Data Structures and Algorithms

(CS210A)

Semester I – **2014-15**

Lecture 34

- **A new algorithm design paradigm:** Greedy strategy

part I

Path to the solution of a problem

No formula

Examples

intuition/insight

Strategy

Have perseverance

Failure ☹

Better Insight ☺

Success ☺

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

- 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 ?

Can you formulate the problem theoretically through this example ?

A job scheduling problem

Formal Description

INPUT:

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

  s($\boldsymbol{i}$): start time of job $\boldsymbol{j_i}$

  f($\boldsymbol{i}$): finish time of job $\boldsymbol{j_i}$
- A single server

Constraints:

- Server can execute <u>at most one job </u>at any moment of time and a job.
- **Job** $\boldsymbol{j_i}$, if scheduled, has to be scheduled <u>during[**s**($\boldsymbol{i}$), **f**($\boldsymbol{i}$)] only</u>.

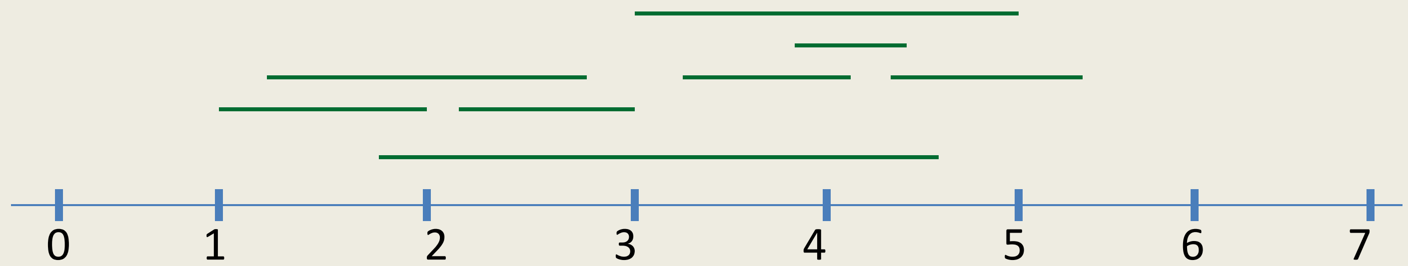
Aim:

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

A job scheduling problem

Example

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



It makes sense to work with pictures than these numbers☺

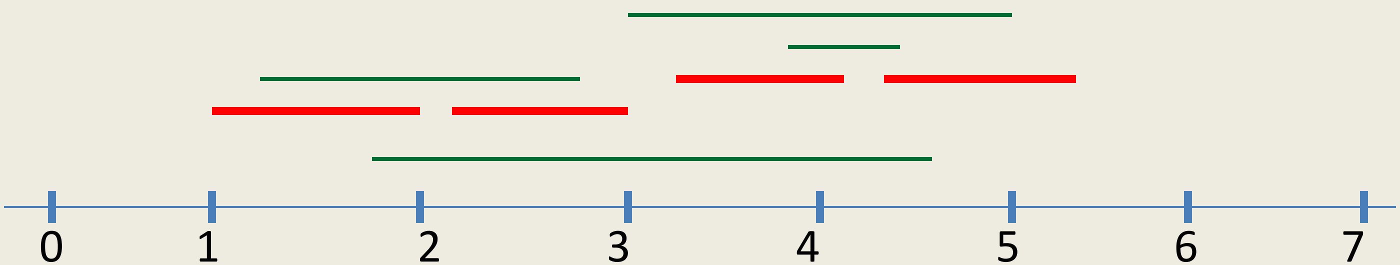
job $\boldsymbol{i}$ is said to be **non-overlapping** with job $\boldsymbol{k}$ if $[\mathbf{s}(\boldsymbol{i}), \mathbf{f}(\boldsymbol{i})] \cap [\mathbf{s}(\boldsymbol{k}), \mathbf{f}(\boldsymbol{k})] = \emptyset$

Try to find solution for the above example.

