Algorithms: Introduction to Dynamic Programming

Learning objective: Students will apply memoization techniques to speed up recursion with overlapping subproblems.

Process objective: Students will communicate high-level insights about recursive functions orally and in writing.

Begin by **skimming** the following model. You do not need to understand all the code right away; the activity will guide you through the process of understanding it.

Model 1: Fibonaccis

Here are three functions to compute Fibonacci numbers, implemented in Python. You may assume that they are all correct.

```
def fib1(n):
    if n <= 1:
        return n
    else:
        return fib1(n-1) + fib1(n-2)


def fib2(n):
    fibs = [0] * (n+1)   # Create initial array of all 0s
    fibs[1] = 1

    for i in range(2, n+1):
        fibs[i] = fibs[i-1] + fibs[i-2]

    return fibs[n]

# Global table of Fibonacci numbers; assume no one will
# ever call fib3(n) with n >= 1000
fib_table = [-1] * 1000

def fib3(n):
    if n <= 1:
        fib_table[n] = n
    elif fib_table[n] == -1:
        fib_table[n] = fib3(n-1) + fib3(n-2)

    return fib_table[n]
```

Critical Thinking Questions: `fib1` *(15 minutes)*

1 Recall that the Fibonacci numbers are defined by the recurrence

$$F_0 = 0$$
$$F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2}$$

Which of the three implementations corresponds most directly to this definition?

2 Draw the call tree for `fib1(5)`.

3 How many times does `fib1(2)` occur in the call tree? What about `fib1(1)`? `fib1(0)`?

4 It turns out that `fib1` is extremely slow.[1] Intuitively, what do you think makes it so slow?

[1] In fact, it takes $\Theta(\varphi^n)$ time.

- Send your **reporter** to another group to check your understanding of what makes `fib1` slow.
- While the reporter is gone, the **reflector** should take a minute to think of one **strength** of your group's work so far today and one **improvement** to suggest.
- When the reporter gets back, they should share what they learned.
- The reflector should then share their insights with the group.

Critical Thinking Questions: `fib2` and `fib3` (25 minutes)

5 Trace the execution of `fib2(5)`. As a group, come up with one or two complete English sentences to explain how it works.

6 Which does more work, `fib2(5)` or `fib1(5)`? Why?

7 In terms of Θ, how long does `fib2(n)` take?[2]

8 Suppose we switch the direction of the `for` loop in `fib2`, so `i` loops from `n` down to `2`. Would it still work? Why or why not?

9 Trace the execution of `fib3(5)` and explain how it works in one or two complete English sentences.

10 In terms of Θ, how long does `fib3(n)` take?

11 Fill in this statement: `fib3` is just like `fib1` except that

_____.

[2] For the purposes of this activity, you should assume that each addition takes constant time. However, as you may know from a previous activity, it is more accurate to say that addition takes linear time in the number of bits, which actually makes a difference here since Fibonacci numbers can get quite large. You will analyze the situation more precisely on a homework assignment.

12 Fill in this statement: `fib2` is just like `fib3` except that

_____.

13 Consider the following recursive definition of $Q(n)$ for $n \geq 0$:

Note that there are *three* base cases.

$$Q(0) = 0$$
$$Q(1) = Q(2) = 1$$
$$Q(n) = Q(n-1) + 2 \cdot [Q(n-3)]^2$$

(a) Calculate $Q(3)$, $Q(4)$, and $Q(5)$.

(b) Using pseudocode, or any language your group agrees to use, write an algorithm to calculate $Q(n)$ efficiently.

Facilitation plan

CTQs 1

15 minutes, including sending reporters and having reflectors share. Use best judgment to see if a bigger class discussion is needed, but hopefully just send them on to the next section.

CTQs 2

25 minutes. This section is more involved. Have reporters share out at the end to check understanding.

Wrap up

10 minutes for questions and wrap-up. Explain that they have learned today the essence of "dynamic programming" and "memoization". DP can sometimes be presented in confusing ways, but it's really just about using memoization to speed up recursion with overlapping subproblems. Fibonacci numbers are too simple to really need DP, but make a nice simple case study. In the next activities we will move on to more realistic applications.

Author notes

F'19: This one seems short but working through it completely, then having reporters compare with other teams took until around the 40-minute mark or so. And it was really good, they all got it. Took the last 10 minutes to sum up what we had done and put everything in a big-picture context.