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

Greedy Algorithms

WHAT: Local optimize choice at each step, helps some global optimum (maxima/minima)

APPLICATIONS:

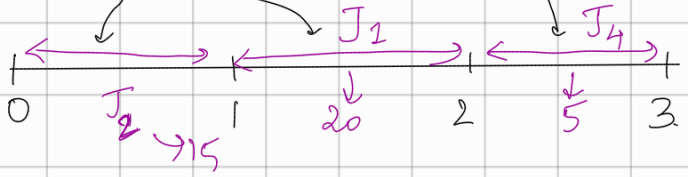
1) Fractional Knapsack
2) Activity Selection / Overlapping Intervals / Meeting Rooms
3) Minimum Spanning tree
   (Prims (PQ), Kruskal (DSU))
4) Huffman Encoding & Decoding / Optimal merge pattern
5) Single Source Shortest path Algo (Dijkstra)
6) Job Sequencing
7) Problems based on sorting

① JOB SEQUENCING (Jobid, deadline, Profit)

$X \rightarrow J_3$ cannot be allocated

JOB	J_1	J_2	J_3	J_4	J_5
Profit	20	15	10	5	1
Deadline	2	2	1	3	3

$X \rightarrow J_5$ cannot be allocated.



} this allocation gives maximum profit

① Sort Jobs in decreasing order of profit

② Picked Job should be placed as last as possible & place the current job there, so now the slot is booked and

$\approx n^2$

Time Complexity = $O(d \times n + n \log n)$

where d = deadline (max)

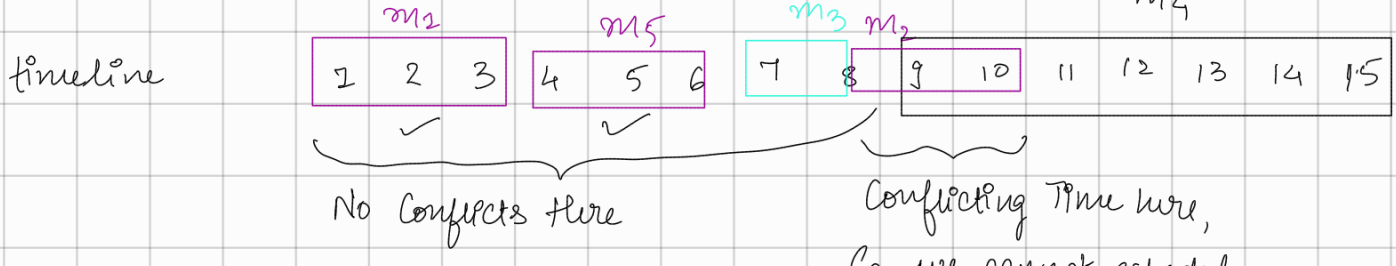
n = number of elements

② Meeting Rooms. I. (Same as merge overlapping intervals)

$\{\{1,3\}, \{8,10\}, \{7,8\}, \{9,15\}, \{4,6\}\}$

m_1 m_2 m_3 m_4 m_5

one room ⇒ one meeting at a time



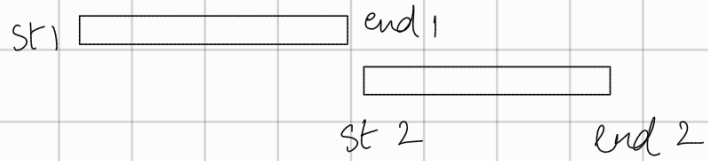
No Conflicts Here

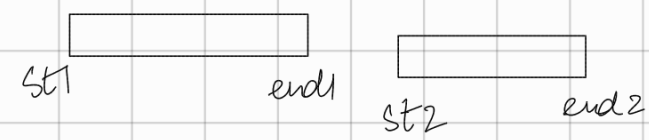
Conflicting Time here, So we cannot schedule meetings in one room

So return false

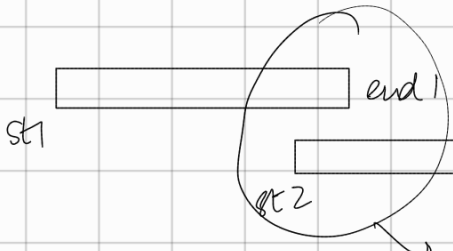
Step①: Sort based on ending times

This gives 3 possible cases

① 

② 

→ True

③ 

→ False

So, to check for conflict, we only need to check between end1 & st2 times.

