Advanced Data Structure and Algorithms fo Problem Solving

Lecture # 04

Vineet Gandhi

Center for Visual Information Technology (CVIT),

IIIT Hyderabad

Recursion

- Recursion is a mathematical technique that evaluates a function by calling the same function repeatedly on smaller inputs.
- Most programming languages support such a style of programming.
  - Often very elegant to study.
- Helps in problem solving too.

Recursion

- **Q:** How many twists does it take to screw in a light bulb?
  A: Is it already screwed in? Then zero. If not, then twist it once, ask me again, and add 1 to my answer.

Recursion

- Relates to mathematical induction
- Divide and Conquer algorithms

Lets start with an examples

- A mathematical view of computer science
- $\boxed{\text{factorial(n) = n* factorial(n-1)}}$ equation

factorial (1) = 1

n x (n-1 x n-2 x n-3 x . . . .3x2x1)

Lets start with an examples

- A mathematical view of computer science
- $\boxed{\text{factorial(n) = n* factorial(n-1)}}$ equation

```
int fact (int n)
{   if (n==1) return 1;
  else return n × fact(n-1);
}
```

Three Laws of Recursion

- Recursion must have a base case
- A recursive algorithm must change its state and move toward the base case.
- A recursive algorithm must call itself, recursively.

Factorial – deeper look

fact(5)

if(n==1) return 1

else return n x fact(n-1)

		return	state
→	F(5)	5 x F(4)	P.
→	F(4)	4 x F(3)	P
→	F(3)	3 x F(2)	P
→	F(2)	2 x F(1)	R
	F(1)	1	

Factorial – deeper look

fact(5)

if(n==1) return 1

else return n x fact(n-1)

	return	state
→ F(5)	5 x F(4)	R
→ F(4)	4 x F(3)	R
→ F(3)	3 x F(2)	R
→ F(2)	2 x F(1)	R
F(1)	1	

Factorial – deeper look

fact(5)

if(n==1) return 1

else return n x fact(n-1)

	return	state
→ F(5)	5 x F(4)	R
→ F(4)	4 x F(3)	R
→ F(3)	3 x F(2)	R
→ F(2)	2 x F(1)	R
F(1)	1	

$T(n) = T(n-1) + 3$

$T(n) = T(n-2) + 6$

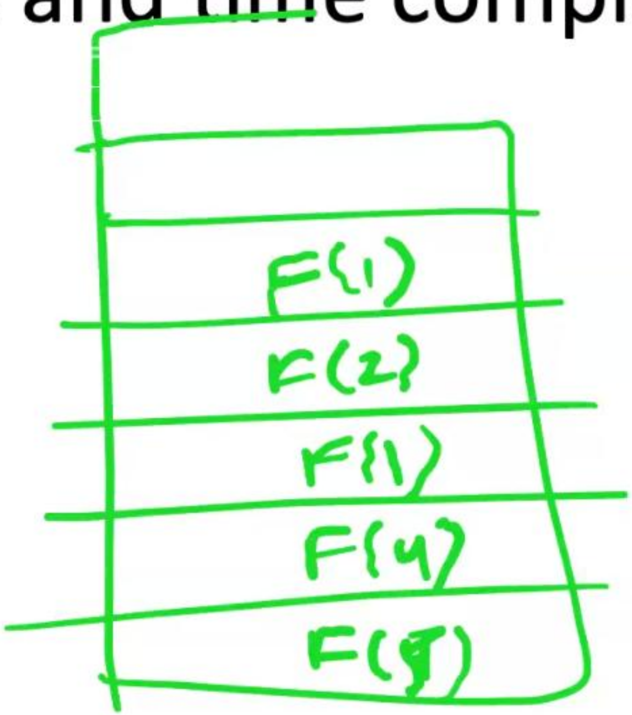
$T(n) = T(n-k) + k \cdot 3$

$\boxed{T(n) = T(0) + n \cdot 3}$

$n - k = 0 \quad k = n$

Factorial – deeper look

- Space and time complexity



Inductive Reasoning

Factorial (n) = n × Fact (n−1)

- ~~A recursive algorithm must have a **base case**~~.
- If I call factorial(n) with n=1, I am done
- If I call factorial(n) with n>1, it makes a recursive call with a smaller value of n; must eventually reach n=1.

