

Interval Scheduling and Exchange Argument

CS/MCS 401: Computer Algorithms I

Lecture 14

Professor: Jonathan Liu

Icebreaker: Did you have a favorite stuffed animal growing up? Where is it now?

Midterm Debrief

Midterm scores have been released. They are a bit lower than I was hoping they'd be.

- Please fill out the **Midterm Debrief Form** (linked on Piazza) to let me know what you thought of the midterm! It's totally okay to be honest – you won't be penalized for any criticism about the exam. I just want to know how it was for you.
- **You are getting the chance to redo one problem on the midterm.** This will consist of a 15-minute one-on-one meeting with me. Signups are gated behind the Debrief form.
- Solutions have been released, and regrade requests are open this week.

Today's Overview

Today is the start of **Unit 4: Greedy Algorithms**.

Greedy Algorithms are easy to write, but there are a few catches:

1. It's also incredibly easy to write an incorrect greedy algorithm.
2. Proofs about greedy algorithms are surprisingly tricky, so it can be hard to tell whether your algorithm is correct or not.
3. Many problems *feel* like they can be solved greedily, but actually can't.

Unit 4: Greedy Algorithms



Greedy Algorithm Blueprint

Pseudocode:

while not done:

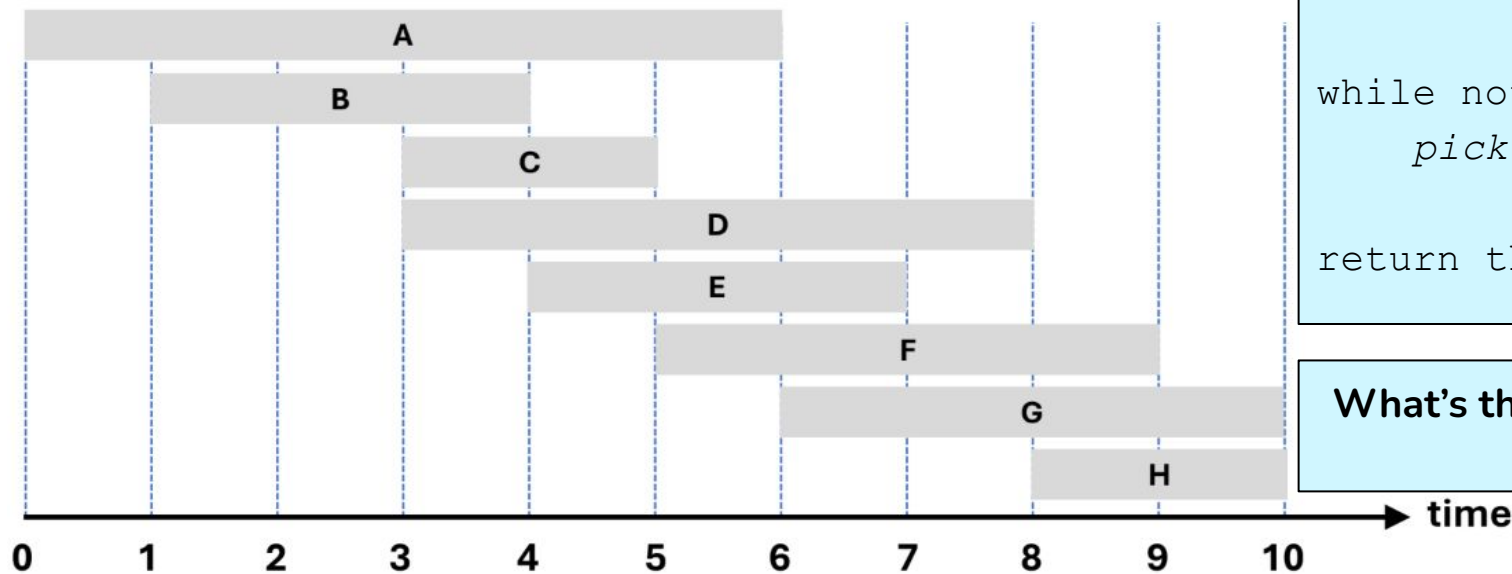
pick the best thing

return the result

This is a *heuristic*.

