

Greedy Algorithms with Structural Proofs

CS/MCS 401: Computer Algorithms I

Lecture 16

Professor: Jonathan Liu

Icebreaker: We're supposed to have a milder winter this year. How do you feel about that?

Announcements

I am behind. I am very sorry.

Here is my release calendar for this week:

- HW 3 Solutions:
 - Out now!
- HW 3 Grades:
 - Today.
- HW 4 + Worked Examples:
 - Tomorrow.
- Project Specification Document:
 - Thursday lecture.
- All other grades I'm behind on (HW resubmissions especially):
 - Friday.

Today's Overview

Today, we are going to learn about the other major type of proof about greedy algorithms: the **structural proof**.

While this type of proof is more rare than the exchange argument, it is a more direct proof which can make it easier to write the proof and easier to understand the proof.

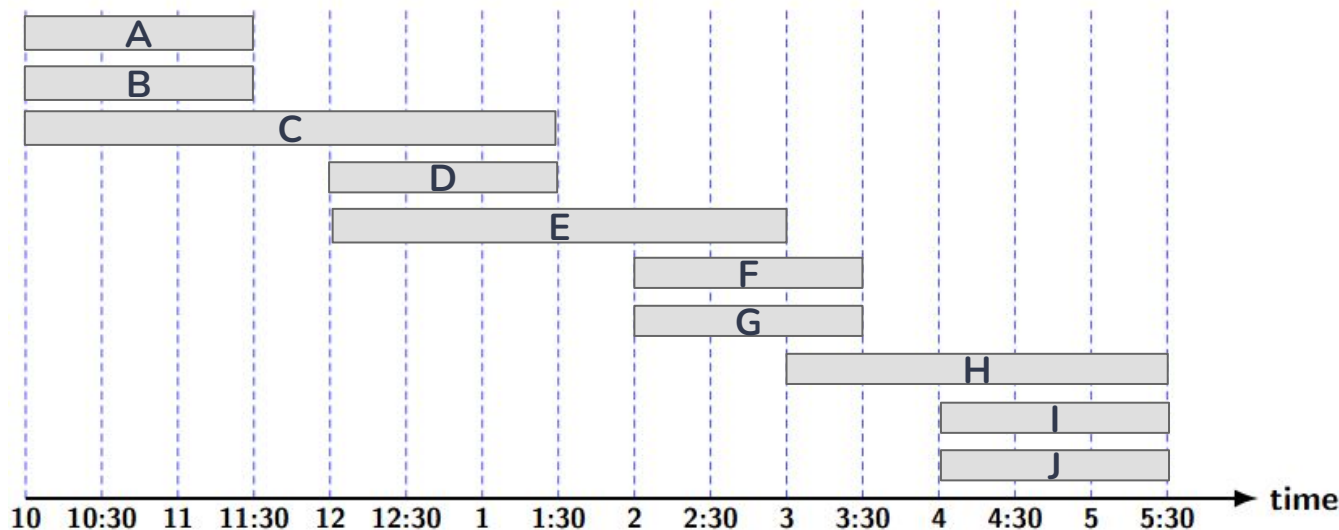
However, it is worth noting that both these types of proofs essentially prove “no other algorithm is better than ours,” which is not exactly the same as “ours is the best.” If nothing else, it makes it no easier to find a heuristic.

Classroom Filling



Classroom Scheduling

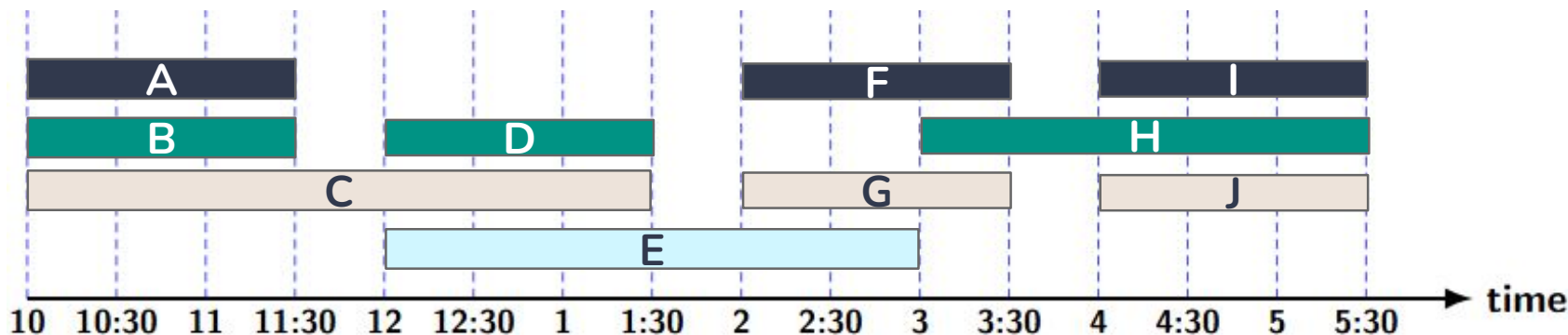
Professor Theys is handling scheduling all of the CS classes for the Spring. He has a list of every class being offered. The school would like him to use as few classrooms as possible so that they can free up other classrooms for the other departments. Find the fewest classrooms he needs to ensure he can schedule them all without conflict.



