

N meetings in one room

Problem Statement: There is one meeting room in a firm. You are given two arrays, start and end each of size N. For an index 'i', start[i] denotes the starting time of the ith meeting while end[i] will denote the ending time of the ith meeting. Find the maximum number of meetings that can be accommodated if only one meeting can happen in the room at a particular time. Print the order in which these meetings will be performed.

Example:

Input: N = 6, start[] = {1,3,0,5,8,5}, end[] = {2,4,5,7,9,9}

Output: 1 2 4 5

Explanation: See the figure for a better understanding.

Meeting No.	1 ✓	2 ✓	3	4 ✓	5 ✓	6
Start Time	1	3	0	5	8	5
End Time	2	4	5	7	9	9

Solution:

Disclaimer: Don't jump directly to the solution, try it out yourself first.

Initial Thought Process:-

Say if you have two meetings, one which gets over early and another which gets over late. Which one should we choose? If our meeting lasts longer the room stays occupied and we lose our time. On the other hand, if we choose a meeting that finishes early we can accommodate more meetings. Hence we should choose meetings that end early and utilize the remaining time for more meetings.

Approach:

To proceed we need a vector of three quantities: the starting time, ending time, meeting number. Sort this data structure in ascending order of end time.

We need a variable to store the answer. Initially, the answer is 1 because the first meeting can always be performed. Make another variable, say *limit* that keeps track of the ending time of the meeting that was last performed. Initially set *limit* as the end time of the first meeting.

Start iterating from the second meeting. At every position we have two possibilities:-

- If the start time of a meeting is strictly greater than *limit* we can perform the meeting. Update the answer. Our new *limit* is the ending time of the current meeting since it was last performed. Also update *limit*.
- If the start time is less than or equal to *limit*, skip and move ahead.

Let's have a dry run by taking the following example.

$N = 6$, $start[] = \{1, 3, 0, 5, 8, 5\}$, $end[] = \{2, 4, 5, 7, 9, 9\}$

Initially set $answer = [1]$, $limit = 2$.

For Position 2 –

Start time of meeting no. 2 = 3 > limit. Update answer and limit.

Answer = [1, 2], limit = 4.

For Position 3 –

Start time of meeting no. 3 = 0 < limit. Nothing is changed.

For Position 4 –

Start time of meeting no. 4 = 5 > limit. Update answer and limit.

Answer = [1, 2, 4], limit = 7.

For Position 5 –

Start time of meeting no. 5 = 8 > limit. Update answer and limit.

Answer = [1,2,4,5], limit = 9.

For Position 6 –

Start time of meeting no. 6 = 8 < limit. Nothing is changed.

Final answer = [1,2,4,5]

Code:

```
class meeting {
    int start;
    int end;
    int pos;

    meeting(int start, int end, int pos)
    {
        this.start = start;
        this.end = end;
        this.pos = pos;
    }
}

class meetingComparator implements Comparator<meeting>
{
    @Override
    public int compare(meeting o1, meeting o2)
    {
        if (o1.end < o2.end)
            return -1;
```

```

        else if (o1.end > o2.end)
            return 1;
        else if(o1.pos < o2.pos)
            return -1;
        return 1;
    }
}

public class Meeting {
    static void maxMeetings(int start[], int end[], int n) {
        ArrayList<meeting> meet = new ArrayList<>();

        for(int i = 0; i < start.length; i++)
            meet.add(new meeting(start[i], end[i], i+1));

        meetingComparator mc = new meetingComparator();
        Collections.sort(meet, mc);
        ArrayList<Integer> answer = new ArrayList<>();
        answer.add(meet.get(0).pos);
        int limit = meet.get(0).end;

        for(int i = 1; i < start.length; i++) {
            if(meet.get(i).start > limit) {
                limit = meet.get(i).end;
                answer.add(meet.get(i).pos);
            }
        }
    }
}

```

```

        System.out.println("The order in which the meetings will be
performed is ");
        for(int i = 0;i<answer.size(); i++) {
            System.out.print(answer.get(i) + " ");
        }
    }
}

```

Time Complexity: $O(n)$ to iterate through every position and insert them in a data structure. $O(n \log n)$ to sort the data structure in ascending order of end time. $O(n)$ to iterate through the positions and check which meeting can be performed.

Overall : $O(n) + O(n \log n) + O(n) \sim O(n \log n)$

Space Complexity: $O(n)$ since we used an additional data structure for storing the start time, end time, and meeting no.

Geeks for geeks problem solution;

<https://practice.geeksforgeeks.org/problems/n-meetings-in-one-room-1587115620/1>

```

public static int maxMeetings(int start[], int end[], int n)
{
    // add your code here
    int[][] time=new int[n][2];
    for(int i=0;i<n;i++)
    {
        time[i][0]=start[i];
        time[i][1]=end[i];
    }
    Arrays.sort(time,(a,b)->Integer.compare(a[1],b[1]));
    if(n<=1)

```

```
return n;
int taken=0;
int count=1;
for(int i=1;i<n;i++)
{
    if(time[i][0]>time[taken][1])
    {
        //System.out.println("Start "+start[i]+" end :
"+end[taken]);
        count++;
        taken=i;
    }
}
return count;
}
```