CS F364
Design & Analysis of Algorithms

Algorithm Design Techniques

Top-Down Design and Divide & Conquer

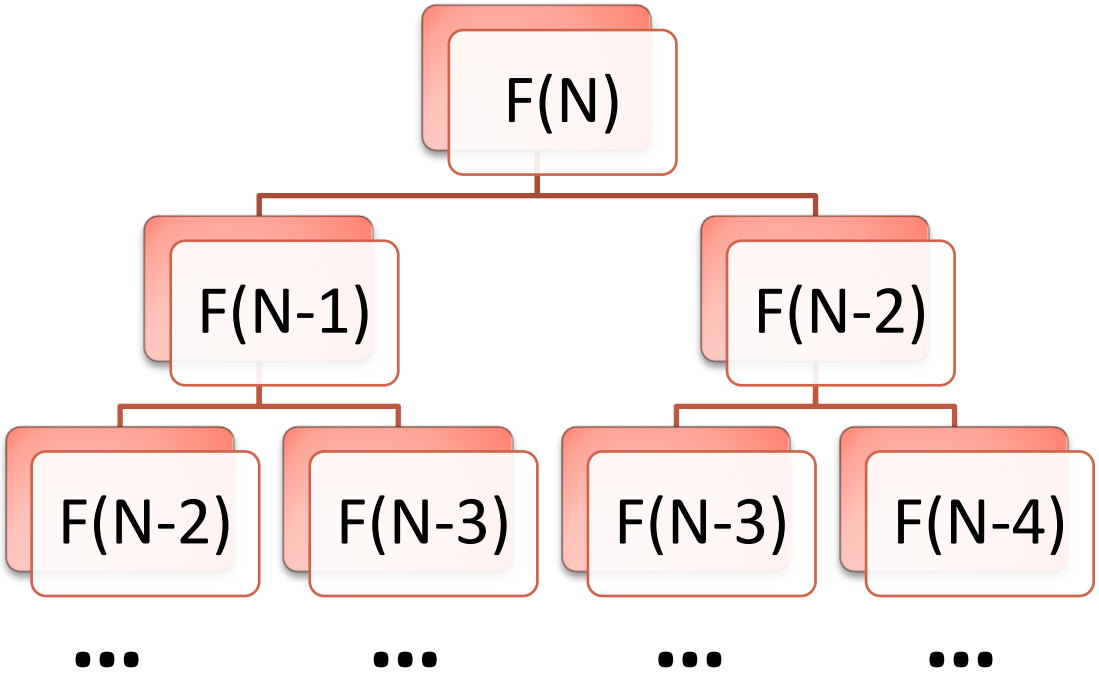
- Limitation: Overlapping Sub-problems
- Solution: Re-using solutions

Top Down Design And Divide & Conquer

- Sub-problems may be identical
  - But design structure may not recognize / reconcile them
    - Results in repeated work
- Consider the problem of

  computing the Nth term of the Fibonacci Sequence

Example – Fibonacci Sequence



Typical Recursive Implementation:

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

Example – Fibonacci Sequence

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

How many recursive calls?

Solve:

T(n) = T(n-1)+T(n-2) if n>=2

= 1 otherwise

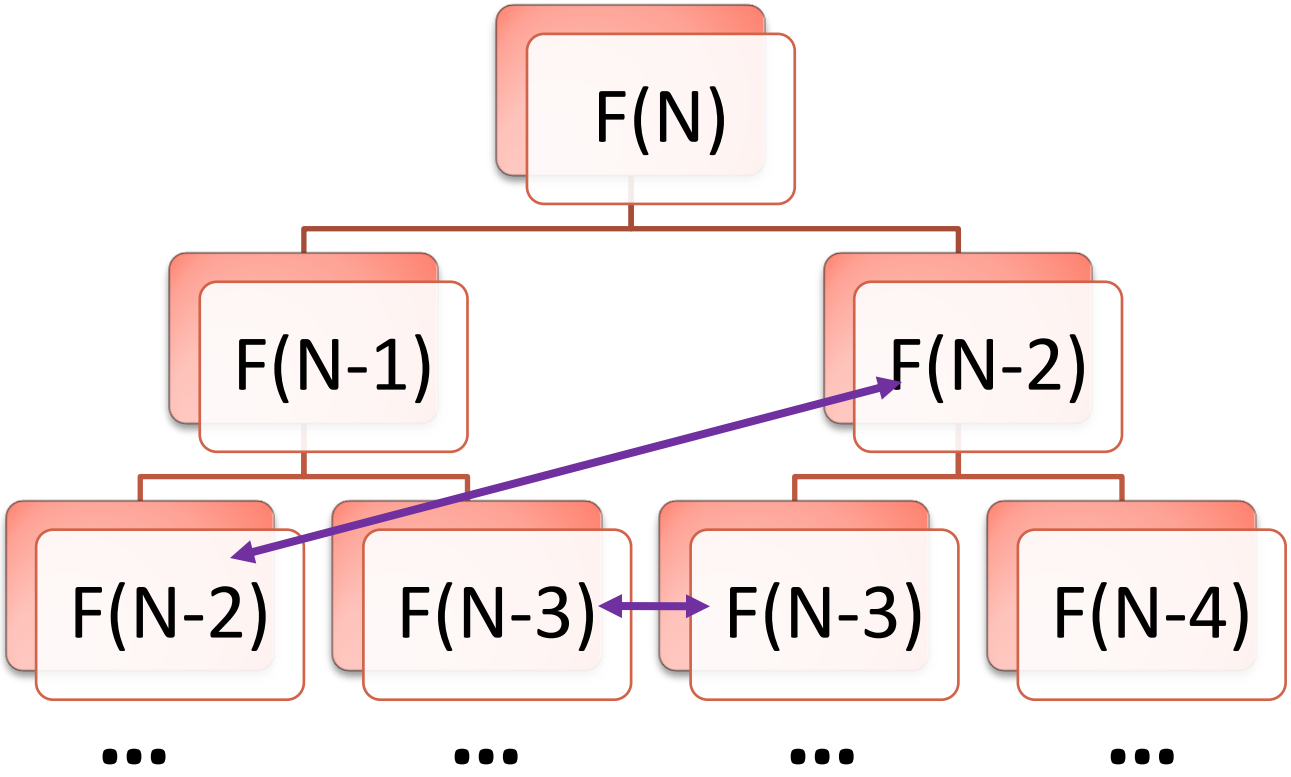
Characteristic equation: t^2 – t^1 – 1 = 0

Solution: t=(1+√5)/2

Thus $T(n) = O(c^n)$ where $c=(1+\sqrt{5})/2$.

Example – Fibonacci Sequence

Overlapping Sub-Problems



Reusing Solutions

- If overlapping sub-problems can be recognized as such
  - can we reuse the solutions?
- Approach: ***Store the results of the sub problems.***

```
F(N)
 // array fib of <done: boolean, val: int>
 // initialize:
//        fib[0].done=fib[1].done = true;
//        fib[0].val=fib[1].val=1;
//        for i=2 to N  fib[i]. done=false
// Pre-condition: ∀i  (fib[i].done ==> F(i)=fib[i].val)
 {

}
```

Algorithm for Fibonacci Numbers: Reusing Solutions

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

Algorithm for Fibonacci Numbers: Reusing Solutions

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

Time Complexity - How many recursive calls?

Space Complexity – How many locations?