Fibonacci numbers

$$F(0) = 0$$

$$F(1) = 1$$

$$F(n) = F(n-1) + F(n-2)$$

A recursive algorithm

```
function  fib-recur(n)
   if n=0: return 0
   if n=1: return 1
   return  fib-recur(n-1)+fib-recur(n-2)
```

Run time?

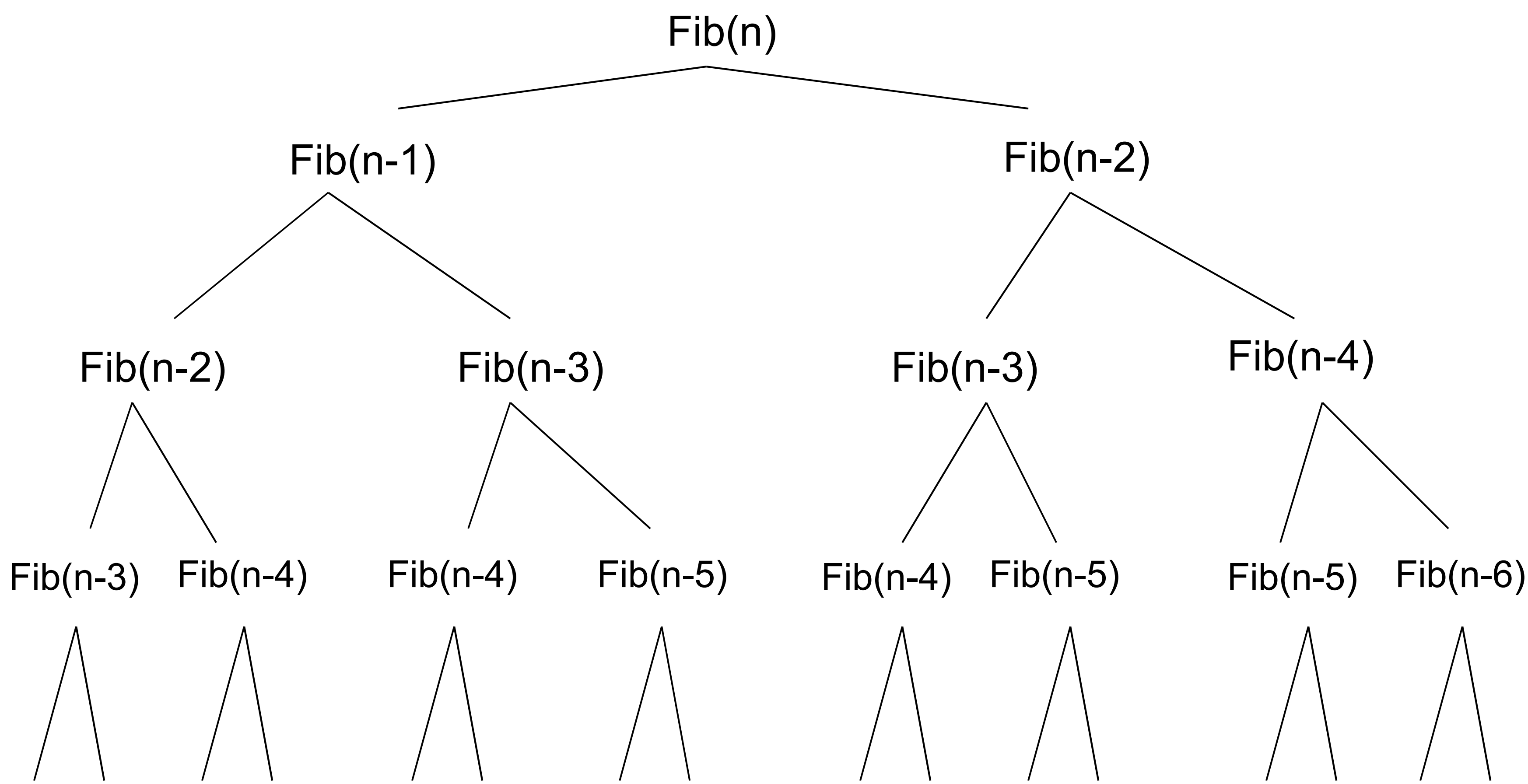
Recurrence relation

$$T(n) = T(n-1) + T(n-2) + c$$

$$O(2^n), \Omega\left(2^{\frac{n}{2}}\right)$$

Why so slow?

