Algorithms: Introduction to Dynamic Programming

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

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


fibtable = [0,1]  # global table of Fibonacci numbers

def fib3(n):

    while len(fibtable) < n+1:
        fibtable.append(-1)

    if fibtable[n] == -1:
        fibtable[n] = fib3(n-1) + fib3(n-2)

    return fibtable[n]
```

(This page is intentionally blank so that the model can be on it's own physical sheet of paper.)

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)`. Start by placing `fib1(5)` as the root of the tree, and then draw its children `fib1(4)` and `fib1(3)`. In general, a node in the call tree represents a single function call (with its parameters); a node's parent is the function that called it, and its children are any function(s) that it calls.

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] What do you think makes it so slow?

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

Once you reach this point, elect one member of your group to venture out to another team. (And if the other team is a two-person group, they should also be sending one of their members to meet with the remaining members of your group.) Discuss your answer to the previous question.

- Do you both cite the same reasons for `fib1`'s slow performance?
- Do you agree with all of each others' reasons?

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

5 Trace the execution of `fib2(5)` and explain how it works using one or two complete sentences.

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

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

[2] For the purposes of this activity, you should assume that each addition takes constant time.

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. Draw the call tree and explain how it works using one or two complete sentences (in whichever order you find easiest).

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

Once again, elect one member of your group to venture out to another team. Discuss your answer to the previous question, and compare your call trees for `fib3(5)`.

- Do you both agree on `fib3(n)`'s big-Theta performance?
- Do your call trees match?

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

____.

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

____.

13 Why don't we do something like `fib2` or `fib3` in the case of merge sort?

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

Note that there are *three* base cases.

$$
\begin{aligned}
Q(0) &= 0 \\
Q(1) &= Q(2) = 1 \\
Q(n) &= \max \begin{cases} Q(n-3)^2 \\ Q(n-1) + Q(n-2) \end{cases}
\end{aligned}
$$

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