Introduction to Dynamic Programming

CS/MCS 401: Computer Algorithms I
Lecture 18
Professor: Jonathan Liu

Icebreaker: If you had to guess what *dynamic programming* was, what would you guess?

Announcements

Canvas updates

- All future assignments, with due dates, are now on Canvas.
- Your grades from the first half of the course are now on Canvas as well.

Project

- **Teams are due by the end of the day today.** That is as far as I am willing to extend it. If you do not submit today, I will assume you are working alone.
- **Phase 1 (Inputs) is due on Friday.** Remember to get started early so you don't get surprised by the autograder.

One-on-One OH will be announced Thursday.

Today's Overview

Today, we're moving on to a new unit: Dynamic Programming. This is the final algorithmic paradigm of the course.

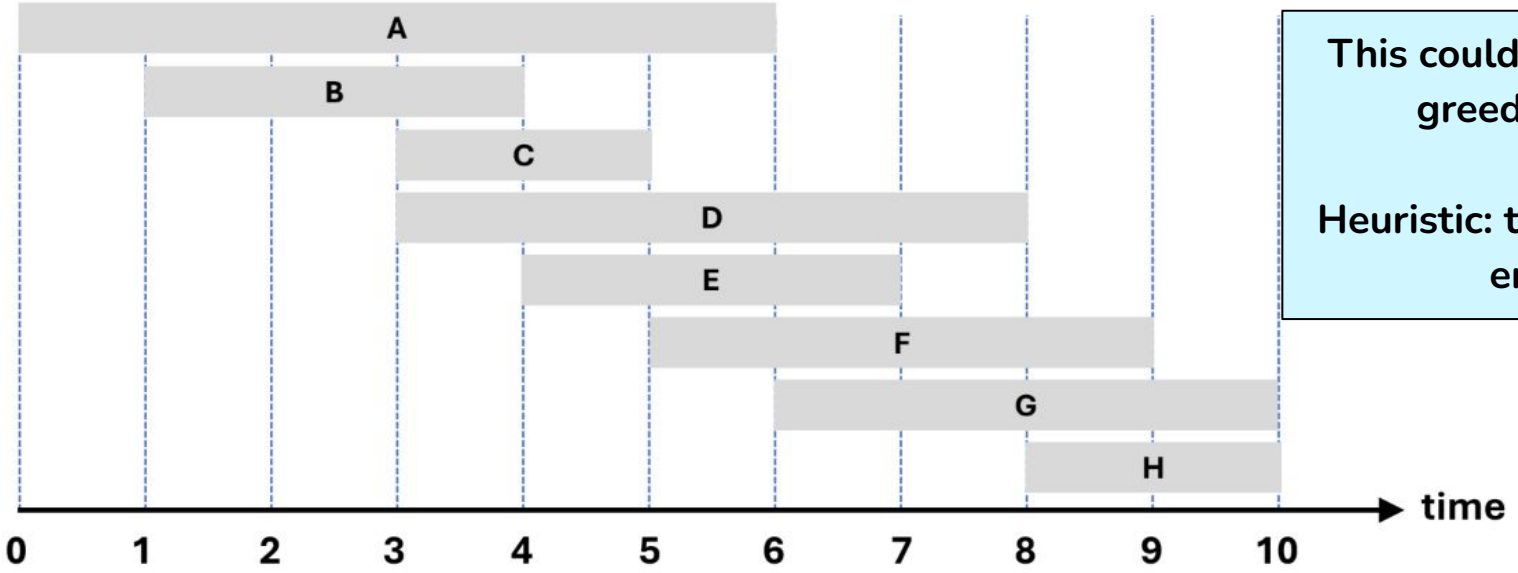
Dynamic Programming is the most versatile paradigm, but it can be more complex to design these algorithms than the ones prior units. That said, they (like the other units) have a recurrent underlying structure.

Today will be spent *understanding* these algorithms: future lectures will talk more about designing them.

Weighted Interval Scheduling

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.

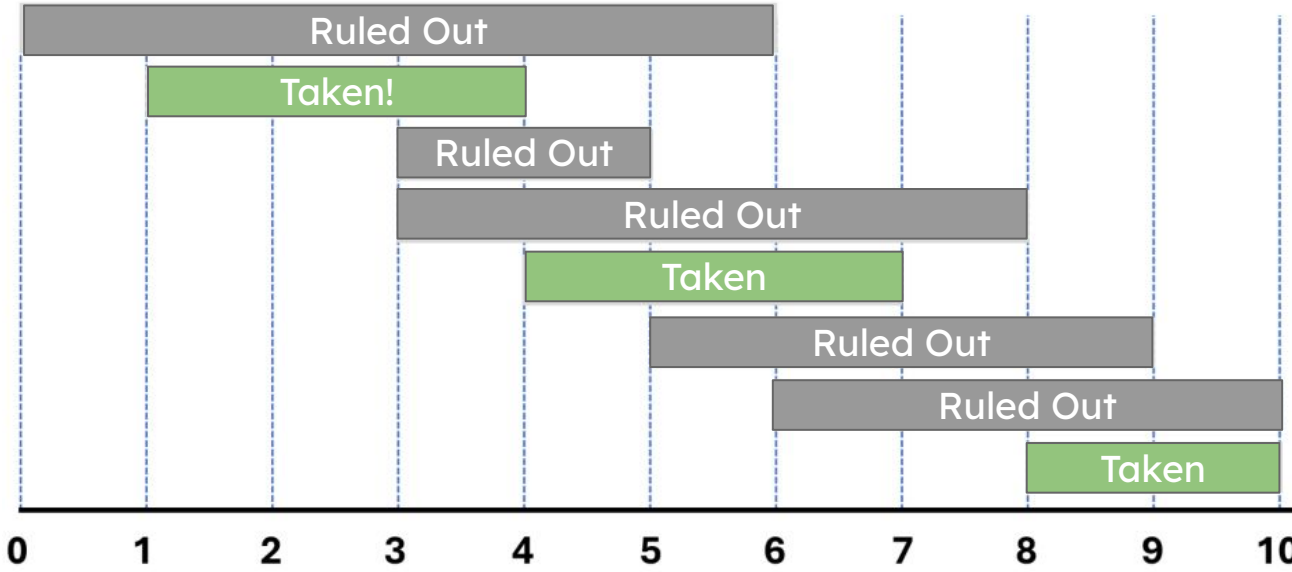


This could be solved with a greedy algorithm!

Heuristic: take the class that ends first.

