Greedy Algorithms

we will start by 9:10.

WHAT ⟹ local optimal choice at each step helps solving global optimum (minima/ maxima)

APPLICATIONS

1. Fractional Knapsack
2. Activity Selection / Overlapping Intervals / Meeting Rooms
3. Minimum Spanning Tree (Prims (PQ), Kruskal (DSU))
4. Huffman Encoding & Decoding / optimal Merge Pattern
5. Single Source Shortest Path Algo {DIJKSTRA}
6. Job Sequencing
7. Problems based on SORTING. (nlogn)

Lecture ①

1. Job Sequencing
2. Meeting Rooms - I
3. Disjoint Intervals - I
4. Disjoint Intervals - II
5. Maximum Chain Length
6. Minimum Balloon Burst

Saturday Morning

Lecture ②

1. Meeting Rooms - II
2. Minimum Platforms
3. Car Pooling
4. Merge overlapping Intervals
5. Insert Interval
6. Intervals Intersection

Saturday Evening

lecture (3)

(1) Circular Tour (salesforce)

(2) Car Fleet (sprinklr)

(3) Two City Scheduling

(4) Candy/Temple offerings

(5) Chocolate Distribution

(6) Queue Reconstruct Height

Sunday Morning

lecture (4)

(1) Biased Standings

(2) Defense Kingdom

(3) Georgivia

(4) Amplifiers

(5) load Balancing

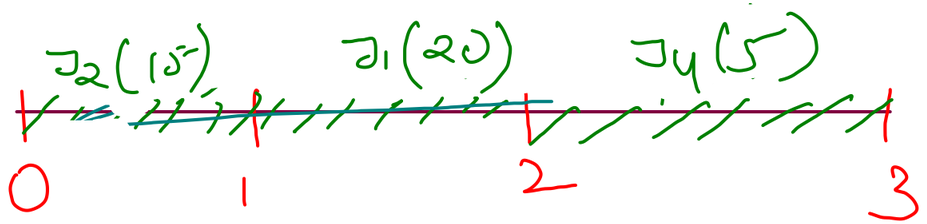
Huffman Coding & Decoding

Sunday Evening

Job Sequencing

1 job → 1 unit of time
Maximum profit

n=5	i⇒ 0	1	2	3	4
Jobs	J_1	J_2	J_3	J_4	J_5
profits	20	15	10	5	1
deadlines	2	2	1	3	3



(1) Order of picking jobs
⇒ decreasing order of profit

(2) Picked job should be placed as last as possible

JA : 1 2 3
Profit = 20 + 40

```java
public static class MyComparator implements Comparator<Job>{
    public int compare(Job obj1, Job obj2){
        if(obj1.profit != obj2.profit){
            return obj2.profit - obj1.profit;
        }
        return obj2.deadline - obj1.deadline;
    }
}

//Function to find the maximum profit and the number of jobs done.
int[] JobScheduling(Job arr[], int n)
{
    Arrays.sort(arr, new MyComparator()); ⇒ nlogn
    int maxDeadline = 0;
    for(int i=0; i<n; i++){
        maxDeadline = Math.max(arr[i].deadline, maxDeadline);
    }

    boolean[] slots = new boolean[maxDeadline];
    int maxProfit = 0;
    int jobsAllocated = 0;

    for(int i=0; i<n; i++) {
        for(int j=arr[i].deadline-1; j>=0; j--){
            if(slots[j] == false){
                slots[j] = true;
                jobsAllocated++;
                maxProfit += arr[i].profit;
                break;
            }
        }
    }

    return new int[]{jobsAllocated, maxProfit};
}
```

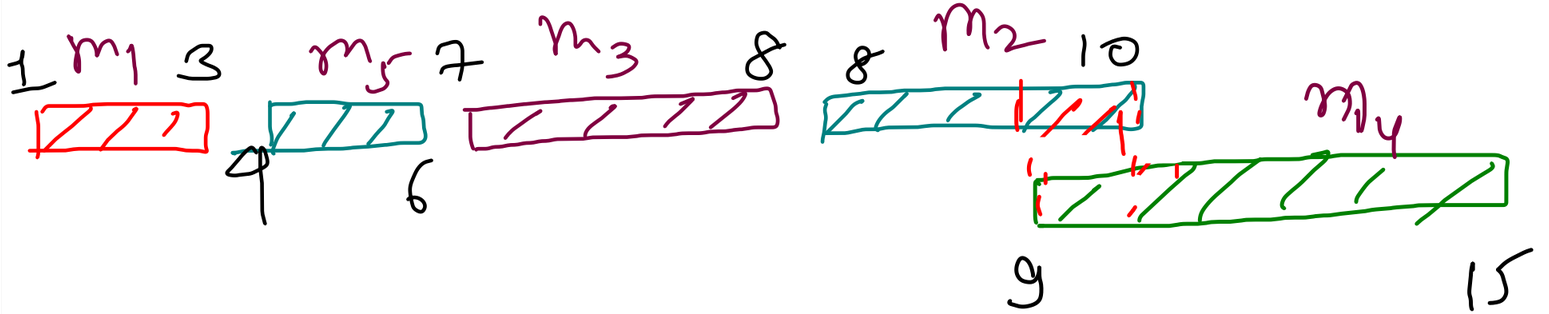
d * n
max deadline

O(dn + nlogn)

Meeting Rooms - I

One room ⇒ one meeting at a time

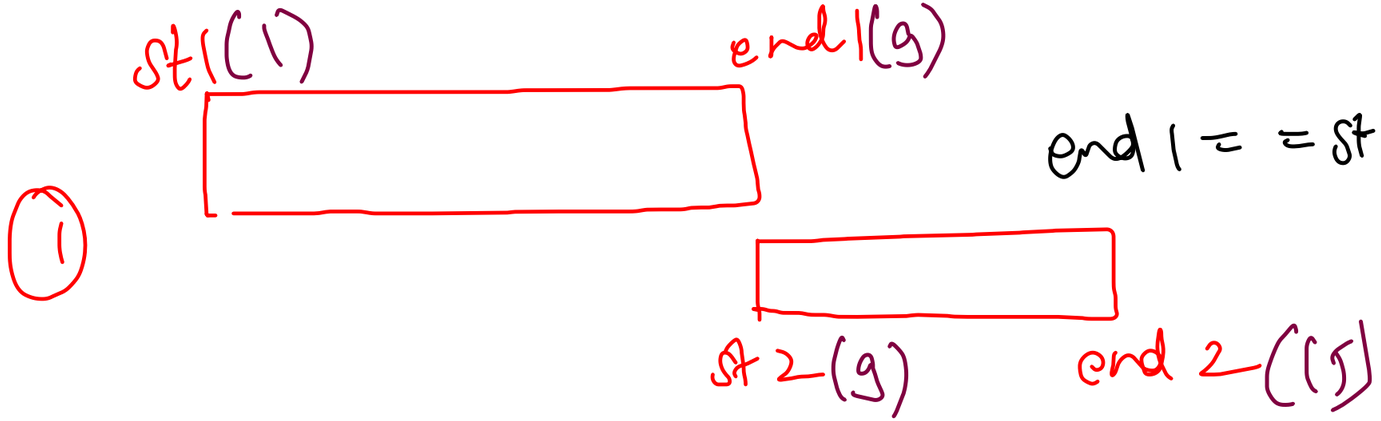
		0	1
		st	et
①	m_1	1	3
②	m_2	8	10
③	m_3	7	8
④	m_4	9	15
⑤	m_5	4	6



