Lecture 4

Divide & Conquer

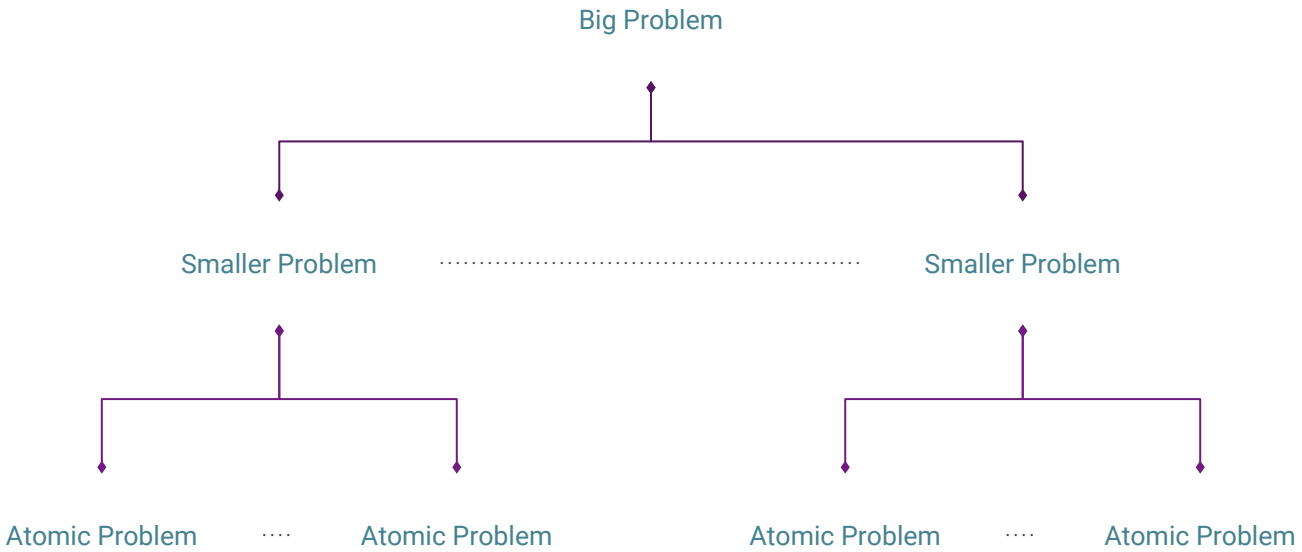
Md. Asif Bin Khaled

Divide & Conquer

In divide-and-conquer, we solve a problem recursively, applying three steps at each level of the recursion:

1. **Divide:** Divide the problem into a number of subproblems that are smaller instances of the same problem.
2. **Conquer:** Conquer the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.
3. **Combine:** Combine the solutions to the subproblems into the solution for the original problem.

Divide & Conquer Continued



Divide & Conquer Continued

The most well known algorithm design strategy:

1. Divide instance of problem into two or more smaller instances.
2. Solve smaller instances recursively.
3. Obtain solution to original (larger) instance by combining these solutions.

