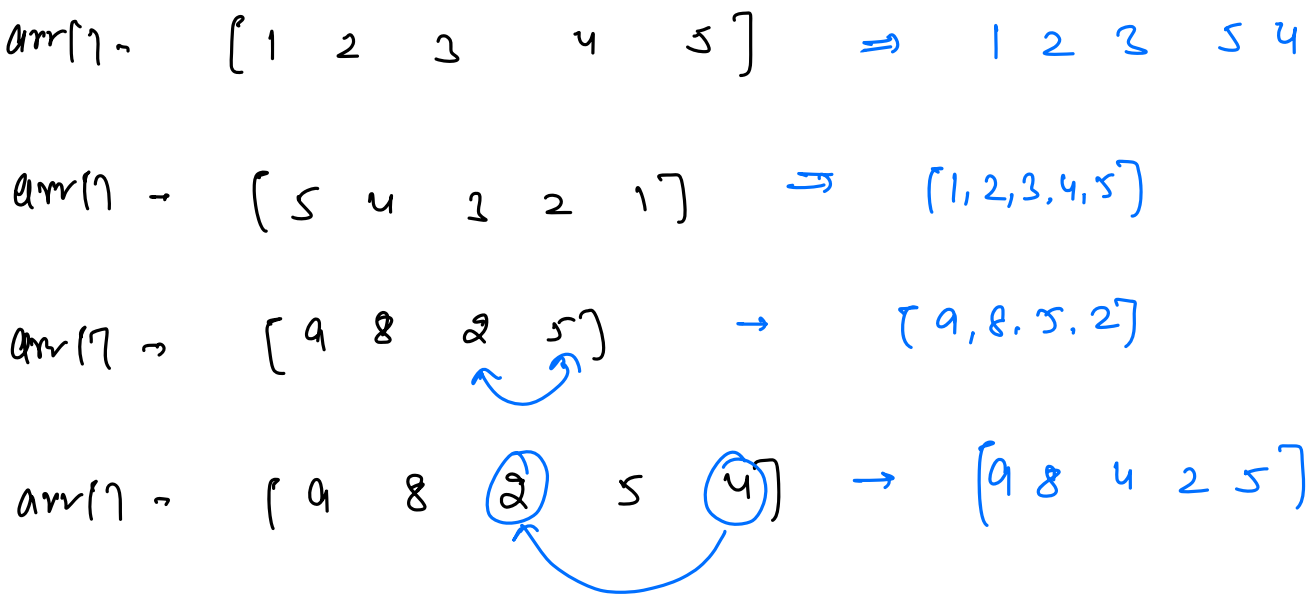
Next Permutation

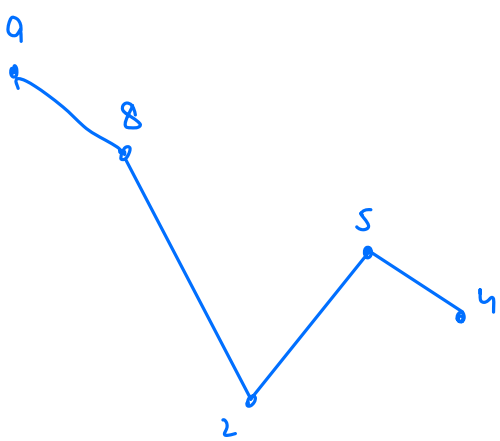
< *Question* > : Implement the next permutation, which rearranges numbers into the numerically next greater permutation of numbers for a given array A of size N.

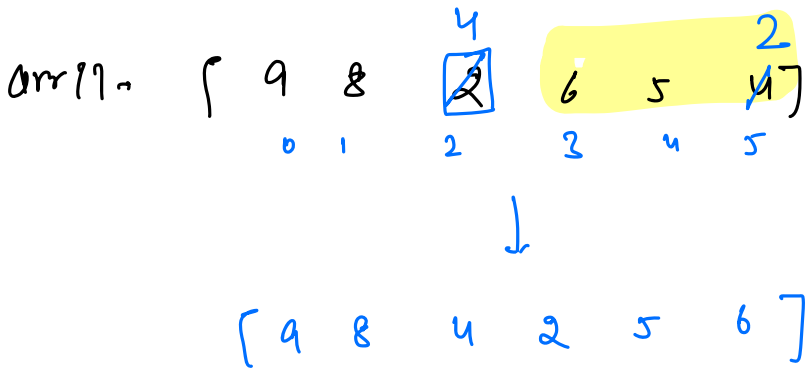
If such arrangement is not possible, it must be rearranged as the lowest possible order, i.e., sorted in ascending order.