Recursion and Induction

L.H.S R.H.S

$$1 + 2 + 3 + 4 + \cdots + n = n(n+1)/2$$

$n = 0$ $0 = 0$ ✓

$k+1$

$$\underbrace{1 + 2 + 3 + 4 + \cdots\cdots + k} + k+1 = \frac{(k+1)(k+2)}{2}$$

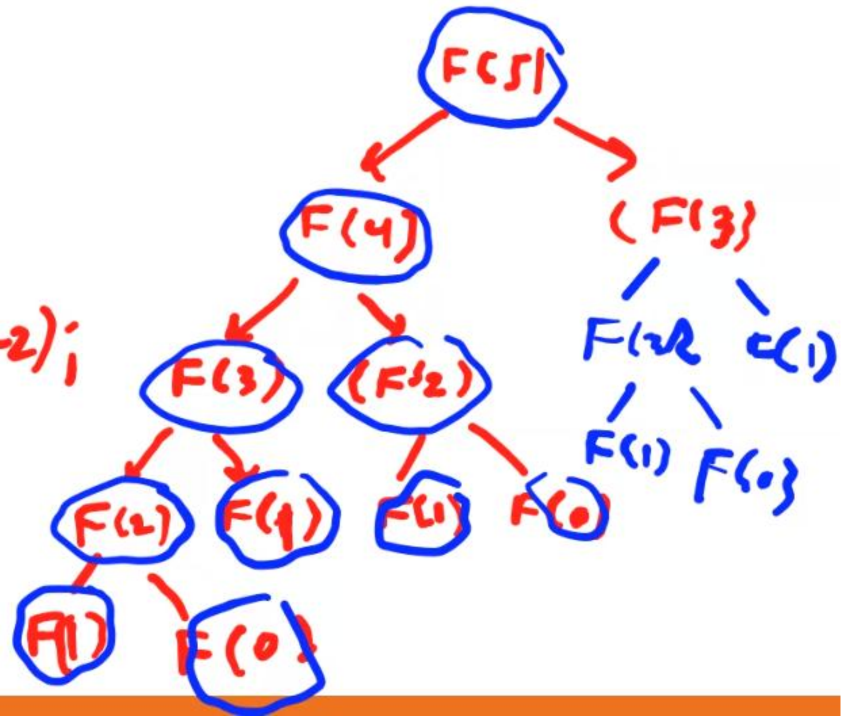
$$\frac{k(k+1)}{2} + (k+1)$$

$$= \frac{k(k+1) + 2(k+1)}{2} = \frac{(k+1)(k+2)}{2}$$

Recursion with multiple base cases

$$fib(n) = \begin{cases} fib(n-1) + fib(n-2) & \text{if } n > 1 \\ n & \text{if } n = 0, 1 \end{cases}$$

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



Fibonacci deeper look

- Time and space complexity

$$\boxed{T(n) = T(n-1) + T(n-2) + 4}$$

$$T(0) = T(1) = 1$$

$$T(n-1) \approx T(n-2)$$

$$\boxed{T(n) = 2\,T(n-2) + c}$$

$$\boxed{2^{n/2}}$$

$$T(n) = 2\left[2 \cdot T(n-4) + c\right] + c$$

$$T(n) = 2^k\,T(\underline{n-2k}) + (2^k - 1)\,c$$

This is lower bound as we have replaced n-1 with n-2. To find upper bound, replace n-2 with n-1 (2^n) lecture4 - recursion — vi fib_rec.c — 80×24

```c
#include<stdio.h>

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

int main(){

        int a;
        printf("Enter a number:");
        scanf("%d",&a);
        printf("%d\n",fib(a));
        return 0;

}
~
~
~
~
~
~
~
-- INSERT --
```

lecture4 - recursion — vi fib_iter.c — 80×24

```c
#include<stdio.h>

int fib(int n){
        if(n<=1)
                return n;
        int i,sum=0, f0=0, f1=1;
        for(i=2;i<=n;i++){
                sum =  f0+f1;
                f0 = f1;
                f1 = sum;
        }
        return sum;
}

int main(){
        int a;
        printf("enter a number:");
        scanf("%d",&a);
        printf("%d \n",fib(a));
        return 0;
}
~
~
"fib_iter.c" 21L, 265C
```

Recursion with memoization

```
Fib(n)
{
    if (n<=1) return n;
 if Fn is in memory return Fn
 else
     return ← Fib(n-1) + Fib(n-2)
     Save Fn in memory
     return Fn
}
```

Exponentiation

x^n

$$x^n = \begin{cases} n = 0 & x^n = 1 \\ x \cdot x^{n-1} & \text{oth} \end{cases}$$

$$x^n = \begin{cases} x^{n/2} \cdot x^{n/2} & n \text{ is even} \\ x \cdot x^{n-1} & n \text{ is odd} \\ 1 & n = 0 \end{cases}$$