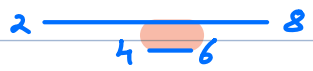
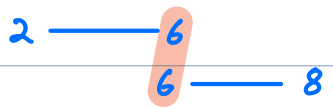
(start <= end)



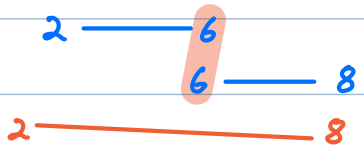


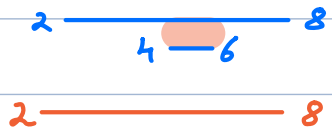
non-overlap

Non-Overlapping (s_1, e_1) (s_2, e_2)

→ $((s_2 > e_1) \,||\, (s_1 > e_2))$

Merge 2 overlapping intervals





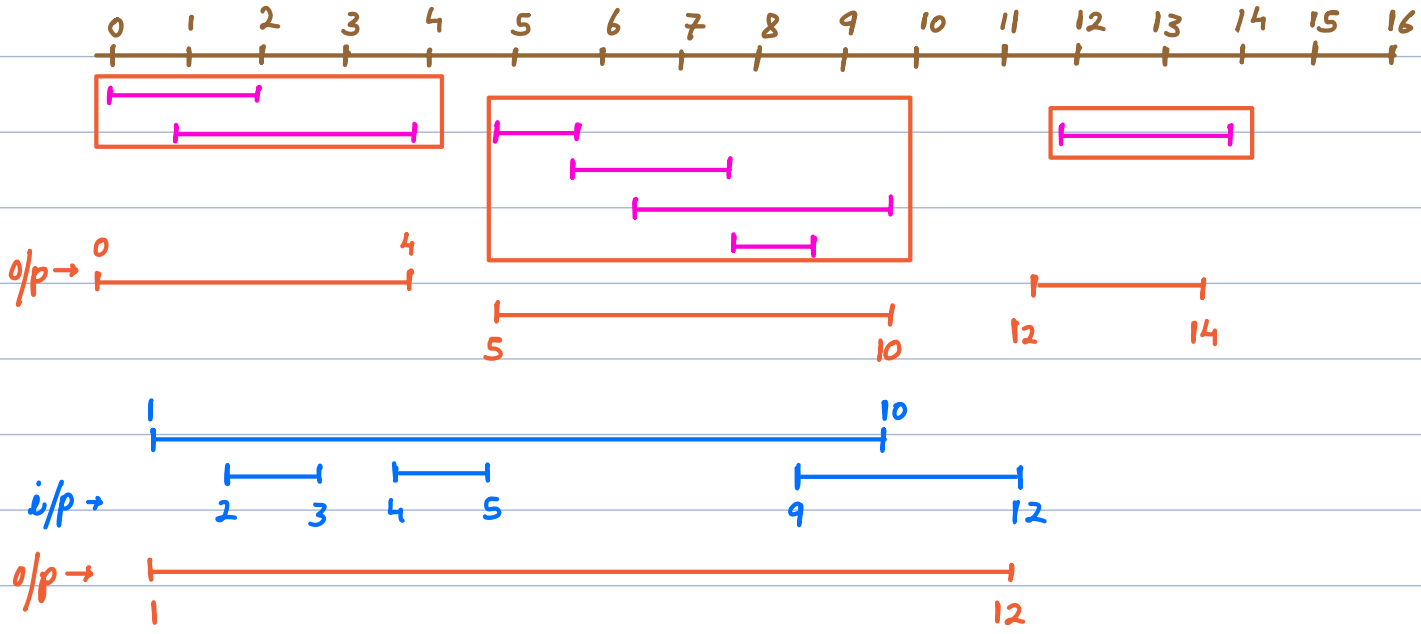
start = min(s1, s2)

end = max(e1, e2)

Q → Given a list of intervals (start <= end), sorted wrt start time. Merge all overlapping intervals & return the result of non-overlapping intervals in sorted order wrt start time.

i/p →

	0	1	2	3	4	5	6
S = [	0	1	5	6	7	8	12]
E = [	2	4	6	8	10	9	14]