① Sort intervals based on ending index

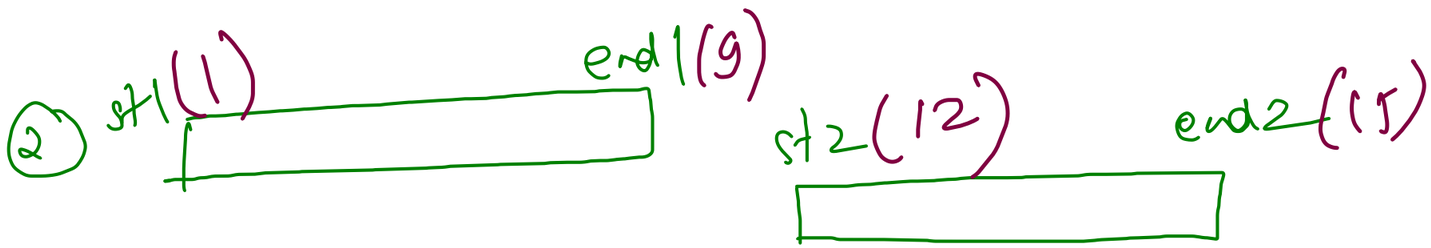
② Check overlapping with previous slot.

false

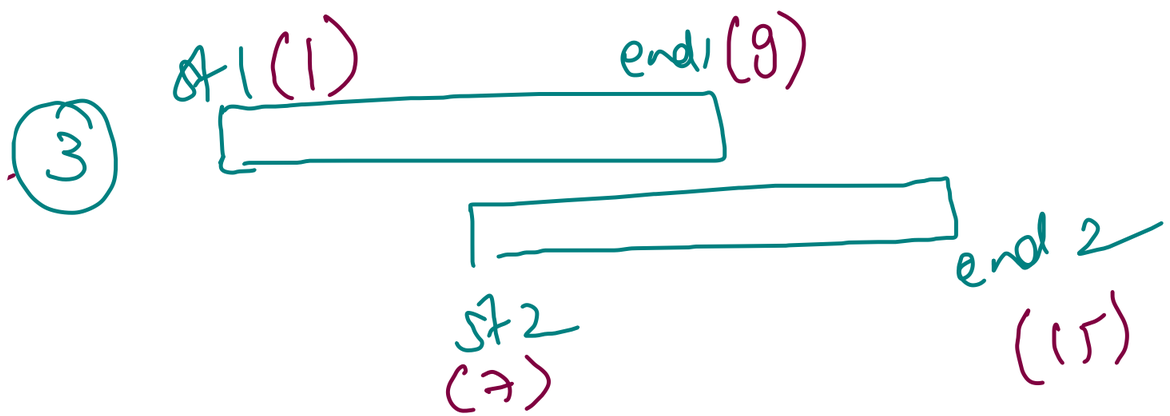


end1 == st2 (T)

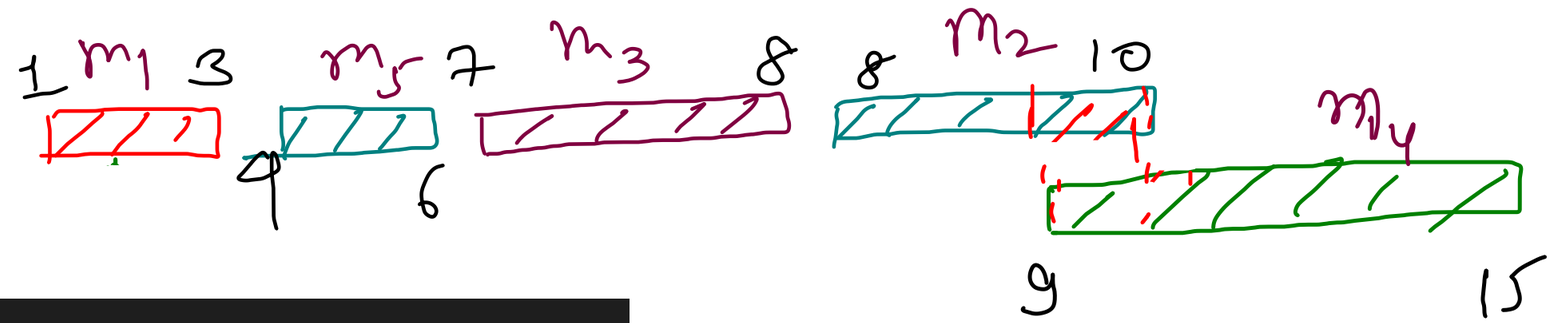
end1 ≤ end2



end1 < st2 (T)



end1 > st2 (F)

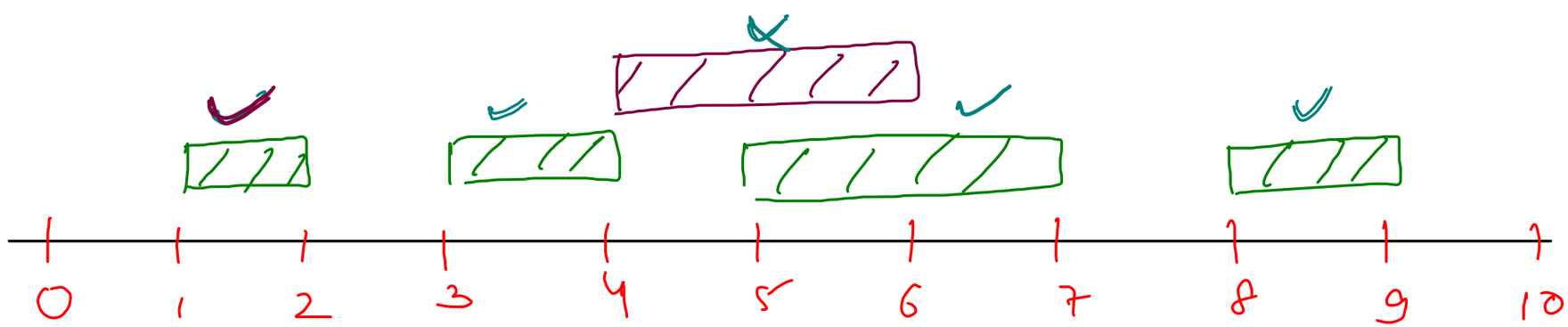



```java
public class Solution {
    public static class MyComparator implements Comparator<Interval>{
        public int compare(Interval obj1, Interval obj2){
            if(obj1.end != obj2.end)
                return obj1.end - obj2.end;
            return obj1.start - obj2.start;
        }
    }

    public boolean canAttendMeetings(List<Interval> intervals) {
        Collections.sort(intervals, new MyComparator());

        int limit = Integer.MIN_VALUE; // last interval's ending time

        for(int i=0; i<intervals.size(); i++){
            if(limit > intervals.get(i).start){
                return false;
            }
            limit = intervals.get(i).end;
        }

        return true;
    }
}
```

