Data Structures and Algorithms

(ESO207)

Lecture 34

- A new algorithm design paradigm: Greedy strategy

  part I

Path to the solution of a problem

No formula

But there is a **systematic approach** which usually works ☺

Today's lecture will demonstrate this approach ☺

Problem : JOB Scheduling

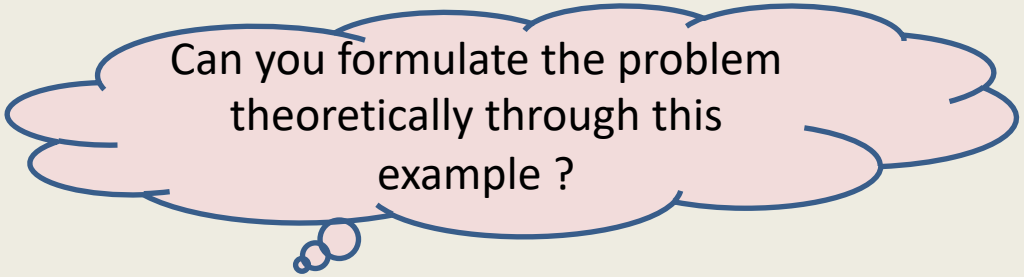
Largest subset of non-overlapping job

A motivating example

Antaragni 2021

- There are n large-scale activities to be performed in Auditorium.
- Each large scale activity has a **start time** and **finish time**.
- There is **overlap** among various activities.

Aim: What is the **largest subset** of activities that can be performed ?



Formal Description

A job scheduling problem

INPUT:

- A set J of n jobs $\{j_1, j_2, \ldots, j_n\}$
- job j_i is specified by two real numbers

  s(i): start time of job j_i

  f(i): finish time of job j_i
- A single server

Constraints:

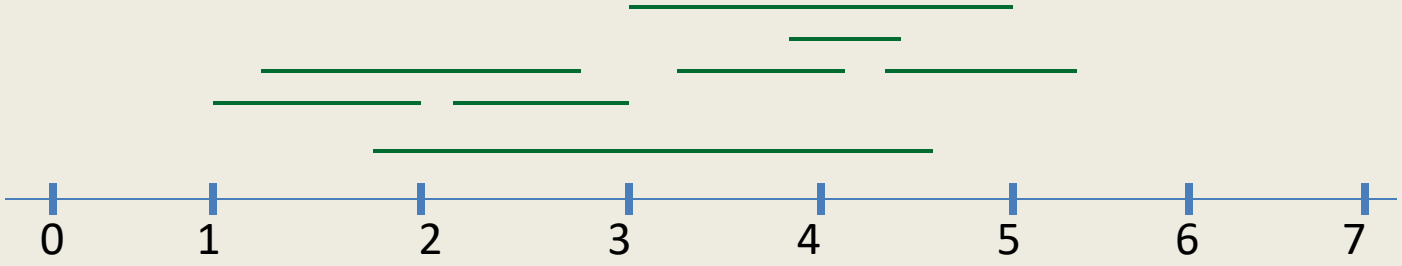
- Server can execute at most one job at any moment of time.
- **Job j_i**, if scheduled,

Aim:

To select the **largest** subset of **non-overlapping** jobs which can be executed by the server.

Example

INPUT: $(1, 2), (1.2, 2.8), (1.8, 4.6), (2.1, 3), (3, 5), (3.3, 4.2), (3.9, 4.4), (4.3, 5.4)$



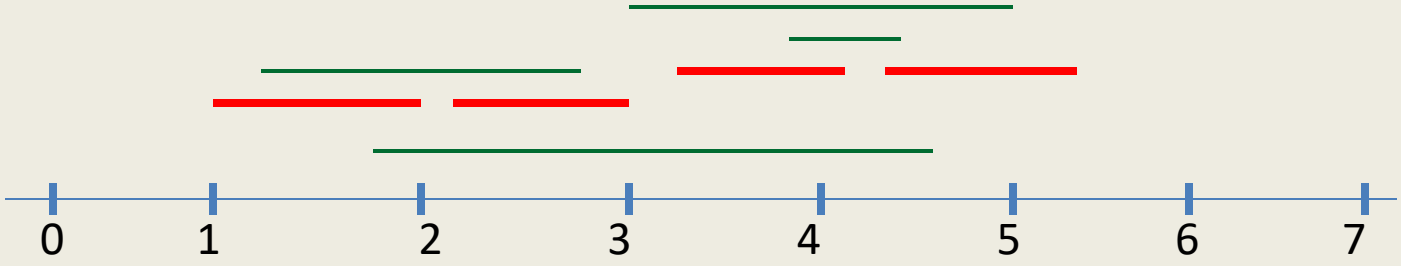
It makes sense to work with pictures than these numbers☺