Check for overlapping

∴ CODE:

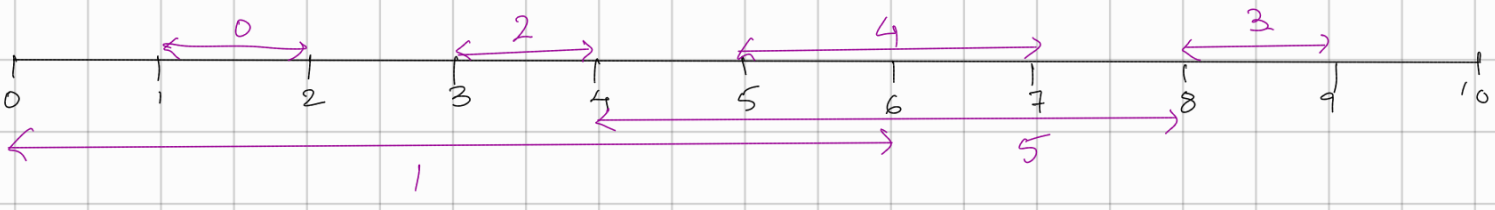
```
public static class MyComparator implements Comparator<Interval>{
    public int compare(Interval obj1, Interval obj2){
        if(obj1.end != obj2.end)
            return obj1.end - obj2.end;
        return obj1.start - obj2.start;
    }
}

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

    int limit = Integer.MIN_VALUE;
```

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

TC: $O(N \log N + N) = O(N \log N)$

③ N meeting in one room

s[]	1	0	3	8	5	4
e[]	2	6	4	9	7	8
idx	0	1	2	3	4	5



Steps

① Sort based on ending times

② Check for overlapping

yes → do not update Count ++

no → update Count ++

⇒ TC: $O(N \log N + N) = O(N \log N)$;

④ Disjoint Intervals

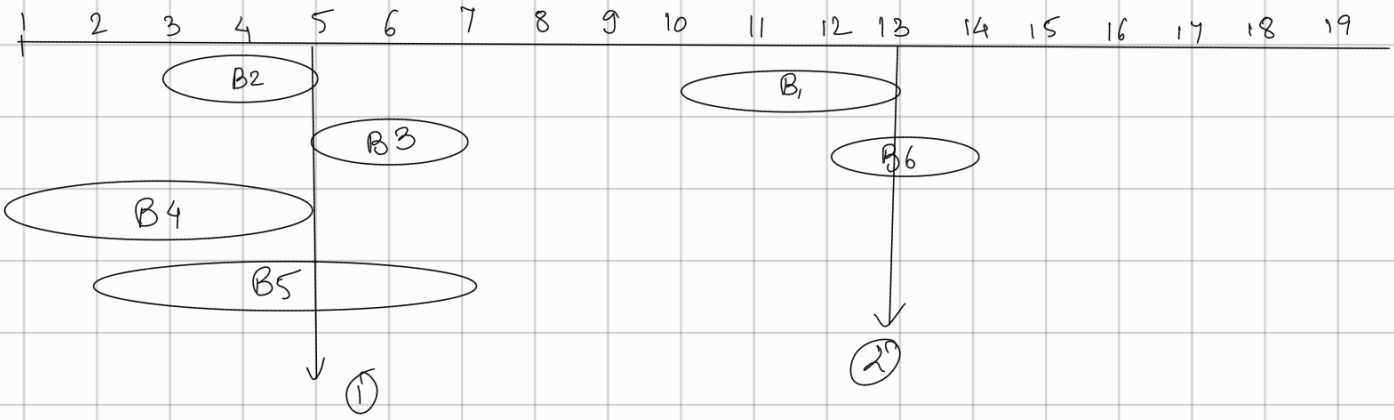
Same as ③, return n - count;