Classroom Scheduling

Professor Theys is handling scheduling all of the CS classes for the Spring. He has a list of every class being offered. The school would like him to use as few classrooms as possible so that they can free up other classrooms for the other departments. Find the fewest classrooms he needs to ensure he can schedule them all without conflict.

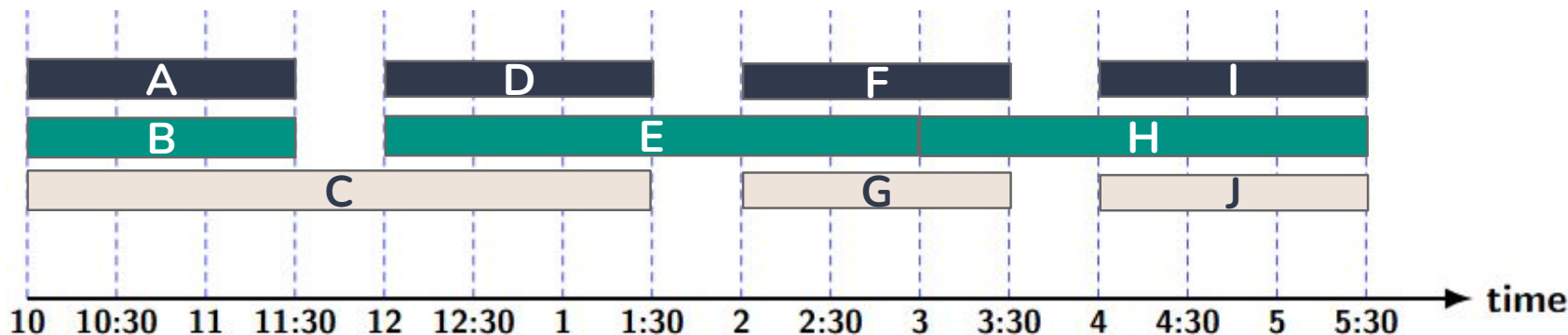
Here's a configuration that uses four classrooms. Can we do better? With what heuristic?



Classroom Scheduling

Professor Theys is handling scheduling all of the CS classes for the Spring. He has a list of every class being offered. The school would like him to use as few classrooms as possible so that they can free up other classrooms for the other departments. Find the fewest classrooms he needs to ensure he can schedule them all without conflict.

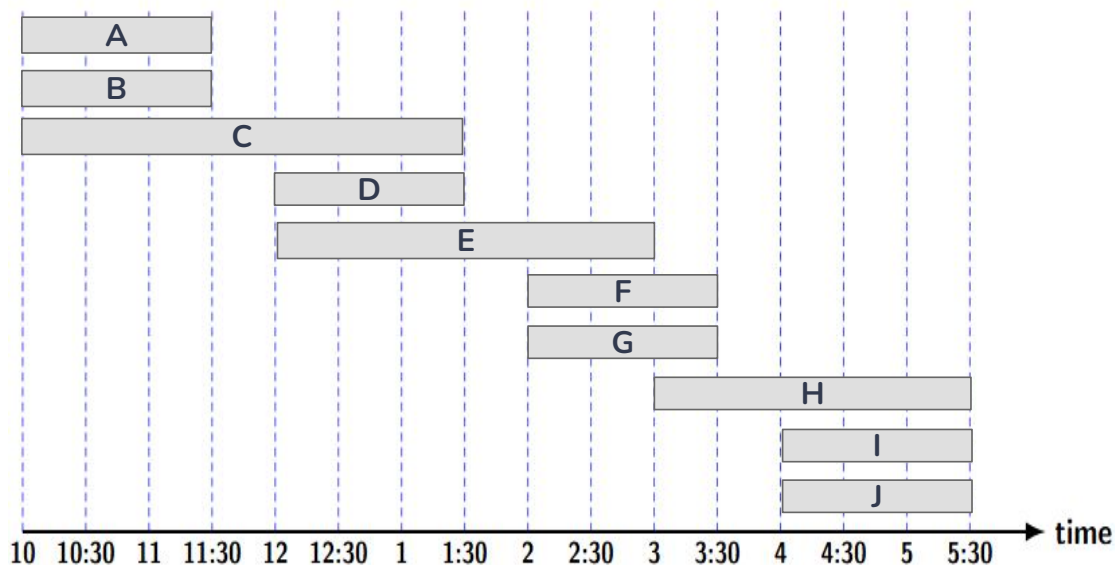
We can do three! But how do we prove that this is the best? And what algorithm?