A job scheduling problem

Example

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



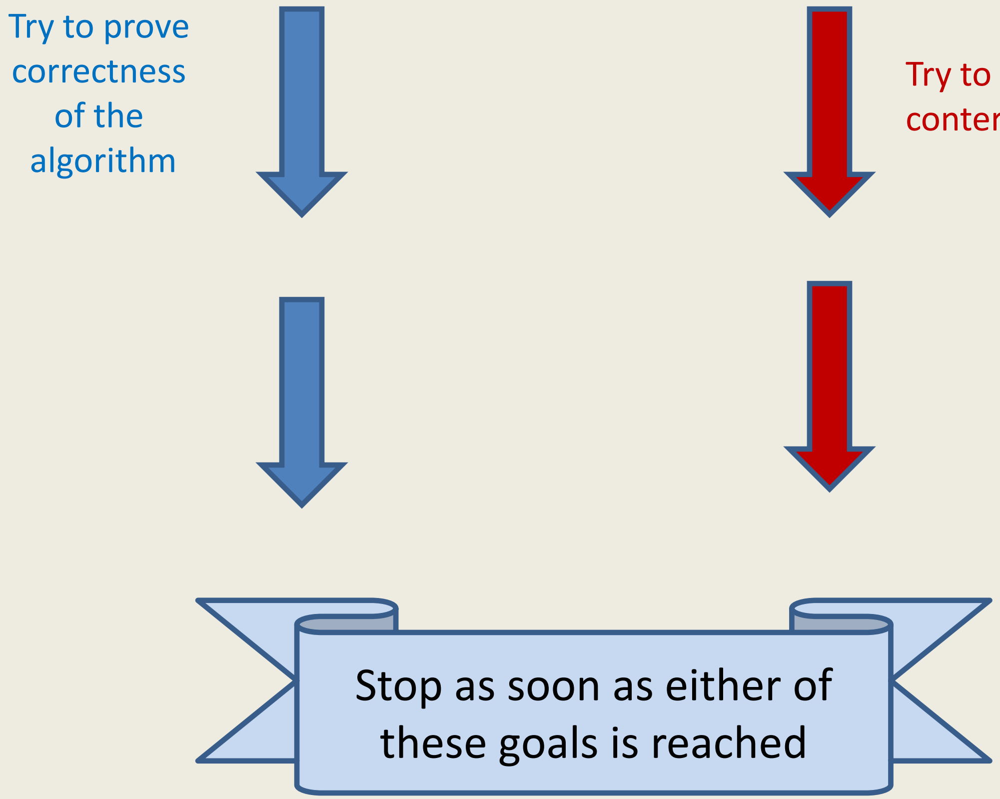
What strategy come to your mind?

job $\boldsymbol{i}$ is said to be **non-overlappi**

$(\boldsymbol{k})$, **f**$(\boldsymbol{k})$**]** = $\emptyset$

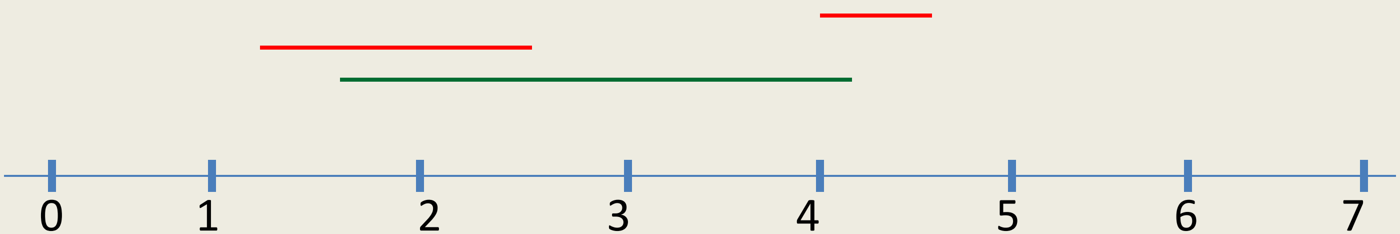
Designing **algorithm** for any problem

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



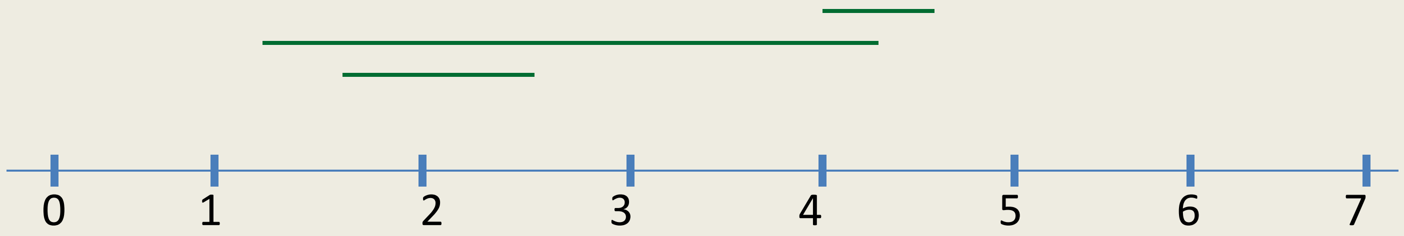
When one fails to prove the correctness of this strategy, one should search for a counterexample. Can you transform the above example into a counterexample ?

