

TEMA 1

Recursividad

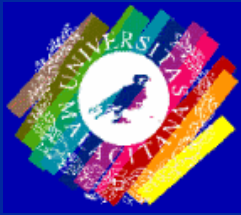


TEMA 1

Recursividad

CONTENIDO DEL TEMA

- 1.- Introducción.
- 2.-Asignación estática y dinámica de memoria.
- 3.-Verificación de funciones y procedimientos recursivos.
- 4.-Escritura de programas recursivos.



Introducción

- **Definición de Recursividad:** Técnica de programación muy potente que puede ser usada en lugar de la iteración.
- **Ambito de Aplicación:**
 - General
 - Problemas cuya solución se puede hallar solucionando el mismo problema pero con un caso de menor tamaño.
- **Razones de uso:**
 - Problemas “casi” irresolubles con las estructuras iterativas.
 - Soluciones elegantes.
 - Soluciones más simples.
- **Condición necesaria:** ASIGNACIÓN DINÁMICA DE MEMORIA



Introducción

- ¿En qué consiste la recursividad?
 - En el cuerpo de sentencias del se invoca al propio
“una versión más pequeña” del problema original.

- Aspecto de un subalgoritmo recursivo.

Algoritmo Recursivo(...);

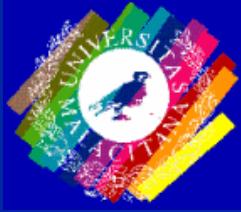
Inicio

...

Recursivo(...);

...

Fin.



Introducción

- Ejemplo: Factorial de un natural.

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

Algoritmo Factorial($n:\mathbf{N}$): \mathbf{N}

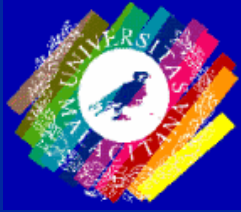
Inicio

SI $n=0$ **ENTONCES** DEVOLVER 1

ENOTROCASO DEVOLVER $n * \text{Factorial}(n-1)$

FINSI

Fin



Introducción

- ¿Cómo funciona la recursividad?