⑤ Max length of pair chain

⑥ Min arrows needed to burst balloons.

coordinates[]: [[10,13],[3,5],[5,7],[1,5],[2,6],[12,14]]
① ② ③ ④ ⑤ ⑥

output = 2



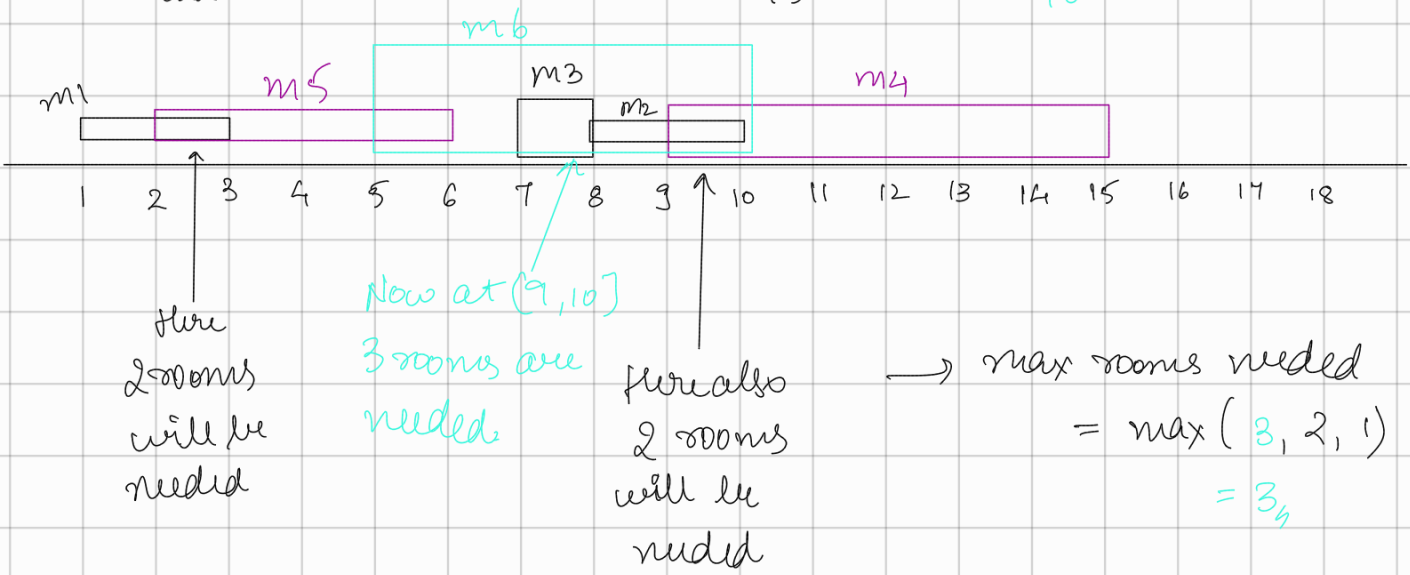
∴ Min. Balloons = no. of non-overlapping intervals

∴ ① Sort on end times

②

Meeting Rooms - II Category

	m1	m2	m3	m4	m5	m6
① Eg: st	1	8	7	9	2	5
end	3	10	8	15	6	10



→ max rooms needed = max (3, 2, 1) = 3

Using meeting rooms - I

Room - 1 → (1 - 3)

Room - 2 → (2 - 6)

for [7, 8] → 1 & 2 rooms are empty again

So, allocate (7, 8) to ①

then for (5, 10) → ③ ∵ ② still had [2, 6]

for [8, 10] → 7-8 ended so ① ✓

for (9, 15) → ② is empty room

So, allocate in ②

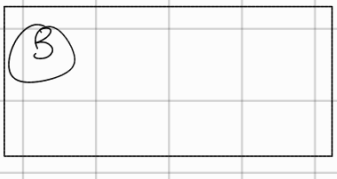
∴ max rooms = 3

To optimize: allot the room that got empty first

Rooms = 0

for [1, 3]