Designing a Greedy Algorithm

Pseudocode:

```
while not done:  
    pick the best thing  
return the result
```



Designing a Greedy Algorithm

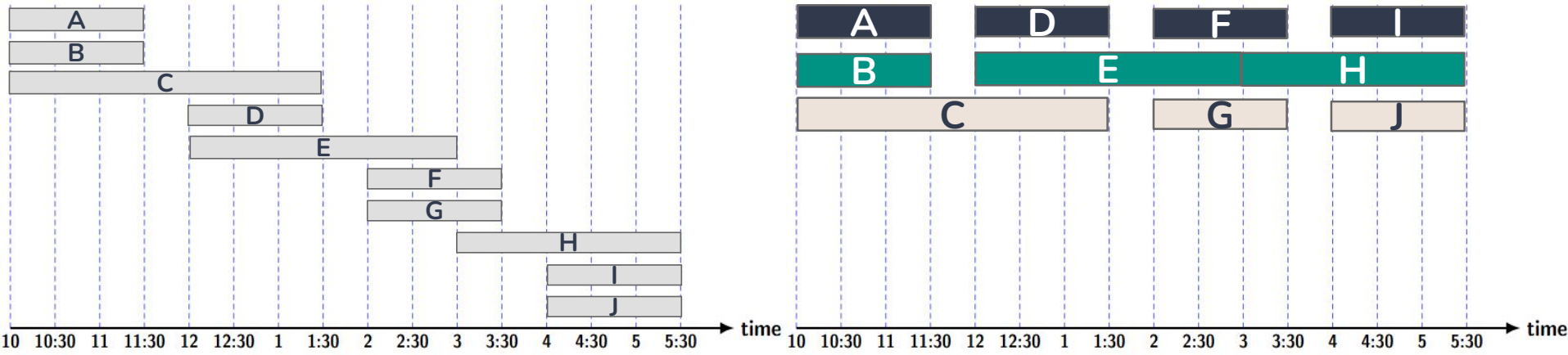
Pseudocode:

```
while not done:
```

```
    Take the next class
```

```
    Put it in the first classroom you can or a new one if none work
```

```
return the result
```



Designing a Greedy Algorithm

Pseudocode:

sort the classes ←

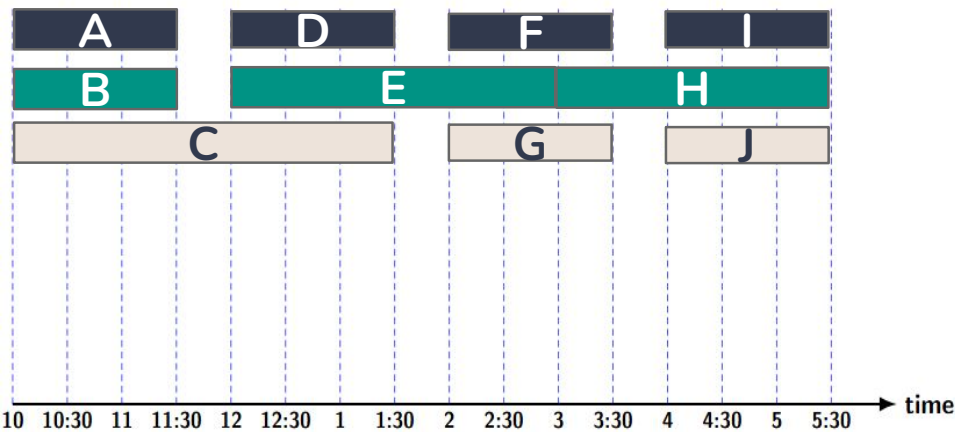
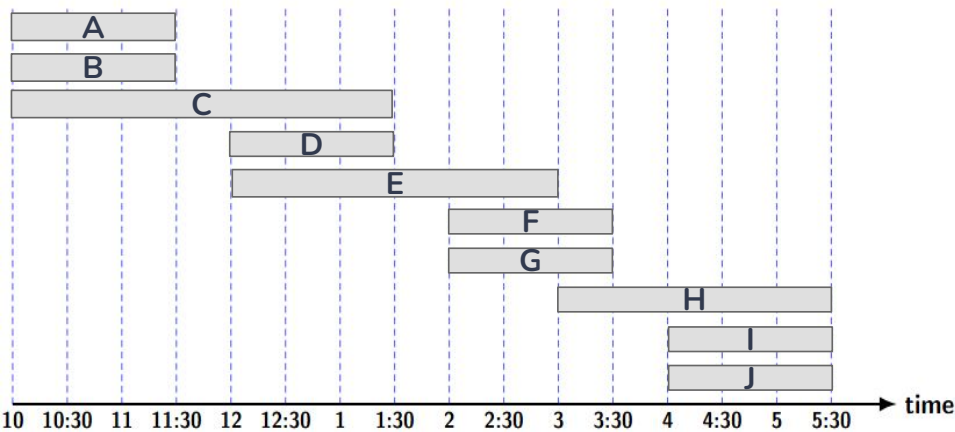
So how do we sort the classes?

while not done:

 Take the next class

 Put it in the first classroom you can or a new one if none work

return the result



Designing a Greedy Algorithm

Pseudocode:

```
sort the classes
```

```
while not done:
```

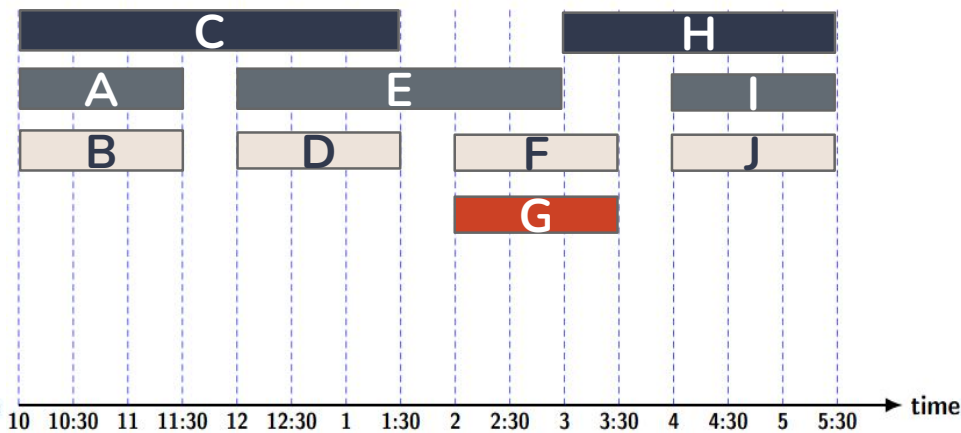
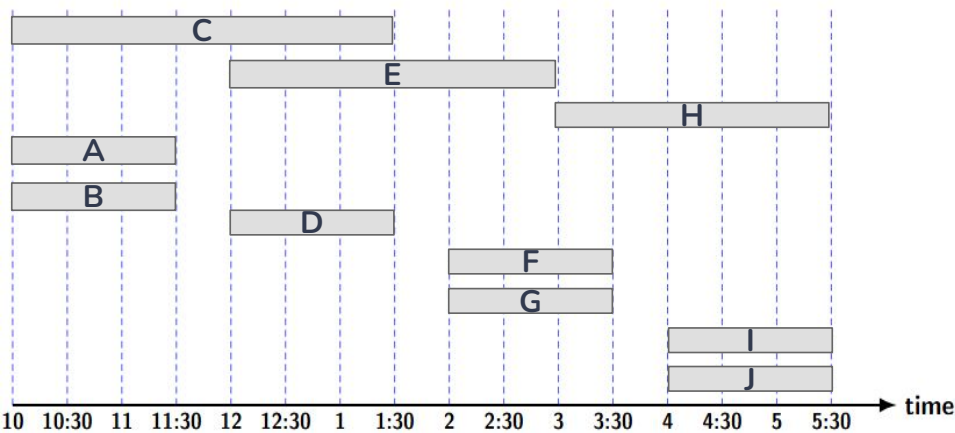
```
    Take the next class
```

```
    Put it in the first classroom you can or a new one if none work
```

```
return the result
```

One Idea: Long to Short

Worksheet: Find the right idea.



Designing a Greedy Algorithm



A counterexample to both “shortest first” and “ends first”.

Designing a Greedy Algorithm

Pseudocode:

```
sort the classes
```

Correct Idea: Earliest Start Time

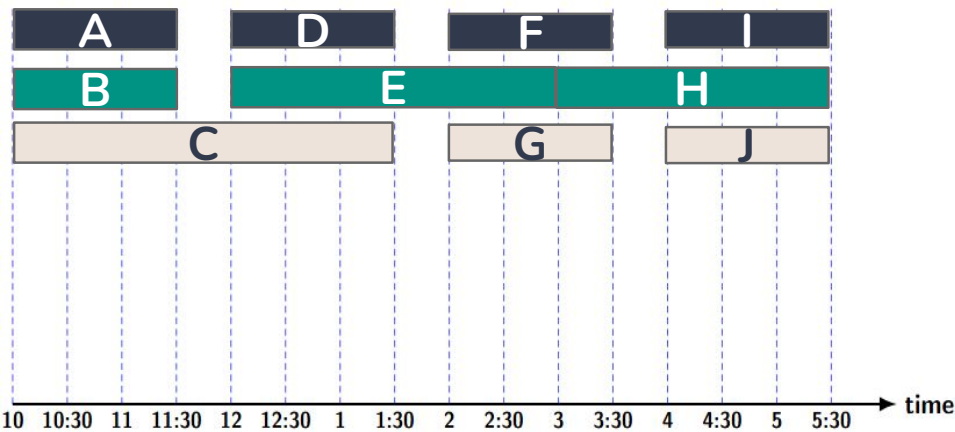
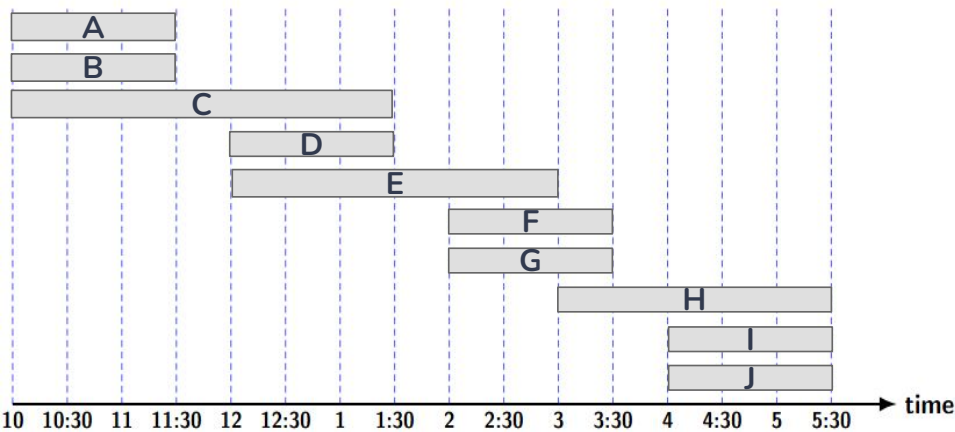
```
while not done:
```

```
    Take the next class
```