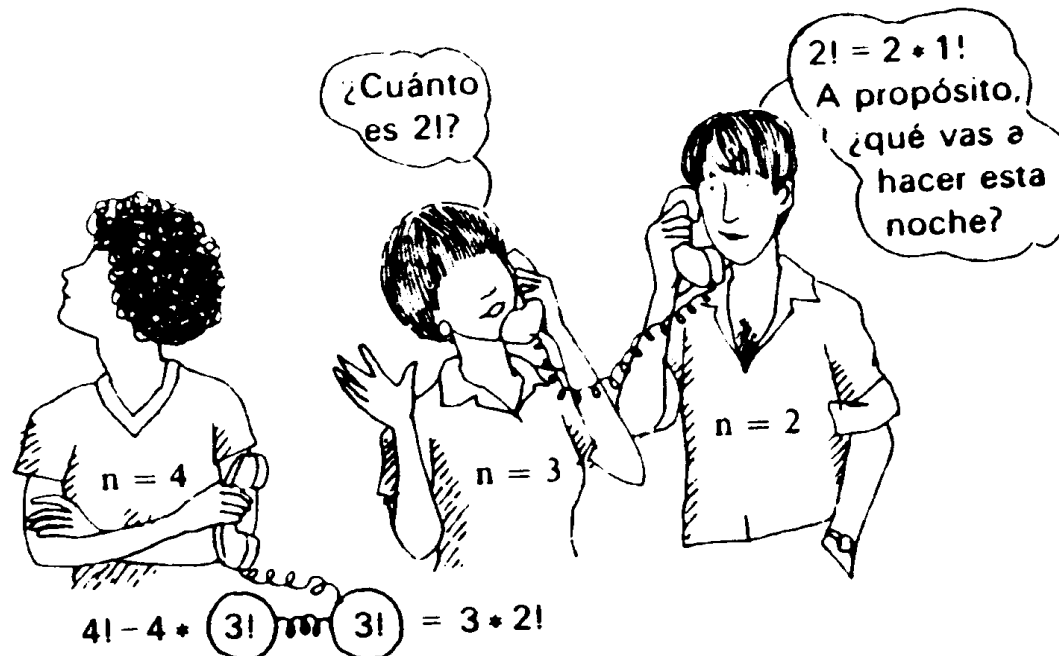
$$4! = 4 * 3!$$

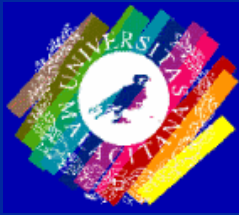




Introducción

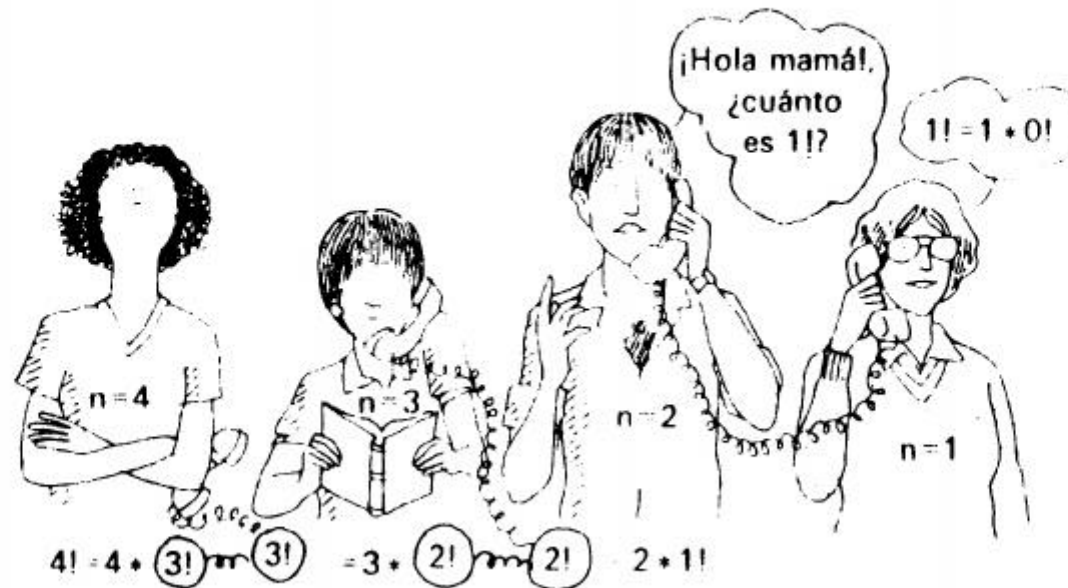
$$3! = 3 * 2!$$

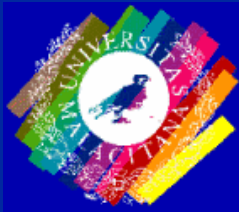




Introducción

