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
- factorial(n) = n* factorial(n-1)

equation

factorial (1) = 1

Factorial – deeper look

if(n == 1) return 1

else return n x fact(n-1)

fact(5)

		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	

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

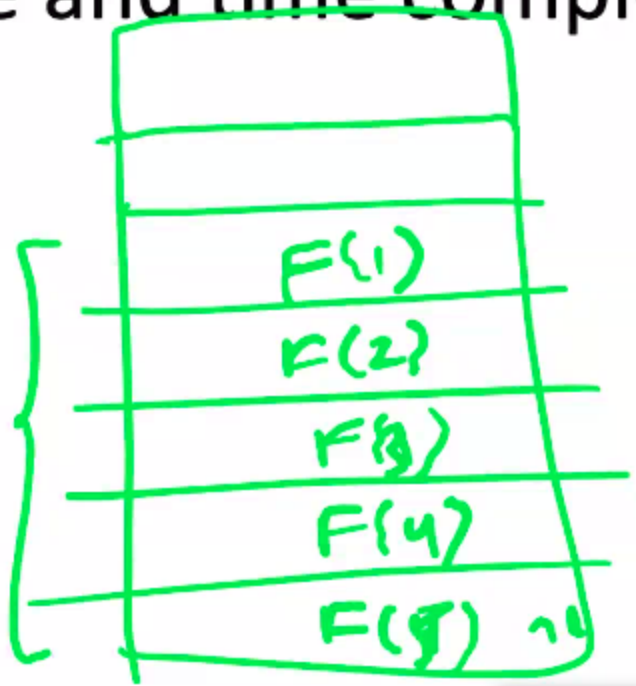
$T(n) = T(n-2) + 6$

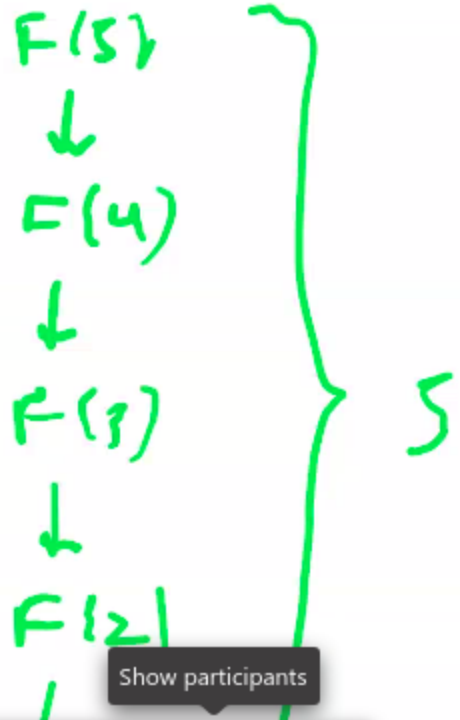
$T(n) = T(n-k) + k \cdot 3$

$T(n) = T(0) + n$

Factorial – deeper look

- Space and time complexity





Inductive Reasoning

- 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

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

→ $n = 0$ $\quad 0 = 0$ ✓

→ $\underbrace{1 + 2 + 3 + + + \ldots\ldots + k}_{k+1} + k+1 = \dfrac{(k+1)(k+2)}{2}$

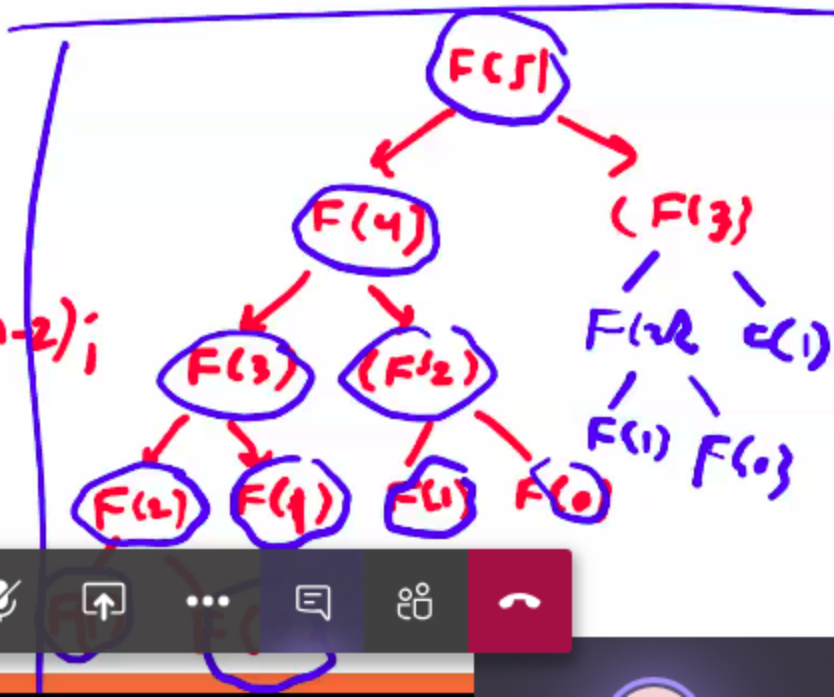
$\dfrac{k(k+1)}{2} + (k+1)$

$= k(k+1) + 2(k+1) = (k+1)(k+2)$

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

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

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

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

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

$$2^{n/2}$$

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

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

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