But how do we know?

```
    Put it in the first classroom you can or a new one if none work
```

```
return the result
```



Designing a Greedy Algorithm

Pseudocode:

```
sort the classes
```

```
while not done:
```

```
    Take the next class
```

```
    Put it in the first classroom you can or a new one if none work
```

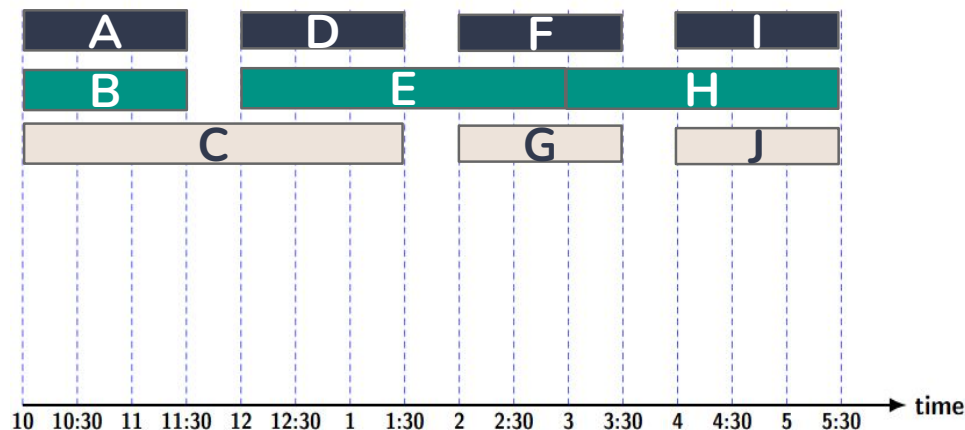
```
return the result
```

Correct Idea: Earliest Start Time

But how do we know?

Take a look at this solution.
How can we be sure there's
no solution that uses two
classes?

It created layer 3 because
there were 3 classes
happening at the same time.



Designing a Greedy Algorithm

Pseudocode:

sort the classes ←

Correct Idea: Earliest Start Time

while not done:

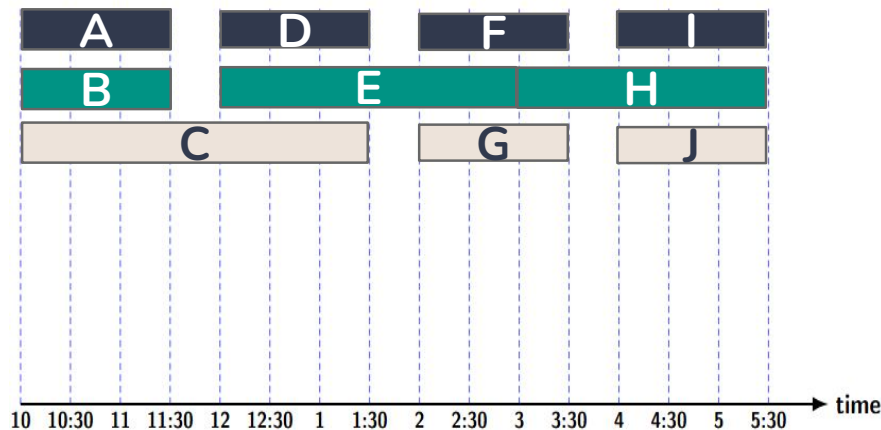
 Take the next class

 Put it in the first classroom you can or a new one if none work

return the result

Proof of Correctness

Say the greedy solution uses n classrooms. Take any class assigned to layer n , say G .



Designing a Greedy Algorithm

Pseudocode:

sort the classes ←

Correct Idea: Earliest Start Time

while not done:

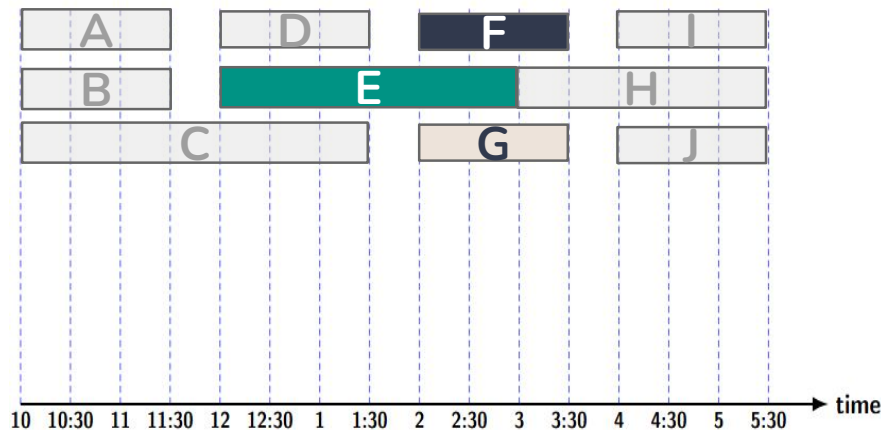
 Take the next class

 Put it in the first classroom you can or a new one if none work

return the result

Proof of Correctness

Say the greedy solution uses n classrooms. Take any class assigned to layer n , say G . If it was assigned to layer n , what does that tell us?