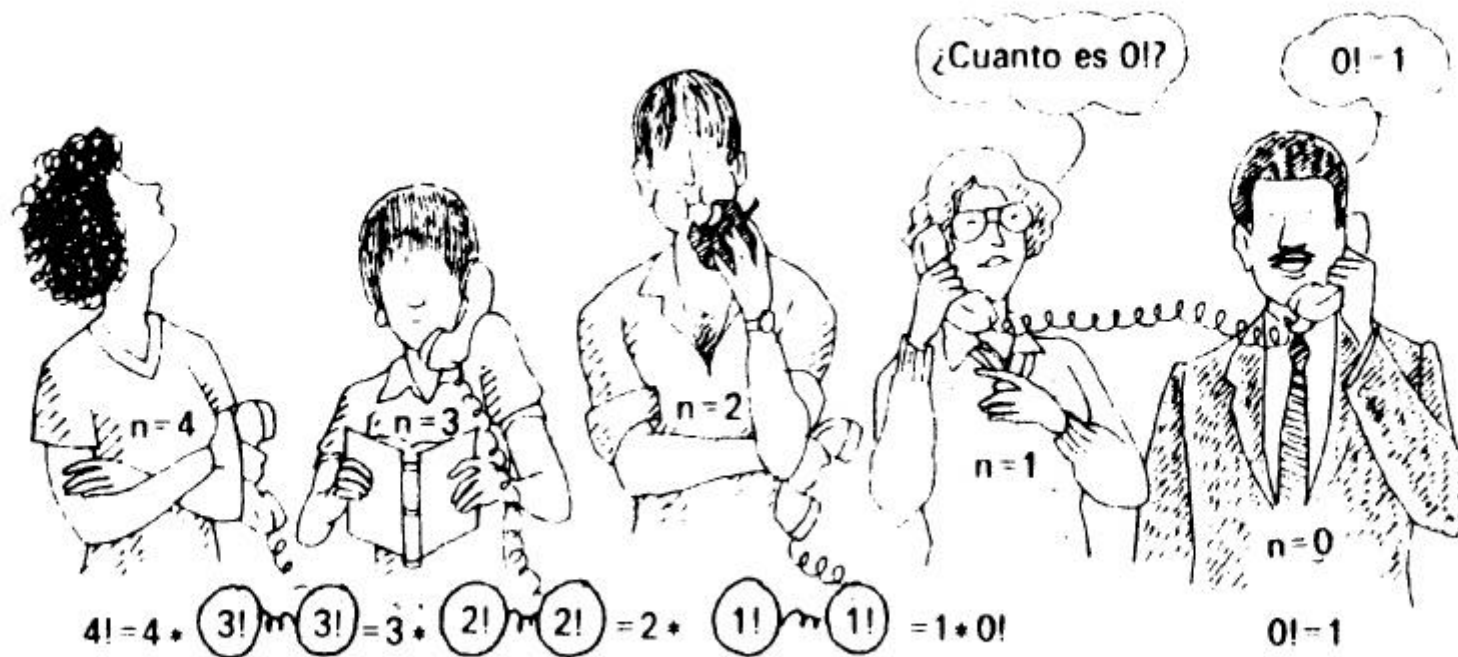
$$2! = 2 * 1!$$





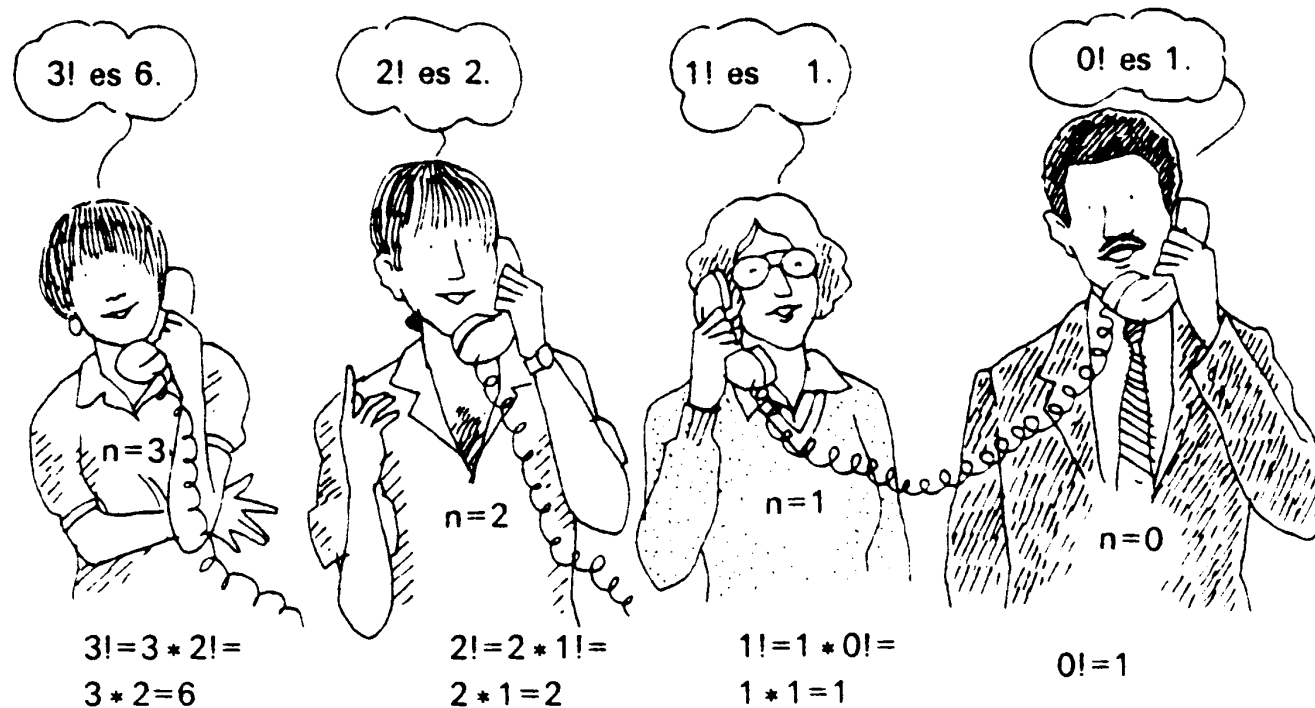
Introducción

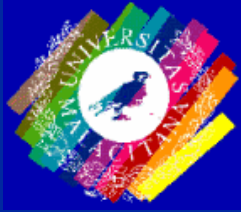
$$1! = 1 * 0! = 1 * 1$$





Introducción





Introducción

Registro de Activación.

Dirección de Retorno.

Pila (Stack).

Vinculación.



Asignación estática y dinámica de memoria

- Partimos del siguiente ejemplo

Algoritmo uno($x, y: \mathbb{R}$)

Variables

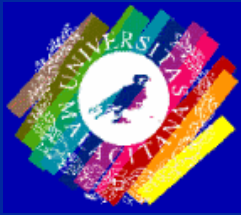
$z: \mathbb{N}$

Inicio

...

...

Fin.