job i is said to be **non-overlapping** with job k if

Try to find solution for the above example.

Example

INPUT: $(1, 2), (1.2, 2.8), (1.8, 4.6), (2.1, 3), (3, 5), (3.3, 4.2), (3.9, 4.4), (4.3, 5.4)$

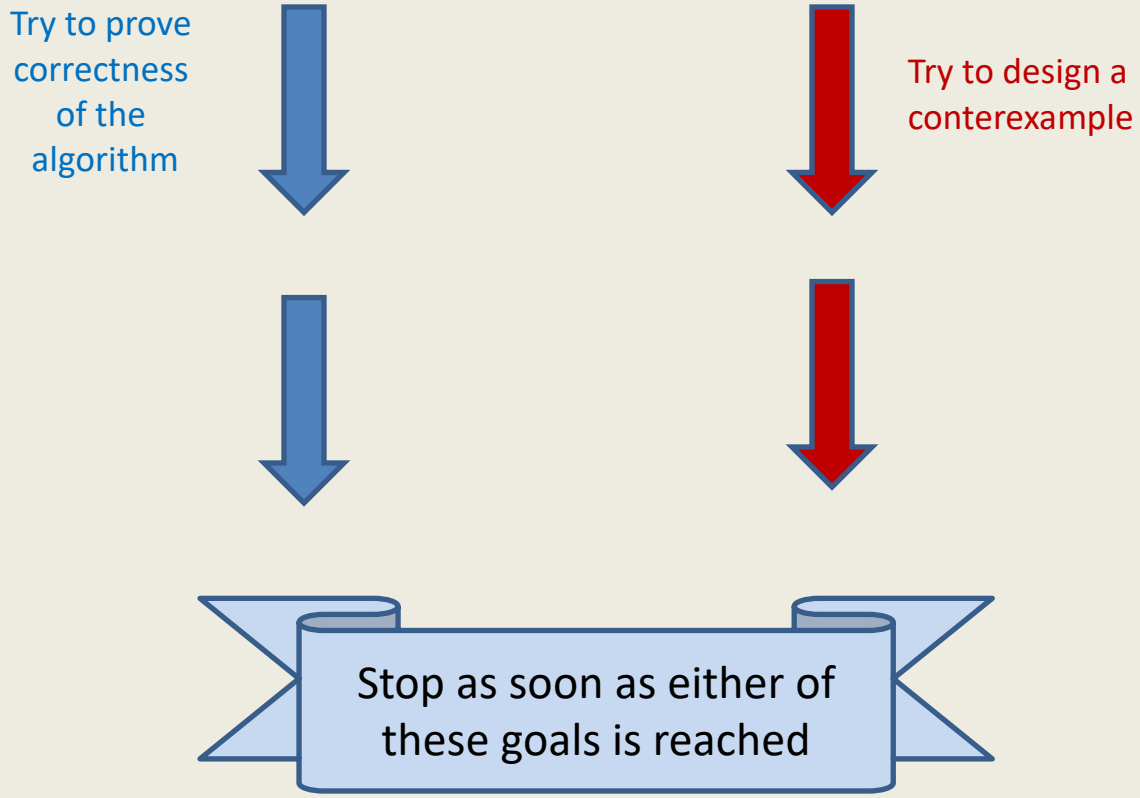


job i is said to be **non-overlappi** $[s(k), f(k)] = \emptyset$

What strategy come to your mind?