$\Rightarrow O(N \log N)$

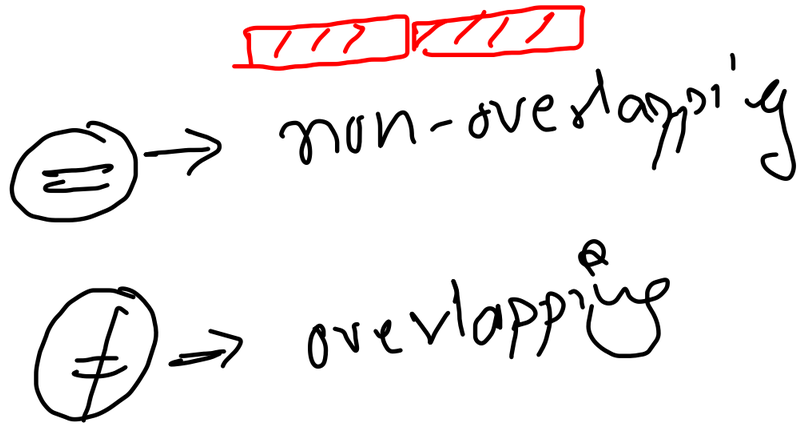
$\Rightarrow O(N)$

S[] =	1	0	3	8	5	~~8~~ 4
F[] =	2	~~6~~ 8	4	9	7	~~7~~ 6
	1	2	3	4	5	6

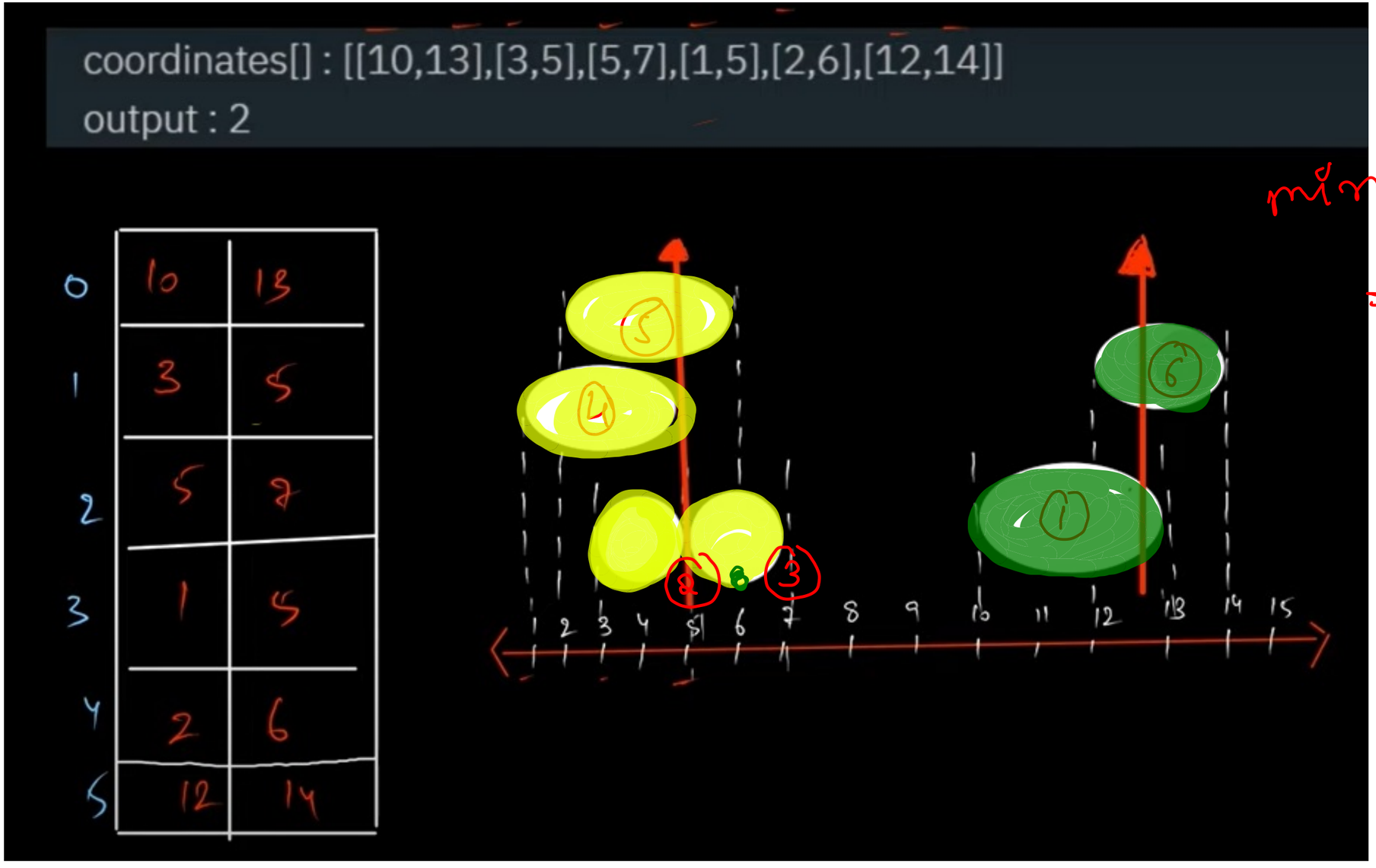


```
int limit = Integer.MIN_VALUE; // last interval's ending time
int count = 0;

for(int i=0; i<n; i++){
    if(limit < intervals[i].start){
        count++;
        limit = intervals[i].end;
    }
}

return count;
```