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

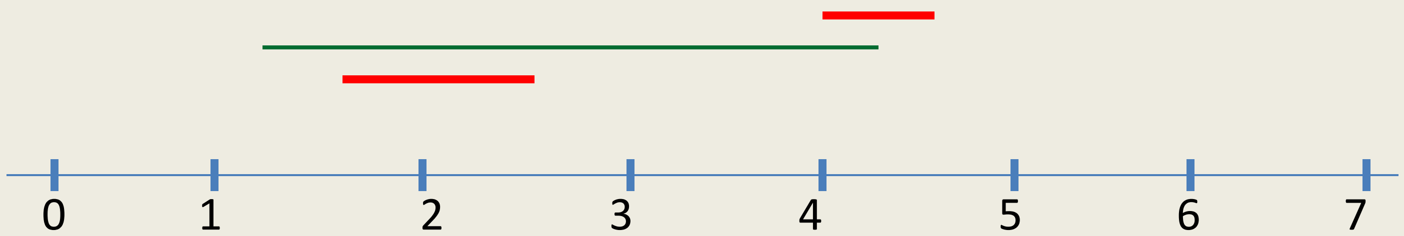


Intuition:

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



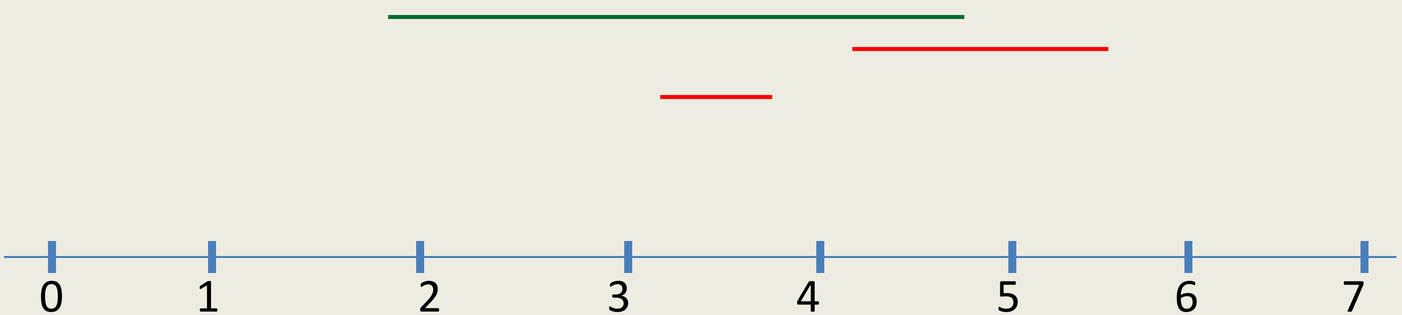
Instead of getting disappointed, try to realize that this counterexample points towards some other strategy which might work.

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**



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 ?

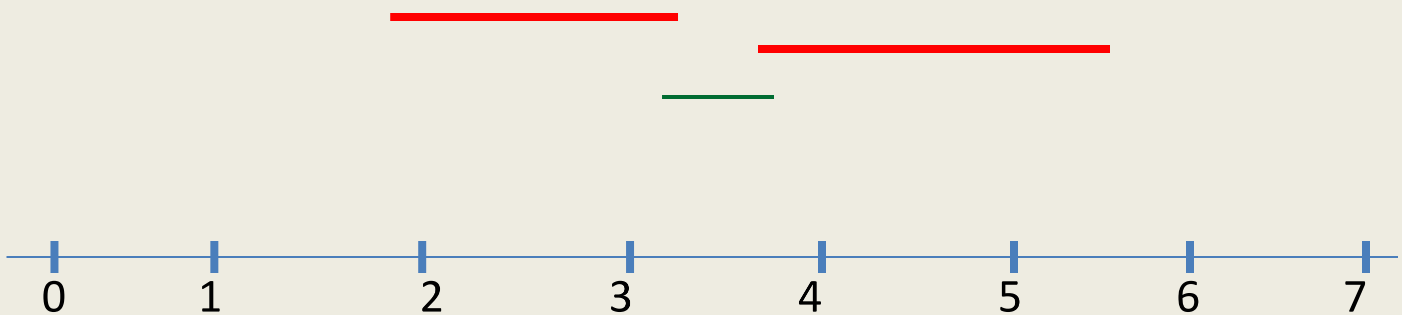
Intuition:

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**



counterexample

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

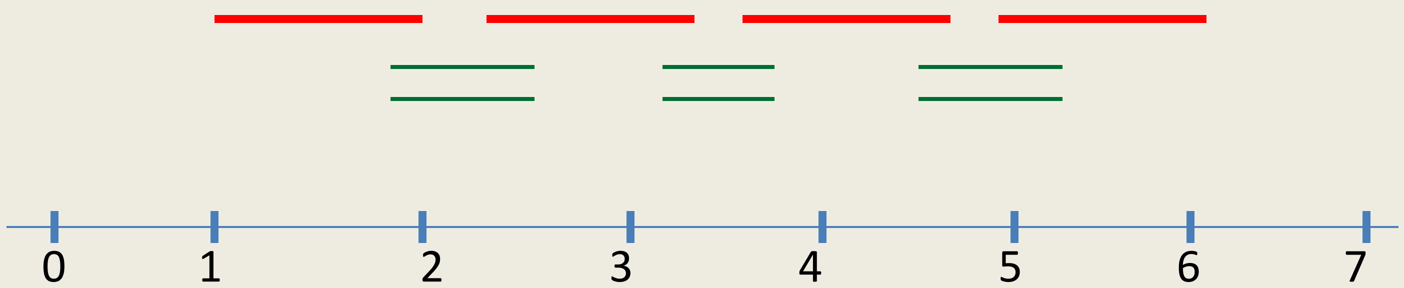
Intuition:

Such a job will

➔ 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**



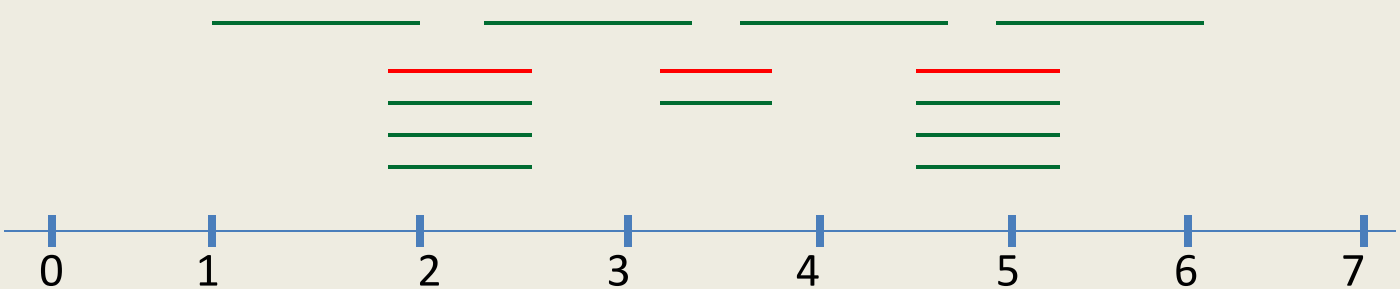
When one fails to prove the correctness of this strategy, one should search for a counterexample. Can you transform the above example into a counterexample ?

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**



Have perseverance.
Think fresh.