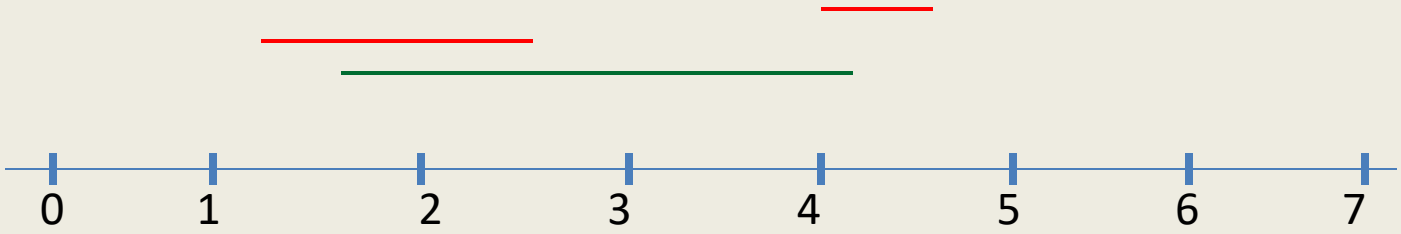
Designing algorithm for any problem

1. Choose a strategy based on some intuition
2. Transform the strategy into an algorithm.



Designing algorithm for the problem

Strategy 1: Select the earliest start time job



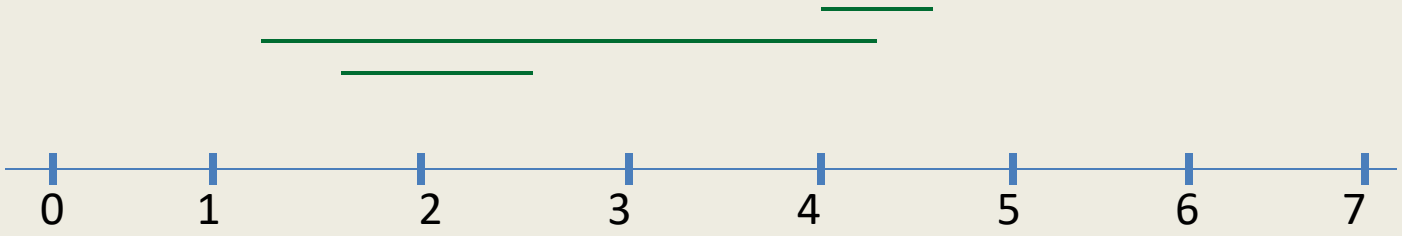
When one fails to prove the correctness of this strategy, one should search for a counterexample. Can you transform the above example into a counterexample ?

Intuition:

It might be better to assign jobs as early as possible so as **to make optimum use of server**.

Designing algorithm for the problem

Strategy 1: Select the earliest start time job

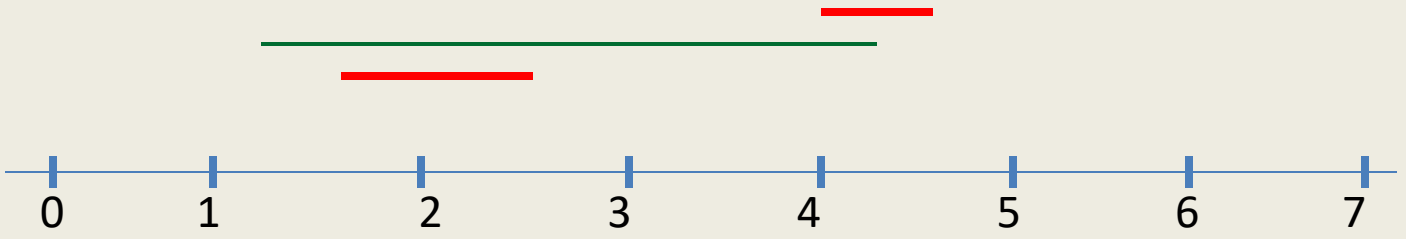


Intuition:

It might be better to assign jobs as early as possible so as **to make optimum use of server.**

