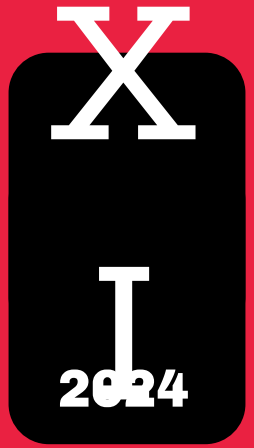
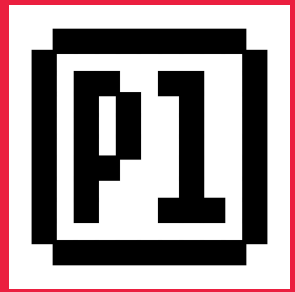
Algoritmos III





UNRN
Universidad Nacional
de Río Negro

rXXI

Complejidad II

Operaciones de la secuencia `Arreglo`

`a = arreglo_t[i]`	como L-Value
`arreglo_t[i] = a`	como R-Value
`ordena(arreglo_t)`	Ordenamiento
`busca(arreglo_t, valor)`	Búsqueda
`inserción(arreglo_t, valor, pos)`	Inserción
`inserción_inicio(arreglo_t, valor)`	Inserción
`eliminar(arreglo_t, pos)`	Eliminar
`eliminar_inicio(arreglo_t)`	Eliminar

Operaciones de la secuencia Lista Enlazada

`modifica(lista_t, posicion, valor)`	como L-Value
`obtiene(lista_t, posicion, valor)`	como R-Value
`ordena(lista_t)`	Ordenamiento
`busca(lista_t, valor)`	Búsqueda
`inserción(lista_t, valor, pos)`	Inserción
`inserción_inicio(lista_t, valor)`	Inserción
`eliminar(lista_t, pos)`	Eliminar
`eliminar_inicio(lista_t)`	Eliminar

Operaciones de Pila (Stack)

`push(stack_t, e)`	meter
`pop(stack_t)`	sacar
`peek(stack_t)`	chusmear
`esta_vacia(stack_t)`	verificar


¿Preguntas?

Recursividad


sividad


SAFELY ENDANGERED
SWEET JESUS, POOH! THAT'S NOT HONEY
SAFELY ENDANGERED
YOU'RE EATING RECURSION

Recursión

La recursión ocurre cuando algo está definido en términos de sí mismo o su tipo.

Estructura de una función recursiva

Estructura base

```
funcion( valor ){
    if ( valor == 0 )
    {
        return value
    }
    else
    {
        return funcion( valor - 1 )
    }
}
```

Estructura base

```
funcion( valor ){
    if ( valor == 0 )
    {
        return value
    }
    else
    {
        return funcion( valor - 1 )
    }
}
```

sobre el caso base...

```
int factorial(int n) {
    // ups, me olvidé el caso base
    return n * factorial(n - 1);
}
```

El caso base es *importante*


¿Preguntas?

Ventajas

El código es intuitivo en su proposito.
La división de un problema en subproblemas más simples es directa.

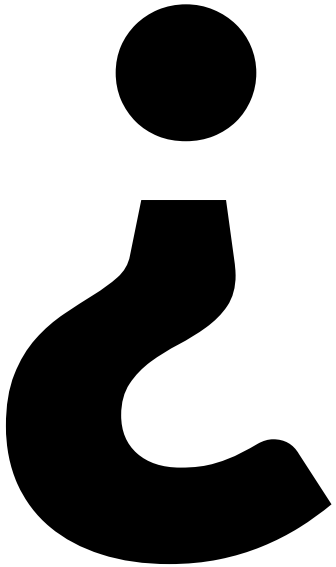
Desventajas

Consumo de memoria elevado[1]

¡Es más fácil caer en un lazo infinito!