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:

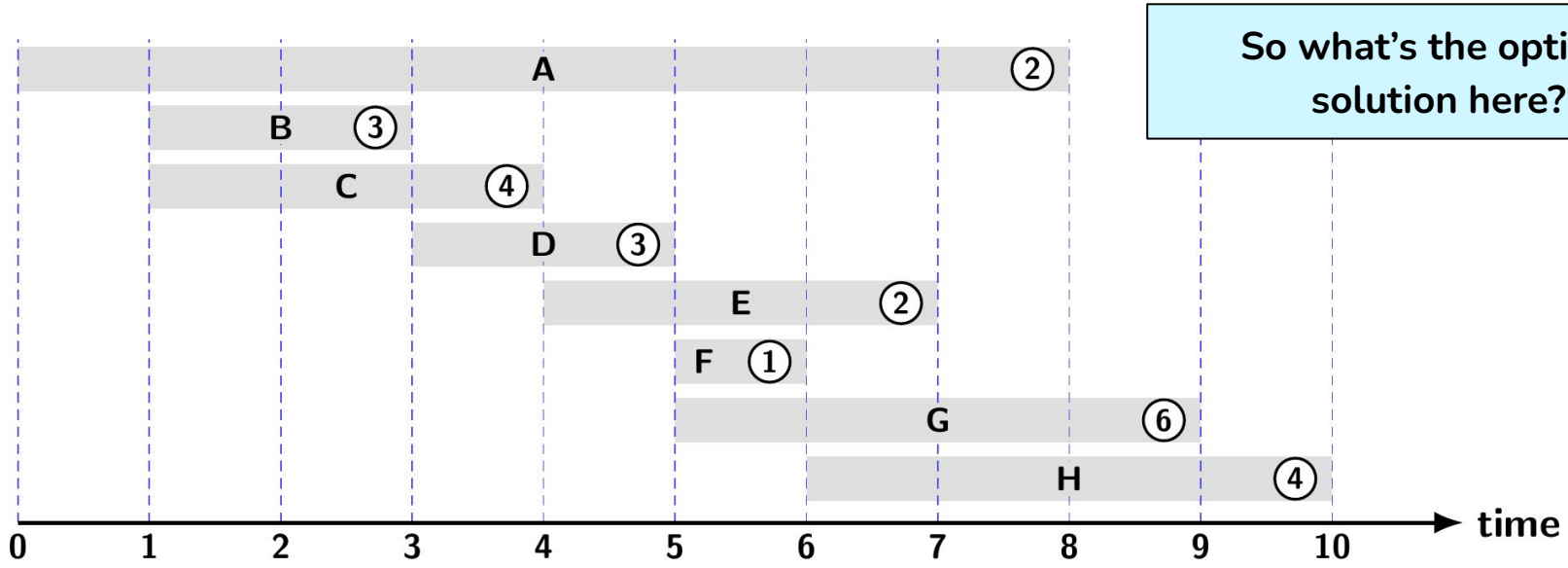
```
while not done:
    pick the best thing

return the result
```

Heuristic: Take the class that ends first.

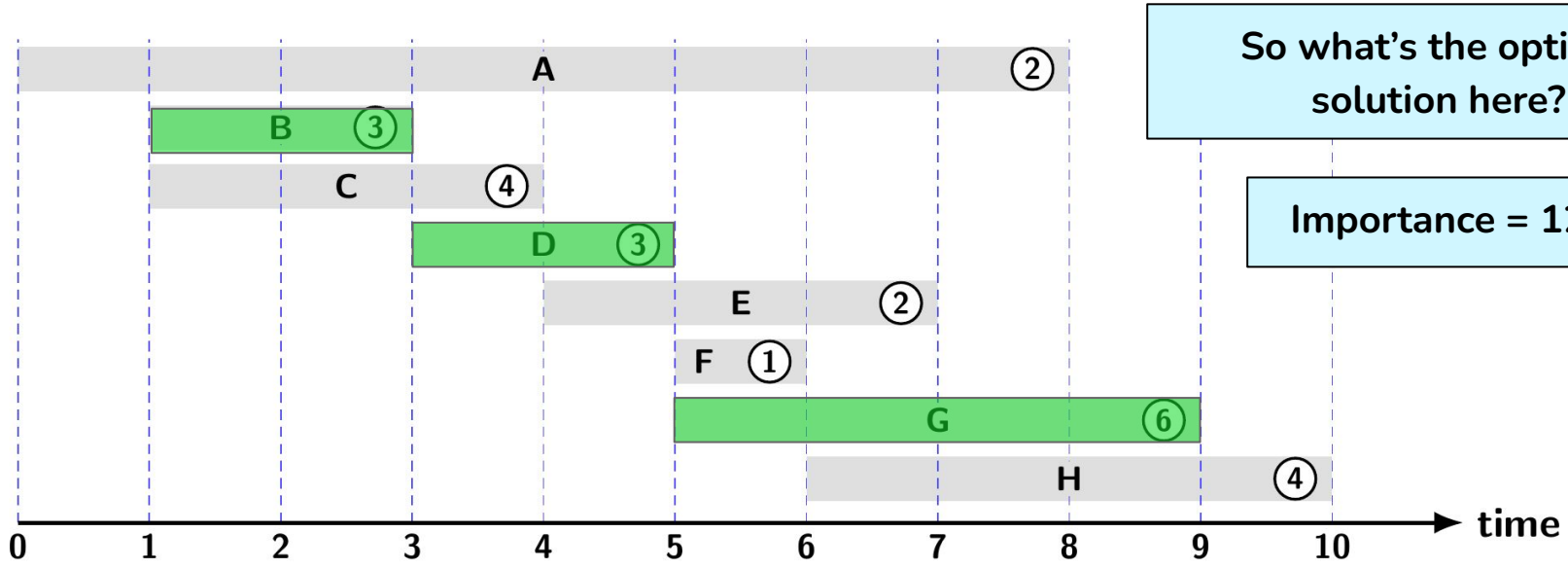
Weighted Course Scheduling

You're only willing to come onto campus one day per week. Each class has an importance level displayed. Find the feasible schedule with maximum importance.



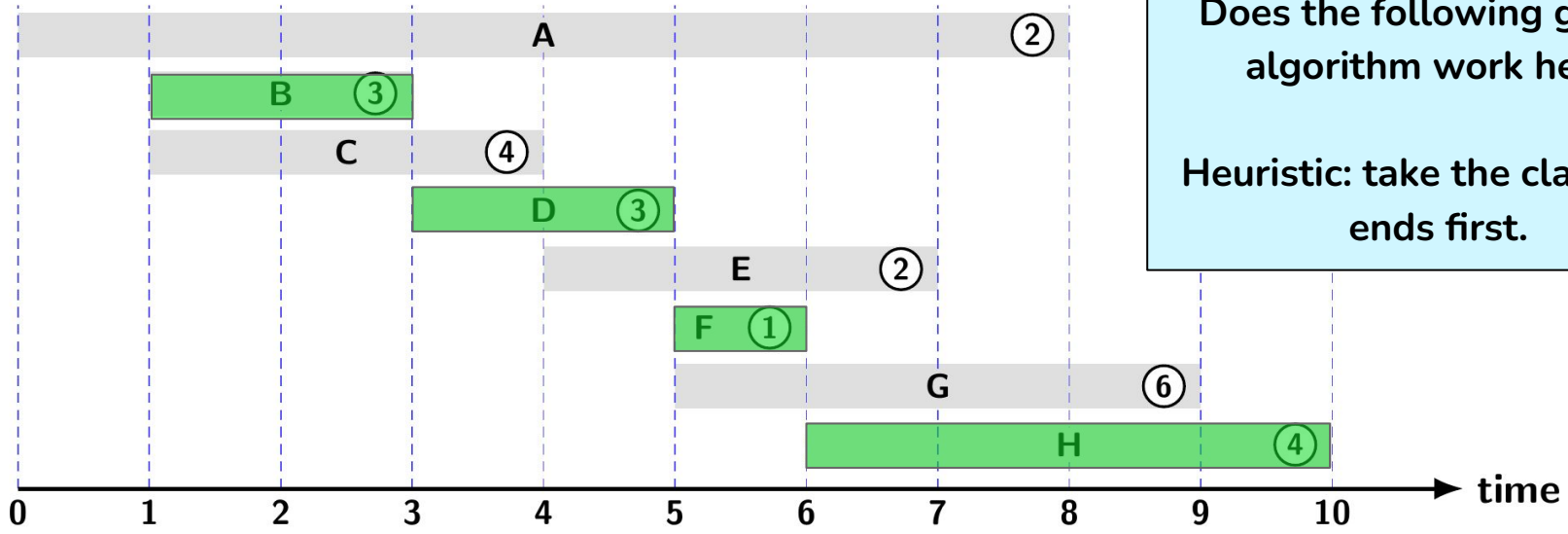
Weighted Course Scheduling

You're only willing to come onto campus one day per week. Each class has an importance level displayed. Find the feasible schedule with maximum importance.



