Dynamic Programming: Weighted Interval Scheduling

Warm Up: we are given a set of n intervals, numbered $1 \dots n$, each of which has a start time s_i and a finish time f_i. For each interval, we want to compute a value $p(i)$, which is the interval j with the *latest* finish time f_j such that $f_j \leq s_i$; that is, the last-ending interval that finishes before interval i starts. If no intervals end before interval i begins, then $f(i) = 0$.

Give an $O(n \log n)$ time algorithm that computes $p(i)$ for all intervals. You may assume that the intervals are already sorted by finish time.

The Big Problem: Fed up after CSCI 570, your friend has decided to change majors to one that grades based only on attendance. The only question is which classes your friend should take in Fall semester. The classes all meet once a day, at different times and lengths, and are worth different amounts of credits. Your friend's goal is to maximize the amount of credits earned in Fall semester without having to skip any classes (as this may interfere with passing those classes).

Problem Statement: More formally, we are given a set of n intervals, each of which has a start time s_i, a finish time f_i, and a value v_i. Our goal is to select a subset of the intervals such that no two selected intervals overlap and the total value of those taken is maximized.

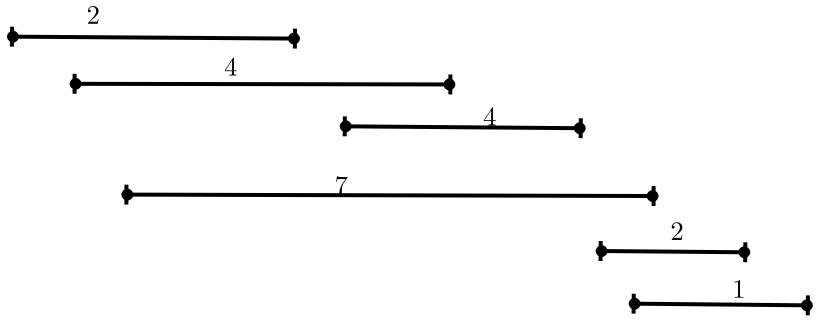
An example input is provided on the back of this paper. Please be aware that sample input will not always be provided in CSCI 570; one of the educational objectives is for you to be able to solve a problem in the abstract.

Let's solve this *recursively* (yay!). We will write a function `OPT(`i`)` that returns the optimal number of credits obtainable among intervals (classes) $1 \dots i$. We can then call `OPT(n)` to figure out the optimal number of credits obtainable among all intervals.

The key observation here is that your friend will either take class i or your friend won't take class i.

```
OPT(i)
// Base Case:



// If my friend doesn't take class $i$:
value_if_not_taken =



// If my friend takes class $i$:
value_if_taken =



//return something:
```

Example Input:



To provide a dynamic programming algorithm as a solution, we need:

- A general recursive solution, along with an explanation of why it is correct:

- Base Case(s):

- The order in which to memoize the recursive solutions

- The running time of the iterative version of the algorithm

i	$p(i)$	v_i	OPT(p(i))	OPT(p(i)) + v_i	OPT($i-1$)	OPT(i)
0		N/A	N/A	N/A	N/A	0
1		2				
2		4				
3		4				
4		7				
5		2				
6		1				

Dynamic Programming is not about filling in tables.
Dynamic Programming is about smart recursion.

Dynamic Programming Exercise: Free Food

Exercise: Graduate students get a lot of free food at various events. Suppose you have a schedule of the next n days marked with those days when you get a free dinner, and those days on which you must acquire dinner on your own. On any given day you can buy dinner at the cafeteria for \$6. Alternatively, you can purchase one week's groceries for \$20, which will provide dinner for each day that week. However, because you don't have a fridge, the groceries will go bad after seven days (including the day of purchase) and any leftovers must be discarded. Due to your very busy schedule, these are your only two options for dinner each night.

Write a dynamic programming algorithm to determine, given the schedule of free meals, the minimum amount of money you must spend to make sure you have dinner each night and to print the dates on which you should purchase groceries. Prove that your algorithm is correct and has running time polynomial in n, the number of days on the schedule.

Dynamic Programming: Subset Sum

Problem Statement: Given a set S of n positive integers, as well as a positive integer T, determine if there is a subset of S that sums to exactly T.

- General recursive solution:

```
// base case:

is_sum_not_using_nth =

is_sum_with_using_nth =

// return something:
```

- In what order do you memoize the recursive solutions?

- What is the running time of your algorithm?

- How would you modify this to output the subset, if one exists?

Example 1: $S = \{2, 3, 4\}$, $T = 6$.

	0	1	2	3	4	5	6
{}							
{2}							
{2, 3}							
{2, 3, 4}							

Example 2: $S = \{2, 3, 5\}$, $T = 6$.

	0	1	2	3	4	5	6
{}							
{2}							
{2, 3}							
{2, 3, 5}							