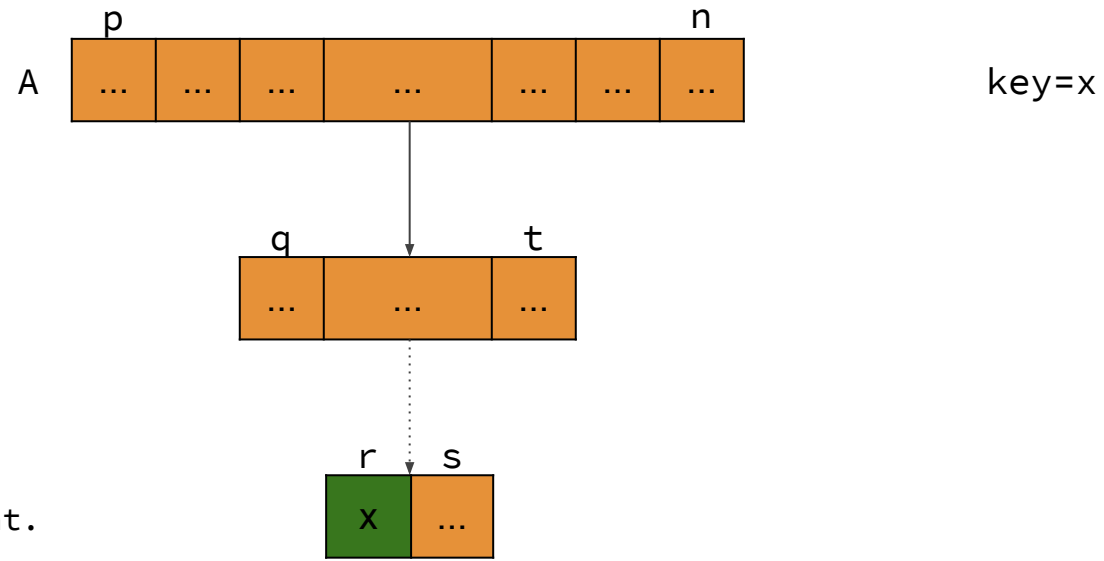
Binary Search

```
Binary_Search(A, l,r, key):
    if (r >= l)
        int mid =(l + r)/2;
        if (A[mid] == key):
             return mid
        if (A[mid] > key):
             return Binary_Search(A, l, mid-1, key);
        return Binary_Search(A, mid+1, r, key);
```

Binary Search Continued

$T(n) = 1T(n/2) + \Theta(1)$

Number of Subproblems	1
Subproblem Size	n/2
Work Dividing and Combining	$\Theta(1)$



Divide: Check middle element.

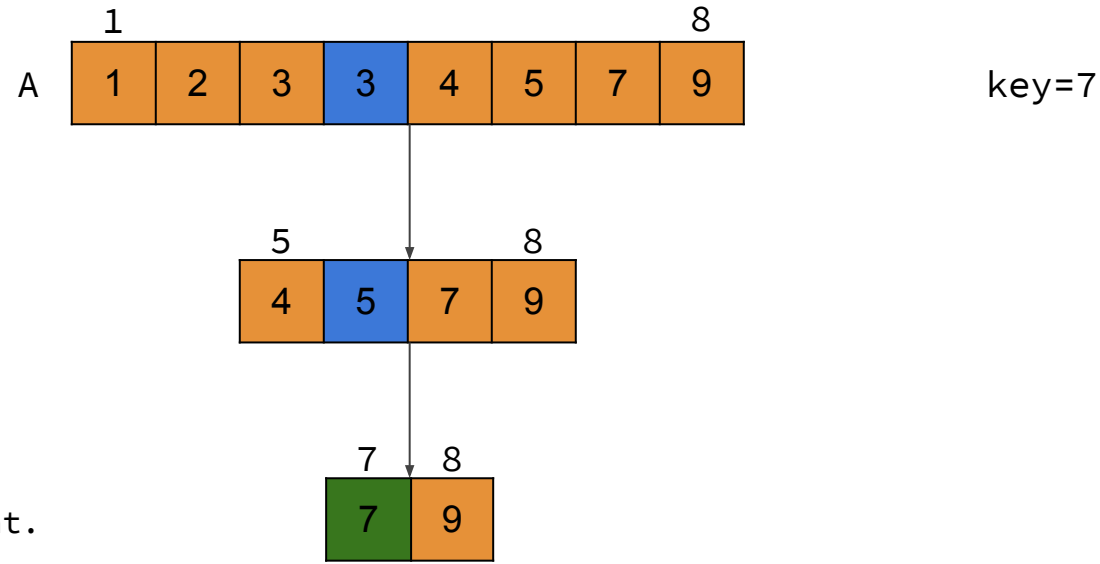
Conquer: Recursively search 1 subarray.

Combine: Trivial.

