A lot of people tell me that this algorithm seems easy. It's too obvious, so it must be wrong. But that's the beauty of greedy algorithms: they're easy! A greedy algorithm is simple: at each step, pick the optimal move. In this case, each time you pick a class, you pick the class that ends the soonest. In technical terms: *at each step you pick the locally optimal solution*, and in the end you're left with the globally optimal solution. Believe it or not, this simple algorithm finds the optimal solution to this scheduling problem!

Obviously, greedy algorithms don't always work. But they're simple to write! Let's look at another example.

The knapsack problem

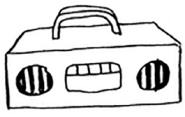
Suppose you're a greedy thief. You're in a store with a knapsack, and there are all these items you can steal. But you can only take what you can fit in your knapsack. The knapsack can hold 35 pounds.

You're trying to maximize the value of the items you put in your knapsack. What algorithm do you use?

Again, the greedy strategy is pretty simple:

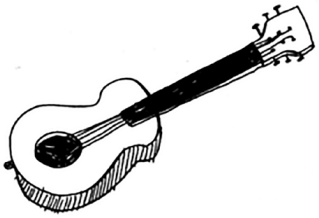
1. Pick the most expensive thing that will fit in your knapsack.
2. Pick the next most expensive thing that will fit in your knapsack. And so on.

Except this time, it doesn't work! For example, suppose there are three items you can steal.



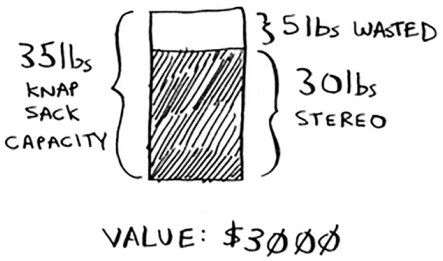
STEREO
$3000
30lbs

LAPTOP
$2000
20lbs

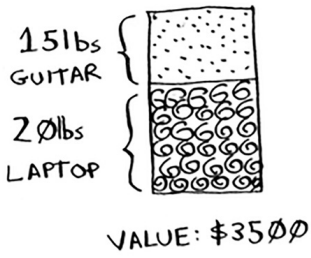


GUITAR
$1500
15lbs

Your knapsack can hold 35 pounds of items. The stereo system is the most expensive, so you steal that. Now you don't have space for anything else.



You got $3,000 worth of goods. But wait! If you'd picked the laptop and the guitar instead, you could have had $3,500 worth of loot!



Clearly, the greedy strategy doesn't give you the optimal solution here. But it gets you pretty close. In the next chapter, I'll explain how to calculate the correct solution. But if you're a thief in a shopping center, you don't care about perfect. "Pretty good" is good enough.

Here's the takeaway from this second example: sometimes, perfect is the enemy of good. Sometimes all you need is an algorithm that solves the problem pretty well. And that's where greedy algorithms shine, because they're simple to write and usually get pretty close.

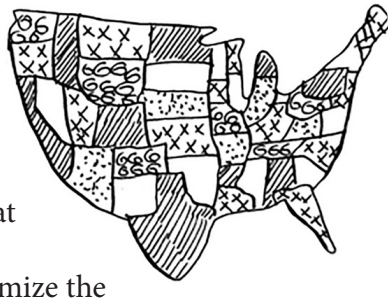
EXERCISES

8.1 You work for a furniture company, and you have to ship furniture all over the country. You need to pack your truck with boxes. All the boxes are of different sizes, and you're trying to maximize the space you use in each truck. How would you pick boxes to maximize space? Come up with a greedy strategy. Will that give you the optimal solution?

8.2 You're going to Europe, and you have seven days to see everything you can. You assign a point value to each item (how much you want

to see it) and estimate how long it takes. How can you maximize the point total (seeing all the things you really want to see) during your stay? Come up with a greedy strategy. Will that give you the optimal solution?

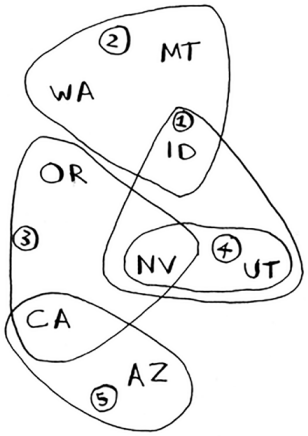
Let's look at one last example. This is an example where greedy algorithms are absolutely necessary.



The set-covering problem

Suppose you're starting a radio show. You want to reach listeners in all 50 states. You have to decide what stations to play on to reach all those listeners. It costs money to be on each station, so you're trying to minimize the number of stations you play on. You have a list of stations.

Each station covers a region, and there's overlap.



How do you figure out the smallest set of stations you can play on to cover all 50 states? Sounds easy, doesn't it? Turns out it's extremely hard. Here's how to do it:

1. List every possible subset of stations. This is called the *power set*. There are 2^*n* possible subsets.