end2 == 8+1



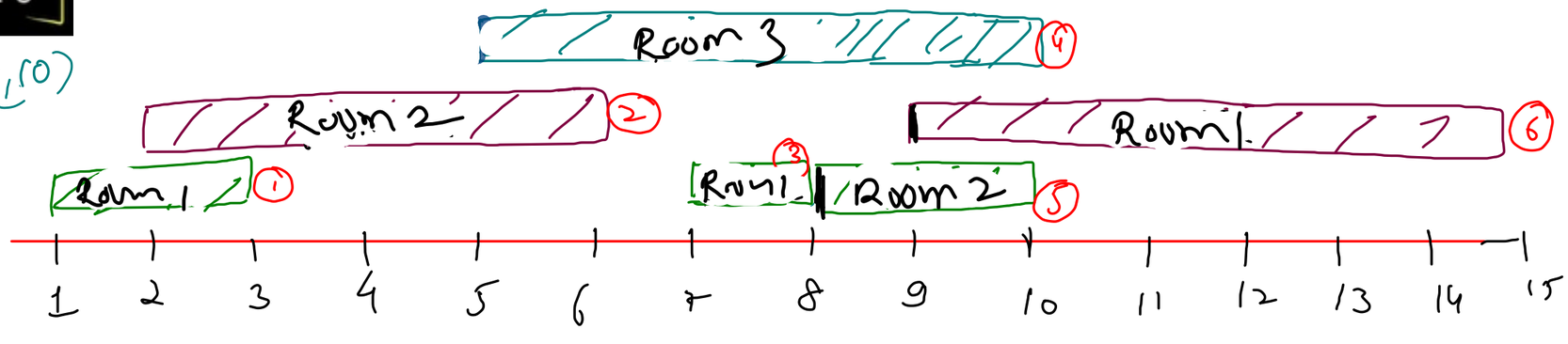
Minimum Balloon Bursts



minimum arrow = maximum meetings that can be accomodated in one room

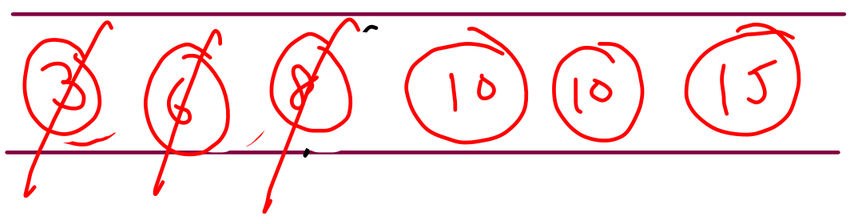
minimum rooms

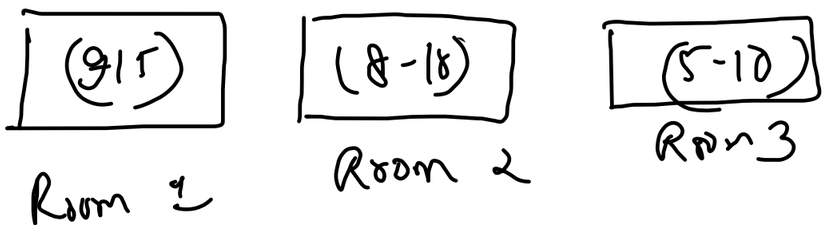
```
5
1 3
8 10
7 8
9 15
2 6
```



(5,10)

Count of Rooms = ~~0~~ ~~1~~ ~~2~~ 3

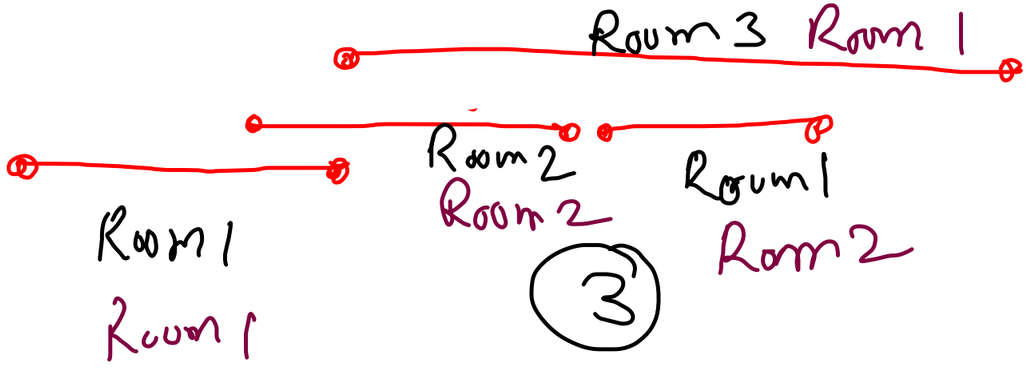




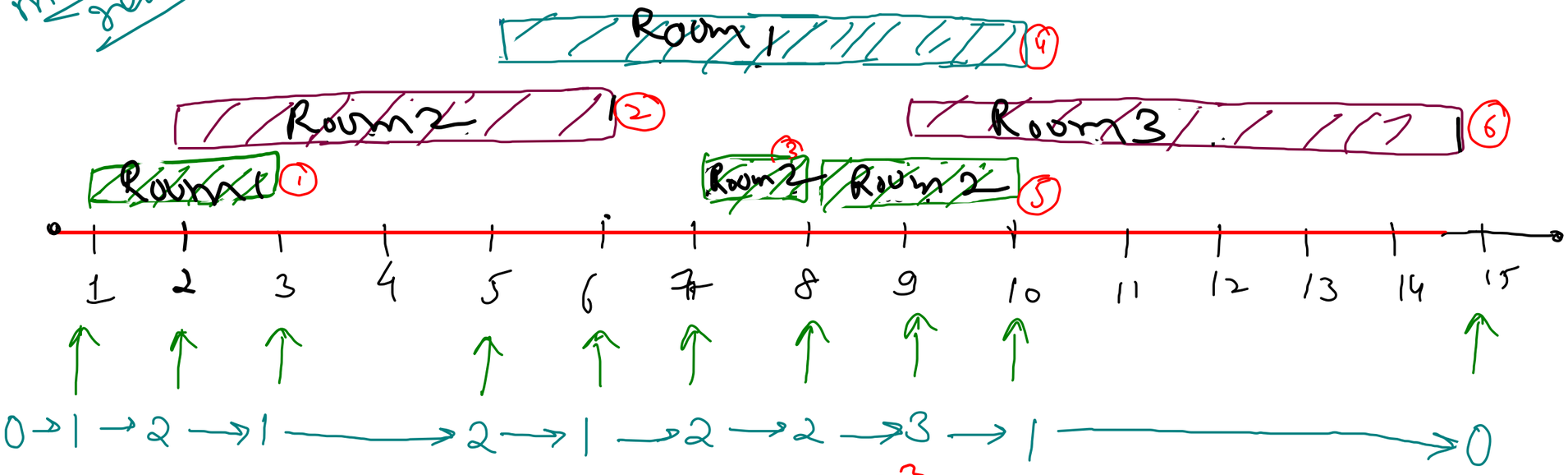

```java
public static int meetingRooms(int intervals[][]) {
    int maxRooms = 0;
    Arrays.sort(intervals, (a, b) -> a[1] - b[1]);

    Queue<Integer> q = new ArrayDeque<>();

    for(int i=0; i<intervals.length; i++){
        if(q.size() == 0 || q.peek() > intervals[i][0]){
            maxRooms++;
        } else {
            q.remove();
        }

        q.add(intervals[i][1]);
    }

    return maxRooms;
}
```