Designing a Greedy Algorithm

Pseudocode:

sort the classes ←

Correct Idea: Earliest Start Time

while not done:

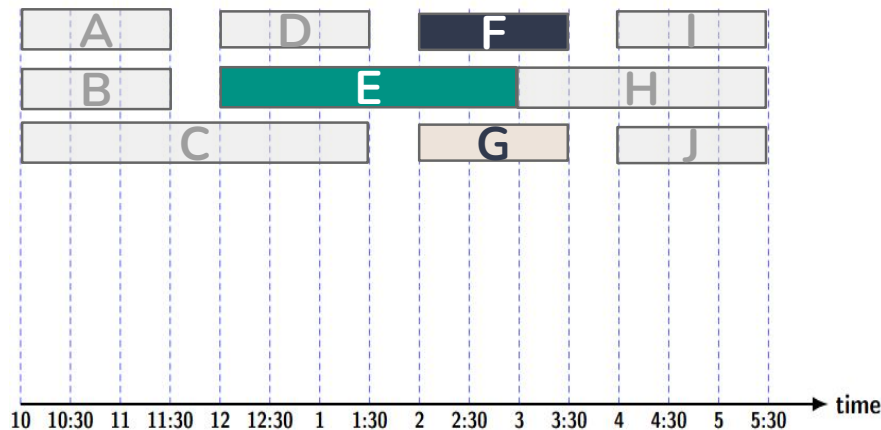
 Take the next class

 Put it in the first classroom you can or a new one if none work

return the result

Proof of Correctness

Say the greedy solution uses n classrooms. Take any class assigned to layer n , say G . Because it was assigned to layer n , there must be $n-1$ classes overlapping with it.



Designing a Greedy Algorithm

Pseudocode:

sort the classes ←

Correct Idea: Earliest Start Time

while not done:

 Take the next class

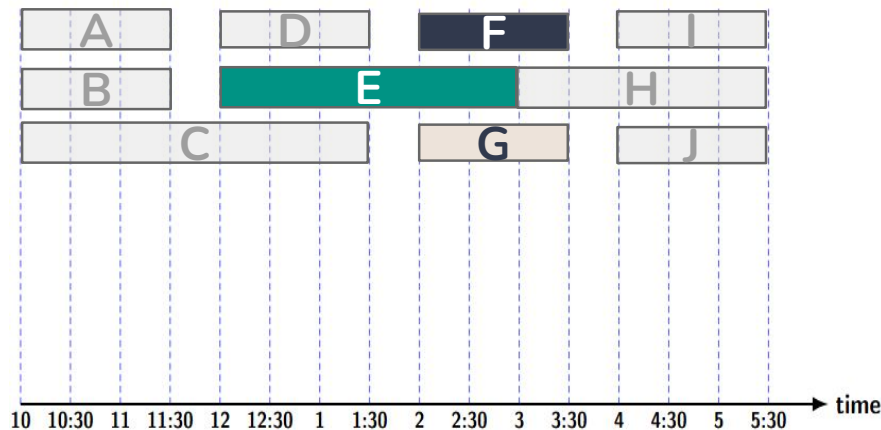
 Put it in the first classroom you can or a new one if none work

return the result

Proof of Correctness

Say the greedy solution uses n classrooms. Take any class assigned to layer n , say G . Because it was assigned to layer n , there must be $n-1$ classes overlapping with it.

What do we know about those $n-1$ classes?



Designing a Greedy Algorithm

Pseudocode:

sort the classes ←

Correct Idea: Earliest Start Time

while not done:

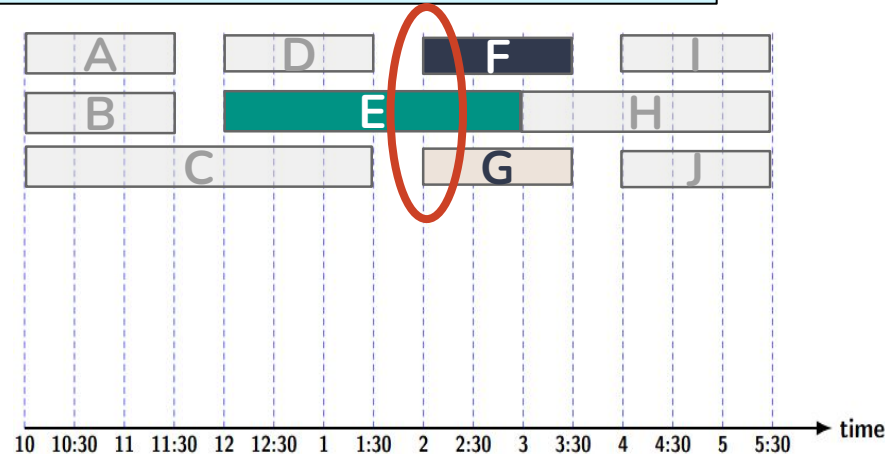
 Take the next class

 Put it in the first classroom you can or a new one if none work

return the result

Proof of Correctness

Say the greedy solution uses n classrooms. Take any class assigned to layer n , say G . Because it was assigned to layer n , there must be $n-1$ classes overlapping with it. Because they are placed by earliest start time, they must all overlap with the start of class G .