Course Scheduling

You're only willing to come onto campus one day per week, so you'd like to squeeze as many classes as you can into the one day.



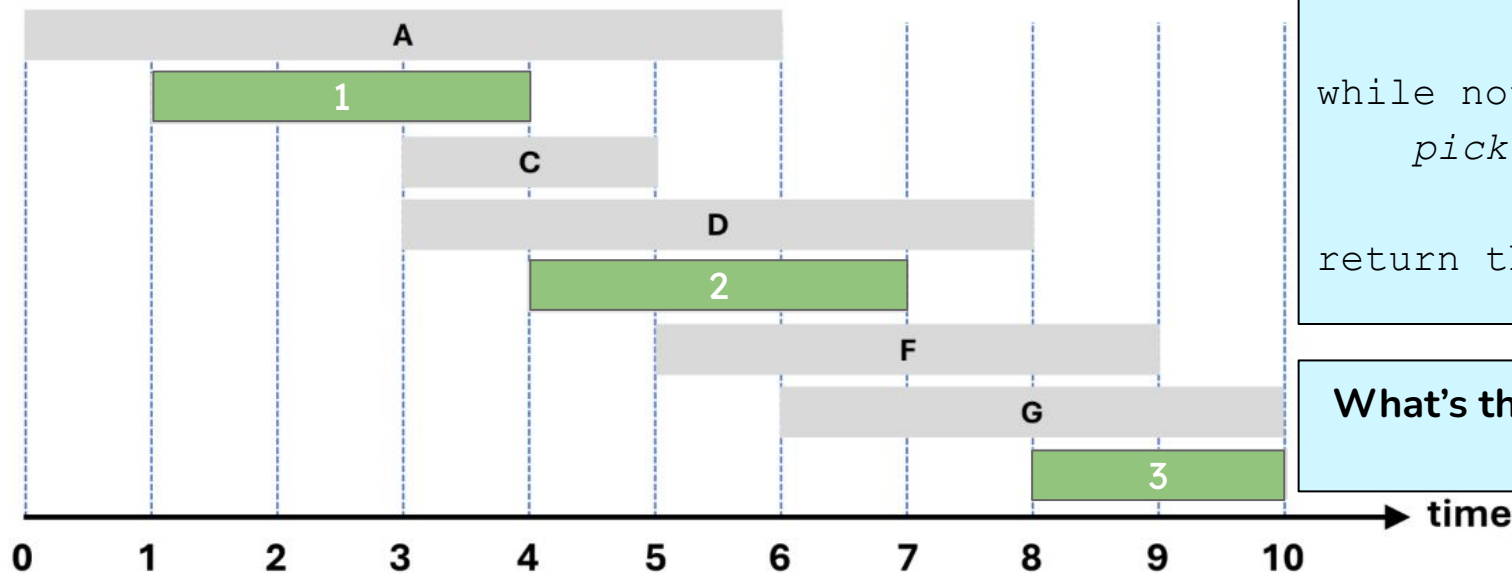
Pseudocode:

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

What's the optimal outcome here?

Course Scheduling

You're only willing to come onto campus one day per week, so you'd like to squeeze as many classes as you can into the one day.



