COL 351:
Analysis and Design of Algorithms

Lecture 2

Algorithm Paradigms

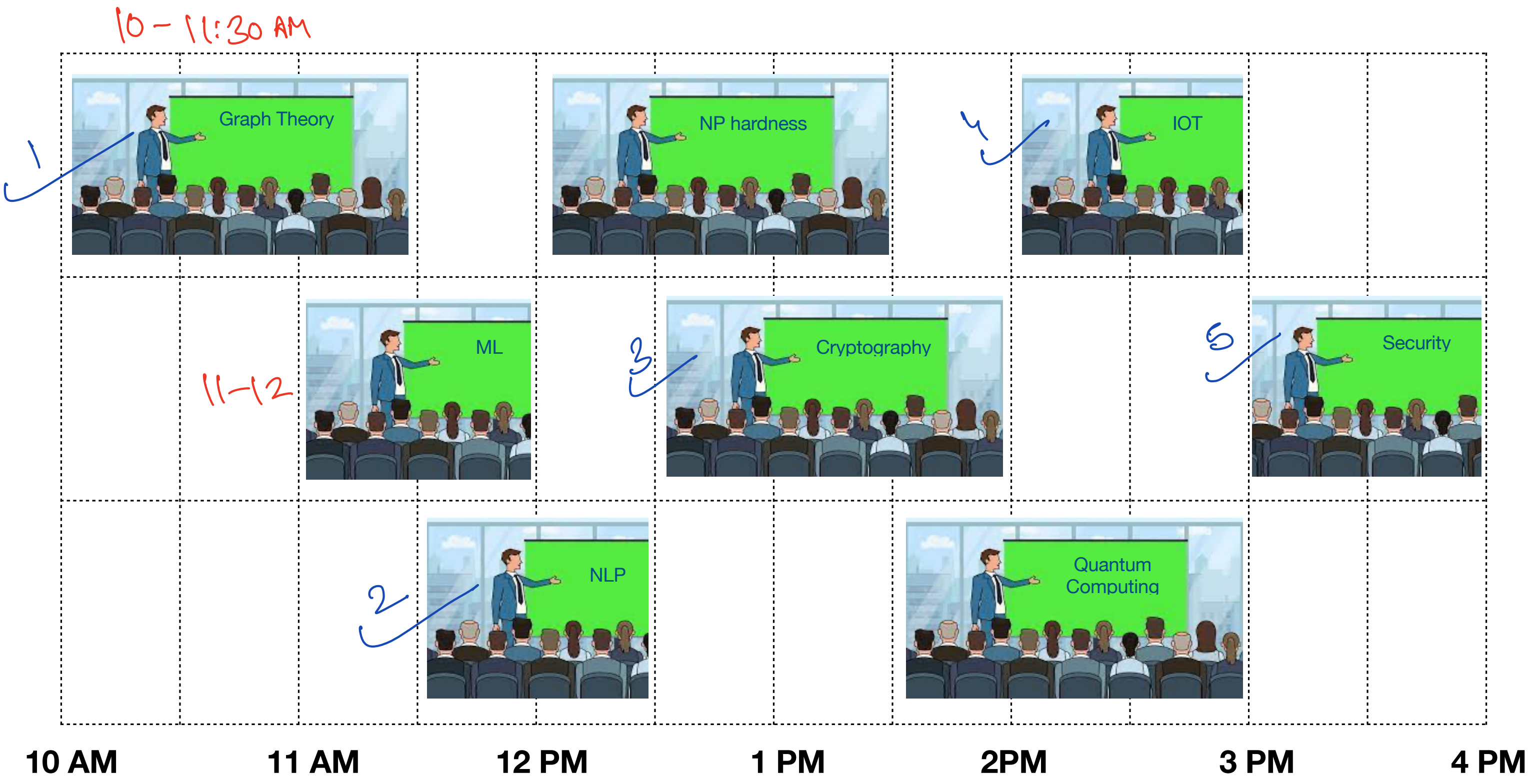
1. **Divide and Conquer**
   — Divide the problem into smaller problems
   — Solve the smaller problems
   — Combine

2. **Dynamic Programming**
   Reduce the problem on an input of size n into problems of size $n-1$, $n-2$, $n-3$,.. etc.

3. **Greedy Strategy**
   Build solution greedily.