Binary Search Continued

$T(n) = 1T(n/2) + 0(1)$

Number of Subproblems	1
Subproblem Size	n/2
Work Dividing and Combining	0(1)



Divide: Check middle element.

Conquer: Recursively search 1 subarray.

Combine: Trivial.

Powering a Number [Fast]

```
Power(a,b):
    if(b==0):
        return 1
    else if(b%2==0):
        x=Power(a,b/2)
        return x*x
    else:
        x=Power(a,b/2)
        return x*x*a
```

Powering a Number [Fast] Continued

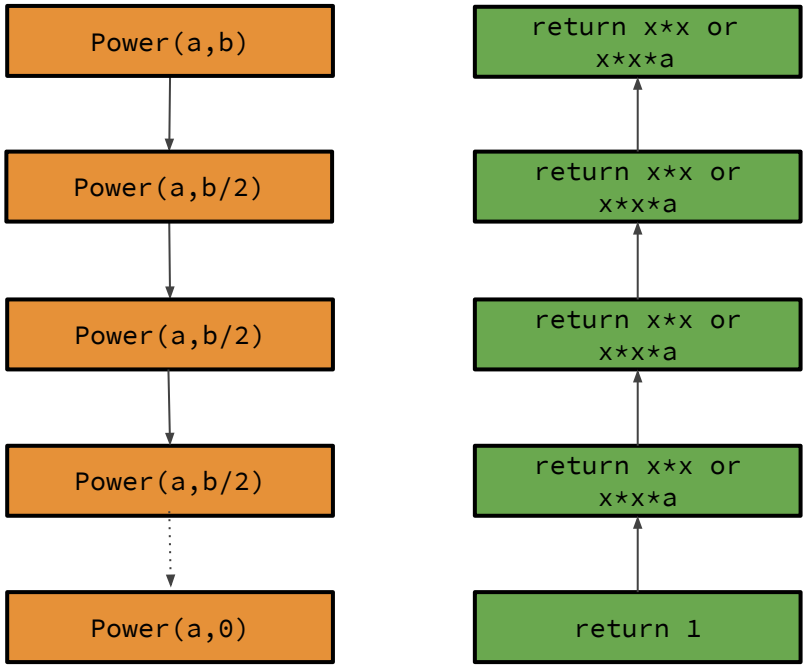
$T(n) = 1T(n/2) + \Theta(1)$

Number of Subproblems	1
Subproblem Size	n/2
Work Dividing and Combining	Θ(1)

Divide: Dividing the Exponent

Conquer: Trivial

Combine: Multiply.



Powering a Number [Fast] Continued

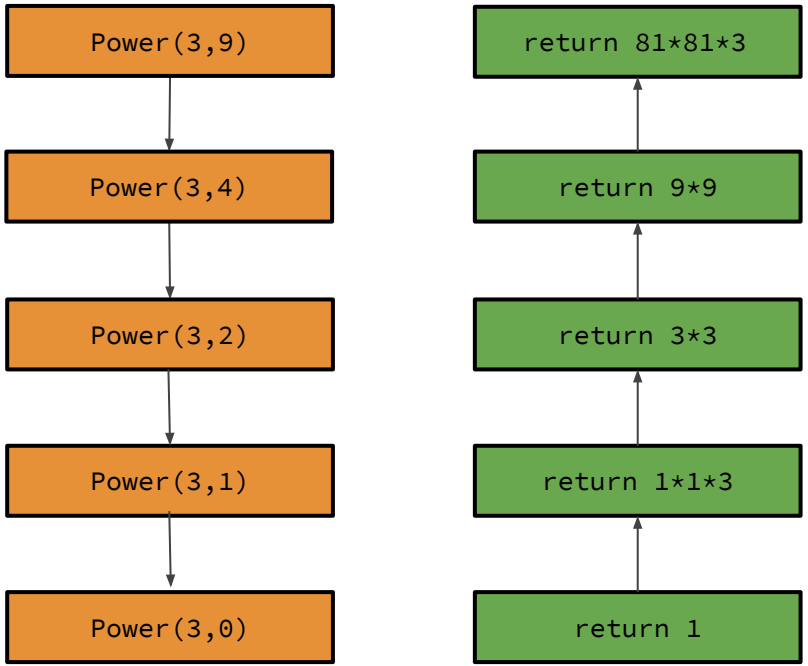
$T(n) = 1T(n/2) + \Theta(1)$

Number of Subproblems	1
Subproblem Size	n/2
Work Dividing and Combining	$\Theta(1)$