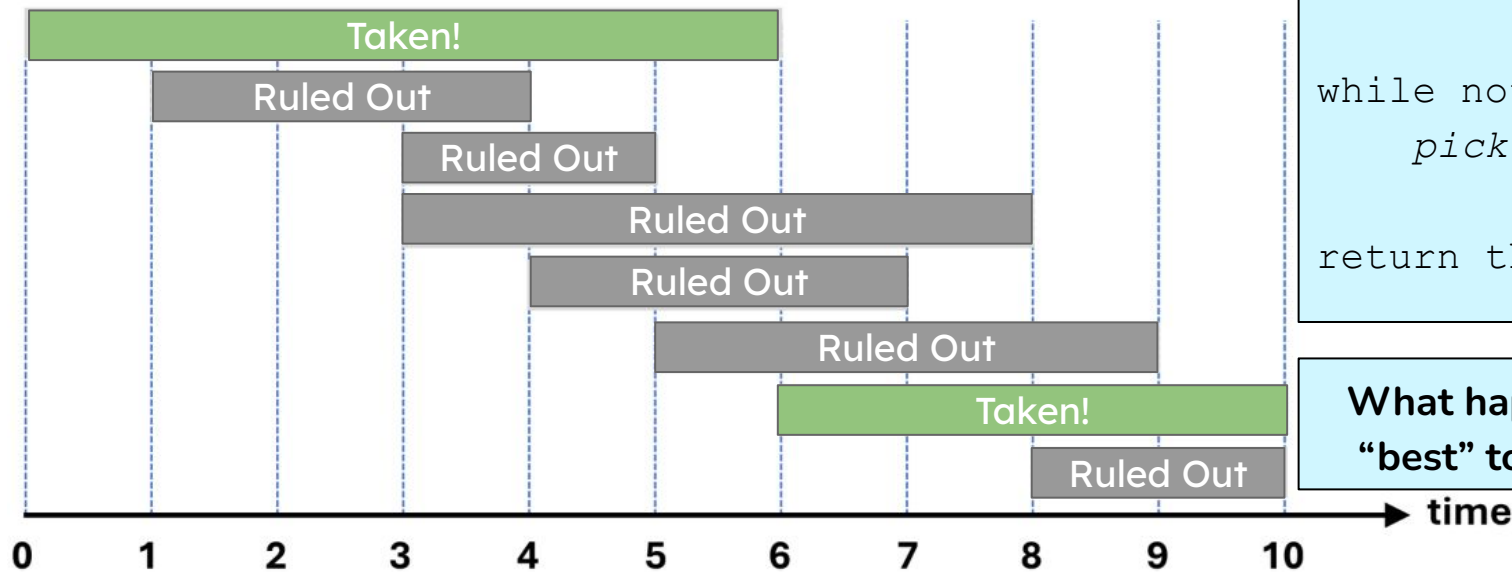
Pseudocode:

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

What's the optimal outcome here?

Course Scheduling

You're only willing to come onto campus one day per week, so you'd like to squeeze as many classes as you can into the one day.



Pseudocode:

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

What happens if we define
“best” to be “starts first”?

Potential Heuristics

You're only willing to come onto campus one day per week, so you'd like to squeeze as many classes as you can into the one day.

Which class is best to choose?

- The one that starts first.
- The one that starts last.
- The one that ends first.
- The one that ends last.
- The shortest class.
- The one that overlaps with the fewest other classes.

Potential Heuristics

You're only willing to come onto campus one day per week, so you'd like to squeeze as many classes as you can into the one day.

Which class is best to choose?

~~— The one that starts first.~~

- The one that starts last.
- The one that ends first.
- **The one that ends last.**
- The shortest class.
- The one that overlaps with the fewest other classes.



Ruled Out

Ruled Out

Potential Heuristics

You're only willing to come onto campus one day per week, so you'd like to squeeze as many classes as you can into the one day.

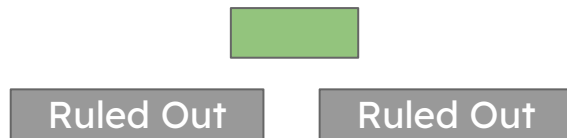
Which class is best to choose?

~~— The one that starts first.~~

- The one that starts last.
- The one that ends first.

~~— The one that ends last.~~

- **The shortest class.**
- The one that overlaps with the fewest other classes.



Potential Heuristics

You're only willing to come onto campus one day per week, so you'd like to squeeze as many classes as you can into the one day.

Which class is best to choose?

~~— The one that starts first.~~

- The one that starts last.
- The one that ends first.