Job- Scheduling

Computer Science Fest



Goal: Attend maximum number of seminars in CS Fest

$|opt| = 5$

Job Scheduling

$J_i = [s_i \; t_i]$

Formal Definition

Given: A collection n jobs, $\{J_1 = [s_1, t_1], \ldots, J_n = [s_n, t_n]\}$.
A single server.

Constraint: If job J_i is scheduled on server, then it occupies the server for time-interval $\underline{[s_i, t_i]}$

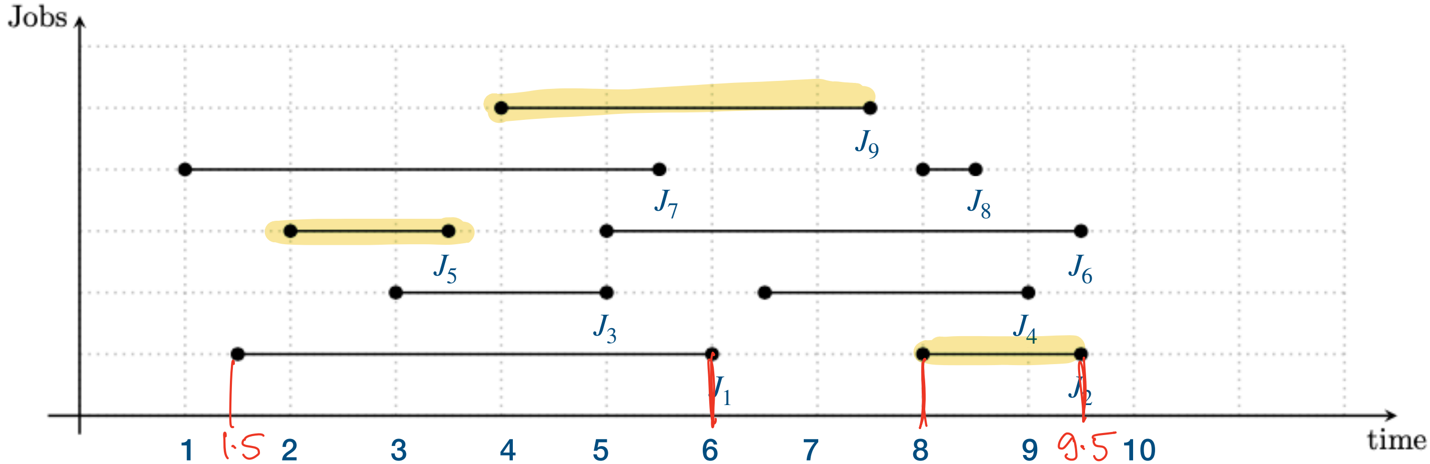
Aim: Find a maximum subset $S \subseteq \{J_1, \ldots, J_n\}$ of non-overlapping jobs.

