CS F364

Design & Analysis of Algorithms

Algorithm Design Techniques

Top-Down Design and Divide & Conquer

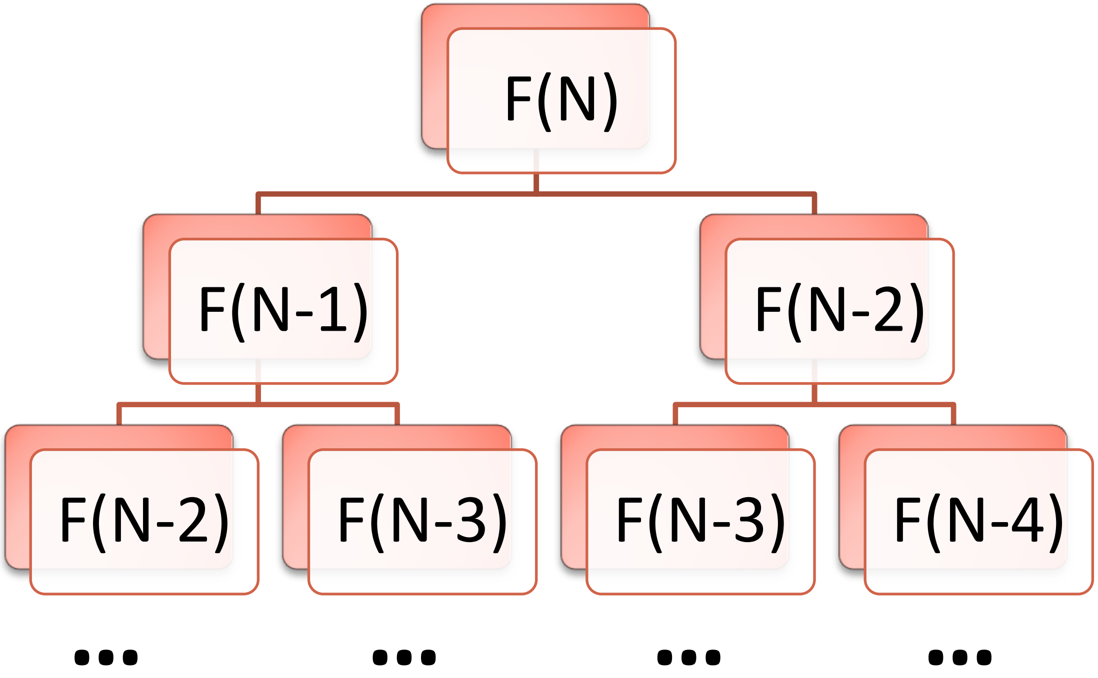
- **Issue: Overlapping Sub-problems**
- **Solution: Re-using solutions**
  - **Memoization.**

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 - Design



Typical Recursive Implementation:

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

EXAMPLE – FIBONACCI SEQUENCE - ALGORITHM

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

- **How many recursive calls?**
- **Solve:**

$$T(n) = T(n-1)+T(n-2) \text{ if } n>=2$$
$$= 1 \text{ otherwise}$$

Example – Fibonacci Sequence – Analysis

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

 # recursive calls:

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

- **Characteristic equation: t^2 – t^1 – 1 = 0**
- **Solution: $t=(1+\sqrt{5})/2$**
- Thus $\mathbf{T(n) = O(t^n)}$ where $t=(1+\sqrt{5})/2$.

Exercise: Verify this;
(while doing so,) review *"the characteristic equation method" of solving recurrence relations*.
Refer to ***Mott, Kandel and Baker***.
End of Exercise.

Example – Fibonacci Sequence – Complexity

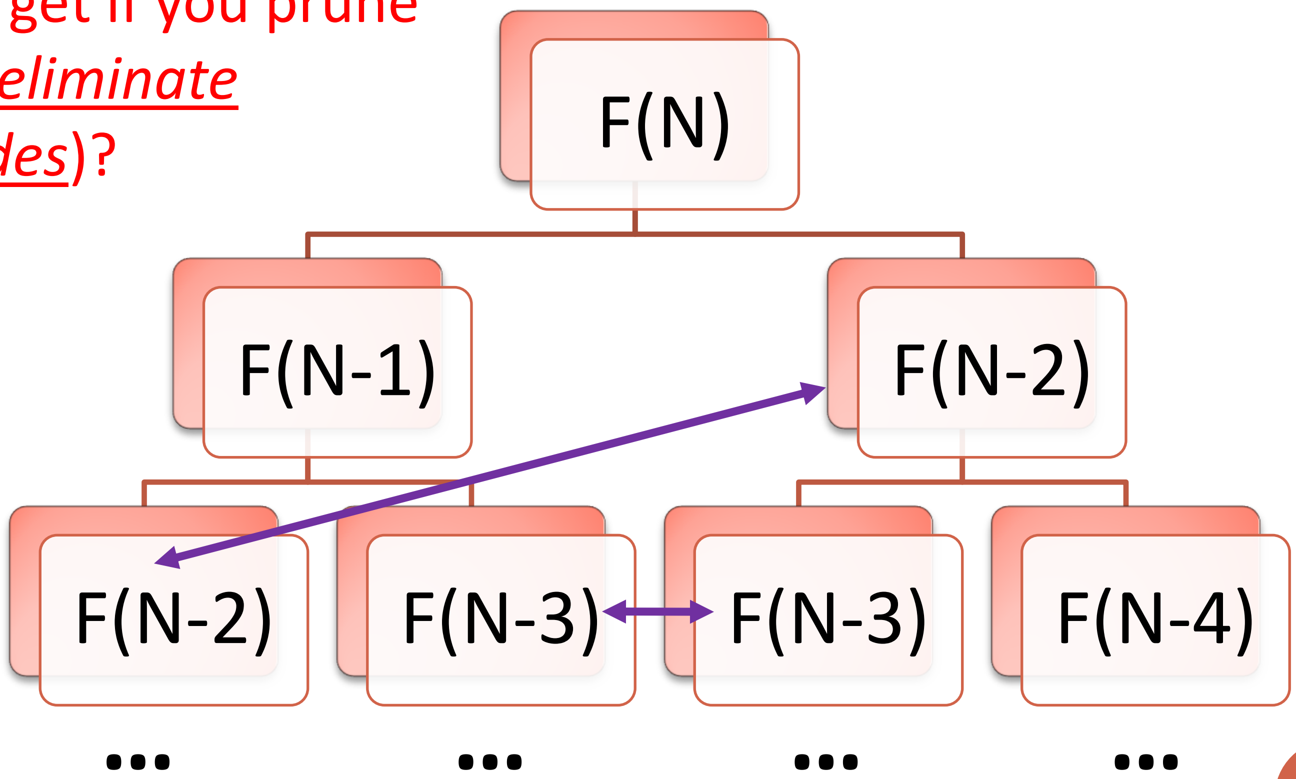
- Time Complexity:
  - $O(1.62^n)$
- Space Complexity:
  - ?

Example – Fibonacci Sequence

Overlapping Sub-Problems

Question:

What do you get if you prune the tree (i.e. *eliminate duplicate nodes*)?



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)
{     ...
}
// Post-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;
{
    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
{
    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?

MEMOIZATION

- Memo (i.e. store) the results of the sub-problems
  - with the hope that *they may be re-used*
- Space-Time tradeoff:
  - If the results are reused time is saved
  - If the results are not reused space is wasted
- How do we know that results will be reused?
  - Consider the structure of the problem
    - If the problem can be specified in terms of the sub-problems
    - then we can identify *whether sub-problems overlap.*