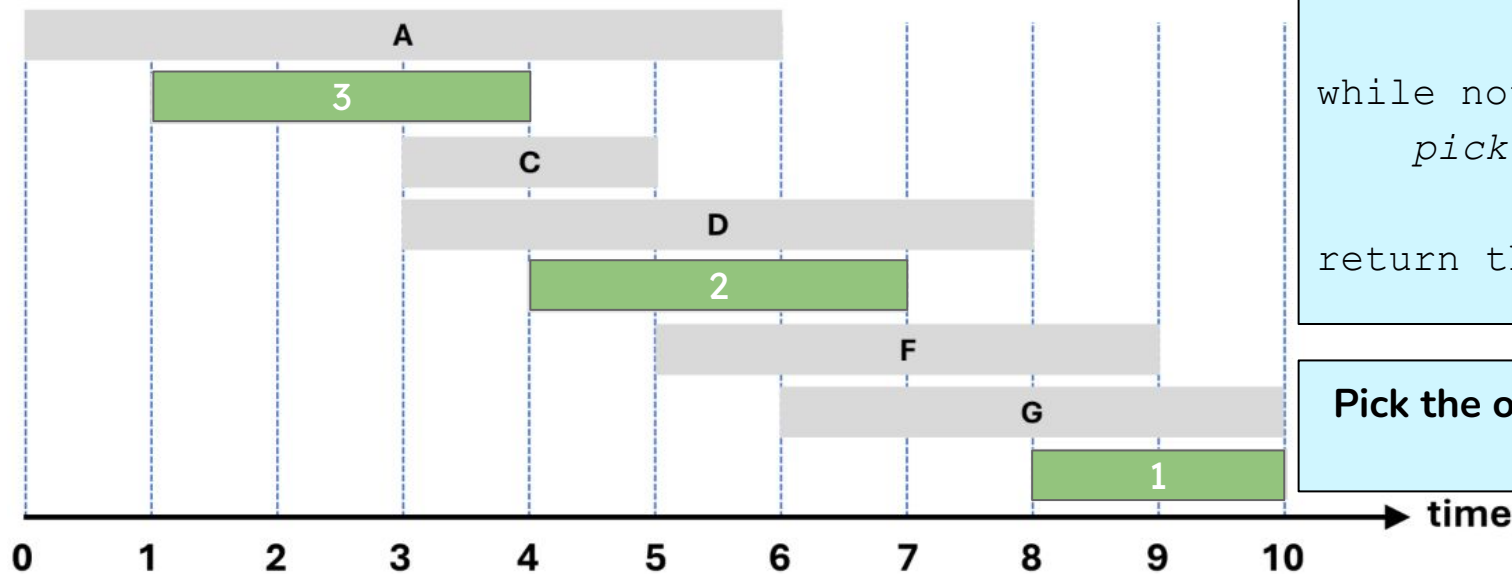
~~— The one that ends last.~~

~~— The shortest class.~~

- **The one that overlaps with the fewest other classes.**

Course Scheduling

You're only willing to come onto campus one day per week, so you'd like to squeeze as many classes as you can into the one day.



Pseudocode:

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

Pick the one with the fewest overlaps.

Potential Heuristics

You're only willing to come onto campus one day per week, so you'd like to squeeze as many classes as you can into the one day.

Which class is best to choose?

~~— The one that starts first.~~

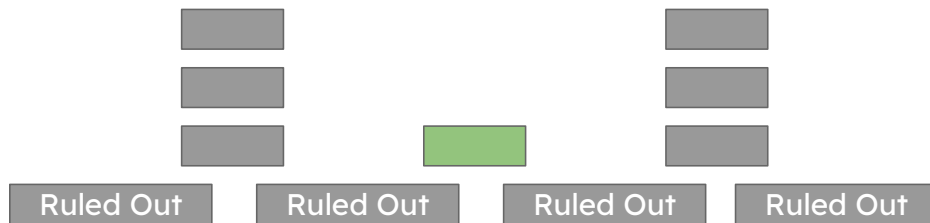
- The one that starts last.

- The one that ends first.

~~— The one that ends last.~~

~~— The shortest class.~~

- **The one that overlaps with the fewest other classes.**



Potential Heuristics

You're only willing to come onto campus one day per week, so you'd like to squeeze as many classes as you can into the one day.