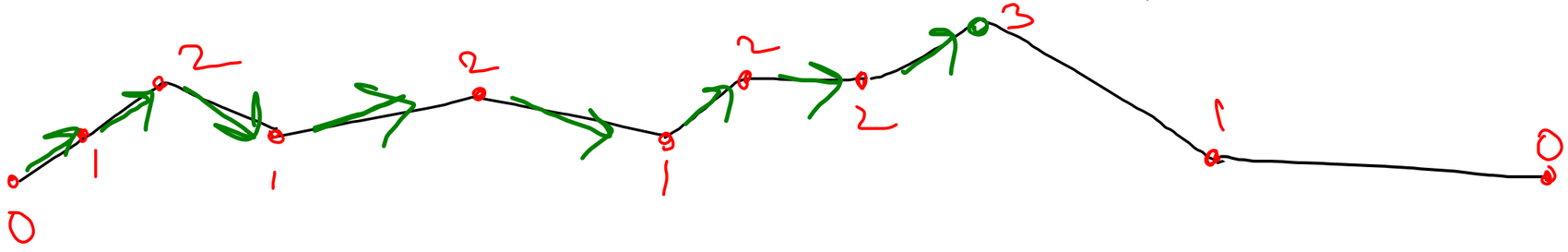
Time → O(NlogN + N)

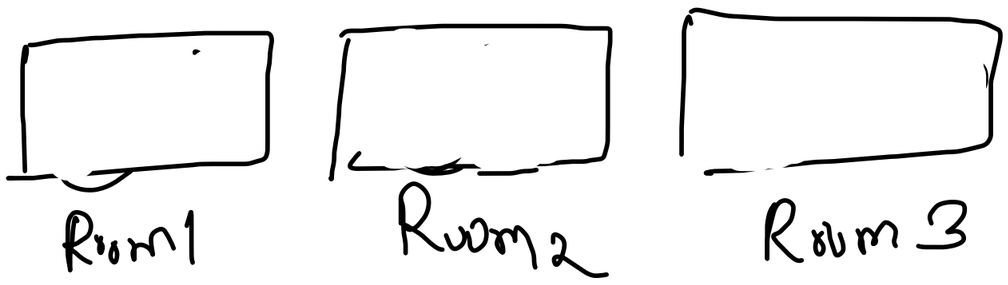
Space → O(N)


Room 3
Room 1
Room 2
Room 2
Room 1
Room 2
Room 1
Room 1
3

minimum rooms







1 2 5 8 9

3 6 8 10 15

```java
public int minMeetingRooms(List<Interval> intervals) {
    ArrayList<Integer> start = new ArrayList<>();
    ArrayList<Integer> end = new ArrayList<>();
    for(int i=0; i<intervals.size(); i++){
        start.add(intervals.get(i).start);
        end.add(intervals.get(i).end);
    }

    Collections.sort(start);
    Collections.sort(end);

    int currentRooms = 0, maxRooms = 0;
    int startIdx = 0, endIdx = 0;

    while(startIdx < intervals.size()){
        if(start.get(startIdx) < end.get(endIdx)){
            startIdx++;
            currentRooms++;
        } else if(end.get(endIdx) < start.get(startIdx)){
            endIdx++;
            currentRooms--;
        } else{
            startIdx++;
            endIdx++;
        }
        maxRooms = Math.max(maxRooms, currentRooms);
    }
    return maxRooms;
}
```

O(N)

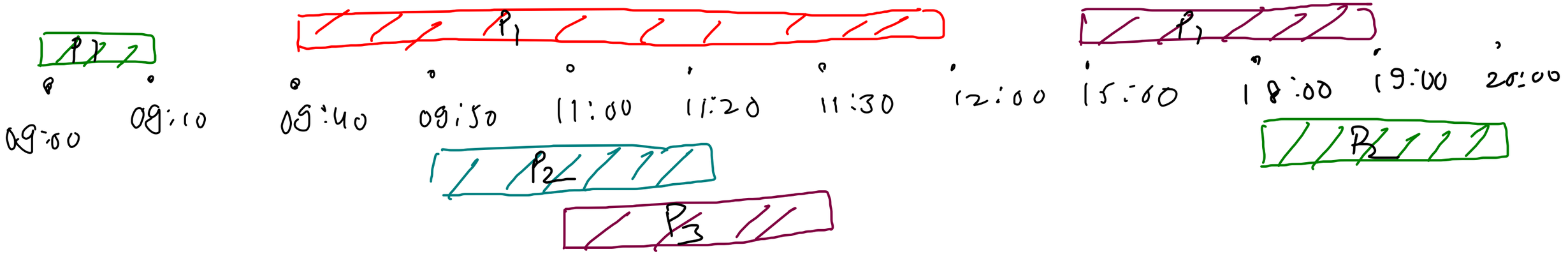
O(NlogN)

O(2N)

Minimum Platforms

{ same as meeting Rooms - II }

```
{0900, 0940, 0950, 1100, 1500, 1800}
{0910, 1200, 1120, 1130, 1900, 2000}
```