Weighted Course Scheduling

You're only willing to come onto campus one day per week. Each class has an importance level displayed. Find the feasible schedule with maximum importance.

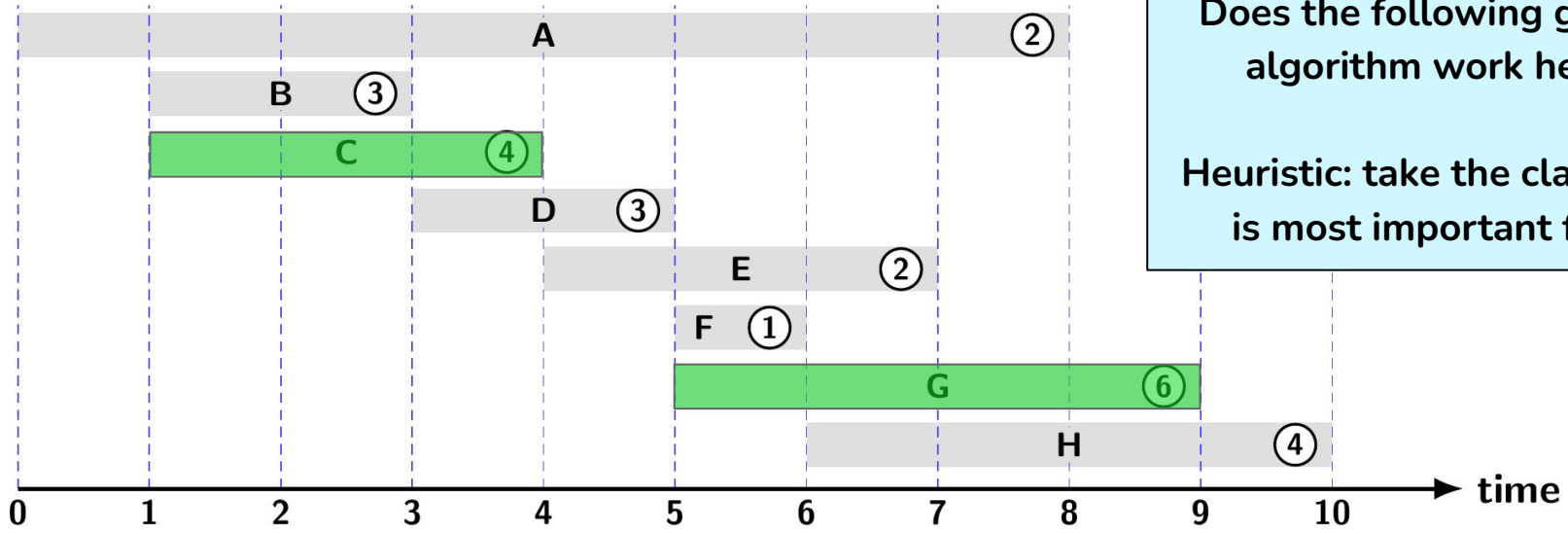


Does the following greedy algorithm work here?

Heuristic: take the class that ends first.

Weighted Course Scheduling

You're only willing to come onto campus one day per week. Each class has an importance level displayed. Find the feasible schedule with maximum importance.



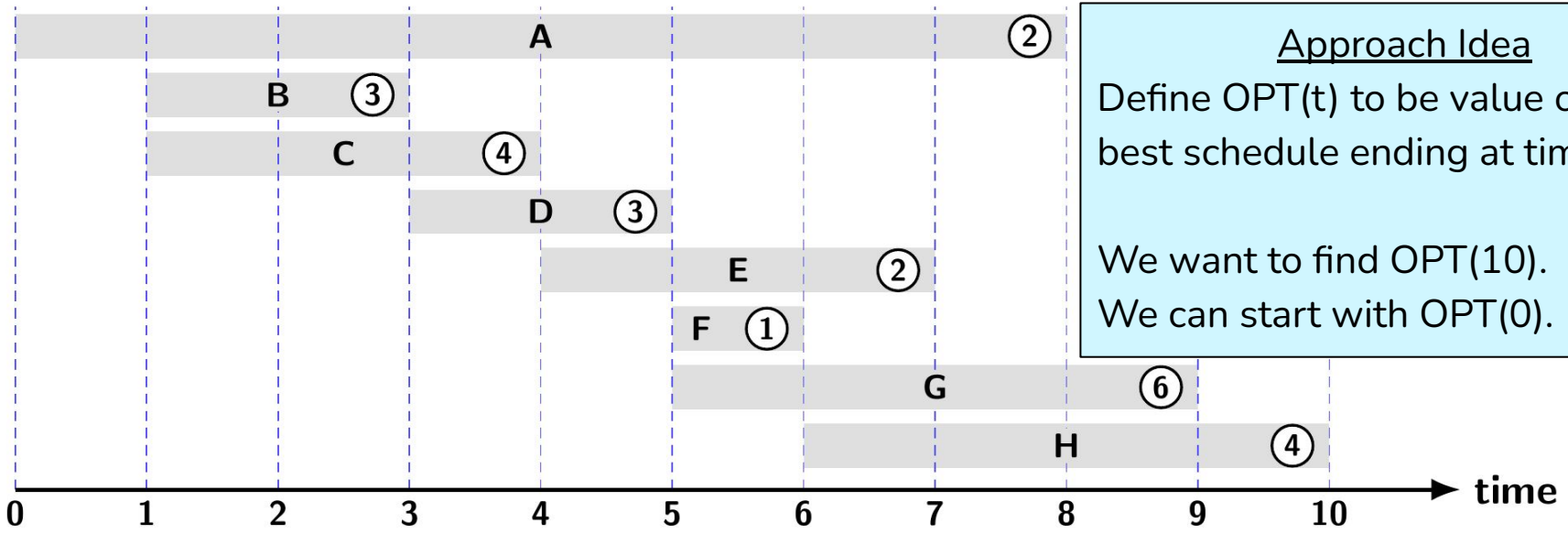
Does the following greedy algorithm work here?

Heuristic: take the class that is most important first.