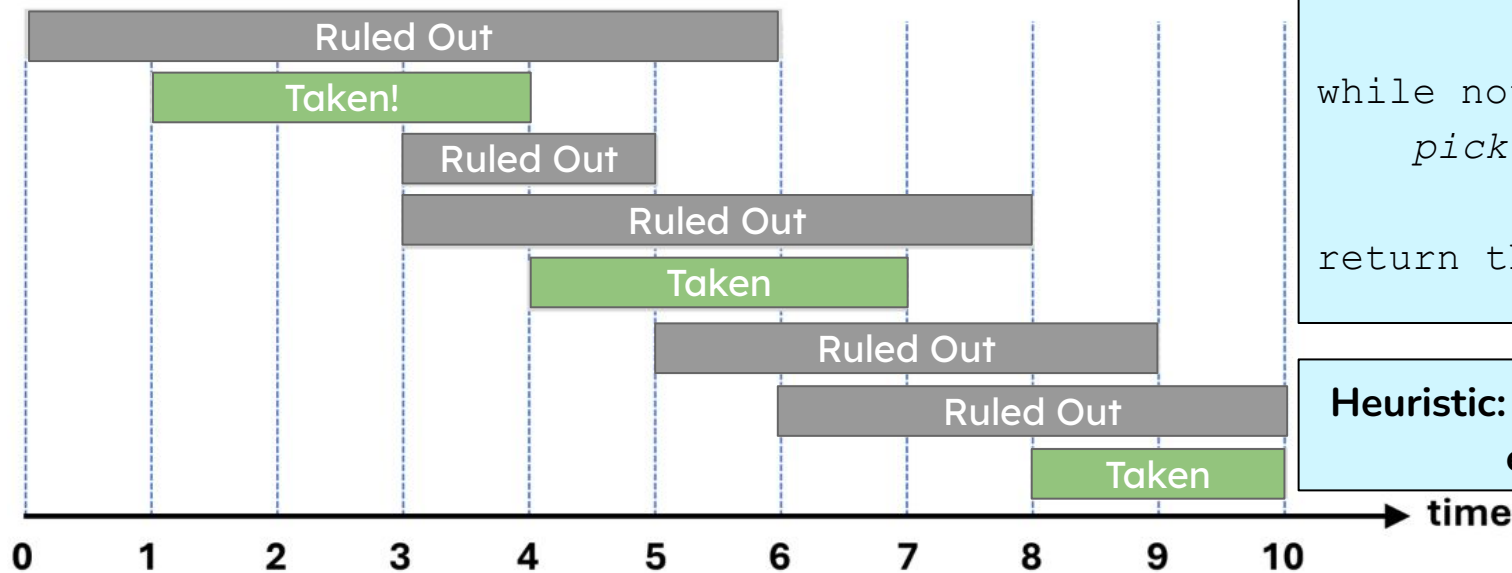
Which class is best to choose?

- ~~— The one that starts first.~~
- The one that starts last.
- **The one that ends first.**
- ~~— The one that ends last.~~
- ~~— The shortest class.~~
- ~~— The one that overlaps with the fewest other classes.~~

**Turns out these are the same!
Let's look at the second one.**

Course Scheduling

You're only willing to come onto campus one day per week, so you'd like to squeeze as many classes as you can into the one day.



Pseudocode:

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

Heuristic: Take the class that ends first.

Course Scheduling Algorithm

You're only willing to come onto campus one day per week, so you'd like to squeeze as many classes as you can into the one day.

Pseudocode:

```
while there are still valid classes:  
    Pick the one that ends first  
  
return the result
```

What's the runtime of this algorithm?

It takes $O(n)$ time to iterate through them.

But first, it takes $O(n \log n)$ time to sort the classes by ending time!

Proof of Correctness

If we want to prove the correctness of this algorithm, what claim do we need to prove?

We can't prove that our algorithm outputs the **best** schedule. It's possible that there's more than one best schedule, and ours is only one of them!

We instead need to prove that:

No other schedule does better than the one output by our algorithm.