Car Pooling

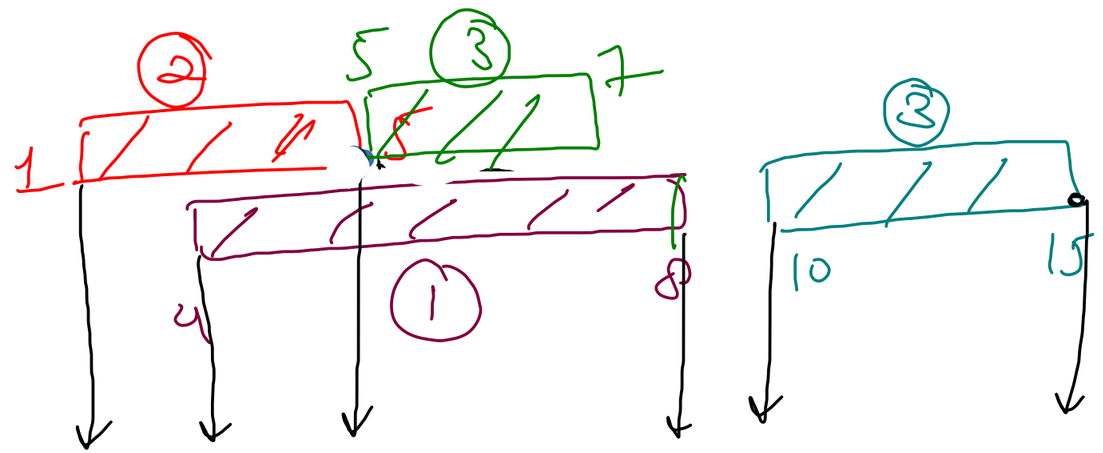
{2, 1, 5}

{1, 4, 8}

{3, 10, 15}

3

direct^n ⇒ left to right , max^m capacity = 3



0 → 2 → 3 → 1 → 0 → 3 → 0

HINT

TreeMap { Red Black Tree }

↓

Ordered (keys)

```
TreeMap<Integer, Integer> changes = new TreeMap<>();
for(int i=0; i<trips.length; i++){
    int passengers = trips[i][0];
    int start = trips[i][1];
    int end = trips[i][2];

    changes.put(start, changes.getOrDefault(start, 0) + passengers);
    changes.put(end, changes.getOrDefault(end, 0) - passengers);
}

int currPassengers = 0;
for(Integer key: changes.keySet()){
    Integer val = changes.get(key);
    currPassengers += val;

    if(currPassengers > capacity) return false;
}
return true;
```

O(N log N)

O(N)