Divide: Dividing the Exponent

Conquer: Trivial

Combine: Multiply.



Fibonacci [Slow]

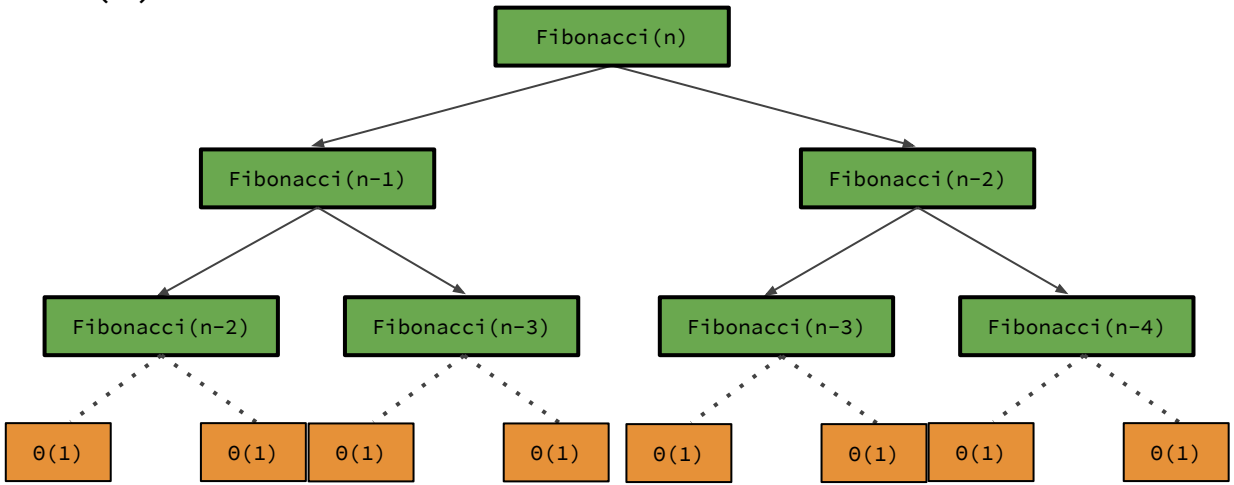
```
Fibonacci(n):
    if (n==0 || n==1):
        return n
    else:
        return Fibonacci(n-1) + Fibonacci(n-2)
```

Fibonacci [Slow] Continued

$$T(n) = T(n-1) + T(n-2) + \Theta(1)$$

$$T(n) \leq 2T(n-1) + \Theta(1)$$

Number of Subproblems	2
Subproblem Size	n-1
Work Dividing and Combining	$\Theta(1)$



Divide: Reducing the problem size.