Pseudocode:

```
while there are still valid classes:  
    Pick the one that ends first  
  
return the result
```

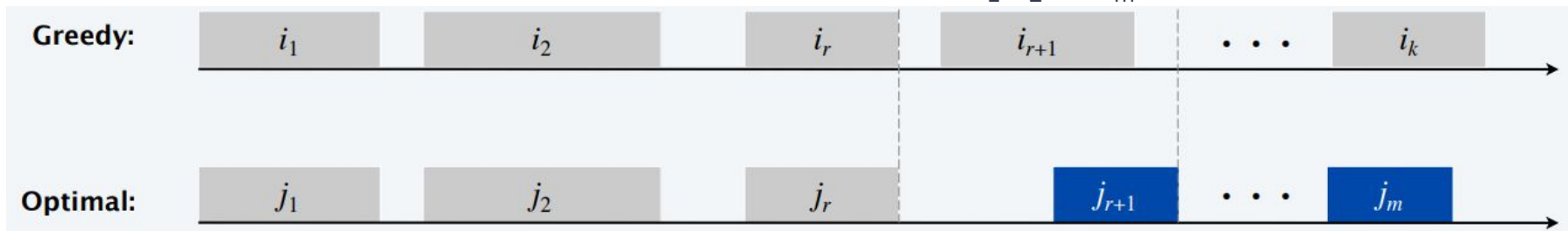
Proof of Correctness

Heuristic:

Pick the valid class that ends first.

Claim: No other schedule does better than the one output by our algorithm.

Take any optimal solution. Label the classes selected by the greedy algorithm as $i_1, i_2, \dots, i_n$ and label the classes selected by the optimal solution as $j_1, j_2, \dots, j_m$.



If they're the same classes, then obviously the greedy solution is optimal. So assume, instead, that they're different somewhere. Say that $i_1=j_1, i_2=j_2, \dots$, all the way until $i_r=j_r$, but $i_{r+1} \neq j_{r+1}$. **What do we know about j_{r+1} in relation to i_{r+1} ?**

We know that j_{r+1} must end later than i_{r+1} . **If we exchange j_{r+1} with i_{r+1} , we'd still have an optimal schedule.**

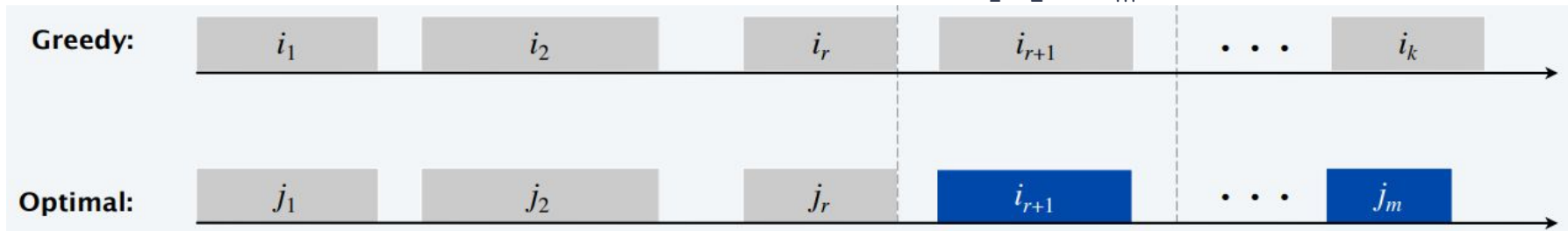
Proof of Correctness

Heuristic:

Pick the valid class that ends first.

Claim: No other schedule does better than the one output by our algorithm.

Take any optimal solution. Label the classes selected by the greedy algorithm as $i_1, i_2, \dots, i_n$ and label the classes selected by the optimal solution as $j_1, j_2, \dots, j_m$.