A Dynamic Programming Approach to Weighted Interval Scheduling

Weighted Course Scheduling

t	0	1	2	3	4	5	6	7	8	9	10
OPT(t)											



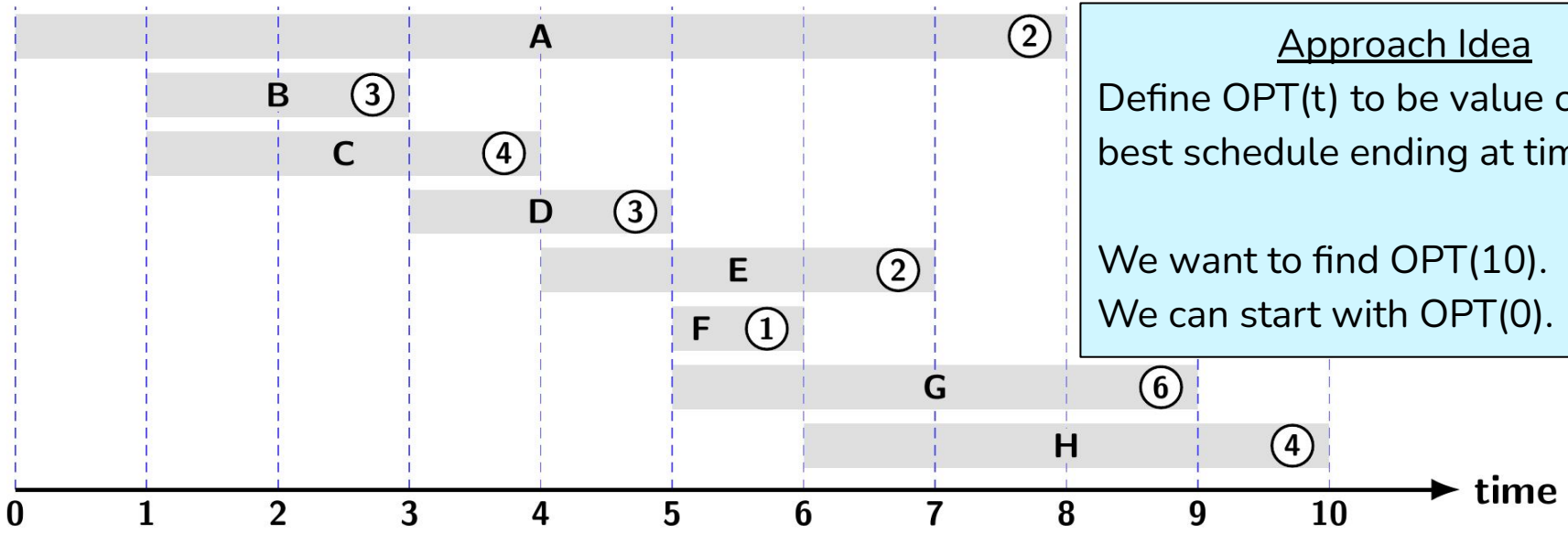
<u>Approach Idea</u>

Define OPT(t) to be value of the best schedule ending at time t.

We want to find OPT(10).
We can start with OPT(0).

Weighted Course Scheduling

t	0	1	2	3	4	5	6	7	8	9	10
OPT(t)	0	0	0								



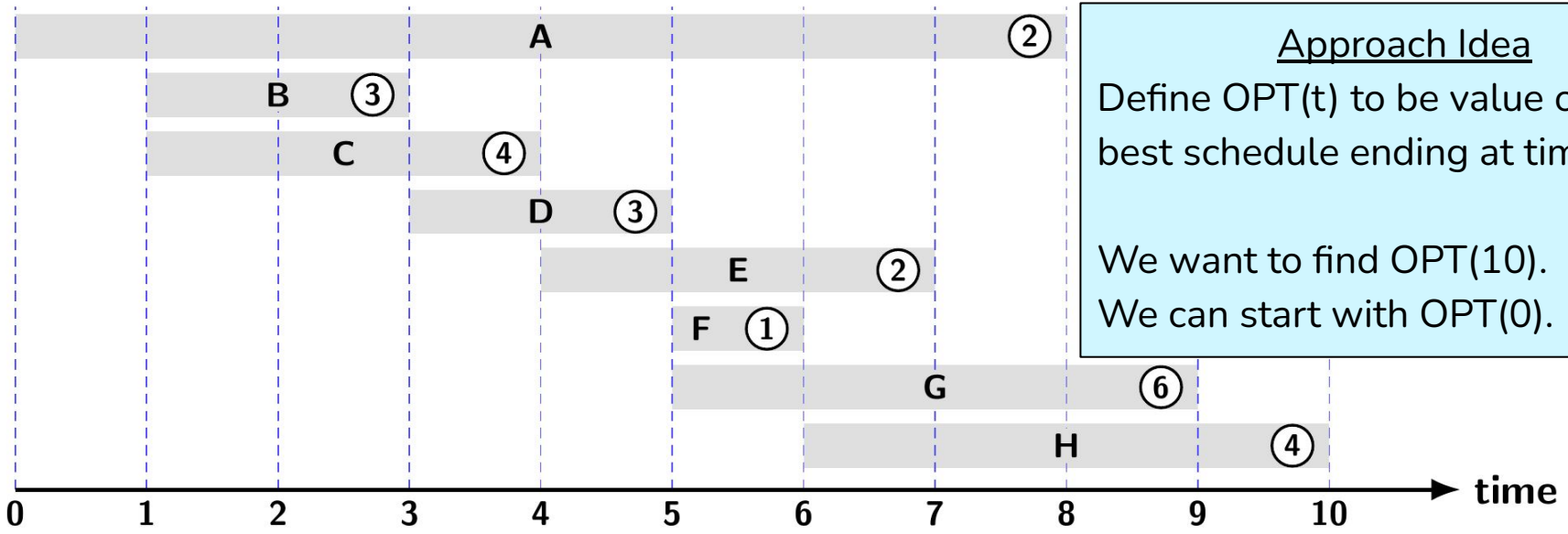
<u>Approach Idea</u>

Define OPT(t) to be value of the best schedule ending at time t.

We want to find OPT(10).
We can start with OPT(0).

Weighted Course Scheduling

t	0	1	2	3	4	5	6	7	8	9	10
OPT(t)	0	0	0	3							



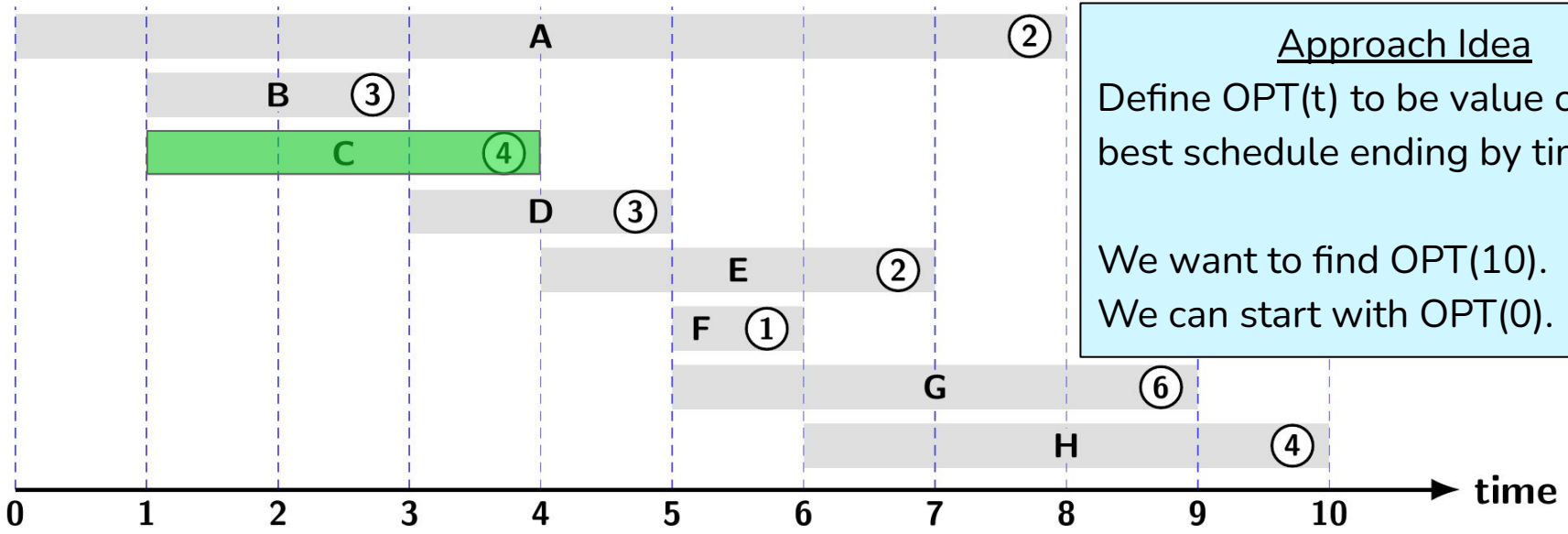
<u>Approach Idea</u>

Define OPT(t) to be value of the best schedule ending at time t.