Designing algorithm for the problem

Strategy 1: Select the earliest start time job



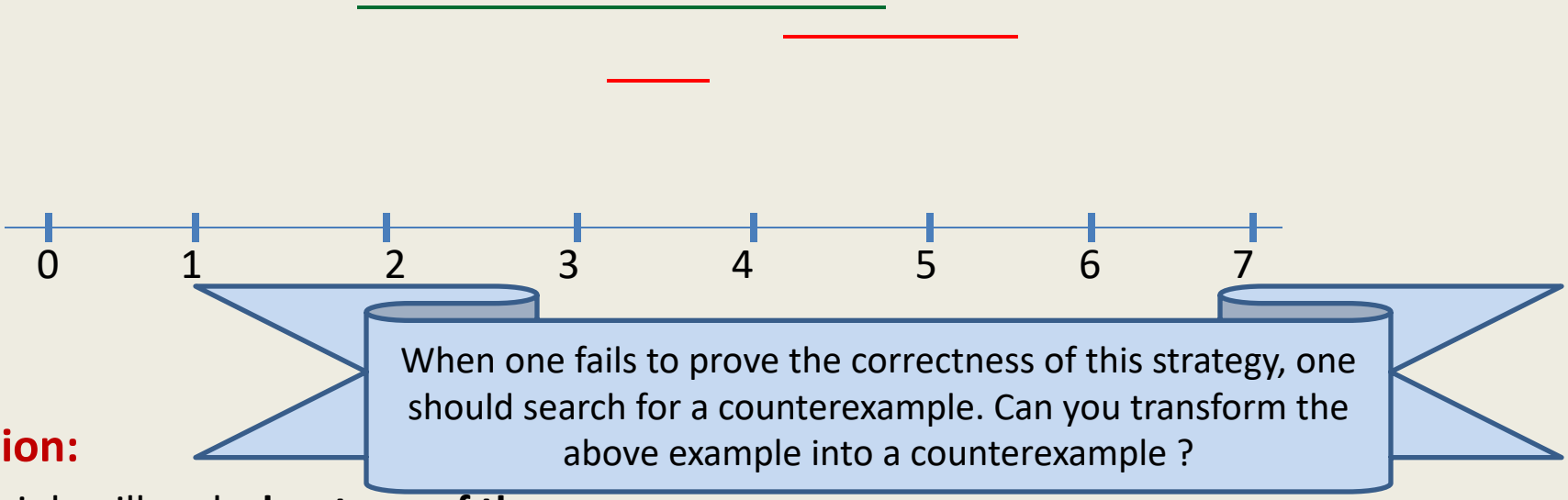
Instead of getting disappointed, try to realize that this counterexample points towards some other strategy which might work.

Intuition:

It might be better to assign jobs as early as possible so as to make optimum use of server.

Designing algorithm for the problem

Strategy 2: Select the job with smallest duration



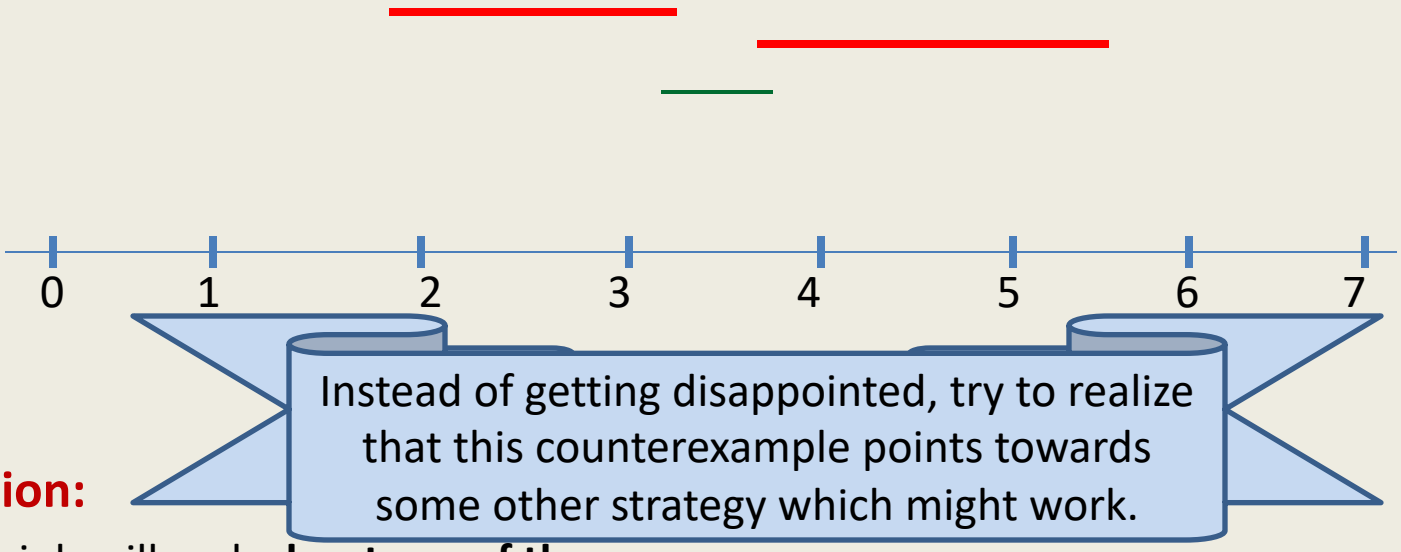
Intuition:

Such a job will make **least use of the server**

➔ this might lead to larger number of jobs to be executed

Designing algorithm for the problem

Strategy 2: Select the job with smallest duration



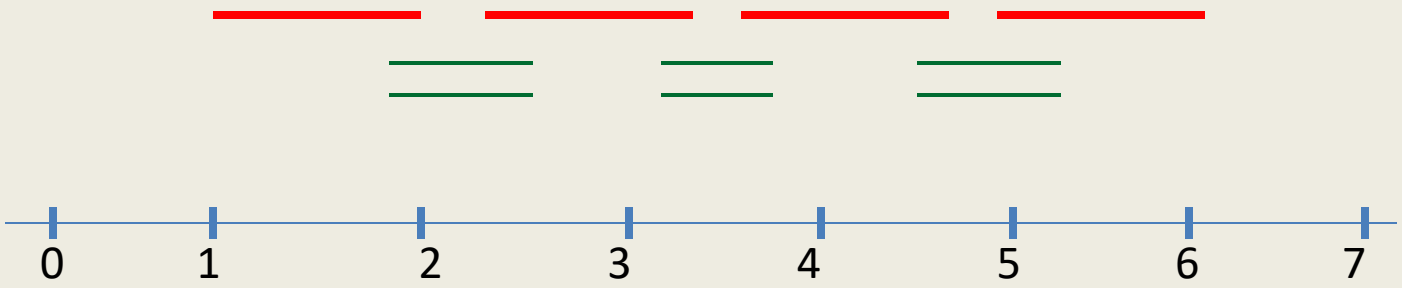
Intuition:

Such a job will make **least use of the server**

➔ this might lead to larger number of jobs to be executed

Designing algorithm for the problem

Strategy 3: Select the job with smallest no. of overlaps



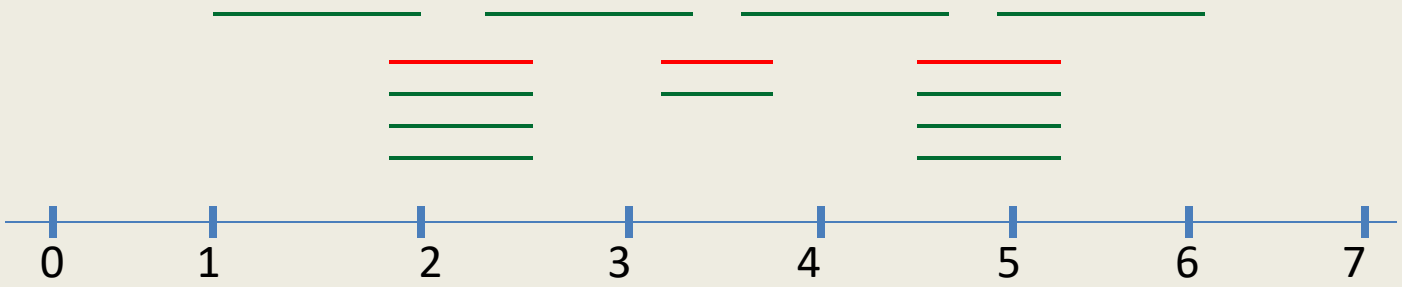
When one fails to prove the correctness of this strategy, one should search for a counterexample. Can you transform the above example into a counterexample ?