Designing a Greedy Algorithm

Pseudocode:

sort the classes ←

Correct Idea: Earliest Start Time

while not done:

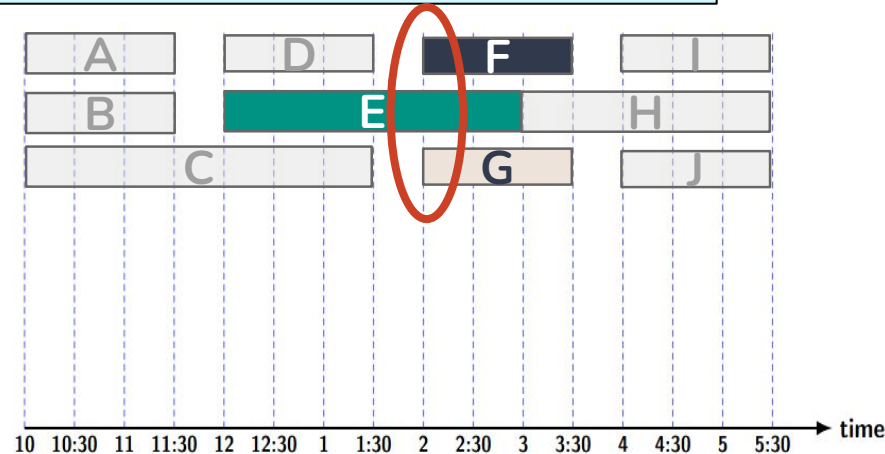
 Take the next class

 Put it in the first classroom you can or a new one if none work

return the result

Proof of Correctness

Say the greedy solution uses n classrooms. Take any class assigned to layer n , say G . Because it was assigned to layer n , there must be $n-1$ classes overlapping with it. Because they are placed by earliest start time, they must all overlap with the start of class G . *What does that tell us?*



Designing a Greedy Algorithm

Pseudocode:

sort the classes ←

Correct Idea: Earliest Start Time

while not done:

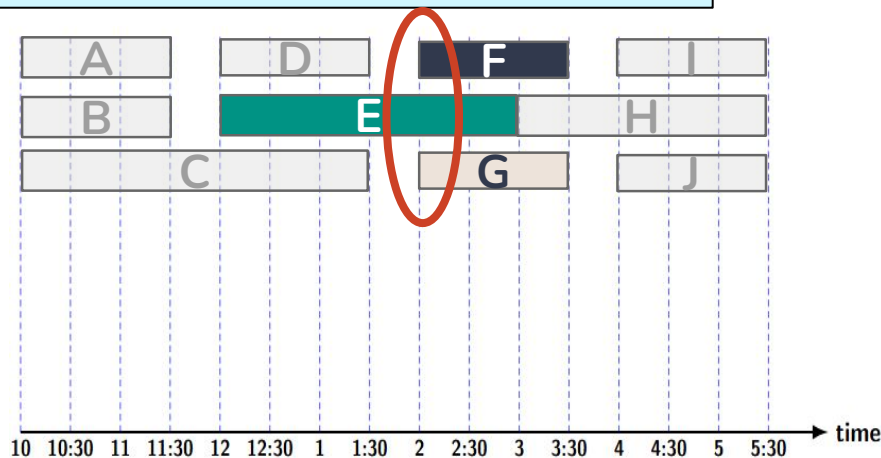
 Take the next class

 Put it in the first classroom you can or a new one if none work

return the result

Proof of Correctness

Say the greedy solution uses n classrooms. Take any class assigned to layer n , say G . Because it was assigned to layer n , there must be $n-1$ classes overlapping with it. Because they are placed by earliest start time, they must all overlap with the start of class G . Those $n-1$ classes, as well as class G , all take place during the start of class G , so these are n different classes occurring at the same time.



Designing a Greedy Algorithm

Pseudocode:

sort the classes ←

Correct Idea: Earliest Start Time

while not done:

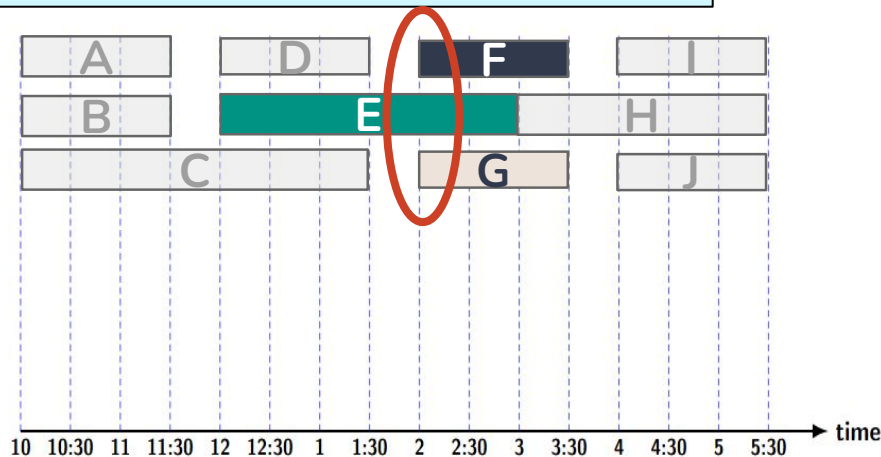
 Take the next class

 Put it in the first classroom you can or a new one if none work

return the result

Proof of Correctness

Say the greedy solution uses n classrooms. Take any class assigned to layer n , say G . Because it was assigned to layer n , there must be $n-1$ classes overlapping with it. Because they are placed by earliest start time, they must all overlap with the start of class G . Those $n-1$ classes, as well as class G , all take place during the start of class G , so these are n different classes occurring at the same time. Thus, no solution with less than n classrooms exists.



Structural vs. Exchange Proof

Proof Outline

Because the greedy algorithm used in classrooms, we know that there must be some point where n classes overlap at the same time.

Therefore, we know that no algorithm can do better than greedy.

This is a **structural proof**: we are making an argument about what this algorithm reveals about the **structure of the input** that ensures no algorithm can do better than ours.

These proofs, like exchange proofs, work by showing that no other solution can be better than the greedy solution.

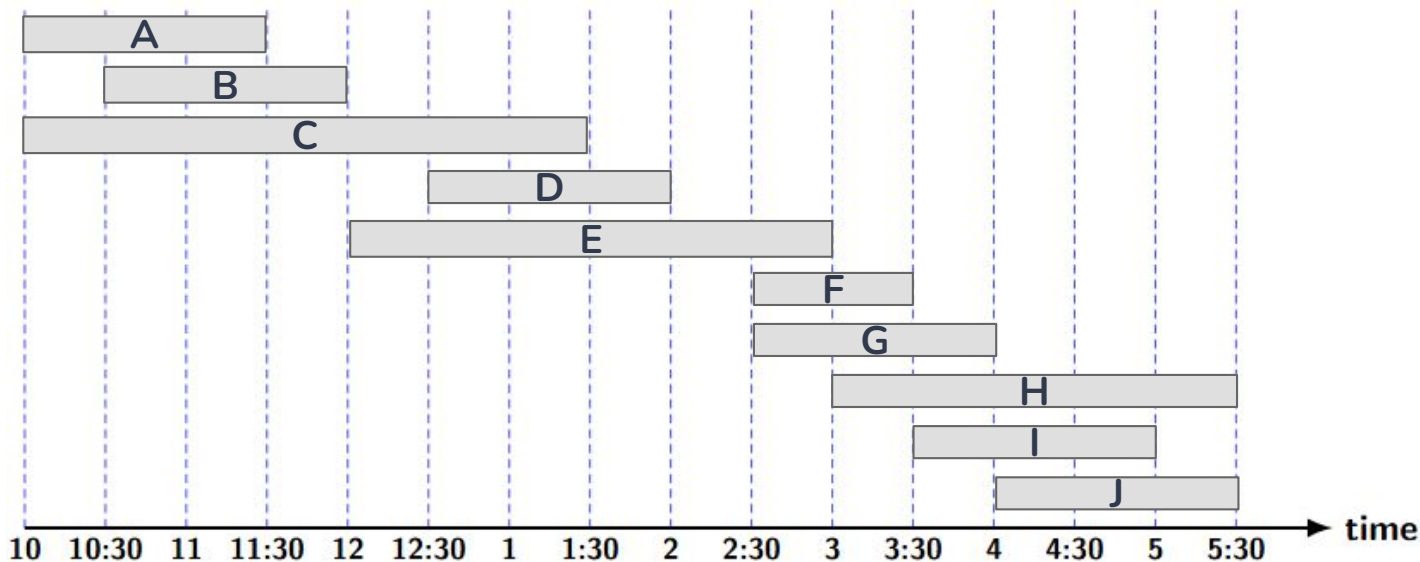
However, no exchange is happening here. Exchange proofs say “any optimal solution can slowly be changed into the greedy one”, while structural proofs say “it’s physically impossible for any solution to do better than the greedy one.” It is an argument about the problem itself, not about another solution.

Missing Meetings



Missing Meetings

It's a busy day at work: you have n scheduled meetings. You'd like to be excused from all of them. However, you can't be excused from a meeting unless you have another meeting at the same time. Thankfully, you have friends at work who'd agree to join a fake 1-minute meeting with you. For example, if you create a fake meeting at 1pm, you'd be excused from meetings C, D, and E. Find the minimum number of fake meetings you need to create to be excused from all n (real) meetings.



Lecture Recap

Today we saw a different style of proof of correctness for Greedy Algorithms: the structural proof.

Structural proofs are less common, and are harder to discover, but once you discover them they are a much easier proof to write.

One common theme of greedy algorithm proofs is that you're never directly proving that your algorithm is the best; you can only prove that no other algorithm can possibly do better.