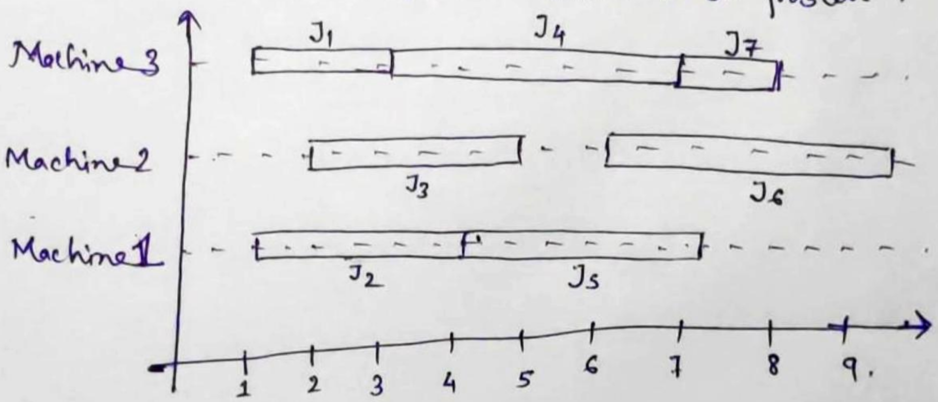
Task Scheduling:

Let us consider the following task scheduling problem, for tasks whose collection of pairs of start time & end time are as follows:

$$T = \{(1,3),(1,4),(2,5),(3,7),(4,7),(6,9),(7,8)\}.$$
$$\quad J_1 \quad J_2 \quad J_3 \quad J_4, \quad J_5, \quad J_6 \quad J_7.$$

here $J_1 \rightarrow$ Start time = 1, end time = 3. Now if we consider 3 machine, which will be used to solve this problem.



Now, Lets formalize the problem & provide an algorithmic Solution for it,–

Let's consider a set T of n tasks, such that each task 'i' has a start time, s_i & finish time f_i (where $s_i < f_i$). Task i must start at time s_i & it is guaranteed to be finished by time f_i. Each task has to be performed on a machine & each machine can execute only one task at a time. Two tasks i & j are "<u>non conflicting</u>" if $f_i \leq s_j$ or $f_j \leq s_i$. Two task can be scheduled to be executed on the same machine only if they are non conflicting.

Here the objective is to schedule all the task in T on the fewest machine in an non conflicting manner. (See the last example).

Algorithm

~~TaskSchedule(T)~~

Input: A set T of tasks, such that each task has a start time s_i & a finish time f_i

Output: A non conflicting schedule of tasks in T using a minimum number of machines.

```
TaskSchedule (T)
{  m ← 0   { optional, no. of machine)
   while (T ≠ ∅)
   {
      remove task i with smallest start time sᵢ from T.
      if (there is a machine j with no task conflicting with task i)
      {
         schedule task i on machine j
      } else
      {
            m ← m+1  { add a new machine }.
            Schedule task i on machine m.
      }
   }
}
```

In this algorithm TaskSchedule, we begin with no machines & we consider the task in a greedy fashion, order by their start times.

Now consider the followng problem:

Given a set of n tasks specified by their start & finish times, Algorithm Taskschedule produces a schedule of the tasks with the minimum number of machine in $O(n\lg n)$ time.

Ans

This problem can be solved by simple Contradiction Argument. So, suppose the algorithm does not work. That is, suppose the Algorithm finds a ~~non~~ non conflicting schedule using K machines but there is a non conflicting schedule that uses only $(K-1)$ machines. Let K be the last machine allocated by our algorithm, & i be the first task scheduled on K. By the structure of the algorithm, when we schedule i, each of the machines 1 through $K-1$ contained tasks that conflict with i. Since they conflict with i & because we consider tasks ordered by their start times, all the task currently conflicting with task i must have start time less than or equal to S_i, the start time of i, & have finish time after S_i. In other words these task not only conflict with task i they all conflict with each other. ~~which implies it is impossible for us to schedule all the task in~~

But this means we have K tasks in our set T that conflicts with each other, which implies it is impossible for us to schedule all the task in T using only $K-1$ machines. Therefore K is the minimum number of machines needed to schedule all the task in T.