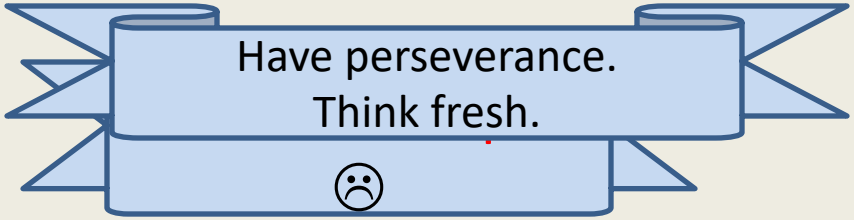
Intuition:

Selecting such a job will result in **least number of other jobs to be discarded**.

Designing algorithm for the problem

Strategy 3: Select the job with smallest no. of overlaps



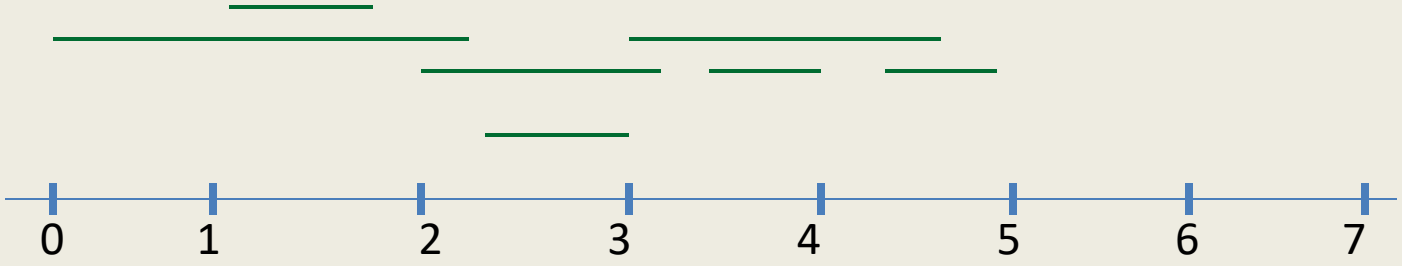


Intuition:

Selecting such a job will result in **least number of other jobs to be discarded.**

Designing algorithm for the problem

Strategy 4: Select the job with earliest finish time



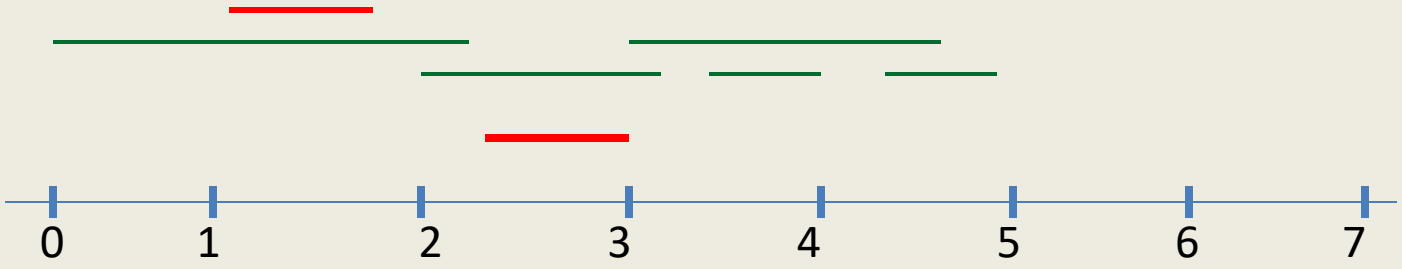
Intuition:

Selecting such a job will **free** the server **earliest**

➔ hence more no. of jobs might get scheduled.