Conquer: Trivial

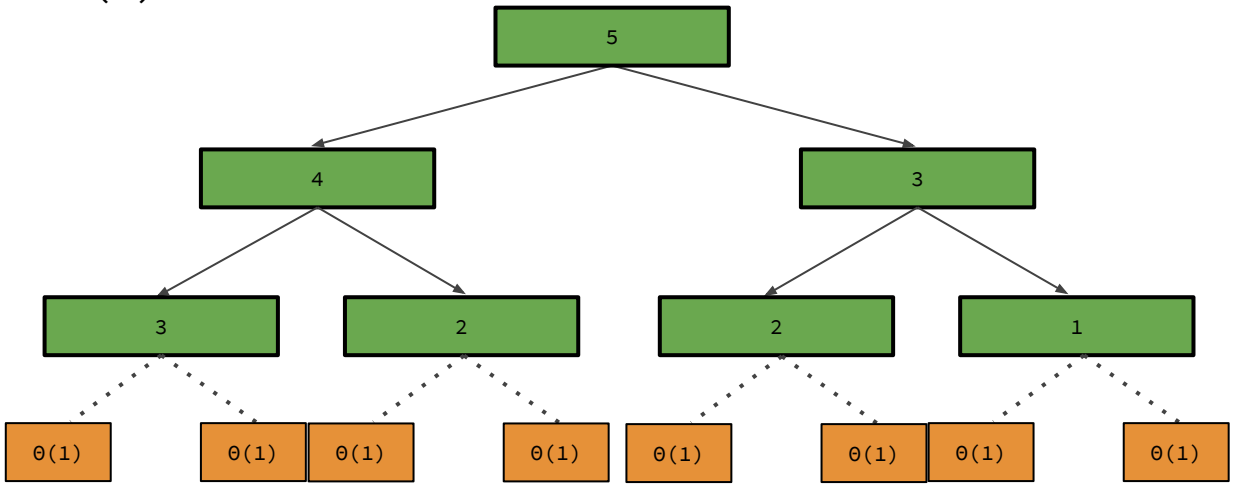
Combine: Addition.

Fibonacci [Slow] Continued

$$T(n) = T(n-1) + T(n-2) + \Theta(1)$$

$$T(n) \leq 2T(n-1) + \Theta(1)$$

Number of Subproblems	2
Subproblem Size	n-1
Work Dividing and Combining	$\Theta(1)$



Divide: Reducing the problem size.

Conquer: Trivial

Combine: Addition.

Fibonacci [Fast] Recursive Squaring

```
Fibonacci(n):
      int F[2][2] = {{1,1},{1,0}};
      if (n == 0)
            return 0;
      F = Power(F, n-1);
      return F[0][0];
Power(F[2][2], n):
      M[2][2] = {{1,1},{1,0}};
      if(n     ==     0     ||     n     ==     1)
            return;
      Power(F, n / 2);
      F=Multiply(F, F)
      if (n % 2 != 0):
            Multiply(F, M);
      return F
```

```
Multiply(F[2][2], M[2][2]):
      x = F[0][0]*M[0][0] + F[0][1]*M[1][0]
      y = F[0][0]*M[0][1] + F[0][1]*M[1][1]
      z = F[1][0]*M[0][0] + F[1][1]*M[1][0]
      w = F[1][0]*M[0][1] + F[1][1]*M[1][1]

      F[0][0] = x
      F[0][1] = y
      F[1][0] = z
      F[1][1] = w
      return F
```

Fibonacci [Fast] Recursive Squaring Continued.

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$$

Fibonacci [Fast] Recursive Squaring Continued.