We want to find OPT(10).
We can start with OPT(0).

Weighted Course Scheduling

t	0	1	2	3	4	5	6	7	8	9	10
OPT(t)	0	0	0	3	4						



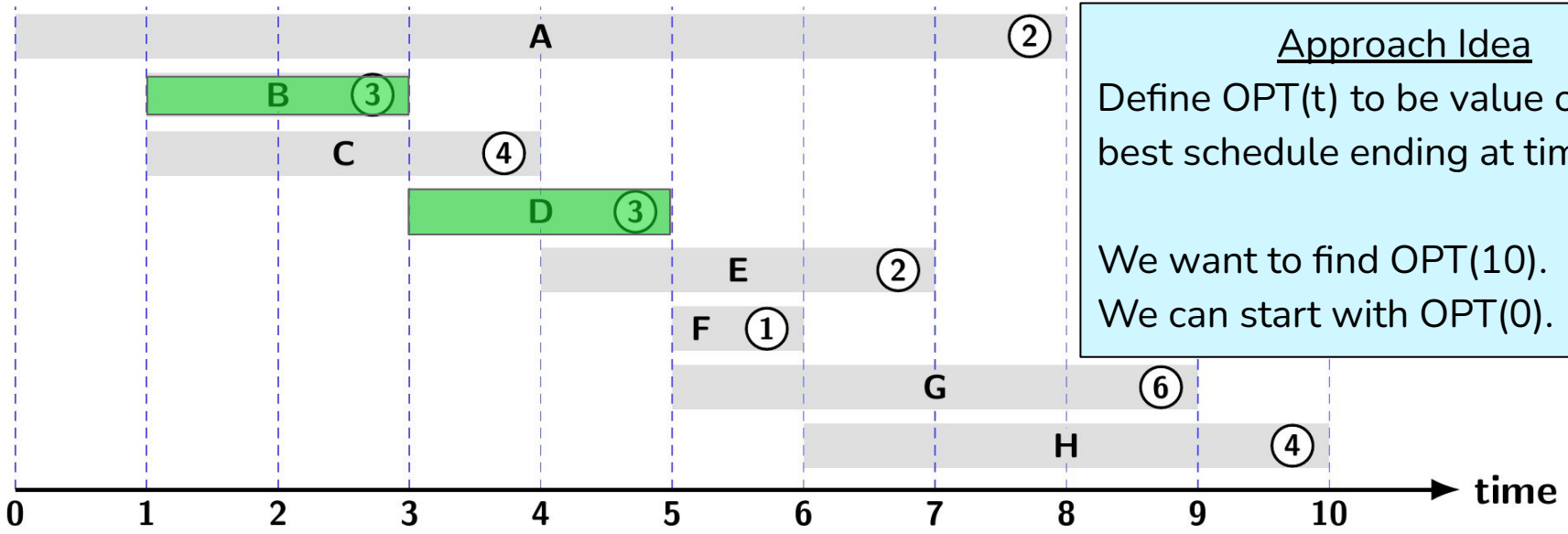
Approach Idea

Define OPT(t) to be value of the best schedule ending by time t.

We want to find OPT(10).
We can start with OPT(0).

Weighted Course Scheduling

t	0	1	2	3	4	5	6	7	8	9	10
OPT(t)	0	0	0	3	4	6					



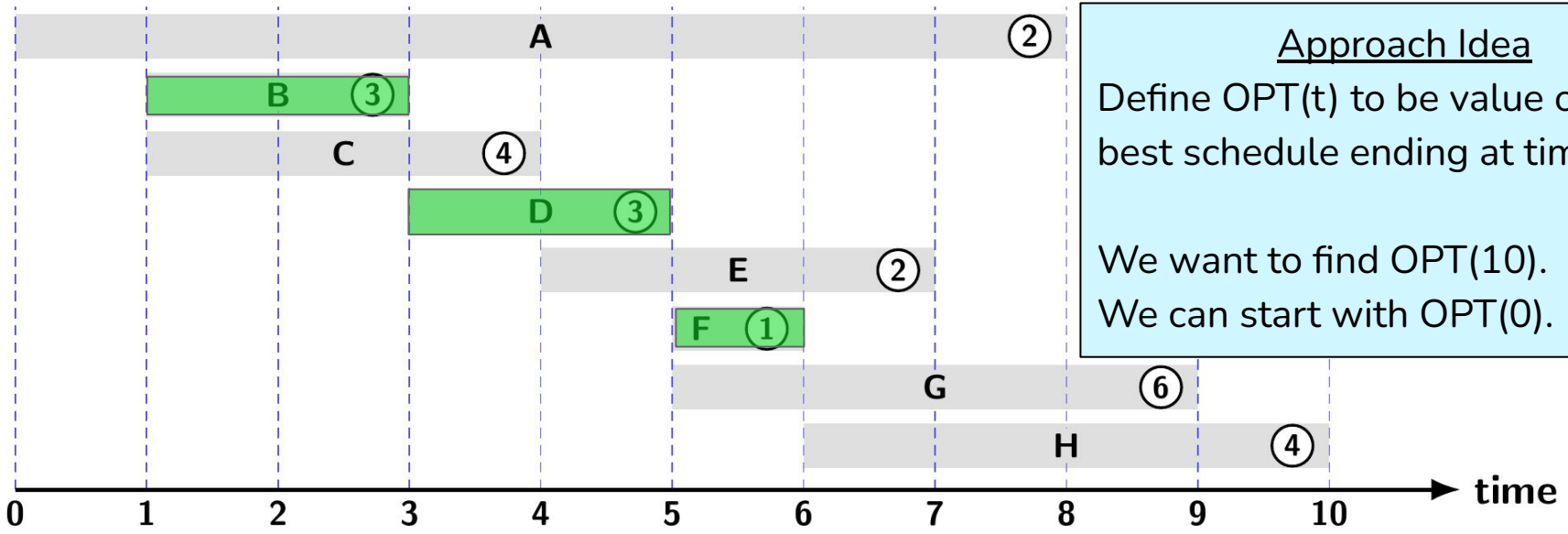
Approach Idea

Define OPT(t) to be value of the best schedule ending at time t.

We want to find OPT(10).
We can start with OPT(0).

