Lecture :- Arrays 3

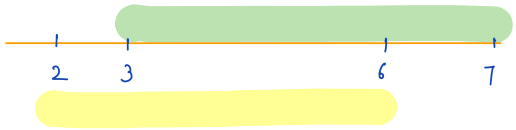
Agenda

- Merge intervals ✓
- Merge overlapping intervals ✓
- first missing +ve no. ✓
- Search an el in a matrix. [Refer from intermediate module]

class starts at 7:05 AM

Qu1 Merge intervals.

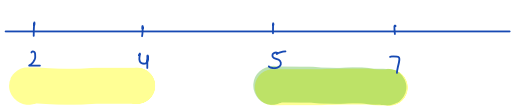
(2, 6) (3, 7)
s1 e1 s2 e2



Overlapping

Merge interval: 2 , 7
↓ ↑
min (s1, s2) max(e1, e2)

(2, 4) (5, 7)



Non-overlapping

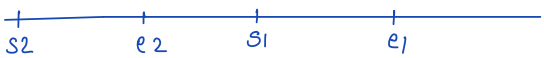
(2, 8) (4, 7)

overlapping

(s1, e1) and (s2, e2)

s2 >= e1 (Non-overlapping)

e2 <= s1 (Non-overlapping)



Given arr[n] of intervals in sorted manner and non-overlapping.

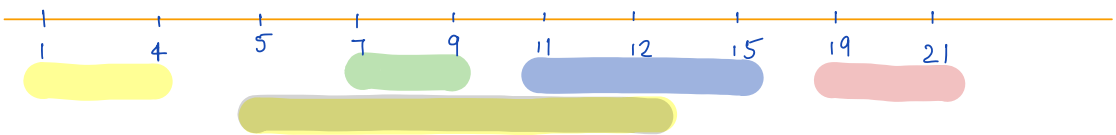
Given one extra interval.

Return new arr with inserting the extra interval

Example: arr[4] = [(1,4) (7,9) (11,15) (19,21)] Non-overlapping

interval = (5, 12)

updated array = [ ]



(1,4) (5,15) (19,21) Ans [Non-overlapping]

Idea 1. Non-overlapping

a. (3,6) (7,8) → (7,8) lies on right of (3,6)
arr interval interval

b. (3,6) (1,2) → (1,2) lies on left of (3,6)
arr[i] interval:

2. overlapping

(3,6) (5,8)
arr[i] interval

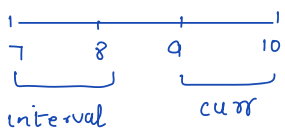
Example:

		new interval	op array
[1, 3]	non-overlapping (12,22) right of [1,3]	[12, 22]	[1,3]
[4, 7]	non-overlapping right	[12, 22]	[4,7]
[10, 14]	overlapping Merge = [10, 22]	[12, 22]	
[16, 19]	overlapping merge = [10, 22]	[10, 22]	
[21, 24]	overlapping merge = [10, 24]	[10, 22]	
[27, 30]	non-overlapping	[10, 24]	[10, 24]
[32, 35]			[27, 30]
			[32, 35]

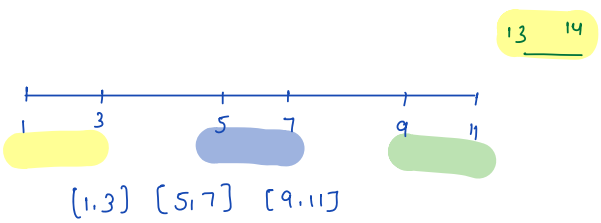
↓ non-overlapping

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

```
List<Interval> merge(List<Interval> arr, Interval interval) {

    List<Interval> ans = new ArrayList<>();

    for(int i=0; i < arr.size(); i++) {          —— O(n)

        Interval curr = arr.get(i);

        if(interval.start >= curr.end) {

            ans.add(curr);

        } else if(curr.start >= interval.end) {

            ans.add(interval);

            while(i<n) {

                ans.add(arr.get(i));

                i++;

            }

            return ans;

        }

        else {          // overlapping interval

            // Merge the intervals.

            interval.start = min(interval.start, curr.start);

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

        }

    }

    ans.add(interval);

    return ans;

}
```



TC O(n)
SC: O(1)



Qu: Given arr[n], merge all overlapping intervals.

Ex: arr: [2,6] [15,18] [1,3] [8,10]

sort on basis on start

[1,3] [2,6] [8,10] [15,18]

op: [1,6] [8,10] [15,18] Ans

Assignment: Do the sorting.

