Lecture :- Arrays Interview problems

Agenda

- Merge overlapping intervals
- Insert new interval.
- first missing +ve integer

Merge intervals

interval is defined by start and end time.

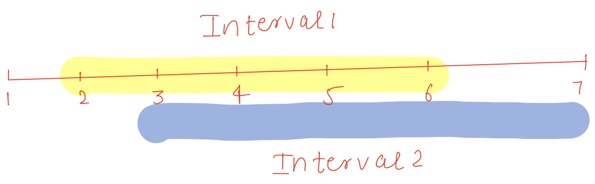
start <= end time.

1.) Interval 1 $(2, 6)$ Interval 2 $(3, 7)$

Merge them.

final merged interval = $(2, 7)$.

Overlapping



2.) $(2, 4)$ $(5, 7)$ $\longrightarrow$ non-overlapping

3.) $(2, 8)$ $(4, 6)$ $\longrightarrow$ overlapping

Merge them.

merged interval = $(2, 8)$

4.) $(3, 9)$ $(1, 6)$ $\longrightarrow$ overlapping $\therefore$ merged interval = $(1, 9)$

Generalisation

(s_1, e_1) (s_2, e_2) $\longrightarrow$ Overlapping intervals

After merging them:

$[\min(s_1, s_2), \max(e_1, e_2)]$

Qu. Merge sorted overlapping intervals.

Given n intervals in sorted manner [overlapping intervals] sorted on start time. Merge all overlapping interval and return sorted list.

Input [(0, 2) (1, 4) (5, 6) (6, 8) (7, 10) (8, 9) (12, 14)].

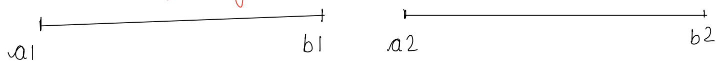
$\underbrace{(0, 2) \quad (1, 4)}_{(0, 4)}$

output [(0, 4) (5, 10) (12, 14)]

Interval 1	Interval 2	is overlapping ?	Answer interval list.
(0, 2)	(1, 4)	Overlapping	(0. 4)
(0, 4)	(5, 6)	No	(0, 4) , (5, 6)
(5, 6)	(6, 8)	Overlapping	(0, 4) (5, 8)
(5, 8)	(7, 10)	Overlapping	(0, 4) (5, 10)
(5, 10)	(8, 9)	Overlapping	(0, 4) (5, 10)
(5, 10)	(12, 14)	No	(0, 4) (5, 10) (12, 14)

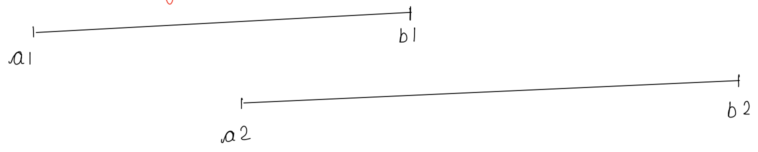
Approach   Overlapping condition ?

case1   Non- overlapping.

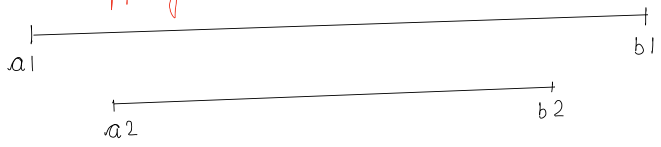


a2 > b1

case2:   Overlapping



case3:   Overlapping



start of int2 <= end of int1

a2 <= b1

Dry run:

interval[] = [(0,2) (1,4) (5,6) (6,8) (7,10) (8,9) (12,14)]

interval1	interval2	isoverlapping?	After merging answer list.
(0,2)	(1,4)	Yes [a2<=b1]	(0,4)
(0,4)	(5,6)	No [a2>b1]	(0,4) (5,6)
(5,6)	(6,8)	Yes [a2<=b1]	(0,4) (5,8)
(5,8)	(7,10)	Yes [a2<=b1]	(0,4) (5,10)
(5,10)	(8,9)	Yes [a2<=b1]	(0,4) (5,10)
(5,10)	(12,14)	No [a2>b1]	(0,4) (5,10) (12,14)
(12,14)	end		

```
class Interval {
        int start;
        int end;
}
```

