CS F364
Design & Analysis of Algorithms

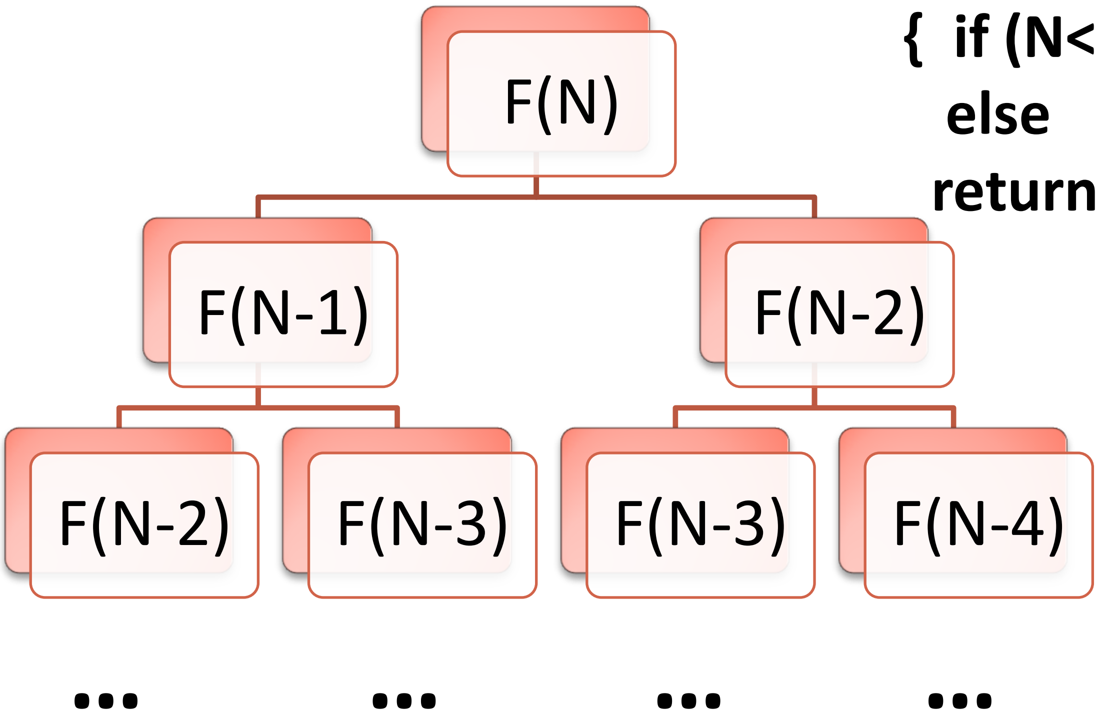
Algorithm Design Techniques

Bottom-up Design

Dynamic Programming

- **- Examples**
- **- Fibonacci sequence**
- **- 0/1 Knapsack**

Example – Fibonacci Sequence – Top Down




```
F(N)
 {  if (N<=1) return 1;
    else
   return F(N-1) + F(N-2);  }
```

Example – Fibonacci Sequence – Memoized Solution

```
// define array fib of <done: boolean, val: int>
 // initialize: for i=0 to N   fib[i]. done=false
//  fib[0].val = fib[1].val = 1;
//  fib[0].done = fib[1].done = true;
F(N)
 {
   if (fib[N].done) return fib[N].val;
   else {
      fib[N].val = F(N-1) + F(N-2);
      fib[N].done = true;
      return fib[N].val;
    }
 }
```

Example – Fibonacci Sequence – DP Solution

Known (Atomic) Solutions: F(0) = 1; F(1) = 1;

Recursive structure: F(j) = F(j-1) + F(j-2)   for j>=2

i.e. using F(0), and F(1) we can compute F(2)

using F(1) and F(2) we can compute F(3) …

This results in a bottom-up algorithm (referred to as a Dynamic Programming algorithm)

```
F(N)
 // define array fib[0..N] of int
 {
    fib[0] = fib[1] = 1; // atomic solutions
    for (j=2; j<=N; j++)   fib[j]  =   fib[j-1]  +  fib[j-2];
    return fib[N];
}
```

Straightforward conversion from Memoized version:

Time Complexity? Space Complexity?

EXAMPLE – FIBONACCI SEQUENCE – DP SOLUTION

Optimal space: fib[j] only requires fib[j-1] and fib[j-2]

```
F(N)
 {
    fib0 = fib1 = 1; // atomic solutions
     for (j=2; j<=N; j++)   { fib2  =   fib1  +  fib0;
                              fib0 = fib1;
                              fib1 = fib2;
                            }
     return fib2;
}
```

Space Complexity?

Example – Fibonacci Sequence – DP Solution

Exercise: Derive a linear-time (i.e. logN time) version of F(N).

[Hint: (1) Formulate this is as a matrix recurrence:

$$F_{n+1} = F_n + F_{n-1}$$

$$F_n = F_n$$

(2) Use repeated squaring to compute M^k

End of Hint.]