Asignación estática y dinámica de memoria

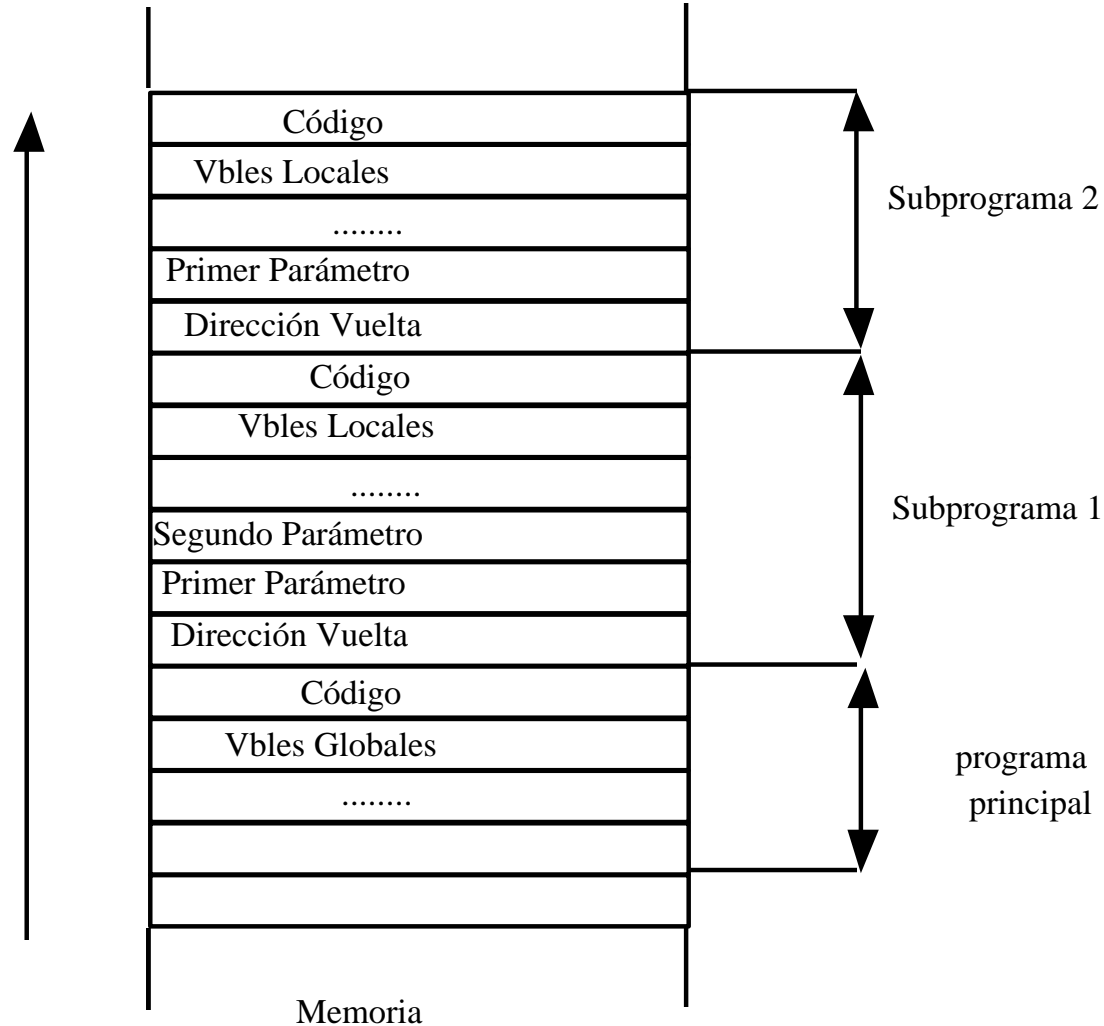
- **Asignación estática**

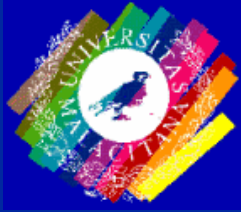
- Se reserva espacio en memoria a partir de una posición **FIJA**, tanto para el código como para los parámetros formales y variables locales de cada subprograma.
- - x ←-----→
 - y ←-----→
 - z ←-----→
- La zona reservada para variables locales y parámetros formales usualmente preceden al código del subprograma



Asignación estática y dinámica de memoria

Direcciones
altas

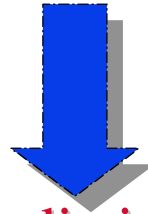




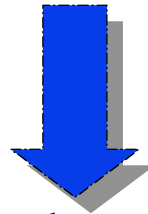
Asignación estática y dinámica de memoria

•PROBLEMA

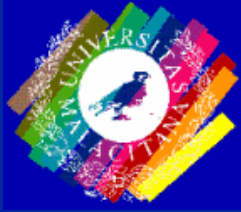
Vinculación de variables en tiempo de compilación



¿almacenamiento de las distintas llamadas recursivas?