Algorithm code

```
List<Interval> mergeOverlappingIntervals (List<Interval> intervals) {
        List<Interval> ans = new ArrayList<>();
        int cs = intervals.get(0).start;
        int ce = intervals.get(0).end;

        for(i=1; i< intervals.size(); i++) {
                int s = intervals.get(i).start;
                int e = intervals.get(i).end;
                if( s <= ce)  {
                        ce = max( e, ce);
                } else {
                        Interval temp = new Interval ( cs, ce);
                        ans.add(temp);
                        cs = s;
                        ce = e;
                }
        }
        Interval temp = new Interval ( cs, ce);
        ans.add(temp);
        return ans;
}
```

TC: O(n)

SC: O(n) || O(1)

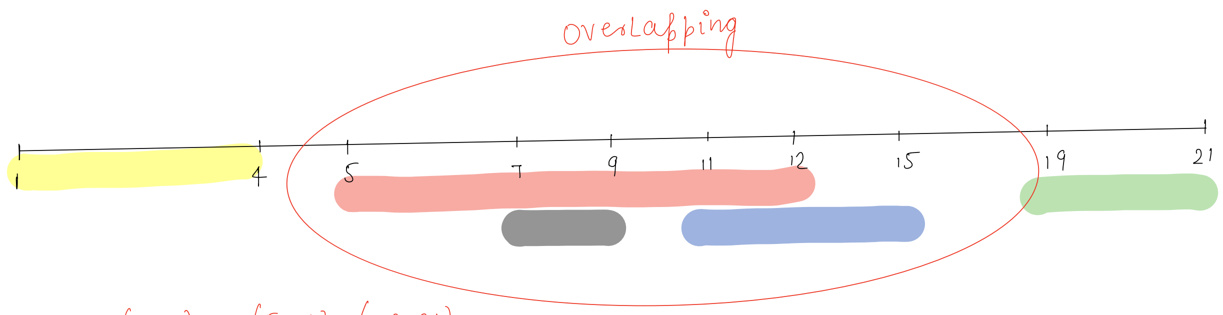
Qu2 Given arr[n] intervals in sorted manner and non-overlapping. Given one extra interval.

Insert this extra interval in array.

Input interval[] = [(1, 4) (7, 9) (11, 15) (19, 21)]

new-interval = (5, 12)

output: [(1, 4) (5, 15) (19, 21)]



(1, 4) (5, 15) (19, 21)

Idea: 1.) If intervals are non-overlapping. (sorted)

interval = (3, 6) New interval will come on right.

new interval = (7, 8)

2.) interval = (3, 6) Left

new interval = (1, 2)

3.) interval = (3, 6) (3, 8)

new interval = (5, 8)

Overlapping

Dry run:

intervals[] = [(1, 3) (4, 7) (10, 14) (16, 19) (21, 24) (27, 30) (32, 35)]

new interval = (12, 22)

interval 1	new interval	Placement	Merged.
(1, 3)	(12, 22)	right	(1, 3)
(4, 7)	(12, 22)	right	[1, 3] [4, 7]
(10, 14)	(12, 22)	Overlapping	[1, 3] [4, 7]
(16, 19)	(10, 22)	overlapping	[1, 3] [4, 7]
(21, 24)	(10, 22)	Overlapping	[1, 3] [4, 7]
(27, 30)	(10, 24)	left	[1, 3] [4, 7] [10, 24]
(32, 35)	(27, 30)	left	[1, 3] [4, 7] [10, 24] [27, 30]
(32, 35)	end		[1, 3] [4, 7] [10, 24] [27, 30] [32, 35] Ans

Algorithmic code

```
List<Interval>  mergeIntervals( List<Interval>  intervals,
                                Interval  newInterval)  {
    List<Interval>  ans = new  ArrayList<>();

    for( i = 0;  i < intervals.size();  i++) {
          int  s = intervals.get(i).start;

          int  e = intervals.get(i).end;
               // non-overlapping
          if( newInterval.start  > e)   {

               ans.add intervals.get(i));

          }
curr → (27, 30)
n.i  → (16, 24)
        else if( s  >   newInterval.end)  {
               ans.add( newInterval);
               while( i<n)  {
                    ans.add( intervals.get(i));
                    i++;
               }

             return ans;
        }
        else {     // overlapping.
            newInterval.start = min( newInterval.start , s);

               "        .  end = max(   "        .  end ,  e);

        }
    }
     ans.add( newInterval);
     return  ans;
}
```

TC: O(n)
SC:

Break: 8:30-8:40

Qu Given arr[n], find first missing +ve number.

