M section

25 March 2025 08:20

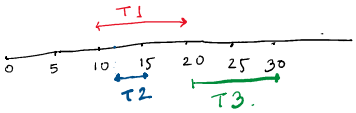
Activity selection problem / maximum disjoint interval

8–9 → DSA

8:30 – 9:30 → ML.

You are given **n** activities with their start and finish times. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a time.

Task	Start Time	End Time
1	10	20
2	12	15
3	20	30



Possible activities that can be performed either

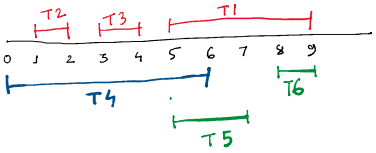
Ans=

T1, T3

T2, T3

Eg-

Start Time (s)	Finish Time (f)	Task Name
5	9	T1
1	2	T2
3	4	T3
0	6	T4
5	7	T5
8	9	T6



T2, T3, T5, T6 → 4

T2, T3, T1 → 3

T4, T6 → 2

Possible ordering –

Algorithm:

1. Sort all activities based on their finish time.
2. Choosing the first activity from the sorted list.
3. Select the next activity from the sorted list only if its `start` time is greater than or equal to the `finish` time of the previously selected activity.
4. Repeat `Step 3` for all the remaining activities in the sorted list.

Question: Maximum tasks that can be performed without any overlapping

Start Time (s)	Finish Time (f)	Task Name
5	9	T1
1	2	T2
3	4	T3
0	6	T4
5	7	T5
8	9	T6

Answer:

1. Sort all activities based on their finish time.

Start Time (s)	Finish Time (f)	Task Name
1	2	T2
3	4	T3
0	6	T4
5	7	T5
5	9	T1
8	9	T6

① Sort

② 1st task को इसीरा Perform Karo.

③ previous task.end time <= next task.start time.

T2, T3, T5, T6. → Max tasks = 4

Code-

```
start[] = {s1, s2, s3}
end[]   = {e1, e2, e3}.

no_of_activities = start.length();

n = No. of tasks
start = {10, 12, 20}
end   = {20, 15, 30}
```

```
class Task{
    int start_time;
    int end_time;
};
```

Task t[3];

t[0].start_time

t[0].end_time.

Integer a[];

```
ArrayList<Task> task = new ArrayList<>();
for(i=0; i<n; i++)
{
    task.add(new Task(start[i], end[i]));
}
```

task.get(1).start_time;
task.get(1).end_time;

Job Sequencing Problem

GFG.

n = No. of tasks.

start[] = {10, 12, 20}

end[] = {20, 15, 30}

Ans Max_no_of activities.

```
class Activity {
    int start time;
    int end time;
    // Constructor;
```

initialize as an empty list.

3.

```
Arraylist<Activity> activities = [ ]
for (i=0; i<n; i++){
    activities.add(new Activity(start time, end time));
```

3.

```
Collections.sort(activities)
```

→ Sort based on their starting time in increasing order.

JAVA Comparator

```
public int compare(Activity a1, Activity a2) {
    return a1.end - a2.end;
}
```

-ive → a1, a2

0 → a1, a2 / a2, a1

+ive → a2, a1.

Start Time (s)	Finish Time (f)	Task Name
		T
		T
		T
		T
		T
		T

Start Time (s)	Finish Time (f)	Task Name	
1	2	T 2	last End = 2
3	4	T 3	last End = 4
0	6	T 4	
5	7	T 5	last End = 7
5	9	T 1	
8	9	T 6	

count = 1

count = 2

count = 3