**Problem 1.** Recall the definition for the sequence of *Fibonacci numbers*:
$F(1) = 1, F(2) = 1, F(n) = F(n-1) + F(n-2)\ \forall n > 2$
Draw the tree of recursive calls that are performed to compute `Fib(6)` using the following code and mark all work that is done repetitively:

```
Fib(n)
    if n < 3 return 1
    else return Fib(n-1) + Fib(n-2)
```

Give an asymptotic upper bound (as tight as possible) for the runtime of this algorithm as a function of $n$.

**Problem 2.** Give the pseudocode for a dynamic programming algorithm to compute $F(n)$ given $n$. State a tight asymptotic bound for the runtime of your algorithm.

**Problem 3.** Read section 15.2 in the book and answer the following questions:

1. What is the matrix chain multiplication problem? Describe it in your own words.
2. What are the different ways of parenthesizing the matrix product $A_1A_2A_3$?
3. How many scalar multiplications are needed to compute the product, for each of the different paranthesizations, assuming the matrix dimensions are $7 \times 50, 50 \times 10$, and $10 \times 100$, respectively?
4. How long would it take to try all possible parenthesizations? State an asymptotic lower bound (as tight as possible), assuming that two numbers of arbitrary length can be multiplied in constant time.
5. What is the runtime of the dynamic programming algorithm?
6. Why does the d.p. algorithm need a two-dimensional array to store the subproblem solutions?
7. How does the d.p. algorithm ensure that solutions to subproblems are available when they are needed?

**Problem 4.** Solve the rod-cutting problem for the input $n = 8, p = [2, 3, 7, 8, 10, 13, 15, 17]$ using dynamic programming. Give the array of optimal revenues $r$ and an optimal cut.

**Problem 5.** Define the problem of giving change as follows: Given a sorted array $A$ of $n$ distinct, positive integers with $A[1] = 1$ and a positive integer $v$, return an array $B$ of length $n$ such that $\sum_{i=1}^{n} A[i] * B[i] = v$ and $\sum_{i=1}^{n} B[n]$ is minimized. $A$ represents the different coins, $v$ is the amount of money needing to be returned, $B$ tells us how many of each kind of coin to use, and the restrictions on $B$ ensure that the correct amount is returned and the number of coins used is minimized. For U.S. currency, for instance, $A = [1, 5, 10, 25, 50, 100]$. Then for $v = 289$, $B = [4, 0, 1, 1, 1, 2]$. Note that $A[1] = 1$ is included to ensure that there is always a solution.

Describe a greedy algorithm to solve this problem. Analyze its runtime. The greedy approach does not guarantee an optimal solution for all $A$. Give an example where the greedy algorithm would return a solution that uses at least 5 more coins than necessary.

**Problem 6.** Describe a dynamic programming algorithm to solve the problem of giving change optimally for all valid inputs. Analyze its runtime.

**Problem 7.** Analyze the runtime of the dynamic programming algorithm for the activity selection problem (see class on 3/26 or CLR(S) 16.1). Given the array $c$ filled in by the algorithm and a two-dimensional array $d$ such that $d[i][j]$ gives the value $k$ that maximizes $c[i][k] + c[k][j] + 1$, give pseudocode to print the indices of the selected activities. Lastly, how would the algorithm have to be adapted if activities were assigned a value and the task was to select not the largest subset but the subset with the greatest value?