3	-2	1	2	7

ans = 4

1	2	5	6	4	3

ans = 7

1	0	-5	-6	4	2

ans = 3

Claim Ans = [1, n+1]

Approach1 Brute force approach.

```
Iterate from 1 to n+1 ——
    Check if it is present in array or not?
        present → continue;
    not   "   → return ans;
```

TC: $O(n^2)$
SC: $O(1)$

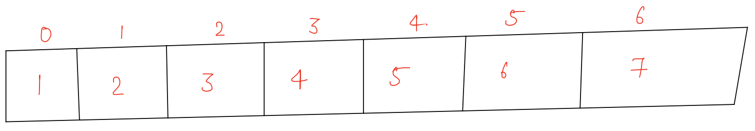
Approach2 Hashset.

TC: $O(n)$
SC: $O(n)$

Expected solution TC: $O(n)$
SC: $O(1)$

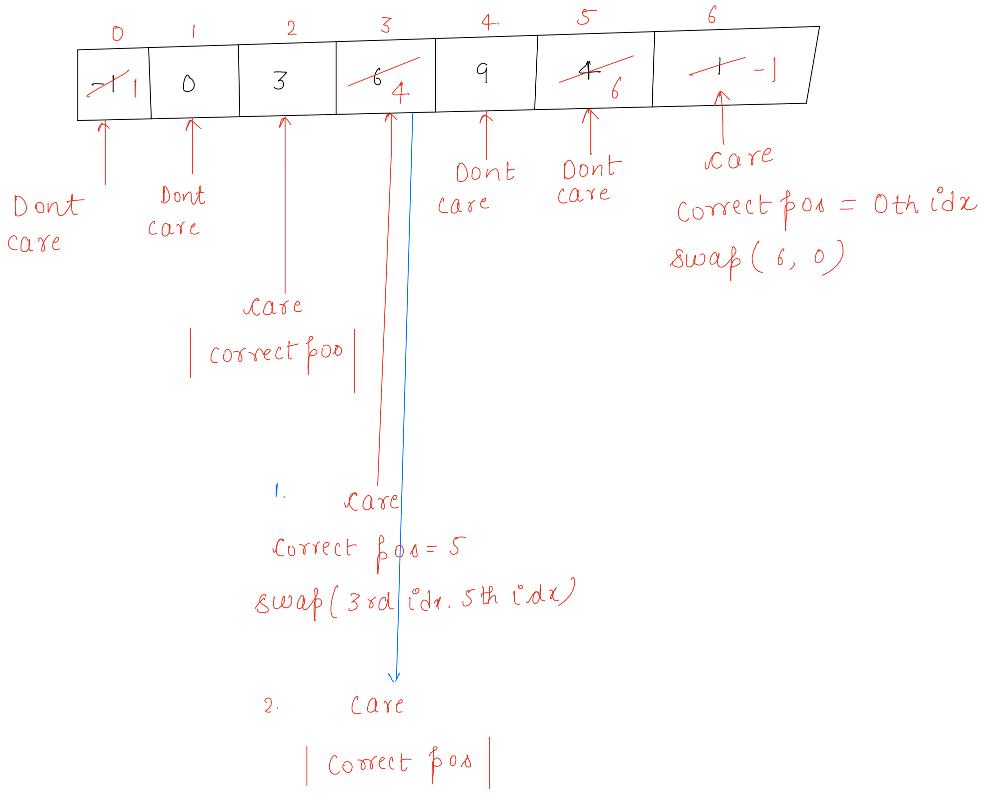
Use same array to store the information?