$J_{set} := \{J_1, \ldots, J_n\}$

Example

Goal – $O(n^c)$, $c \geq 0$

$n = 9$



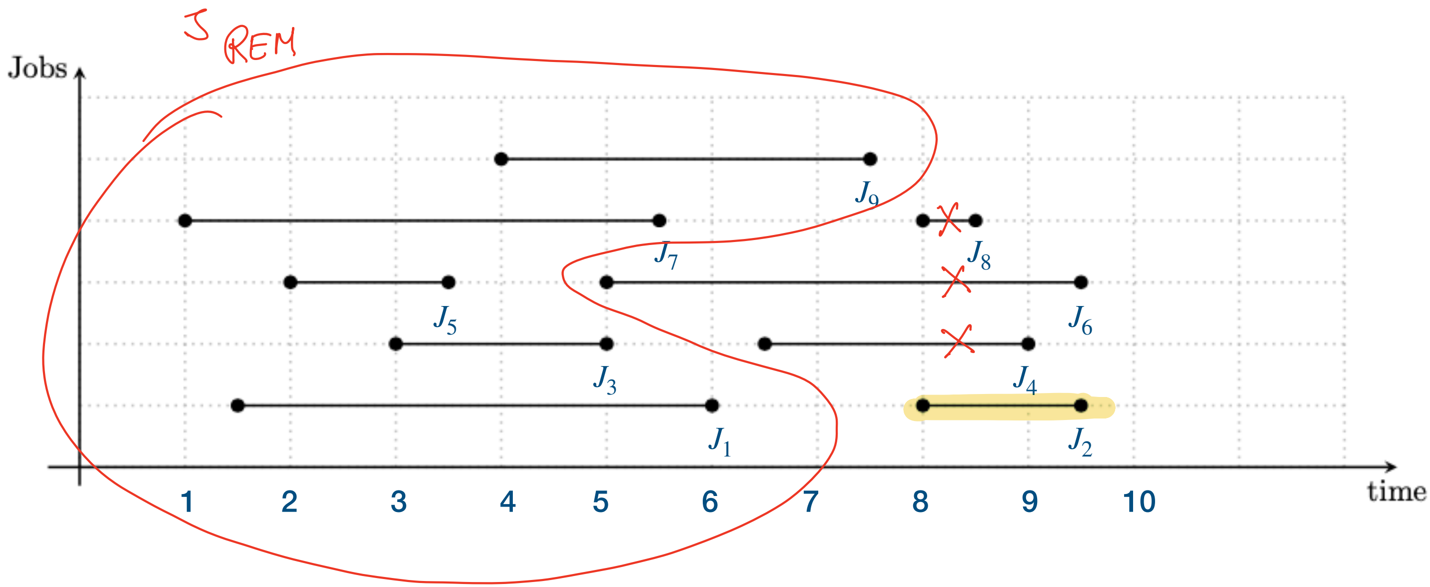
All 2^n subsets

Time $= \Omega(2^n)$

TRIVIAL APPROACH?

Example

∃ an opt sol^n containing job J_2

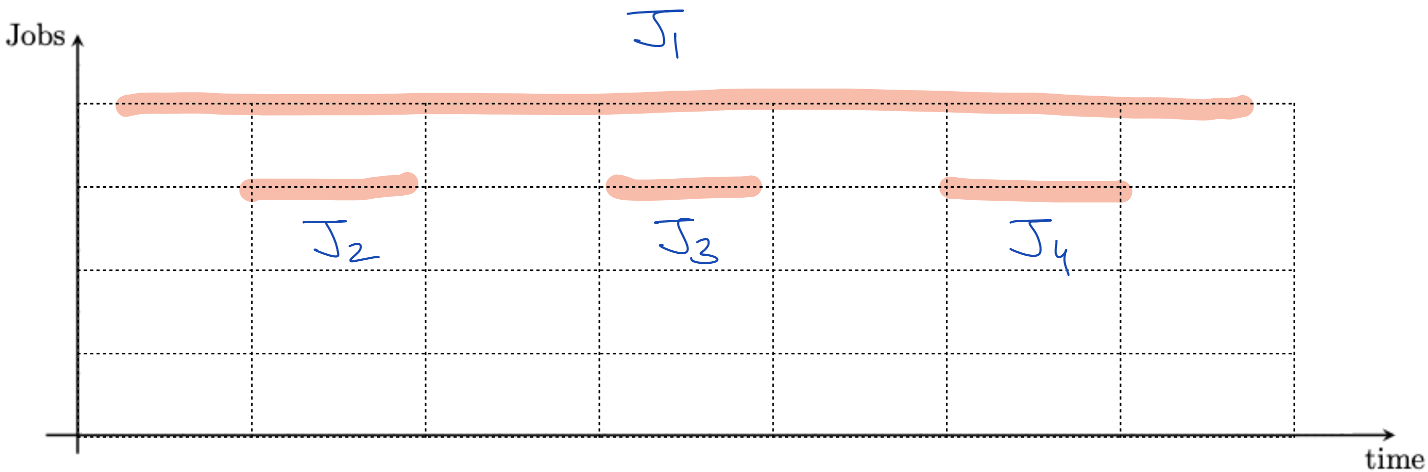


Main Idea: Greedily find one job that lies in an optimal solution.

Ques: How to find a job in the set $\{J_1 \dots J_n\}$ that lies in the opt sol^n?

Which Greedy Strategy works?