Rooms = 1 {1, 3}



for [2,6]

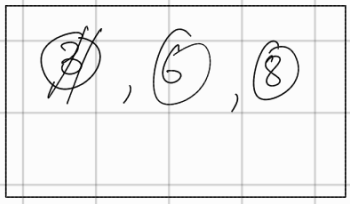
Room 1 ⇒ {1,3} So new Room

Room 2 ⇒ {2,6}

for [7,8]

Room 1 ⇒ empty → {7,8}

Room 2 ⇒ empty at 7



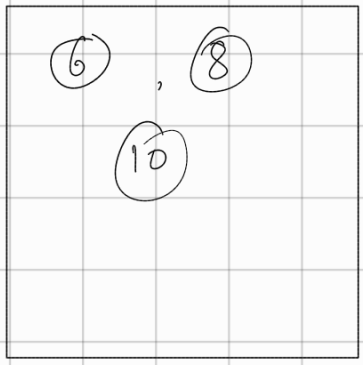
for [5,10]

Room 1 ⇒ {7,8} } full

Room 2 ⇒ {2,6}

So make new room

Room 3 ⇒ {5,10}

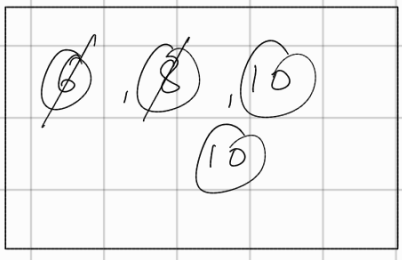


for {8,10}

Room 1 ⇒ empty at 8

Room 2 ⇒ empty at 6 ⇒ (8,10)

Room 3 ⇒ {5,10}

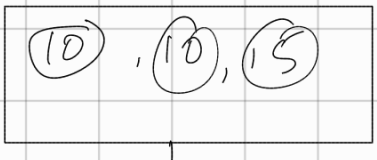


for {9,15}

Room 1 ⇒ empty ⇒ {9,15}

Room 2 ⇒ {8,10}

Room 3 ⇒ {5,10}



Can be Queue or Priority Queue

↓

this gives N log N

TC: O(N log N + N)

SC: O(N)

★★★

The above approach is wrong

Instead give the room that got empty at last as possible and not as early as possible

Optimized Approach

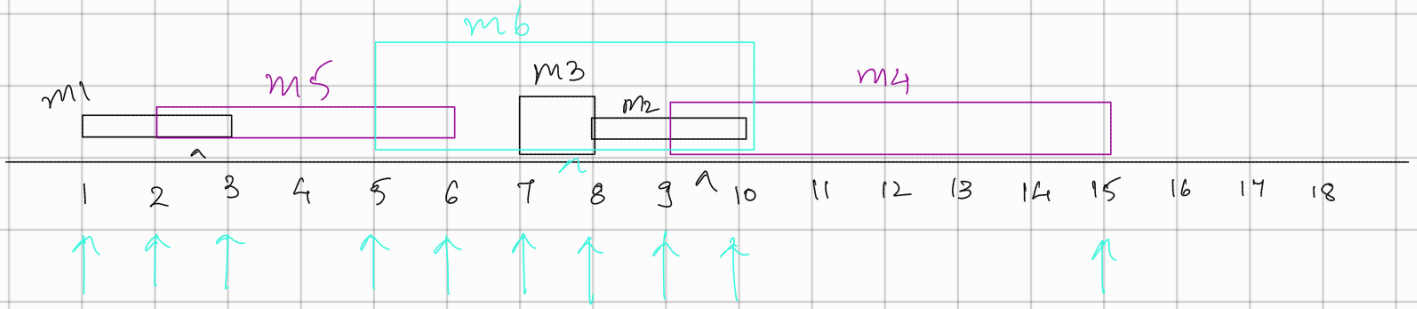
$0 < interval[i] < 10^4$

$0 < start, end < 10^6$

for all start & end points. check for

if (start) → Rooms ++

if (end) → Rooms --



only check for above selected points & after traversing all points ⇒ ans = max(Rooms)