Weighted Course Scheduling

t	0	1	2	3	4	5	6	7	8	9	10
OPT(t)	0	0	0	3	4	6	7				



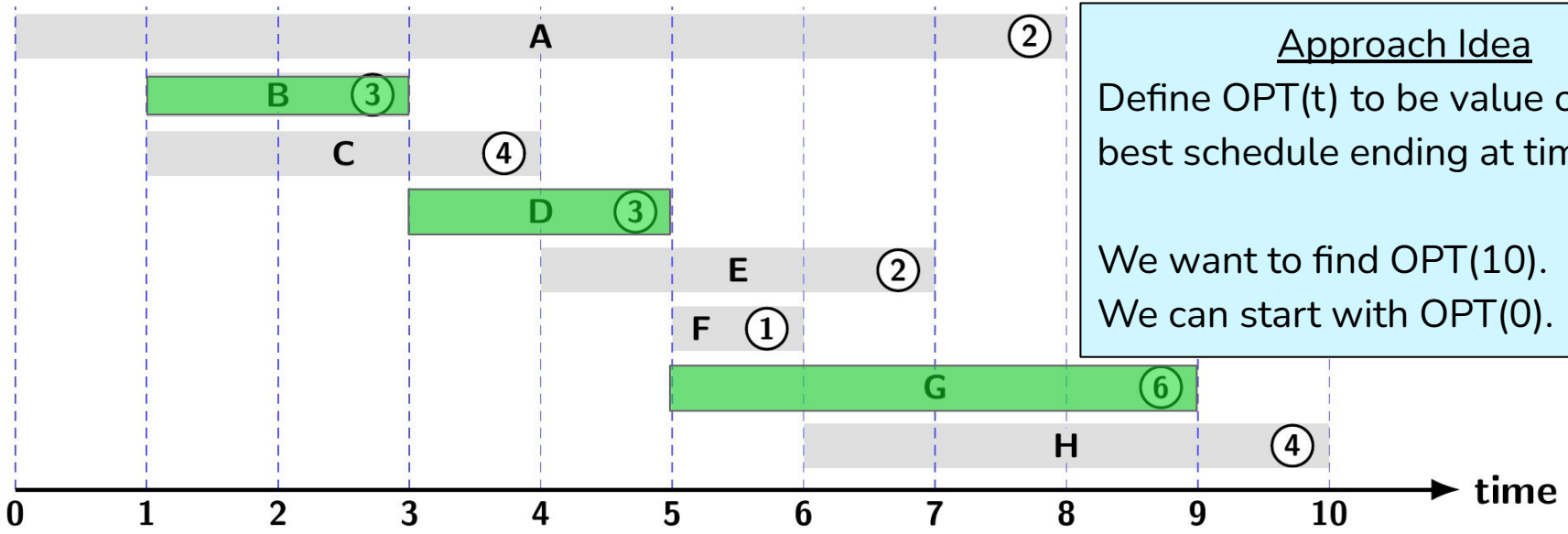
<u>Approach Idea</u>

Define OPT(t) to be value of the best schedule ending at time t.

We want to find OPT(10).
We can start with OPT(0).

Weighted Course Scheduling

t	0	1	2	3	4	5	6	7	8	9	10
OPT(t)	0	0	0	3	4	6	7	7	7	12	



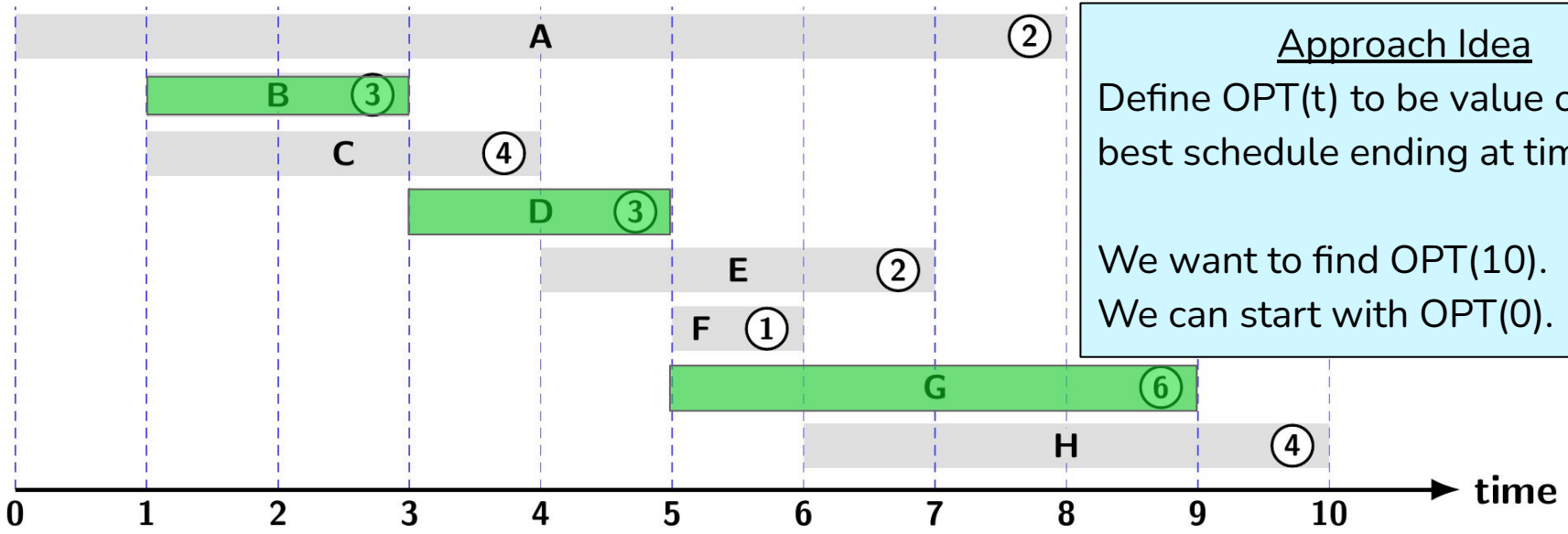
<u>Approach Idea</u>

Define OPT(t) to be value of the best schedule ending at time t.

We want to find OPT(10).
We can start with OPT(0).

Weighted Course Scheduling

t	0	1	2	3	4	5	6	7	8	9	10
OPT(t)	0	0	0	3	4	6	7	7	7	12	**12**



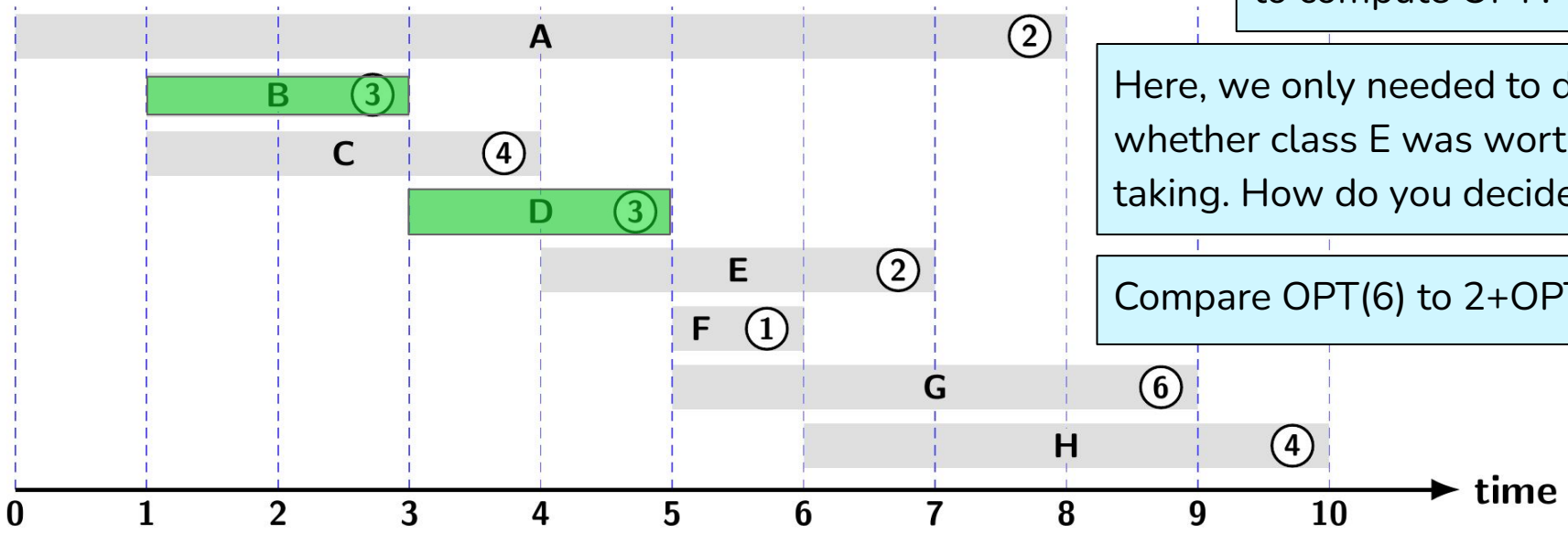
<u>Approach Idea</u>

Define OPT(t) to be value of the best schedule ending at time t.