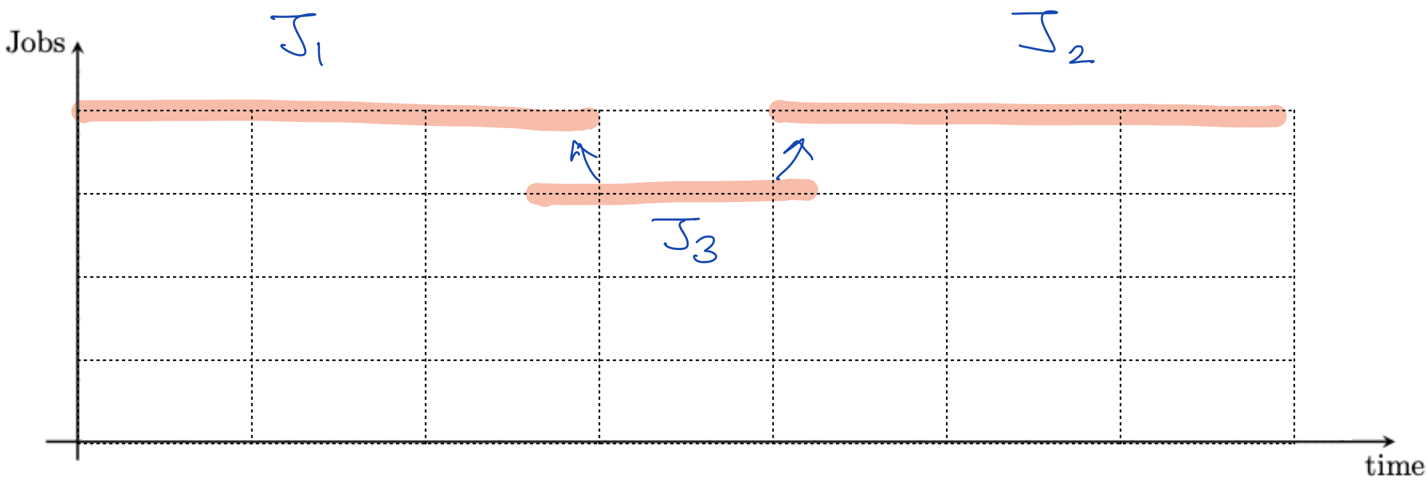
Ques. Can we select a job that **arrives first**, i.e., has smallest s_i? NO



opt = { J2 J3 J4 }

Which Greedy Strategy works?

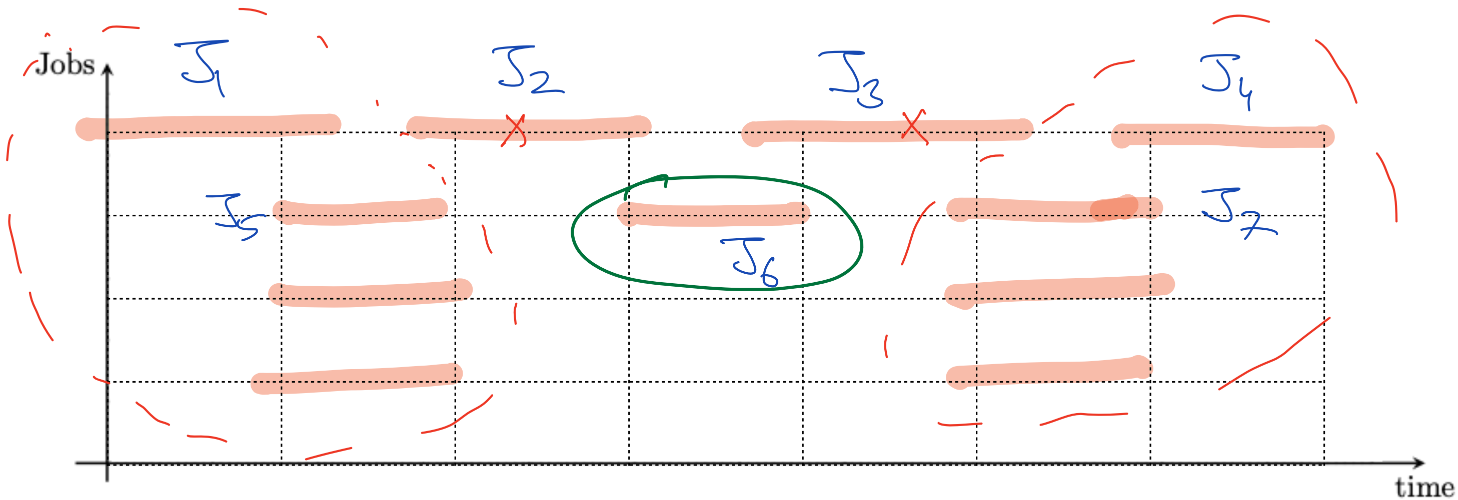
Ques. Can we select a job with **smallest duration**, i.e., with smallest value of $(t_i - s_i)$? No



$opt = \{J_1, J_2\}$

Which Greedy Strategy works?