Designing algorithm for the problem

Strategy 4: Select the job with earliest finish time



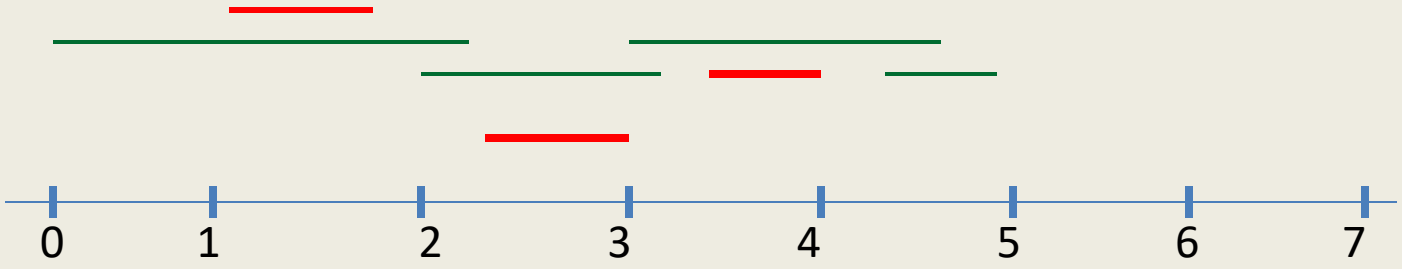
Intuition:

Selecting such a job will **free** the server **earliest**

➔ hence more no. of jobs might get scheduled.

Designing algorithm for the problem

Strategy 4: Select the job with earliest finish time



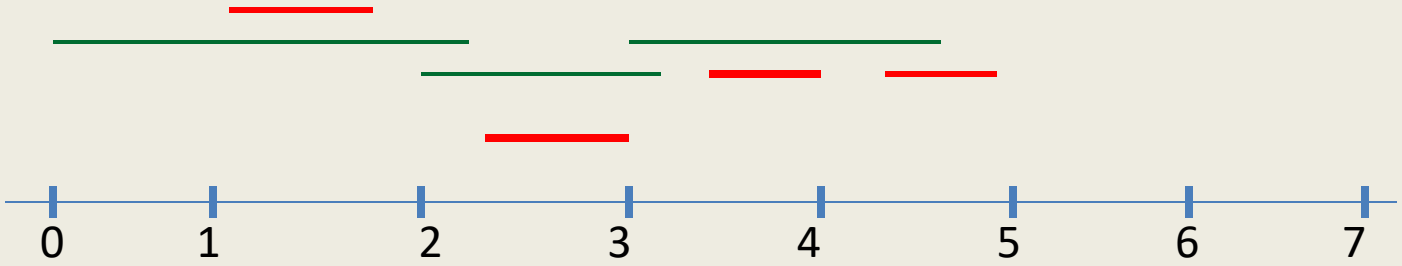
Intuition:

Selecting such a job will **free** the server **earliest**

➔ hence more no. of jobs might get scheduled.

Designing algorithm for the problem

Strategy 4: Select the job with earliest finish time



It is indeed a correct solution for this example. We should try to prove the correctness of the algorithm.☺But first we give a full description of the algorithm based on this strategy.

Intuition:

Selecting such a job will **free** the server **earliest**

➔ hence more no. of jobs might get scheduled.

Algorithm "earliest finish time"

Description

Algorithm (**Input :** set J of n jobs.)

1. Define $A \leftarrow \emptyset$;
2. **While** $J <> \emptyset$ **do**

   { Let x be the job from J with earliest finish time;

   $A \leftarrow A \cup \{x\}$;

   Remove x and all jobs that overlap with x from set J;

   }
3. Return A;

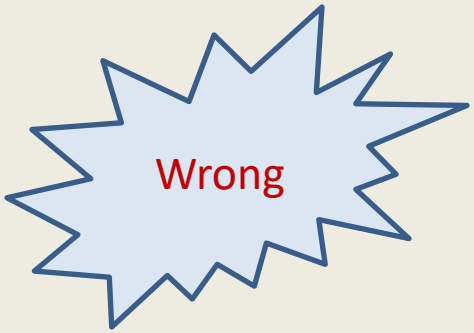
Running time for a trivial implementation of the above algorithm: O(n^2)