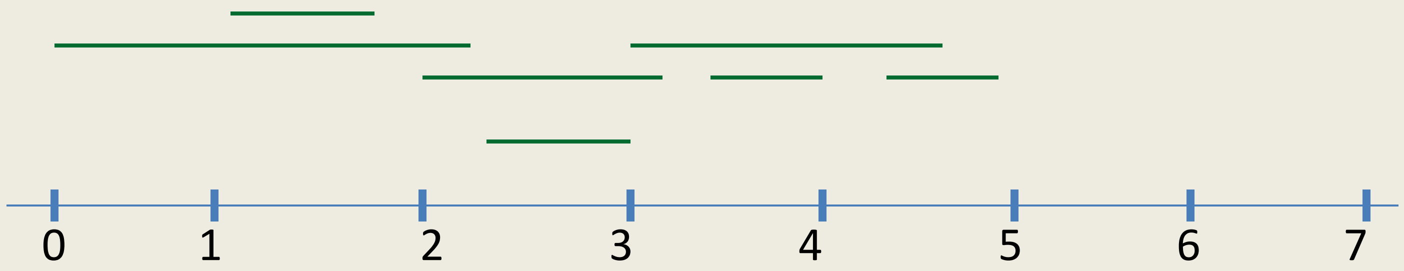
☹

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



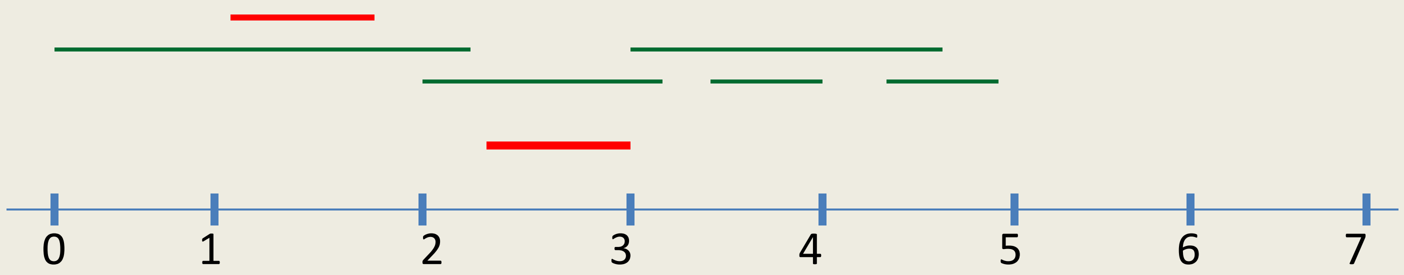
Intuition:

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**



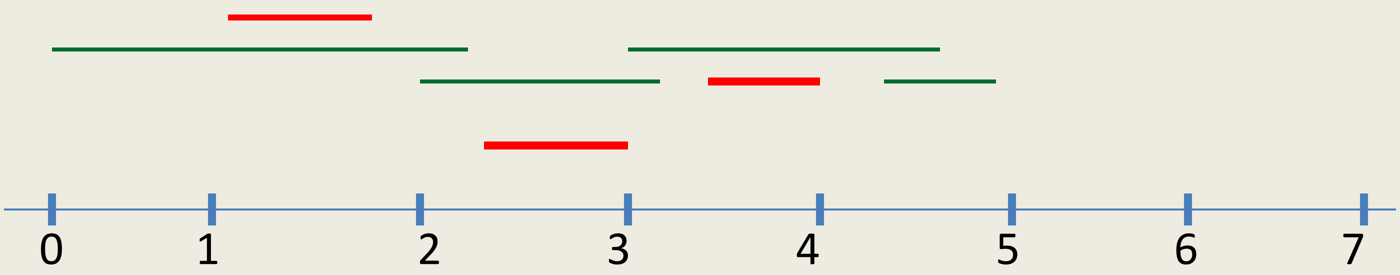
Intuition:

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**



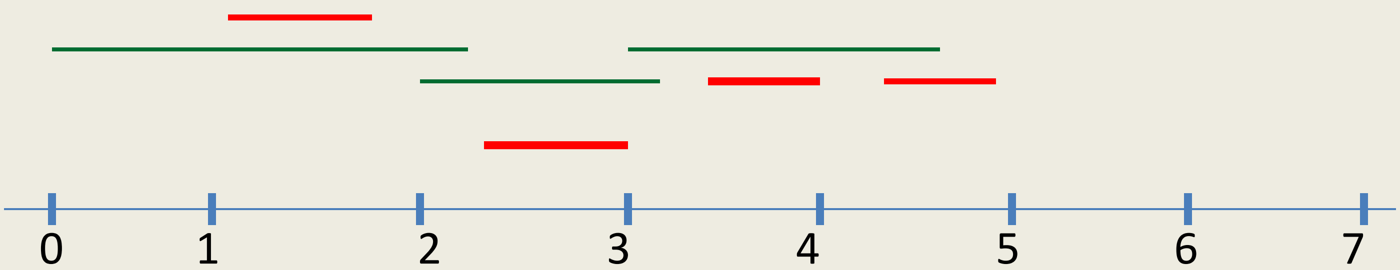
Intuition:

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;