Ques. Can we select a job with **minimum overlap**?



$$Opt = \{ J_1 \ J_2 \ J_3 \ J_4 \}$$

Which Greedy Strategy works?

Ques. Can we select a job that **finishes earliest**, i.e., has smallest value of t_i? Yes.

Greedy Strategy - Choose job with earliest finish time

≡ Set of all jobs.

Lemma: Let "J_0" $\in J_{set}$ be job with earliest finish time. Then there exists an optimal solution containing J_0.

Proof:

S = any opt solⁿ

J* = job with earliest F.T in S

$$F.T.(J_0) \leq FT(J^*)$$

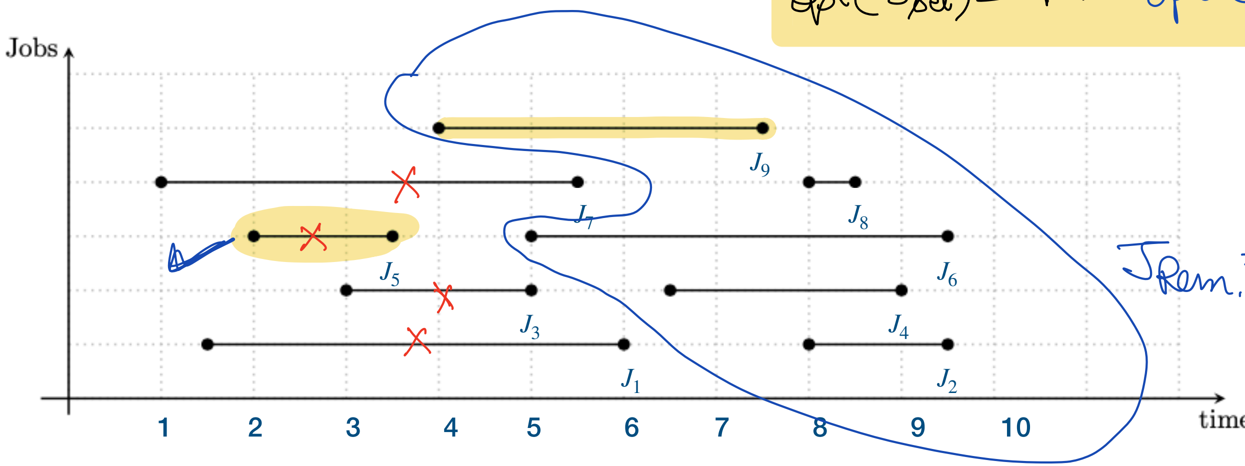
$$S_{new} = (S \setminus J^*) + J_0$$

- Jobs in S_{new} are NON-OVERLAPPING
- $|S_{new}| = |S|$ ⇒ S_{new} must be opt.

Algorithm

J_0 = Job with earliest F. T.

$\boxed{opt(J_{set}) = 1 + opt(J_{Rem})}$



$J_{Rem} = J \setminus \text{OVERLAP}(J_0)$

1. Initialise $S = \phi$.

2. While $J_{set} \neq \phi$:

   (a) Find a job $J_0 \in J_{set}$ with earliest finish time, and **add** it to set S.

   (b) Remove J_0 and jobs overlapping with J_0 from J_{set}.

3. Return S.

Schedule(J_{set})

What will be implementation time?

$\boxed{opt(J_{set}) * O(n)}$

Correctness

Theorem : Let $J_0 \in J_{set}$ be job with *earliest finish time*, and $J_{rem} = J_{set} \backslash \text{Overlap}(\ J_0\ )$. Then,

$$OPT(J_{set}) = OPT(J_{rem}) + 1.$$

LHS ≤ RHS | LHS ≥ R.H.S.

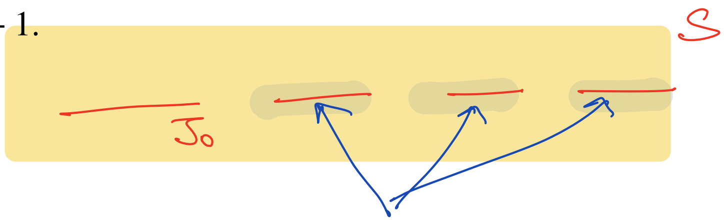
Correctness

$J_{set} = \{J_1, \ldots J_n\}$

Theorem : Let $J_0 \in J_{set}$ be job with *earliest finish time*, and $J_{rem} = J_{set} \setminus \text{Overlap}(J_0)$. Then,

$$OPT(J_{set}) = OPT(J_{rem}) + 1.$$

Proof Part 1: We will show $OPT(J_{set}) \leqslant OPT(J_{rem}) + 1$.