We want to find OPT(10).
We can start with OPT(0).

Weighted Course Scheduling

t	0	1	2	3	4	5	6	7	8	9	10
OPT(t)	0	0	0	3	4	6	7	?			



Reflection Question:
What did you do at t=7 to compute OPT?

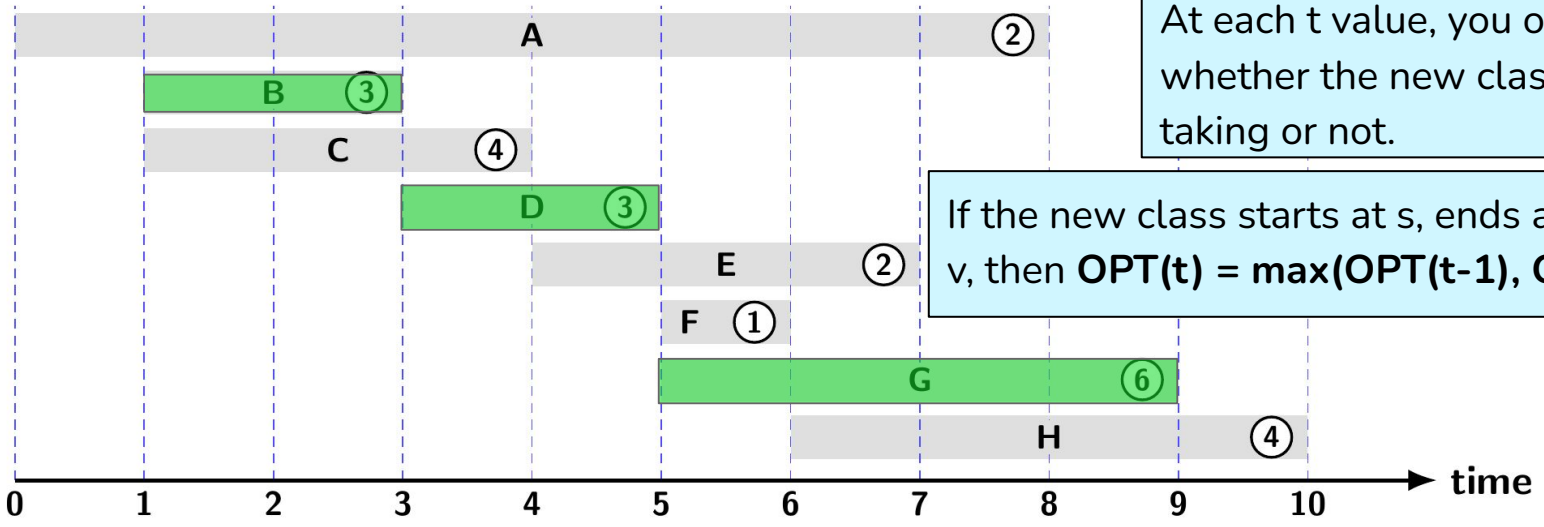
Here, we only needed to decide whether class E was worth taking. How do you decide?

Compare OPT(6) to 2+OPT(4).

Weighted Course Scheduling

t	0	1	2	3	4	5	6	7	8	9	10
OPT(t)	0	0	0	3	4	6	7	7	7	12	**12**

<u>Reflection Question:</u>
What did you do at each t value to compute OPT?



At each t value, you only had to decide whether the new class was worth taking or not.

If the new class starts at s, ends at t, and has value v, then **OPT(t) = max(OPT(t-1), OPT(s) + v)**.

Weighted Course Scheduling: Putting it Together

t	0	1	2	3	4	5	6	7	8	9	10
OPT(t)	0	0	0	3	4	6	7	7	7	12	12

Approach Idea

Define OPT(t) to be value of the best schedule ending at time t.

We want to find OPT(10).

We can start with OPT(0).

At each t value, you only have to decide whether the class ending at time t is worth taking or not.

If that class starts at s, ends at t, and has value v, then **OPT(t) = max(OPT(t-1), OPT(s) + v)**.

Best possible value if you don't take the new class.

Best possible value if you do take the new class.

Weighted Course Scheduling: Putting it Together

Approach Idea

Define OPT(t) to be value of the best schedule ending at time t.

We want to find OPT(10).

We can start with OPT(0).

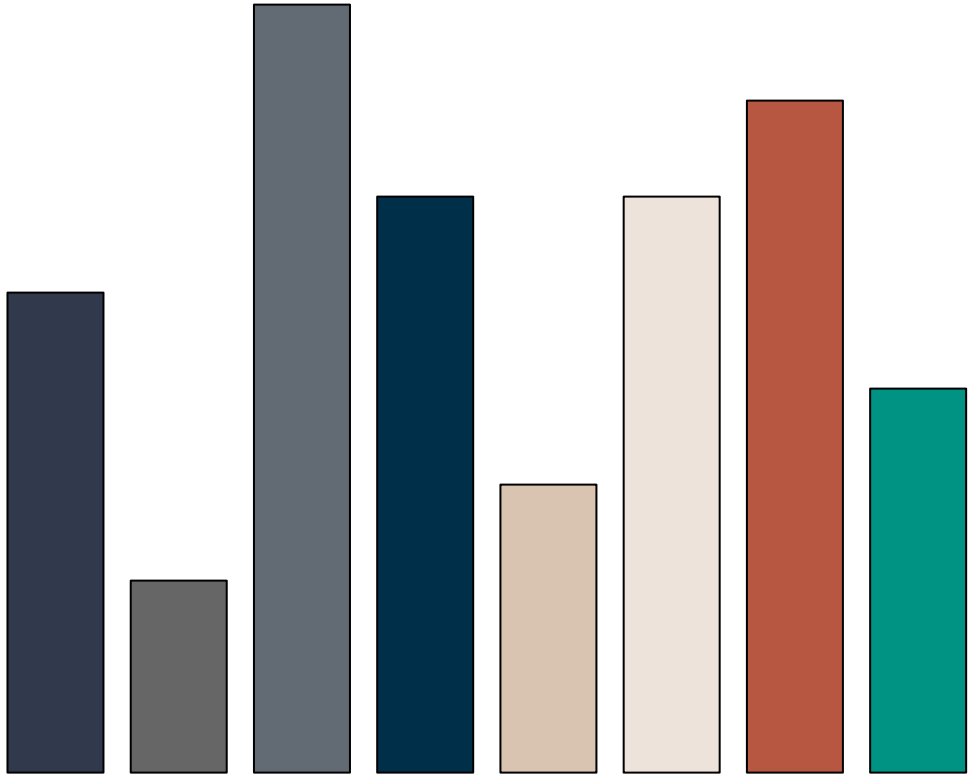
If that class starts at s, ends at t, and has value v, then **OPT(t) = max(OPT(t-1), OPT(s) + v)**.