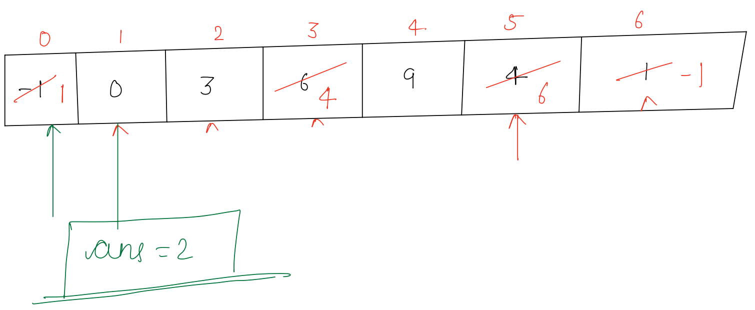
Intuition

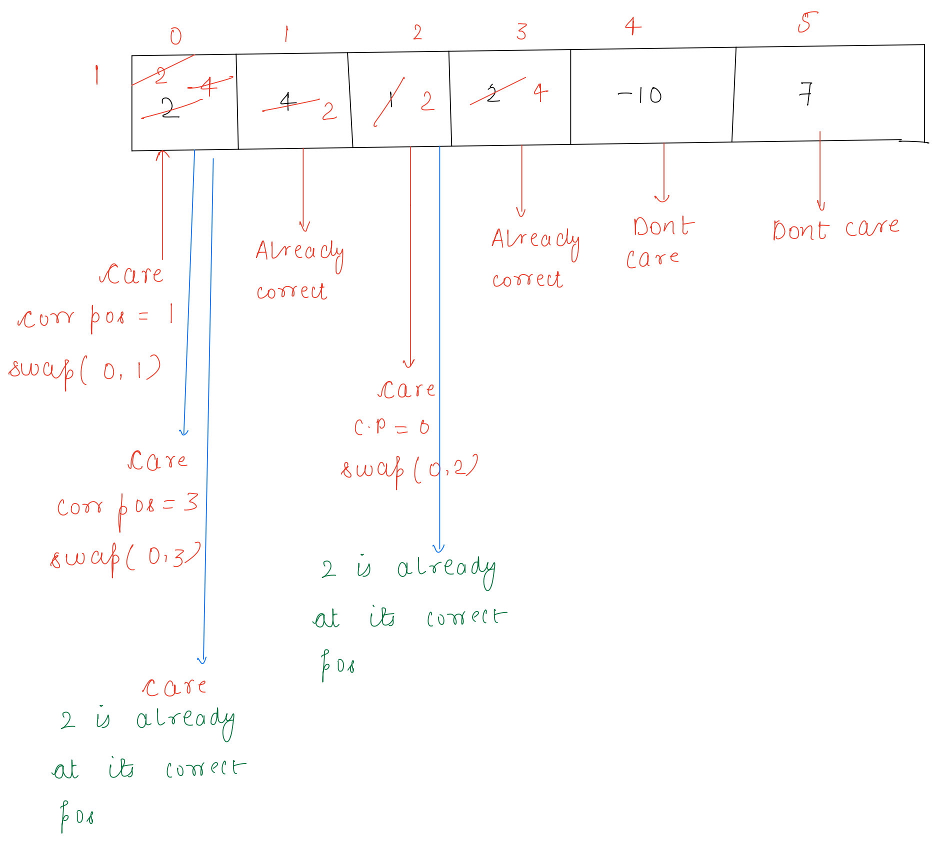


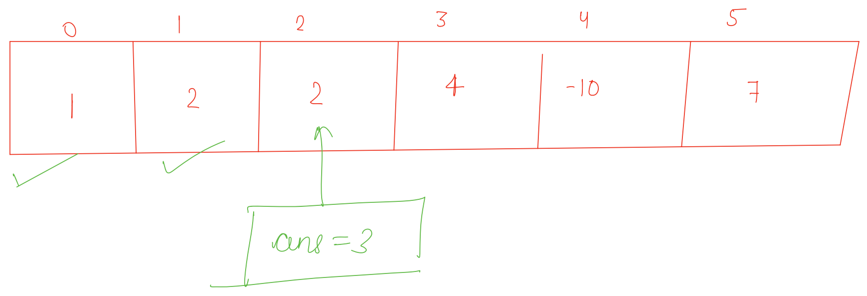
arr[n] ⟶ any el from 1 ~~to~~ n should be placed at idx (el − 1).

Example:






0 1 2 3 4 5
1
~~2~~ ~~4~~
~~2~~
~~4~~ 2
~~1~~ 2
~~2~~ 4
-10
7
care
corr pos = 1
swap(0, 1)
Already correct
Already correct
Dont care
Dont care
care
C.P = 0
swap(0,2)
care
corr pos = 3
swap(0,3)
2 is already at its correct pos
care
2 is already at its correct pos


0 1 2 3 4 5
1
2
2
4
-10
7
ans = 3

Algorithmic code

```
int firstMissingInterval( int[] arr) {
    int i=0;
    while( i < arr.length) {
        int correctIdx = arr[i] - 1;
Elements I care  ——  if( arr[i] >=1 && arr[i] <=n) {
about
        Check if el is  — if( arr[correctIdx] != arr[i]) {
        at its correct
        pos or not?              swap( arr, i, correctIdx);

                          } else { ⟶ el is at its corr
                             i++;       position
                          }
Elements you  ————} else{
dont care
                       i++;
                  }
    }

    for(i=0; i< arr.length; i++) {
        if ( i+1 != arr[i]) {
            return i+1;
        }
    }
    return arr.length +1;
}
```

TC: O(n)
SC: O(1)

Thankyou