- Do not lie in OVERLAP(J_0)
- lie in J_{REM}

S = opt sol^n containing job J_0

CLAIM (1) $S \setminus J_0 \subseteq J_{REM}$

claim (2) $S \setminus J_0$ is non-overlapping.

$$opt(J_{Rem}) \geq |S \setminus J_0| = |S| - 1 = opt(J_{set}) - 1$$

HINT – $\exists$ an opt sol^n of J_{set} containing job J_0

Correctness

Theorem : Let $J_0 \in J_{set}$ be job with *earliest finish time*, and $J_{rem} = J_{set} \setminus \text{Overlap}(J_0)$. Then,

$$OPT(J_{set}) = OPT(J_{rem}) + 1.$$

Proof Part 2: We will show $OPT(J_{set}) \geqslant \underline{OPT(J_{rem})} + \underline{1}.$

$S^* =$ opt sol^n of J_{REM}

$S^* \cup \{J_0\}$ — Non overlapping

$$\underline{\underline{\text{opt}(J_{set})}} \geq S^* \cup \{J_0\} = |S^*| + 1 = \underline{\underline{\text{opt}(J_{Rem}) + 1}}$$

Homework Exercises

- Suppose the n jobs $J_1, \ldots, J_n$ satisfy that $t_1 \leqslant \cdots \leqslant t_n$.
  Then design an $O(n)$ time algorithm to compute an optimal scheduling.

Tut

$O(n \log n)$ time

$s_i, t_i \in \{1, 2, 3, \ldots, cn\}$ — Bucket sorting $\Rightarrow O(n)$

- Design an algorithm to find an optimal scheduling with **two servers.**

k servers.