- Repeated computation



Avoid repeat by memoization

Memoization: a speed up technique that stores the results of expensive function calls and returns the cached result

```
fib-mem(n)
        if n<2 F(n)=n
        else if F(n) is undefined
                F(n)=fib-mem(n-1)+fib-mem(n-2)
        return F(n)
```

Runtime?

Bottom-up: iterative Version

```
function  fib-iter(n)
   f[0]=0
   f[1]=1
   for i=2 to n
          f[i]=f[i-1]+f[i-2]
   return f[n]
```

Runtime?

Dynamic Programming

- A powerful algorithm design technique
- Very common interview questions
- Many applications. For example
  - Unix diff for comparing two files
  - Bellman-Ford for shortest path routing in networks
  - CKY algorithm for natural language parsing
  - ……
- Coined by Richard Bellman before the age of computer programming
  - Dynamic Programming = planning over time

When to use Dynamic Programming?

- When your problem has the following properties:
  - Optimal substructures: solution to a problem can be defined using solutions of smaller sub-problems (similar to Divide and Conquer)
  - Subproblems are overlapping (a key difference from divide and conquer), i.e., we see repeated subproblems

Designing a DP solution

1. Figure out how get the solution to a problem based on solutions to smaller sub-problems
   – Pretend you have a solver but can only be used to solve smaller problems
   – e.g., $F(n)= F(n-1)+F(n-2)$

2. Start from the smallest and build solutions to larger problems – bottom up, iterative
   – Sometimes recursion is used with memoization

3. Sometime we have to re-trace the chain of solutions to construct the final solution

Longest Increasing Subsequences

Problem:

Given a sequence of numbers $a_1, a_2, \dots, a_n$, find the longest increasing subsequence(LIS)

$$5 \quad 2 \quad 8 \quad 6 \quad 3 \quad 6 \quad 9 \quad 7$$

$$\left.\begin{matrix} 5\ 8\ 9 \\ 2\ 6\ 9 \\ 2\ 3\ 6\ 7 \end{matrix}\right\}$$

All three are increasing subsequences
Goal: find the longest one

- Don't need to be contiguous
- May not be unique

5 2 8 6 3 6 9 7

- Q1: what is the longest increasing subsequence if we must end the sequence with 7?
- *A*1: we don't know, but the number before 7 must not be 8, or 9

- Q2: what could the previous number be?
- *A*2: any number < 7

- Q3: if you have a solver that tells you the longest increasing subsequence ending at all previous positions, can you figure out the answer to Q1?

Building our solution

Let L[i] be the length of a longest increasing subsequence ending at position i

$$L[1] = 1$$

$$L[i] = \max_{j:1\leq j<i, a_j<a_i} L[j] + 1 \text{ for } i = 2, \ldots, n$$

Overall solution: $\max_i$ L[i]

Example

5 2 8 6 3 6 9 7

Iterative algorithm

```
LIS (A,n)
L[1]=1
for i=2 to n
        L[i]=1
        for j=1 to i-1
                if a_j < a_i and L[i] < L[j]+1
                        L[i]= L[j]+1
Lis_max=1
for i=1 to n
        if L[i]>Lis_max  Lis_max = L[i]
Return Lis_max
```

$$L[i] = \max_{j:1\le j<i, a_j<a_i} L[j] + 1$$

return $\max_i L[i]$

Run time?