If they're the same classes, then obviously the greedy solution is optimal. So assume, instead, that they're different somewhere. Say that $i_1=j_1, i_2=j_2, \dots$, all the way until $i_r=j_r$, but $i_{r+1} \neq j_{r+1}$. **What do we know about j_{r+1} in relation to i_{r+1} ?**

We know that j_{r+1} must end later than i_{r+1} . **If we exchange j_{r+1} with i_{r+1} , we'd still have an optimal schedule.**

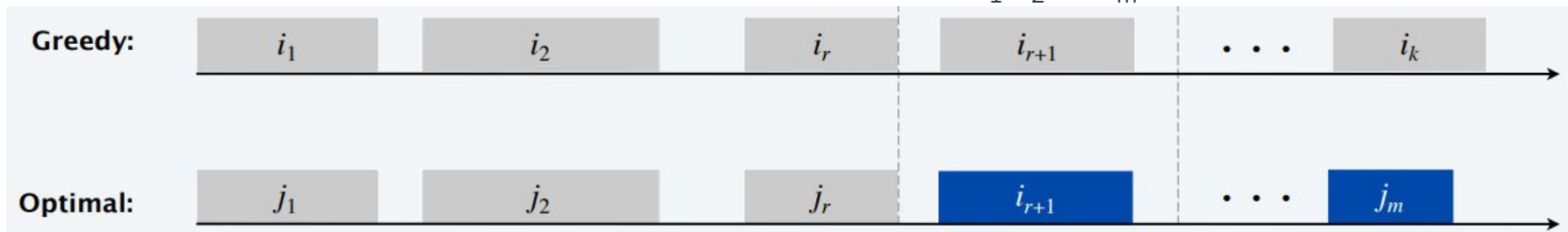
Proof of Correctness

Heuristic:

Pick the valid class that ends first.

Claim: No other schedule does better than the one output by our algorithm.

Take any optimal solution. Label the classes selected by our greedy heuristic as $i_1, i_2, \dots, i_n$ and label the classes selected by the optimal solution as $j_1, j_2, \dots, j_m$.



If they're the same classes, then obviously the greedy solution is optimal. So assume, instead, that they're different somewhere. Say that $i_1=j_1, i_2=j_2, \dots$, all the way until $i_r=j_r$, but $i_{r+1} \neq j_{r+1}$. **What do we know about j_{r+1} in relation to i_{r+1} ?**

We know that j_{r+1} must end later than i_{r+1} . **If we exchange j_{r+1} with i_{r+1} , we'd still have an optimal schedule.** But if we keep exchanging, we'll end up with the greedy schedule!

Exchange Arguments

Heuristic:

Pick the valid class that ends first.

1. Assume there is some optimal schedule that is not the greedy one. Then denote the greedy schedule by $g_1, g_2, \dots, g_n$ and the optimal schedule by $o_1, o_2, \dots, o_m$.
2. Denote by i the first class where they differ, so g_i is not o_i but $g_{i-1} = o_{i-1}$.
3. We know, by the greedy heuristic, that o_i must end later than g_i . Therefore, we know that g_i does not conflict with o_{i-1} because g_i does not conflict with g_{i-1} , and we know g_i does not conflict with o_{i+1} because g_i ends before o_i . Thus, we can replace o_i with g_i in the optimal solution.
4. If $m > n$, then there is some o_{n+1} . By the exchange argument, o_{n+1} does not intersect with g_n , but then the greedy algorithm would have selected an $n+1$ th class as well. Thus, it is impossible for $m > n$.

Blueprint:

1. Assume that there is some optimal solution that is different from the greedy solution.
2. Label the **first** selection where they differ.
3. Argue that, if that selection was replaced with what the greedy algorithm selected, the solution would remain optimal.
4. Conclude that no optimal solution can be better than the greedy solution.

Potential Heuristics

You're only willing to come onto campus one day per week, so you'd like to squeeze as many classes as you can into the one day.

Which class is best to choose?

- ~~— The one that starts first.~~
- The one that starts last.
- **The one that ends first.**
- ~~— The one that ends last.~~
- ~~— The shortest class.~~
- ~~— The one that overlaps with the fewest other classes.~~

Why are these the same?

If you reverse the intervals and pick the one that ends first, it's the same as picking the original interval that started last!

Lecture Recap

Today we began a new unit: Greedy Algorithms.

While greedy algorithms are very easy to understand and implement, they can be surprisingly difficult both because it can be hard to tell which heuristic to use and because proving their correctness is quite tricky.