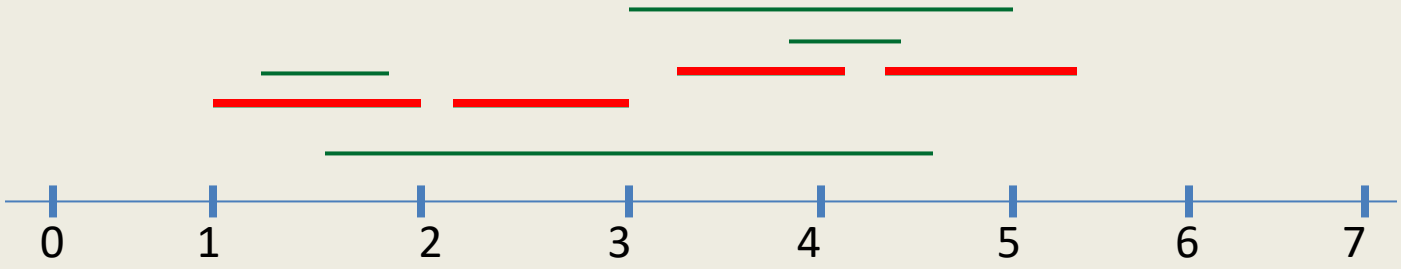
Algorithm "earliest finish time"

Correctness

Let x be the job with earliest finish time.

Lemma1: x is present in the optimal solution for J.





Too strong a claim !

Algorithm "earliest finish time"

Correctness

Let x be the job with earliest finish time.

Lemma1: There exists an optimal solution for J in which x is present.

Proof: Consider any optimal solution O for J.

Let y be the job from O with earliest finish time.

Let $O' \leftarrow O \backslash \{y\}$

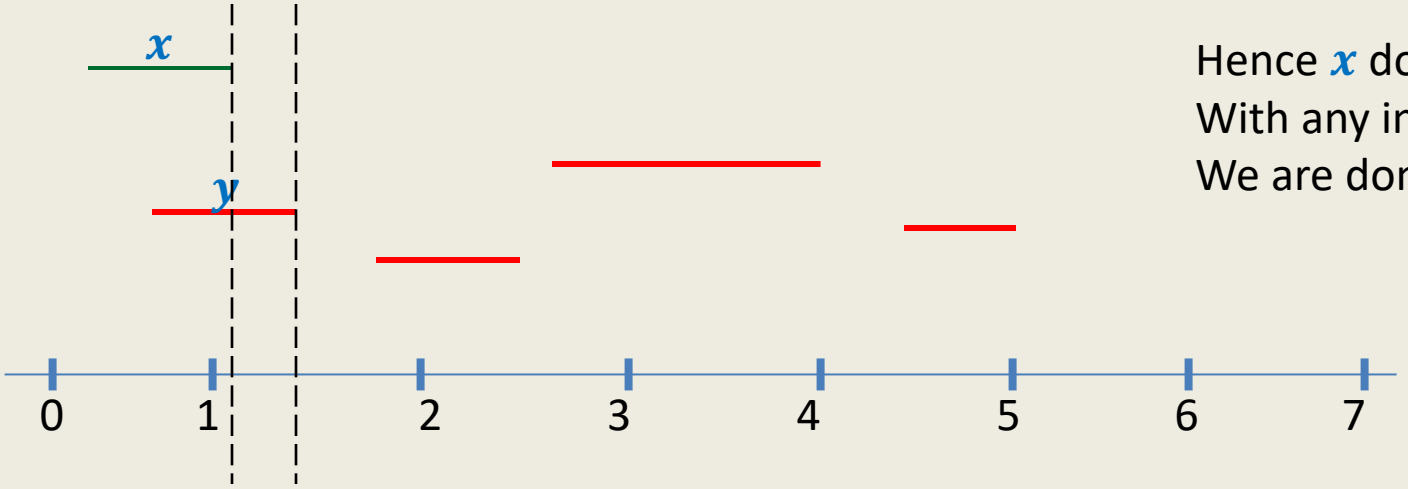
$\rightarrow f(y) < s(z)\ \forall\ z \in O'$

$\rightarrow f(x) \leq f(y)$

$\rightarrow f(x) < s(z)\ \forall\ z \in O'$

$O' \cup \{x\}$ is also an optimal solution.

Reason: $O' \cup \{x\}$ has no overlapping intervals. Give arguments.



Hence x does not overlap
With any interval of O'.
We are done ☺

Homework

Spend 30 minutes today on the following problems.

1. Use **Lemma1** to complete the proof of correctness of the algorithm.

2. Design an **O**($n$log n) implementation of the algorithm.