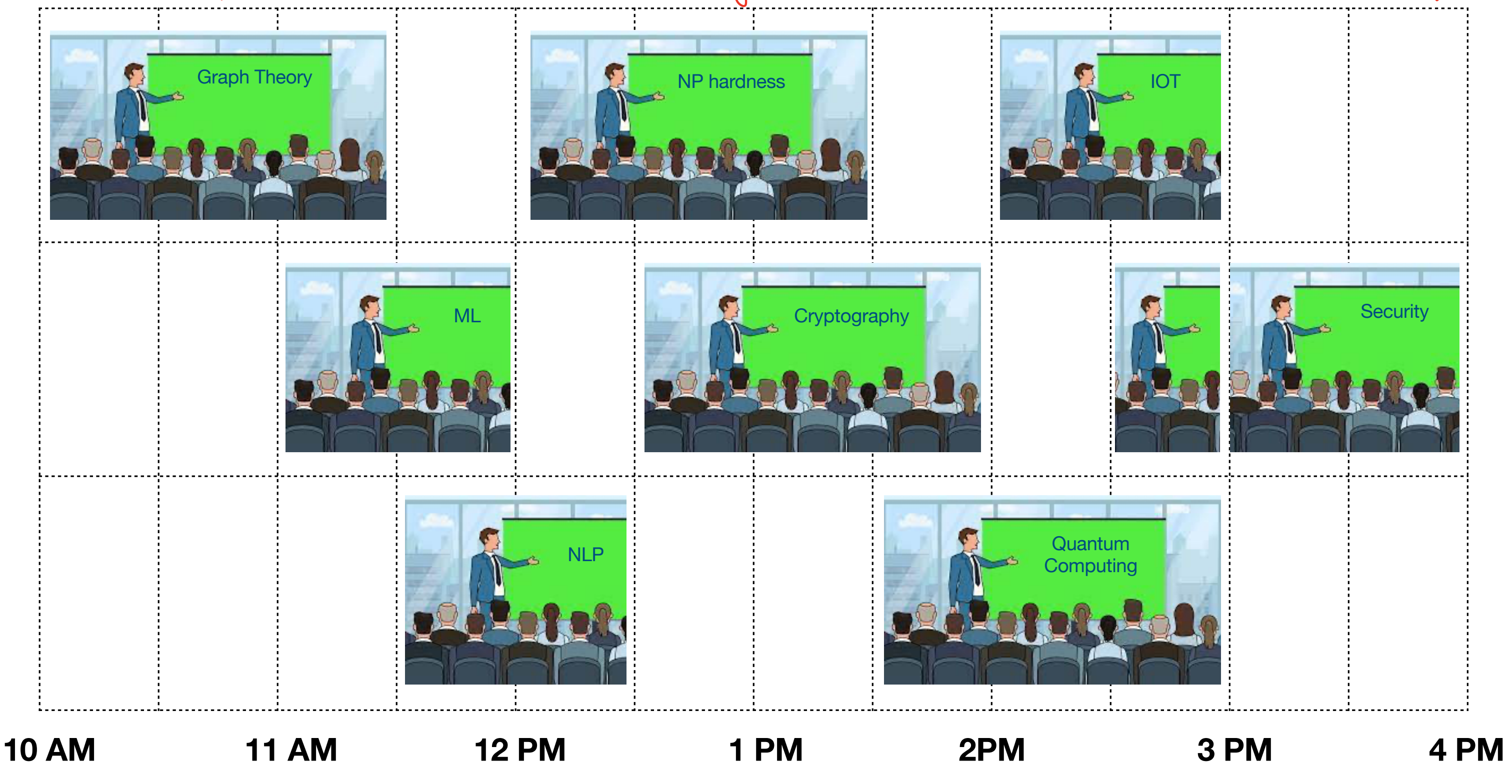
$k = 5, 7$

Scheduling with two servers:

How you and your friend can in total attend maximum possible number of seminars in CS Fest?

Maximum possible cardinality → $|Sem(you) \cup Sem(friend)|$

You friend.



$n = 9$ $|\text{opt sol}^n| = 8$

Unique Solution??

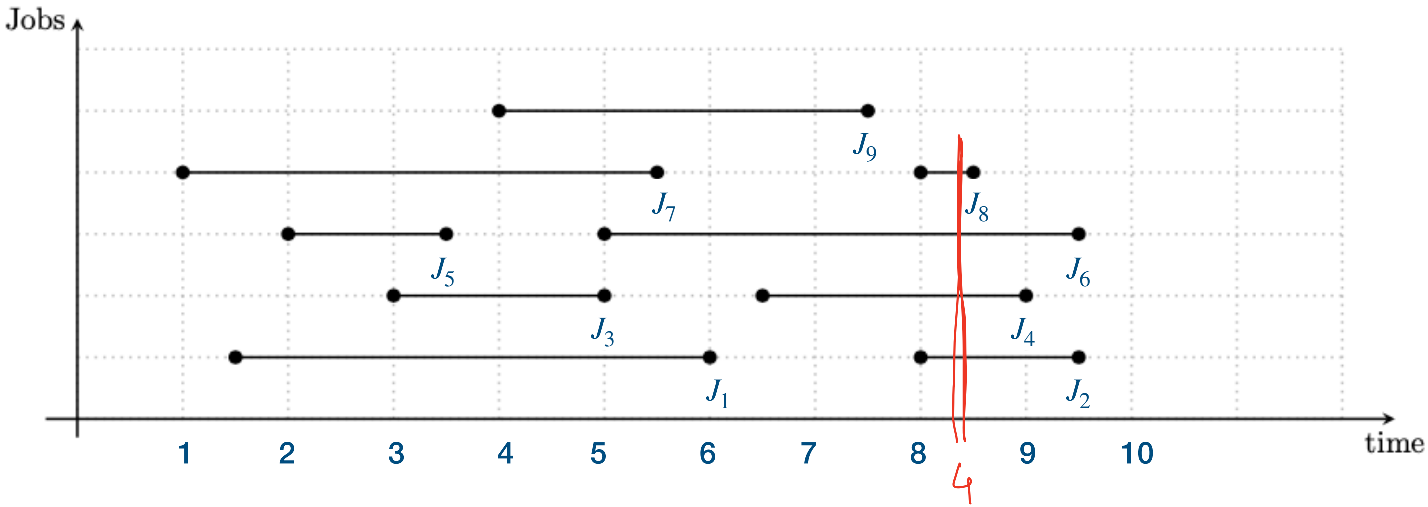
- Design an algorithm to check whether a collection of n given jobs has a **unique** optimal scheduling, with respect to **one given server**.

Challenge Problem (EASY)

Given : A collection of n jobs, $\{J_1 = [s_1, t_1], \ldots, J_n = [s_n, t_n]\}$.

Find : Minimum number of servers required to schedule all jobs. > max-overlap

?



What is the best possible time complexity?