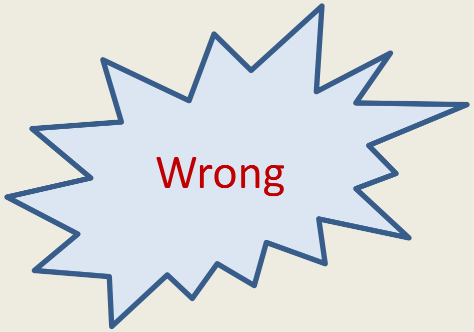
Running time for a trivial implementation of the above algorithm: O(n^2)

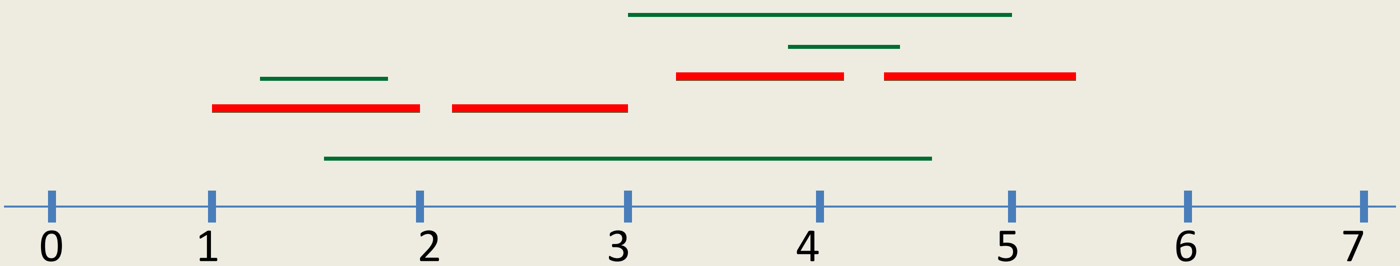
Algorithm "earliest finish time"

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 us suppose $x \notin O$.

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

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

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

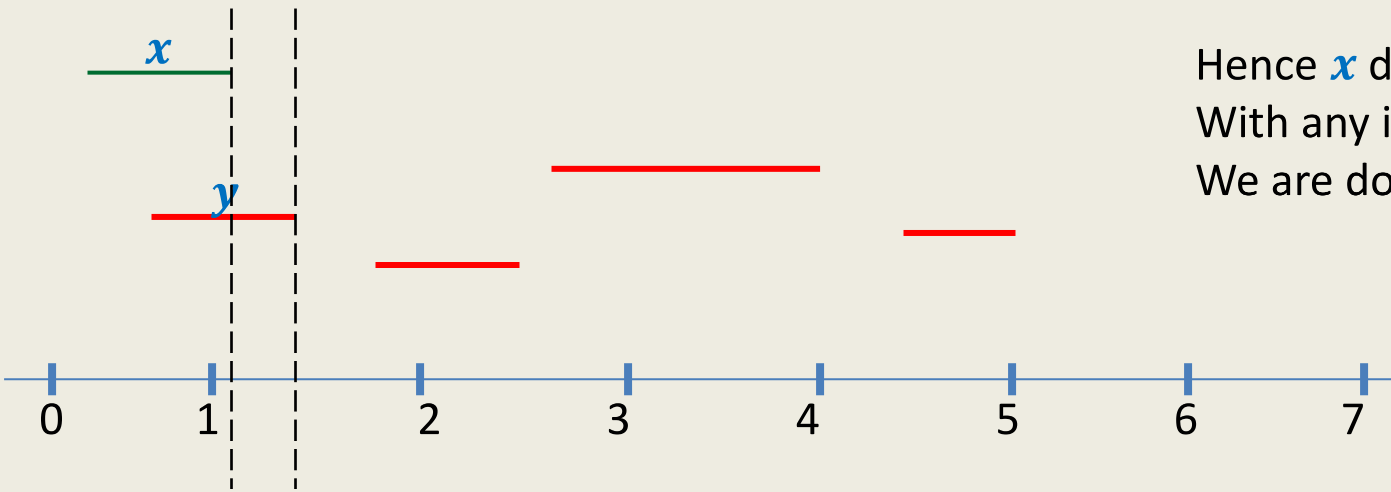
Reason: $O' \cup \{x\}$ has no overlapping intervals. Give reasons.

➔ f(x) ≤ f(y)

➔ f(y) < s(z) ∀ $z \in O'$

➔ f(x) < s(z) ∀ $z \in O'$

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



Homework exercises

Spend 30 minutes today on the following problems.

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

2. Design an **O**($\boldsymbol{n}$log $\boldsymbol{n}$) implementation of the algorithm.