1 → +2

4 → +1

5 → −2

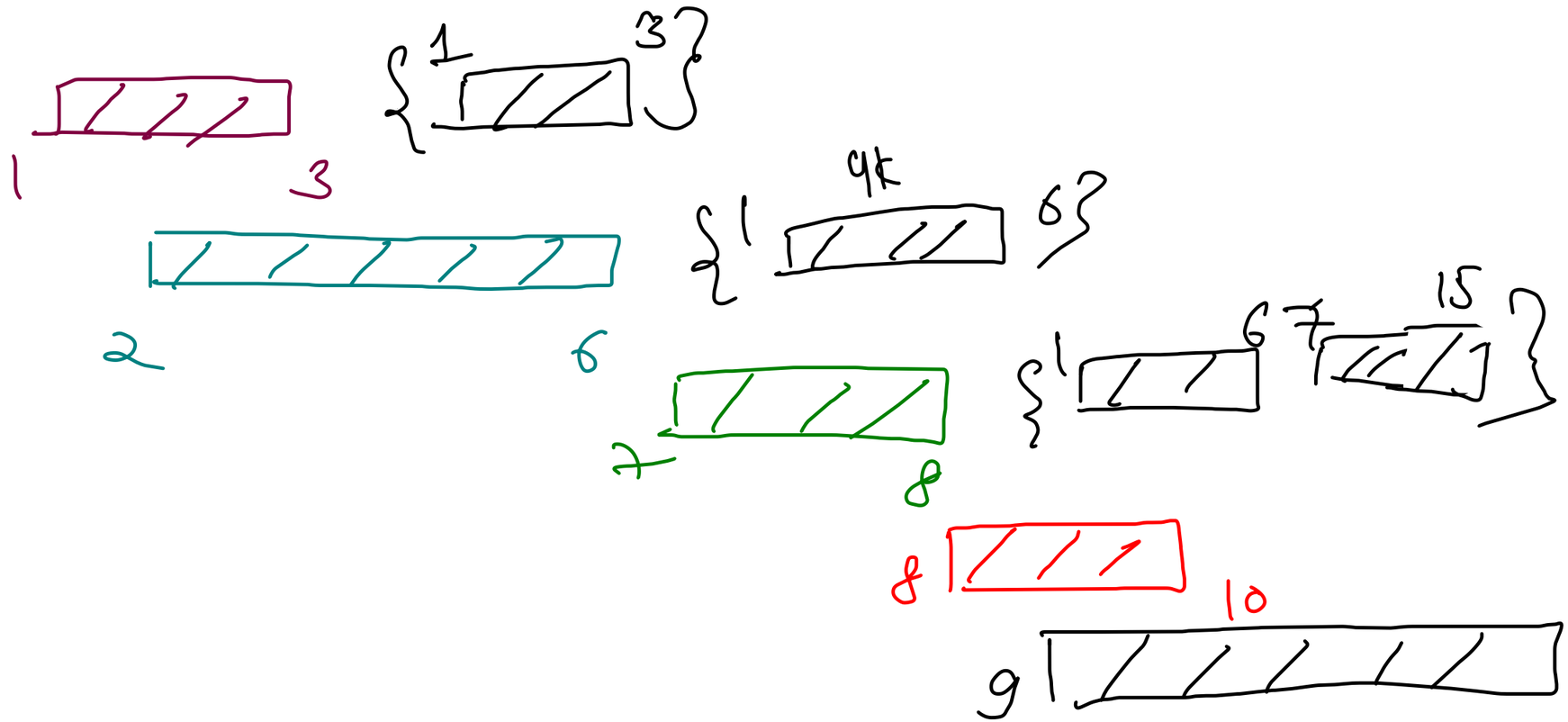
8 → −1

10 → +3

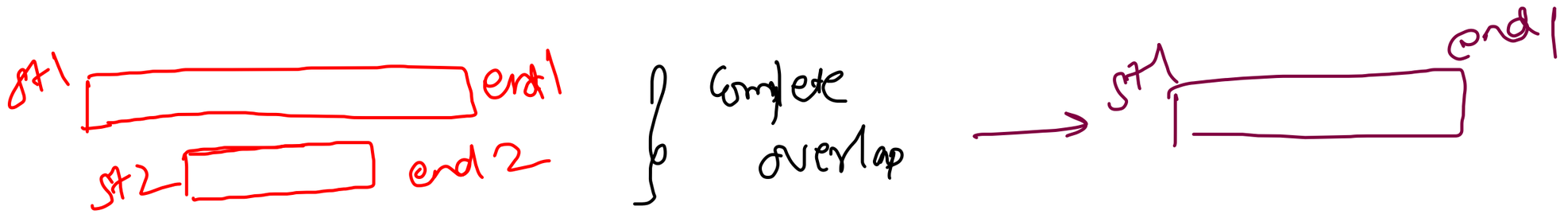
15 → −3

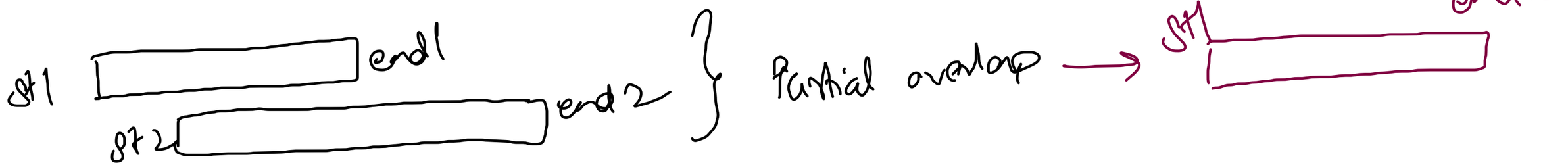
Merge Overlapping Intervals

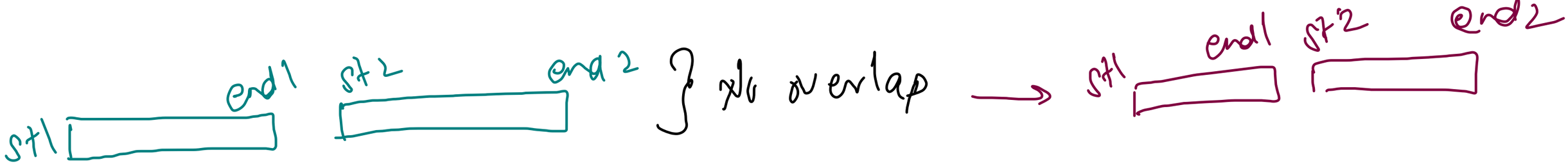
	0	1
0	1	3
1	8	10
2	7	8
3	9	15
4	2	6



Sort on basis of starting index ⇒ st1 <= st2







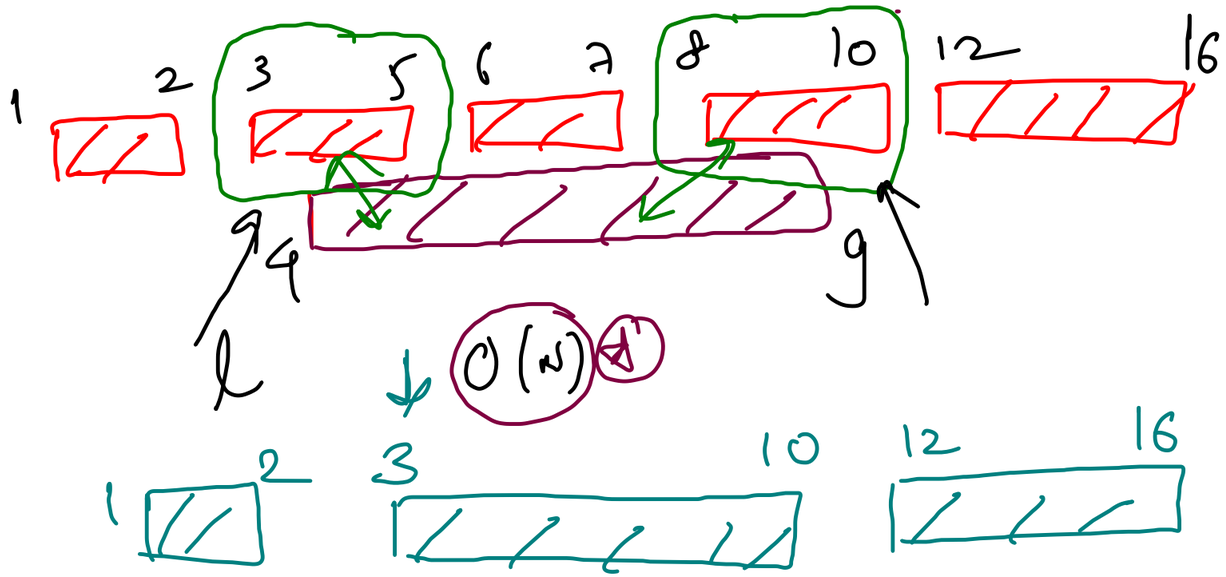
```java
public int[][] merge(int[][] intervals) {
    Arrays.sort(intervals, (a, b) -> a[0] - b[0]);
    ArrayList<int[]> merged = new ArrayList<>();
    merged.add(intervals[0]);

    for(int i=1; i<intervals.length; i++){
        int[] lastInt = merged.get(merged.size() - 1);
        int[] currInt = intervals[i];

        if(lastInt[1] >= currInt[0]){
            // merge
            lastInt[1] = Math.max(lastInt[1], currInt[1]);
        } else {
            merged.add(currInt);
        }
    }

    int[][] res = new int[merged.size()][2];
    for(int i=0; i<merged.size(); i++){
        res[i] = merged.get(i);
    }
    return res;
}
```

TC :→ O(NlogN)

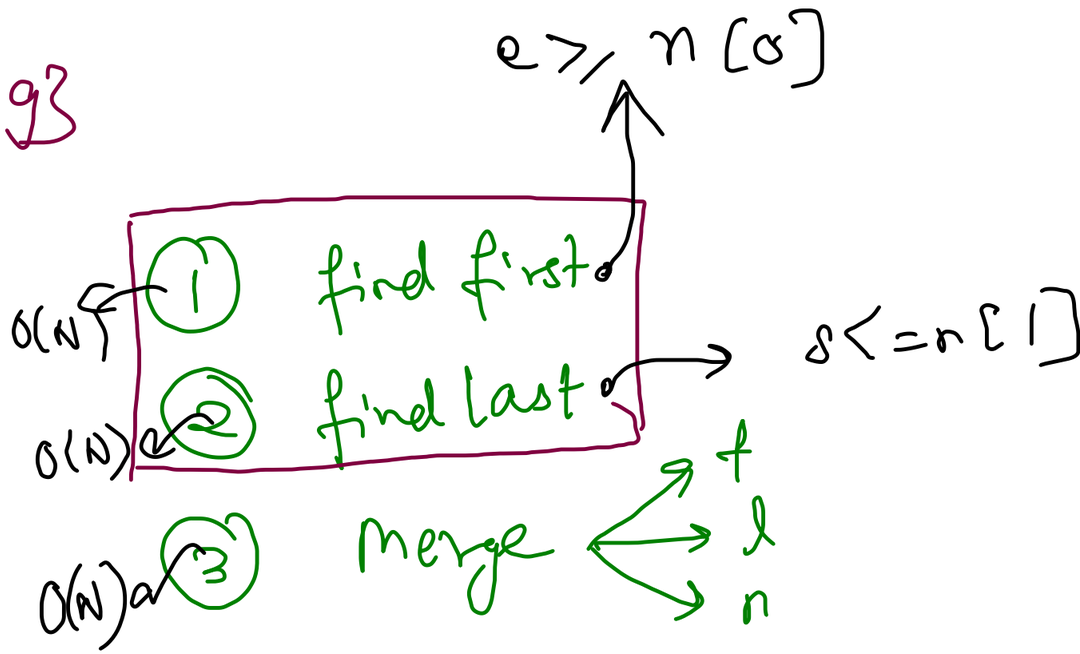
SC
=
O(N) + O(N) + O(1)
output input extra space

