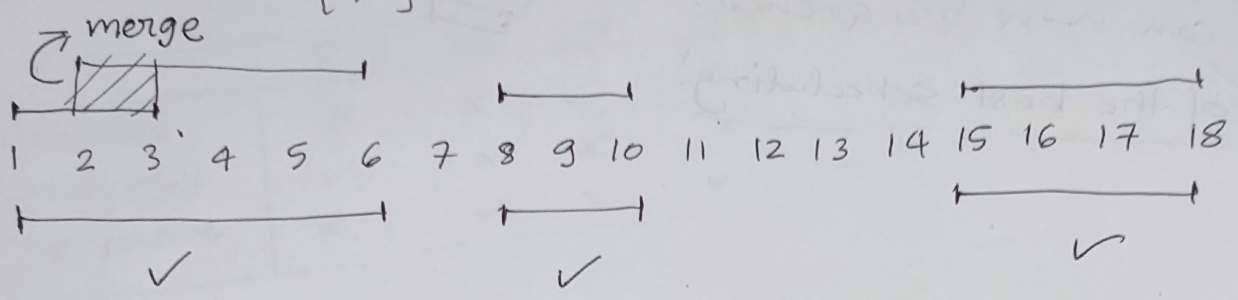
Merge Intervals

ex: i/p: intervals = [[1,3], [2,6], [8,10], [15,18]]

o/p: [[1,6], [8,10], [15,18]]

exp: ∵ intervals [1,3] & [2,6] overlaps, merge to [1,6]

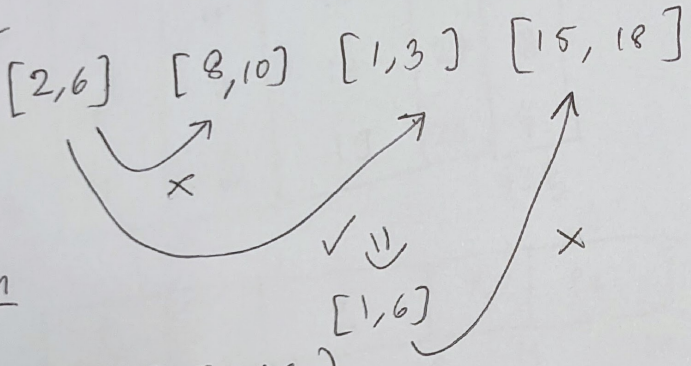


Simmilar Q^{n} (Minimum Platform Required)

Observation

TC. is sorted on basis of 1st interval but isn't nessarary.

Brute:

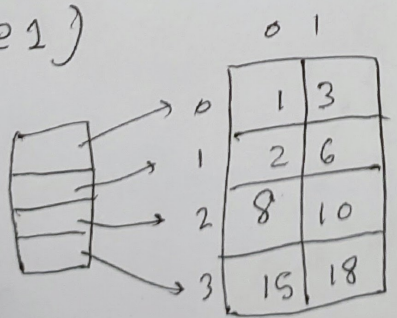


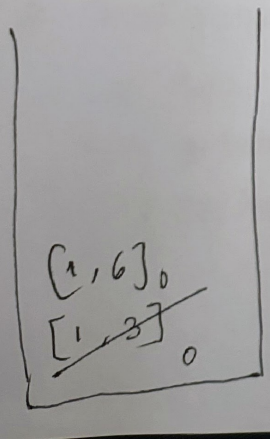
Observation

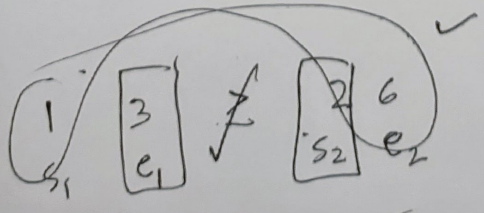
Sort! → ensures (merge) starting is smaller always

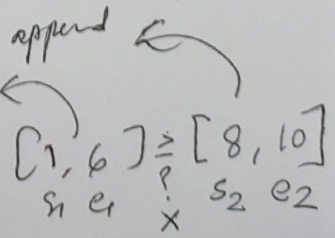
merge when $(e1 \geq s2)$

or $(s2 \leq e1)$









Lambda Functions:

```
int[][] arr = { {1, 4},        <- 2D array
                {5, 7},
                {3, 12},
                {18, 2},
                {6, 10} };

Arrays.sort (arr, (a,b) -> {
        return a[0] - b[0];
});
```

default (inc)

✓ (this - other)

✓ (other - this) → (dec)

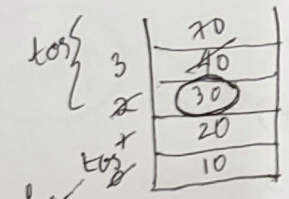
(argument - list) -> {body}

```
class Solution {
  public int[][] merge (int[][] intervals) {
        Arrays.sort (intervals, (a,b) -> {
              return a[0] - b[0];
        });
        Stack <int []> st = new Stack<>();
        st.push (intervals[0]);

        for(int i=1; i < intervals.length; i++) {
            int [] t = st.pop();    // 1   3
            int s1 = t[0];
            int e1 = t[1];

            int s2 = intervals[i][0];    // current interval
            int e2 = intervals[i][1];

            int em = Math.max(e1, e2);
```

```
// merge condition ( s2 <= e1 )
if(s2 <= e1){
    st.push(new
    int[] merge = {s1, em};
    st.push(merge);
} else {
    st.push(x);
    st.push(intervals[i]);
}
}
int[][] ans = new int[st.size()][2];
for(int i = ans.length-1; i>=0; i--){
    int[] tmp = st.pop();
    ans[i][0] = tmp[0];
    ans[i][1] = tmp[1];
}
return ans;
}
```

=> st.push(new int[]{st, em});

Normal Stack (Design)



array as in stack

push(10)
push(20)
push(30)
push(40)

pop() → tos-=1

peek() → arr[tos]

push(70) → arr[tos] = 70 → tos+=1

size = tos + 1;

top of stack

st → underflow

st (empty) → peek or pop → error

tos = -1

st → overflow

st (full) → push → error

tos = data.length - 1

Design / Implementation of Stack using Array. (Normal Stack)

```
public static class CustomStack {

    int[] data;
    int tos; // top of stack

    public CustomStack(int cap) {
        data = new int[cap];
        tos = -1;
    }

    int size() {
        return tos + 1;
    }

    void push(int val) {
        if (tos == data.length - 1) {
            System.out.println("Stack overflow");
        } else {
            tos++;
            data[tos] = val;
        }
    }

    int pop() {
        if (tos == -1) {
            System.out.println("Stack underflow");
            return -1;
        } else {
            int value = data[tos];
            tos--;
            return value;
        }
    }

    int top() {
        if (tos == -1) {
            System.out.println("St overflow");
            return -1;
        } else { return data[tos];
    }
}
```

```
void display() {
    for (int i=tos; i>=0; i--) {
        Sopln (data[i] + " ");
    }
    sopln();
  }
}
```

— < driver - code > —

Dynamic Stack

```
// Change code of push fn
void push (int val) {
    if (tos == data.length-1) {
        Sopln ("St overflow");
    } else {
        tos++;
        data[tos] = val;
    }
}
```

stack condition full!

double the size

```
void push(int val){
    if( tos == data.length-1){
        int[] arr = new int[2*data.length];
        for(i=0; i<data.length; i++){
            arr[i] = data[i];
        }
        data = arr; // shallow copy. (reference changed)
        push(val);
    } else {
        tos++;
        data[tos] = val;
    }
}
```