Son *generalmente* más lentas[2]

No hay forma de manipular la repetición



Cuando se usa

Ejemplos

Suma

La suma de los números entre n y 1

$$\text{suma}(n) = n + n - 1 + \ldots + 1$$

$$\text{suma}(n) = \begin{cases} \text{suma}(1) = 1 & \text{si } n = 0 \\ \text{suma}(n) = n + \text{suma}(n - 1) & \text{si } n > 0 \end{cases}$$

Factorial recursivo

Definición de Factorial

$$n! = n \times (n - 1) \times (n - 2) \times (n - 3) \times \ldots \times 1$$

$$n! = \prod_{i=1}^{n} i$$

$$n! = \begin{cases} n! = 1 & \text{si } n = 0 \\ n! = n \times (n - 1)! & \text{si } n > 0 \end{cases}$$

Factorial

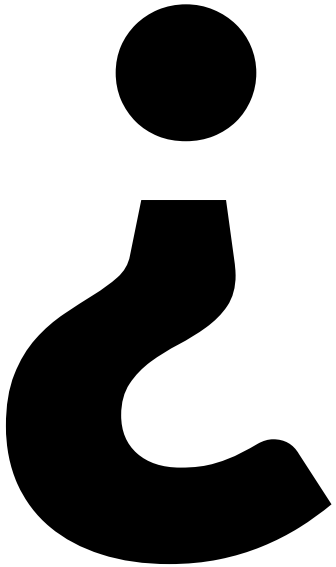
```
long factorial(long valor)
{
    if (valor < 1){
        return 1;
    }
    else
    {
        return valor * factorial(valor - 1L);
    }
}
```


¿Preguntas?

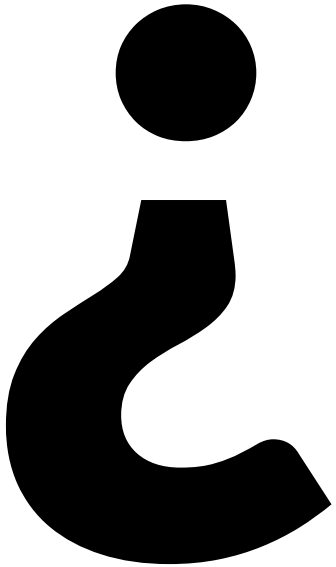
$$\text{fib(n)} = \begin{cases} 0 & \text{si n} = 0 \\ 1 & \text{si n} = 1 \\ \text{fib(n-1) - fib(n-2)} & \text{si} > 1 \end{cases}$$

¡Fibonacci recursivo!

```
long fibonacci(int termino)
{
    if (termino == 0) {
        return 0L;
    } else if (termino == 1) {
        return 1L;
    } else {
        return fibonacci(termino - 1) + fibonacci(termino - 2);
    }
}
```