[1,3] [2,6] [8,10] [15,18] [17,20] [19,26] [25,30] [40,50]

cs = 2 cs = 8 cs = 15 cs = 17 cs = 19 cs = 25 cs = 40

ce = 6 ce = 10 ce = 18 ce = 20 ce = 26 ce = 30 ce = 50

s = ~~1~~ ~~3~~ ~~15~~ 40

e = ~~1~~ ~~6~~ ~~18~~ ~~20~~ ~~26~~ ~~30~~ 50

outside the for loop.

[(1,6), (8,10) (15,30), (40,50)]

```
List<Interval> mergeOverlapping(List<Interval> arr) {
    // sort this array.
    List<Interval> ans = new ArrayList<>();

    int s = arr.get(0).start;

    int e = arr.get(0).end;

    for(i=1; i< arr.size(); i++) {

        // overlapping

        int cs = arr.get(i).start;

        int ce = arr.get(i).end;

        if(    overlapping condition                    ) {

                s = min(s, cs);

                e = max(e, ce);

        } else {

                ans.add(new Interval(s, e));

                s = cs;

                e = ce;

        }

    }

    ans.add(new Interval(s, e));

    return ans;

}
```

TC: O(n log n)

SC: O(1)

Break: 8:47 AM

Qu Given arr[n], find first missing +ve no.

[3, -2, 1, 2, 7] ans = 4.

[1 2 5 6 4 3] ans = 7

Range of ans = max => len of arr + 1.

Brute force: Traverse from 1 to n+1 [i] —

check if arr contains i —

if does not contain [return i]

```
int firstMissing ( int[] arr) {
    int no = 1;
    while ( no <= arr.length + 1) {
        boolean isfound = false;
        for ( int el : arr) {
            if ( el == no) {
                isfound = true;
            }
        }
        if ( !isfound) {
            return no;
        }
        no++;
    }
}
```

TC: $O(n^2)$

SC: $O(1)$

Approach 2: Search the el using hashset.

TC: O(n)

SC: O(n)

Approach3

TC: O(n)

SC: O(1) ⟶ use given array for manipulation

arr:

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

size = 7

arr[i] ⟶ idx = arr[i] - 1

Idea: Every el should be at its correct idx.

1 <= arr[i] <= arr.length [care about)

Example:

0	1	2	3	4	5	6
~~-1~~ 1	0	3	~~8~~ 4	9	~~4~~ 6	~~1~~ -1

ans = 2

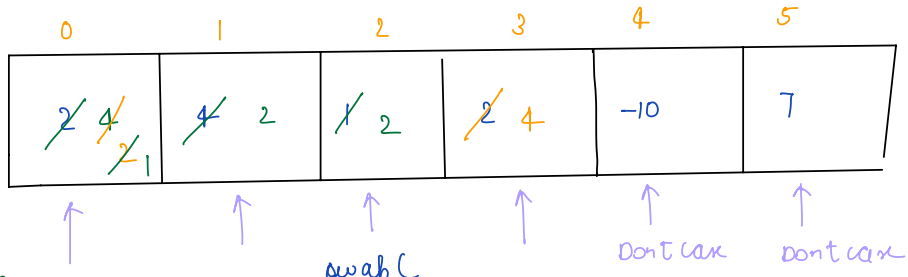
Dont care | Dont care | care ↓ correct position | care swap(3rd, 5th) 4 is at its correct position | care correct position | care swap(6th, 0th)

0	1	2	3	4	5	6
1	0	3	4	9	6	-1

index + 1 = arr[index]

return 2.

Ex2:

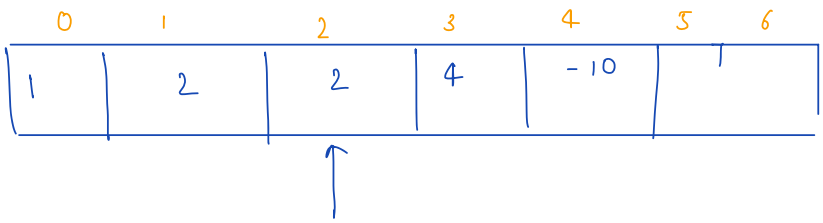


swap (0th, 1st)

swap (0th, 3rd)

swap (0th, 1st) [Dont swap]

2 is at its correct position already



return 3

```
int firstMissingInteger(int[] arr) {
    int i=0;
    while( i < n ) {
        int correct_idx = arr[i]-1;
        if( arr[i] >=1 && arr[i]<=n) {
            if(arr[correct_idx] != arr[i]) {
                swap( arr, correct_idx, i);
            } else {
                i++;
            }
        } else {
            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 ☺