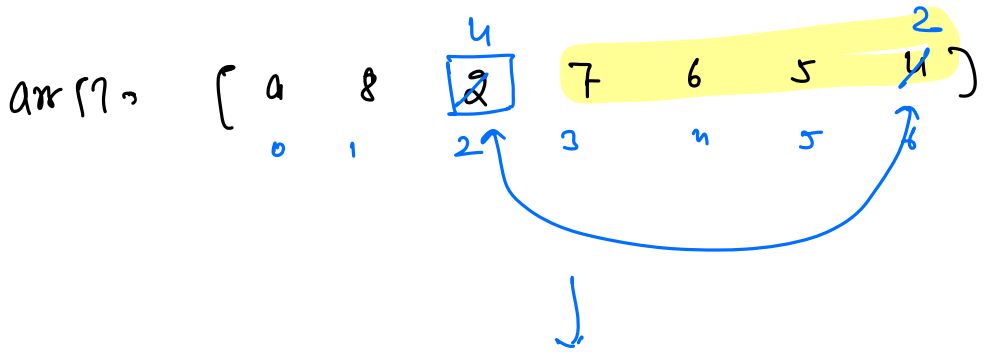
NOTE:

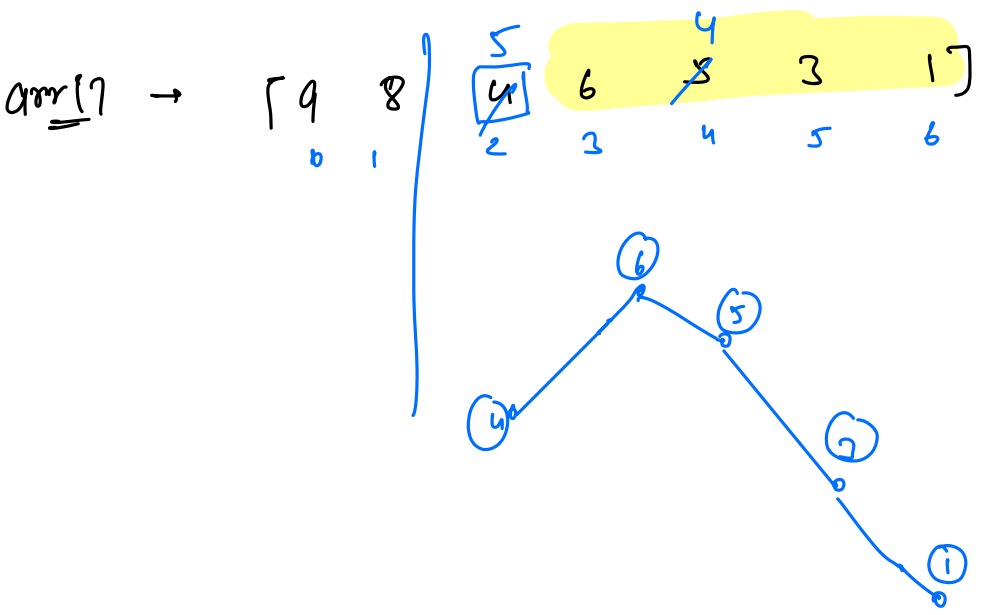
The replacement must be in-place, do not allocate extra memory.
DO NOT USE LIBRARY FUNCTION






arr[] =
9 8 4 6 5 4
0 1 2 3 4 5
4
2
[9 8 4 2 5 6]


arr[] =
9 8 4 7 6 5 4
0 1 2 3 4 5 6
4
2
[9 8 4 2 5 6 7]


arr[] →
9 8 4 6 5 3 1
0 1 2 3 4 5 6
5
4
[9, 8, 5, 1, 3, 4, 6]

idea. →

1. start from r.h.s & keep on iterating till you find a dip.
2. find just greater number than this element (dip) on r.h.s.
3. swap with just greater no.
4. Reverse the subarray from dip+1 to N-1.

code →

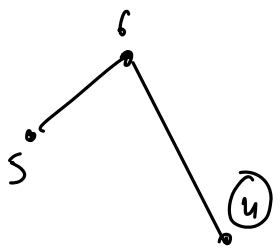
```
dip = -1;
for( i = N-1; i > 0; i--){
    if( arr[i-1] < arr[i]){
        dip = i-1;
        break;
    }
}

if( dip == -1){
    reverse( arr, 0, N-1);
    return arr[];
}
```

```
for( i = n-1 ; i > dip ; i--){
    if( arr[i] > arr[dip]){
        swap arr[i] with arr[dip];
        break;
    }
}
reverse (arr, dip+1, n-1);
```

T.C → O(N)
S.C → O(1)

---

9 8 1 | 6 5 4 ⇒ 9 8 1 6 4 5



n-1-50

p1 + p2 → n

Sieve of Eratosthenes

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47.