Cuantas llamadas a la función son necesarias

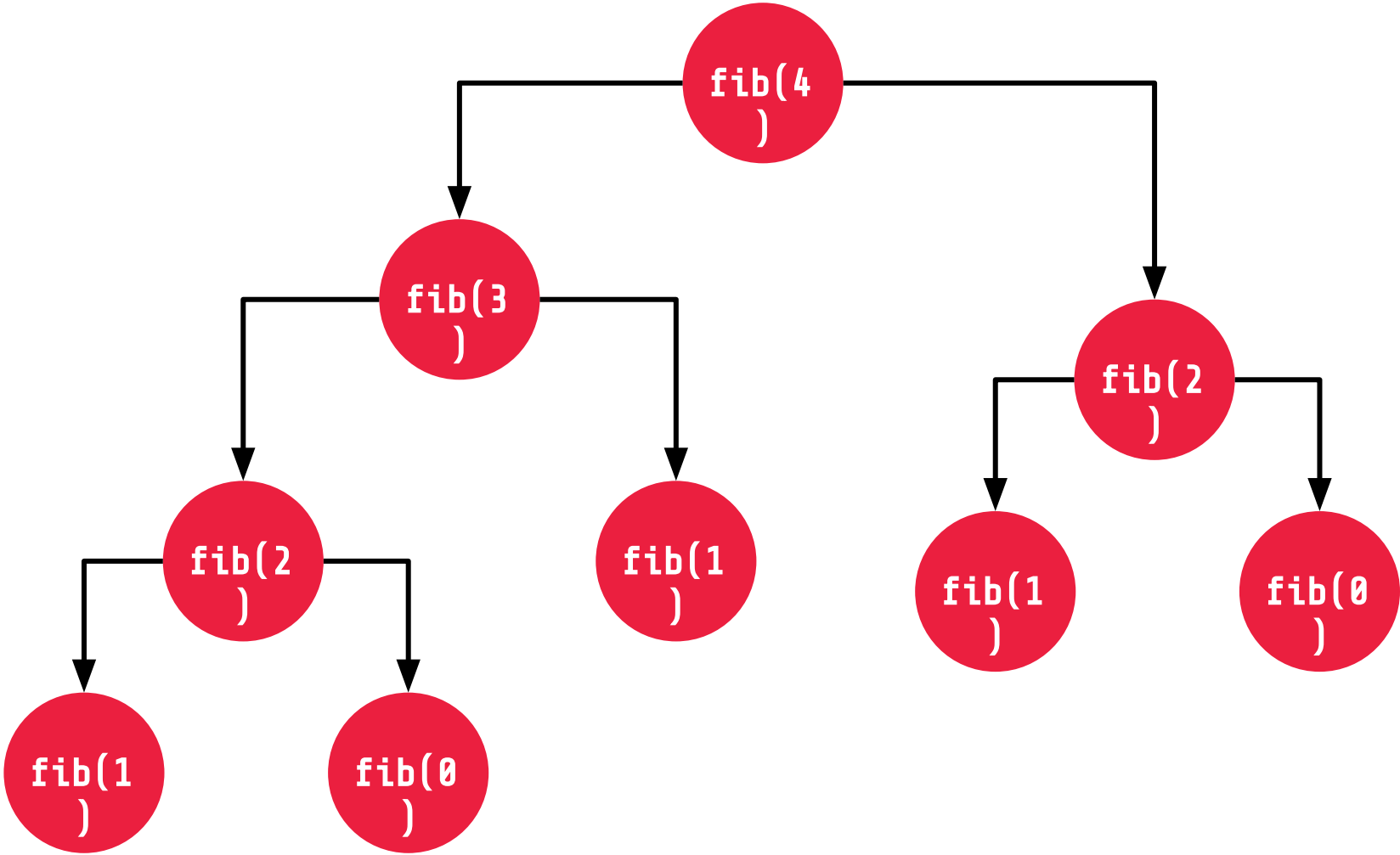


Cuanta memoria consume el algoritmo

$$O(2^n)$$

El nudo

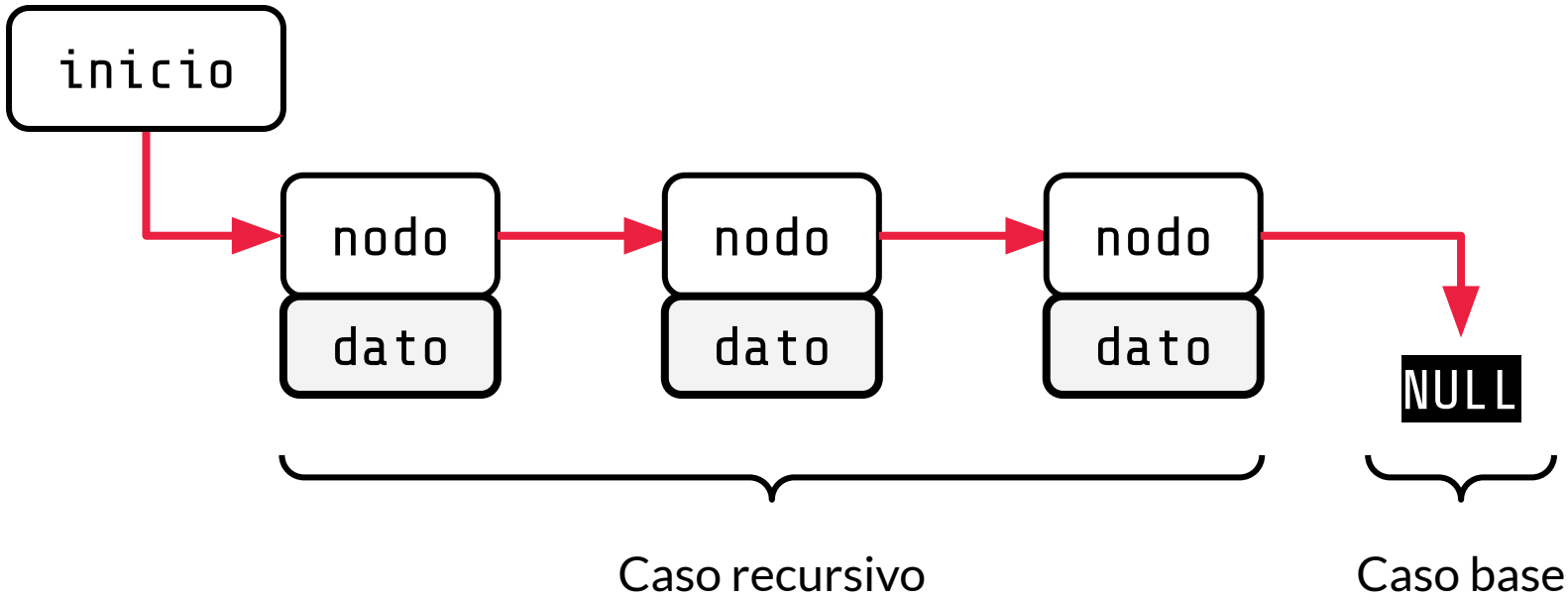
```
long fibonacci(int termino)
{
    if (termino == 0) {
        return 0L;
    } else if (termino == 1) {
        return 1L;
    } else {
        return fibonacci(termino - 1) + fibonacci(termino - 2);
    }
}
```