Pseudocode:

```
def weightedScheduling(A):
    n = len(A)
    if n=0:
        return 0
    opt <- array of length n
    Opt[0] = 0
    sort A by end time
    for e in order from smallest to largest:
        best_if_take_class = v + opt[s]
        best_if_dont_take = opt[e-1]
        opt[e] = max(best_if_take, best_if_dont_take)
    return opt[n]
```

Another Dynamic Programming Algorithm

Reshelving Books

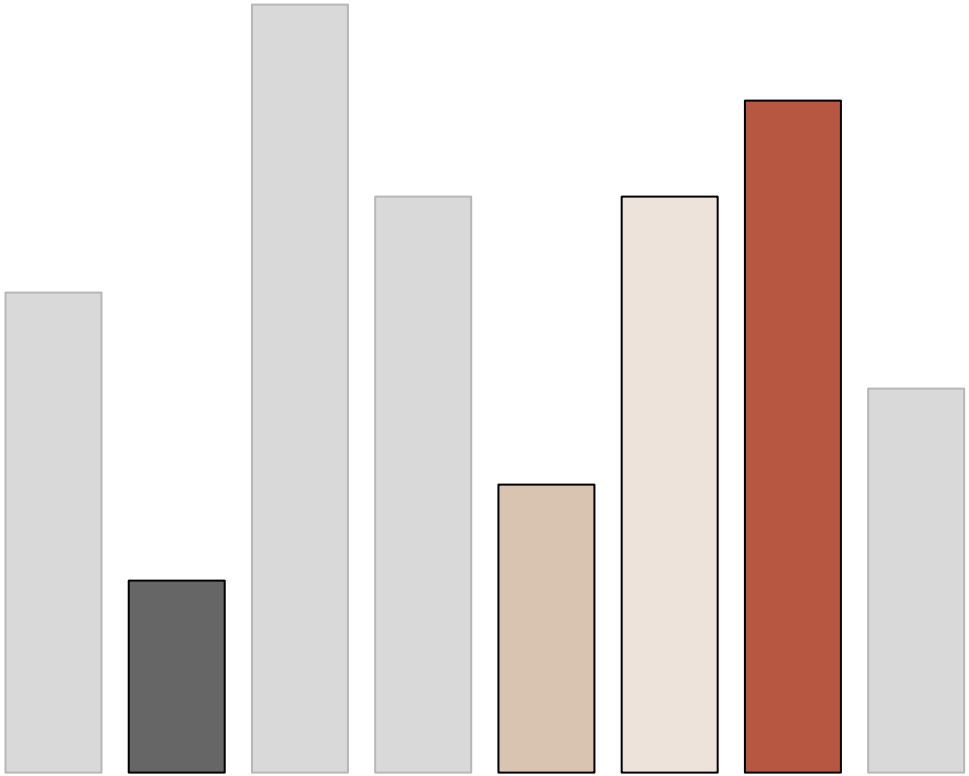


You have a lot of books, and you'd like to rearrange them so that they're in order by height.

The books are heavy, so you'd like to take as few books off the shelf as you can.

What's the largest number of books you can keep on the shelf?

Reshelving Books



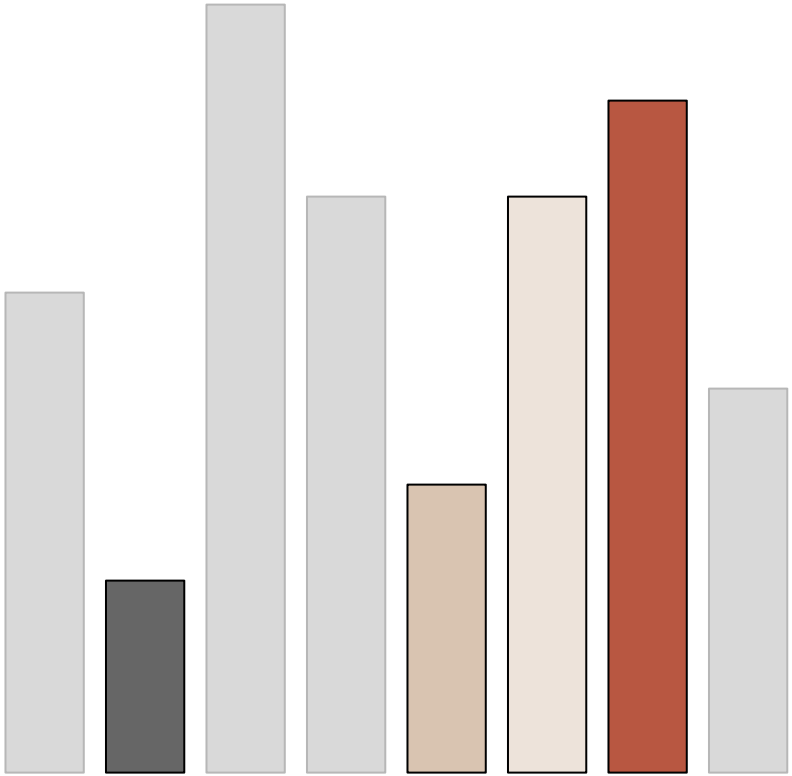
You have a lot of books, and you'd like to rearrange them so that they're in order by height.

The books are heavy, so you'd like to take as few books off the shelf as you can.

What's the largest number of books you can keep on the shelf?

It's the **longest increasing subsequence.**

Reshelving Books



```
def LIS(A):
    n = len(A)
    opt = array of length n
    opt[0] = 1
    for j in range(1, n):
        opt[j] = 1 + max(opt[i]: i < j, A[i] < A[j])
    return max(opt)
```

Take the max opt[i] value among all i where i < j and A[i] < A[j].

Comparing the Algorithms

```
def LIS(A):
    n = len(A)
    opt = array of length n
    opt[0] = 1
    for j in range(1, n):
        opt[j] = 1 + max(opt[i]: i < j, A[i] < A[j])
    return max(opt)
```

```
def WCS(A):
    n = len(A)
    opt = array of length n
    opt[0] = 0
    for class in classes, sorted by end time:
        opt[t] = max(opt[t-1], opt[s] + v)
    return opt[n]
```

Algorithm Similarities:

- Initialize an array
- Each entry in the array represents a *subproblem* – a smaller version of the problem
- Fill out the array one entry at a time, using previous array entries to help you

Why is it called Dynamic Programming?

We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word, research... His face would suffuse, he would turn red, and he would get violent if people used the term, research, in his presence. You can imagine how he felt, then, about the term, mathematical... [The word “dynamic”] has a very interesting property as an adjective, and that is it’s impossible to use the word, dynamic, in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It’s impossible. Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to.



Richard Bellman

Lecture Recap

Today we saw two examples of Dynamic Programming, a new style of algorithm.

This lecture should not be viewed as the fundamentals of Dynamic Programming. Instead, it should be viewed as an introduction to two examples of algorithms in this paradigm, for you to start to get a sense of what these algorithms are like.

Over the next couple of lectures, we'll talk about what dynamic programming algorithms really are, formally, as well as how to analyze their runtime and correctness.

We'll also see that they are much, much more versatile than the previous paradigms, and can even extend to solve some problems that are not efficiently solvable.