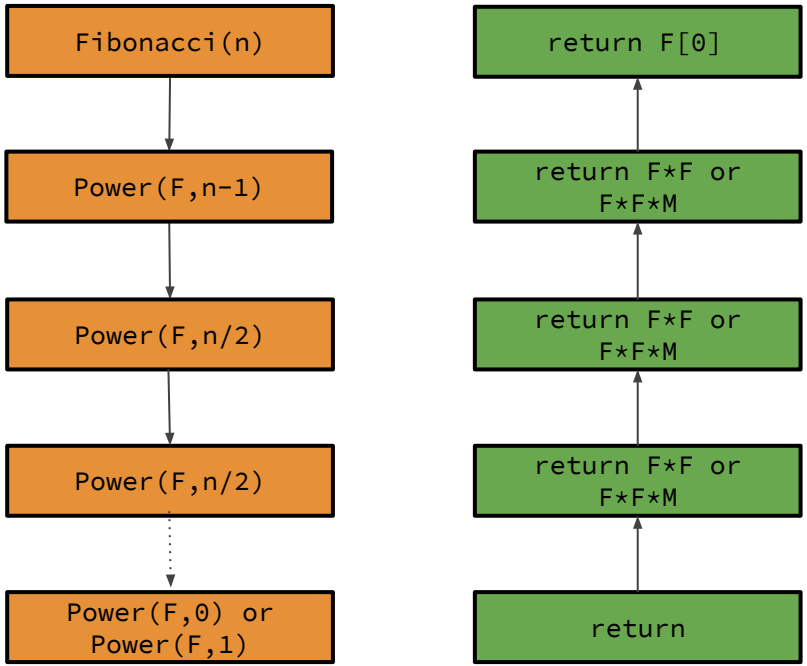
$T(n) = 1T(n/2) + \Theta(1)$

Number of Subproblems	1
Subproblem Size	n/2
Work Dividing and Combining	$\Theta(1)$

Divide: Dividing the Exponent

Conquer: Trivial

Combine: Multiply.



Fibonacci [Fast] Recursive Squaring Continued.

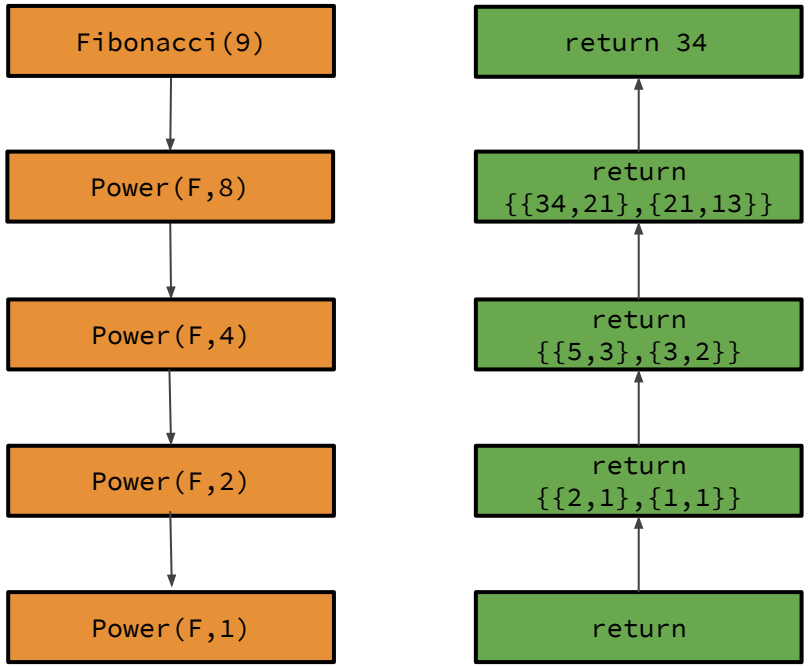
$T(n) = 1T(n/2) + \Theta(1)$

Number of Subproblems	1
Subproblem Size	n/2
Work Dividing and Combining	Θ(1)

Divide: Dividing the Exponent

Conquer: Trivial

Combine: Multiply.



Advantages of Divide & Conquer Paradigm

1. It allows us to solve difficult and often impossible looking problems, such as the Tower of Hanoi, which is a mathematical game or puzzle. Being given a difficult problem can often be discouraging if there is no idea how to go about solving it.
2. It often plays a part in finding other efficient algorithms, and in fact it was the central role in finding the quick sort and merge sort algorithms. It also uses memory caches effectively. The reason for this is the fact that when the sub problems become simple enough, they can be solved within a cache, without having to access the slower main memory, which saves time and makes the algorithm more efficient.

Disadvantages of Divide & Conquer Paradigm

1. Recursion is slow, which in some cases outweighs any advantages of this divide and conquer process. Another concern with it is the fact that sometimes it can become more complicated than a basic iterative approach, especially in cases with a large n.
2. Sometimes once the problem is broken down into sub problems, the same sub problem can occur many times. In cases like these, it can often be easier to identify and save the solution to the repeated sub problem, which is commonly referred to as memoization.