fib(4)
fib(3)
fib(2)
fib(1)
fib(2)
fib(1)
fib(0)
fib(1)
fib(0)


¿Preguntas?

Listas enlazadas

Lista enlazada



Recorriendo la lista

```
void imprime_lista(nodo_t *l)
{
    if(l != NULL)
    {
       printf("%d, ", nodo->valor);
       imprime_lista(nodo->siguiente);
    }
}
```

Una lista se define en términos de sí misma :-D

```
typedef struct nodo
{
    struct nodo *siguiente;
    int valor;
}nodo_t;
```


¿Preguntas?

Divide y conquista

¿Qué es?

Un poco de historia

Pasos

Dividir

Para aplicarlo, un problema dividido debe ser igual que el problema sin dividir.

Conquistar

- Si el problema es complejo[1], dividirlo
- Si el problema es simple[2], resolverlo

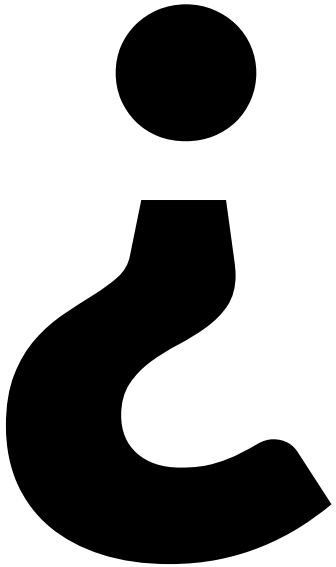
Combinar

los subproblemas simples


¿Preguntas?

Ventajas

Desventajas

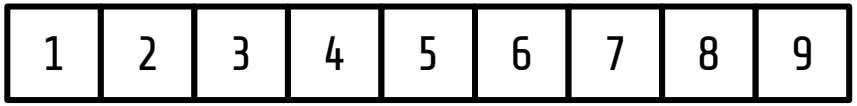


Cuando aplicarlo


¿Preguntas?

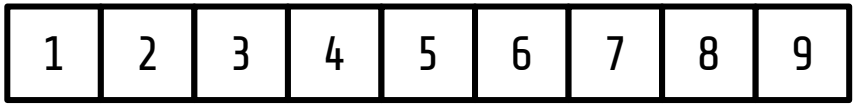
Ejemplos Recursividad + DyC

Suma de elementos recursiva



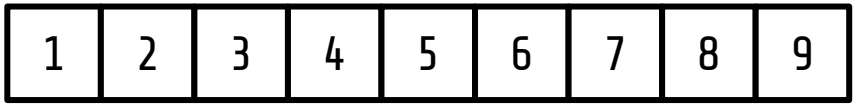

```
int suma(int arreglo[], int largo);
```

¡Aprovechando aritmética de punteros!




```
int suma(int arreglo[], int largo){
    if (largo == 1){
        return arreglo[0];
    } else {
        return arreglo[0] + suma(arreglo + 1, largo - 1);
    }
}
```

Suma de elementos por división y conquista




```
int suma_II(int arr[], int inicio, int fin);
```

Implementación de suma por DyC

```
int suma_DyC(int arr[], int inicio, int fin) {
    if (inicio == fin) {
        return arr[inicio];
    }
    int medio = inicio + (fin - inicio) / 2;

    int izquierda = suma_DyC(arr, inicio, medio);
    int derecha = suma_DyC(arr, medio + 1, fin);

    return izquierda + derecha;
}
```

Base

División

Conquista

unrn.edu.ar

UNRN Universidad Nacional de Río Negro

| unrionegro