Insert Interval

```
[[1,2],[3,5],[6,7],[8,10],[12,16]].
```



24, 93



{ min(f[0], n[0]),
max(l[1], n[1]) }

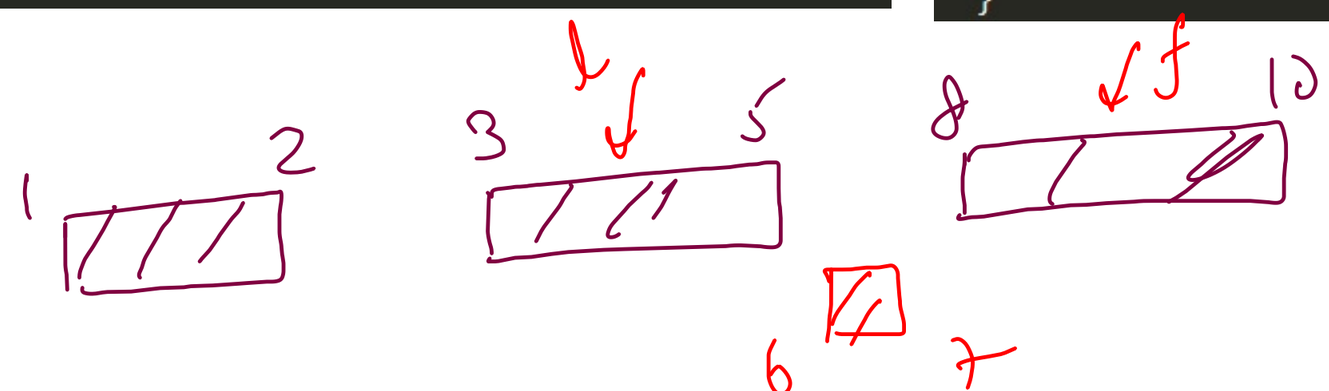
```
int[] merged = new int[2];
merged[0] = Math.min(intervals[firstIdx][0], newInterval[0]);
merged[1] = Math.max(intervals[lastIdx][1], newInterval[1]);
res.add(merged);
```

```java
int findFirst(int[][] intervals, int[] newInterval){
    for(int idx=0; idx<intervals.length; idx++){
        if(intervals[idx][1] >= newInterval[0]){
            return idx;
        }
    }
    return intervals.length;
}

int findLast(int[][] intervals, int[] newInterval){
    for(int idx=intervals.length-1; idx>=0; idx--){
        if(intervals[idx][0] <= newInterval[1]){
            return idx;
        }
    }
    return -1;
}
```

```java
int firstIdx = findFirst(intervals, newInterval);
int lastIdx = findLast(intervals, newInterval);

ArrayList<int[]> res = new ArrayList<>();
```

```java
// non merging -> firstIdx > lastIdx
if(firstIdx > lastIdx){
    for(int i=0; i<=lastIdx; i++){
        res.add(intervals[i]);
    }
    res.add(newInterval);
    for(int i=firstIdx; i<intervals.length; i++){
        res.add(intervals[i]);
    }
```

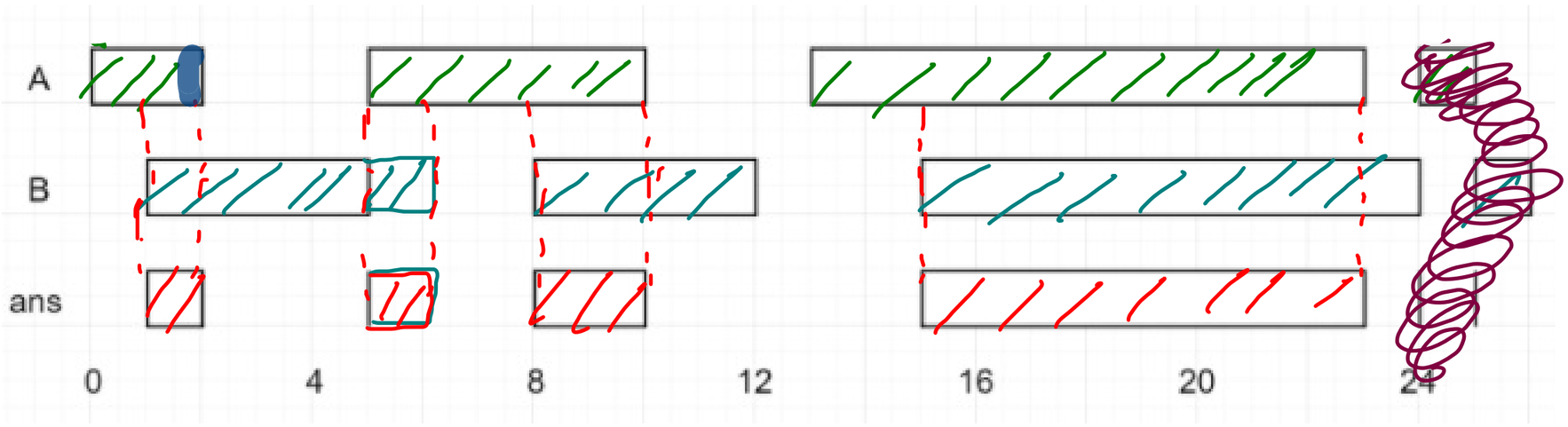
```java
} else {
    for(int i=0; i<firstIdx; i++){
        res.add(intervals[i]);
    }

    int[] merged = new int[2];
    merged[0] = Math.min(intervals[firstIdx][0], newInterval[0]);
    merged[1] = Math.max(intervals[lastIdx][1], newInterval[1]);
    res.add(merged);

    for(int i=lastIdx+1; i<intervals.length; i++){
        res.add(intervals[i]);
    }
}
```



Intervals Intersection

non-overlapping

. Each list of intervals is pairwise **disjoint** and in **sorted order**.



```java
public int[][] intervalIntersection(int[][] firstList, int[][] secondList) {
    int firstIdx = 0, secondIdx = 0;
    ArrayList<int[]> intersect = new ArrayList<>();

    while(firstIdx < firstList.length && secondIdx < secondList.length){
        int s1 = firstList[firstIdx][0];
        int e1 = firstList[firstIdx][1];
        int s2 = secondList[secondIdx][0];
        int e2 = secondList[secondIdx][1];

        int start = Math.max(s1, s2);
        int end = Math.min(e1, e2);

        if(start <= end){
            intersect.add(new int[]{start, end});
        }

        if(e1 < e2) firstIdx++;
        else secondIdx++;
    }

    int[][] res = new int[intersect.size()][2];
    for(int i=0; i<intersect.size(); i++){
        res[i] = intersect.get(i);
    }
    return res;
}
```

1. SRS - GitHub
2. Notes
3. Recordings
4. CB videos
5. R & B - level 2