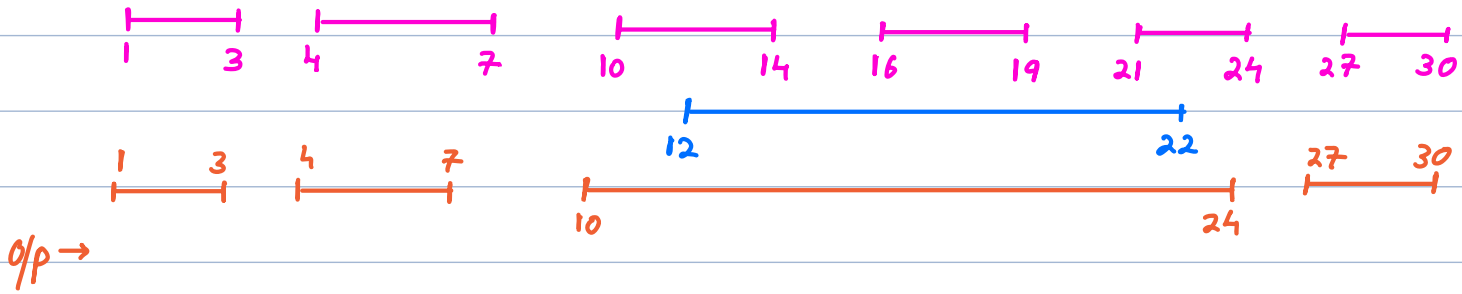

```
start = S[0]    end = E[0]   // current interval
for i → 1 to (N-1) {
    if (S[i] <= end) {        // check if i^th interval overlaps
        end = max(end, E[i])
    } else {
        print(start + " " + end)
        start = S[i]    end = E[i]
    }
} print(start + " " + end)
```

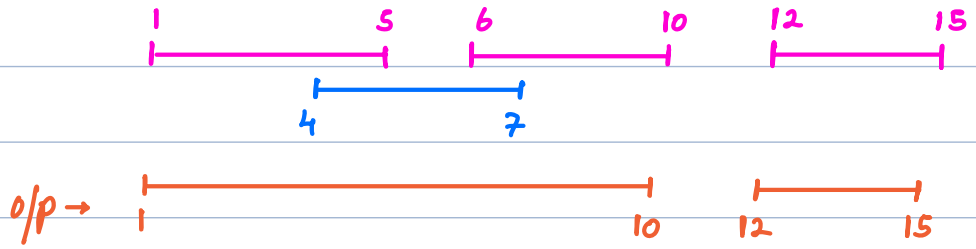
TC = O(N)

SC = O(1)

Q → Given a set of non-overlapping intervals, sorted wrt start time. Insert the given new interval s.t. final list is sorted & non-overlapping.

	0	1	2	3	4	5	
S = [	1	4	10	16	21	27]	L = 12 R = 22
E = [	3	7	14	19	24	30]	





S = [ ] E = [ ] L = 2 R = 4

```
if (N == 0) {
    print (L + " " + R)
    return
}
if ( R < S[0]) {          // L — R  S[0]  E[0]  . . .
    print (L + " " + R)
    for i → 0 to (N-1) {
        print (S[i] + " " + E[i])
    }
    return
} if ( L > E[N-1]) {      // . . .  S[N-1]  E[N-1]  L — R
    for i → 0 to (N-1) {
        print (S[i] + " " + E[i])
    }
    print (L + " " + R)
}
```

```
i = 0
while ( i <= N-1) {   // intervals on left of L — R
    if (E[i] < L) {
        print (S[i] + "  " + E[i])
    } else { break }
    i++
} // 0 — (i-1) ✓
start = L      end = R
while ( i <= (N-1)) {   // overlapping L — R
    if (!(S[i] > end || start > E[i])) {
        start = min (start, S[i])   end = max (end, E[i])
    } else { break }
    i++
} // i —— (i-1)  → merged
print ( start + " " + end)
while ( i <= (N-1)) {   // after L — R
    print (S[i] + "  " + E[i])
    i++
}
```

	0	1	2	3	4	5	
S = [	1	4	7	16	21	27]	L = 8
E = [	3	6	14	19	24	30]	R = 22
						i	

1 — 3

4 — 6

7 — 24

27 — 30

st = ~~8~~ 7

end = ~~27~~ 24

TC = O(N) SC = O(1)

Q → Given an integer array, find the first missing positive integer.

A = [3 ✓ -2 1 ✓ 2 ✓ 7] Ans = 4

A = [-2 -8 0 4] Ans = 1

A = [5 3 1 -1 -2 -4 7 2] Ans = 4

Bruteforce → check every number from 1 to (N+1).

TC = O(N²) SC = O(1)

A = [1 2 3] Ans = 4

Sort →

```
          0   1   2   3   4   5
A = [ -8  -2   1   2   4   8 ]
      ⟶     ↑   ↑   ↑x
            1   2   3 ✓
```

TC = O(N log(N)) SC = O(1)

A = [5 3 1 -1 -2 -4 7 2] SC = O(1)

Keep track of numbers from 1 to N

for any number 1 to N swap it to index 0 to (N-1)

ind

1 ⟶ 0
2 ⟶ 1
3 ⟶ 2
⋮

A = [5 3 1 -1 -2 5 7 2 6]
0 1 2 3 4 5 6 7 8

i

N = 8

-2 1 3 5 6 -2 5

i = 0

1 -2

2

while (i < N){

while (A[i] is [1 N] && A[i] != i+1 && A[i] != A[A[i]-1]){

swap (i, A[i]-1)

}

i++

}

A = [1 2 3 -1 5 6 7 -2 5]

✓ ✓ ✓

4 (Ans)

TC = O(N)

if (A[i] != i+1) = Ans = i+1

SC = O(1)