Pérdida de los valores de las variables



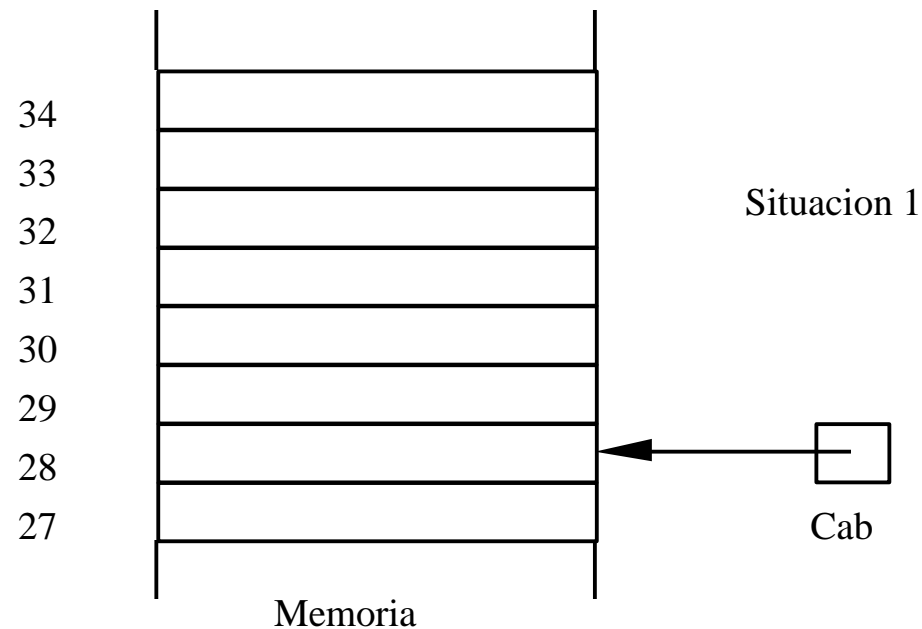
Asignación estática y dinámica de memoria

- Asignación dinámica
 - Asignación de cada variable, parámetro
 - $x \rightarrow 1$
 $y \rightarrow 2$
 $z \rightarrow 3$
 - Dirección de retorno $\rightarrow 0$



Asignación estática y dinámica de memoria

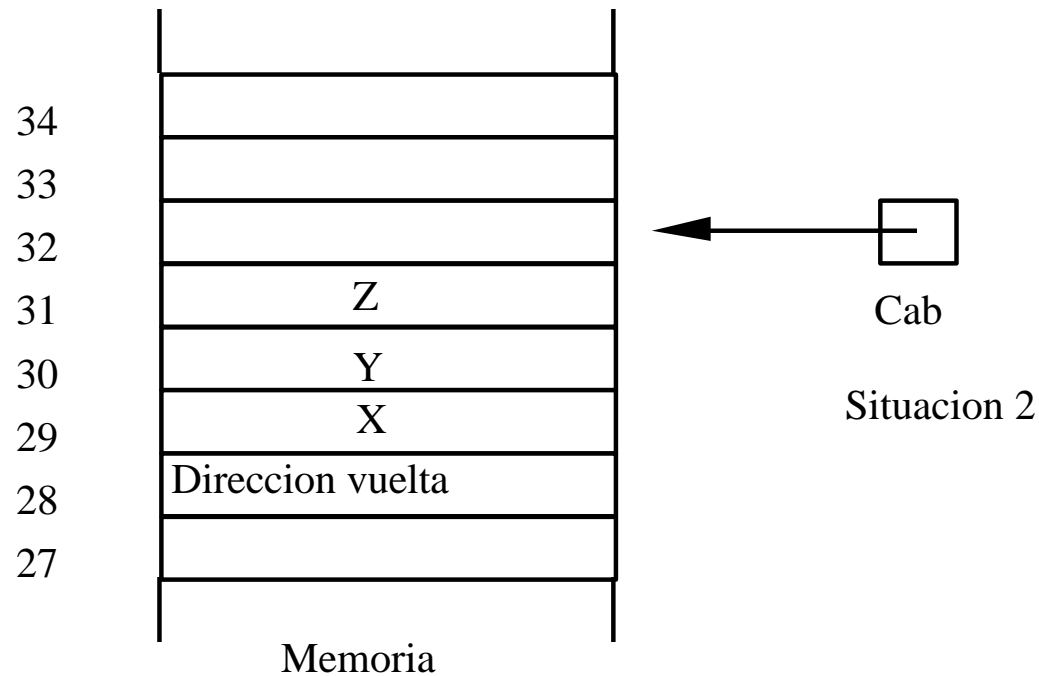
Tiempo de ejecución: Se reserva espacio para las variables y parámetros a partir de la situación actual de CAB

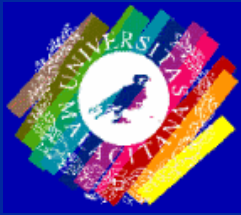




Asignación estática y dinámica de memoria

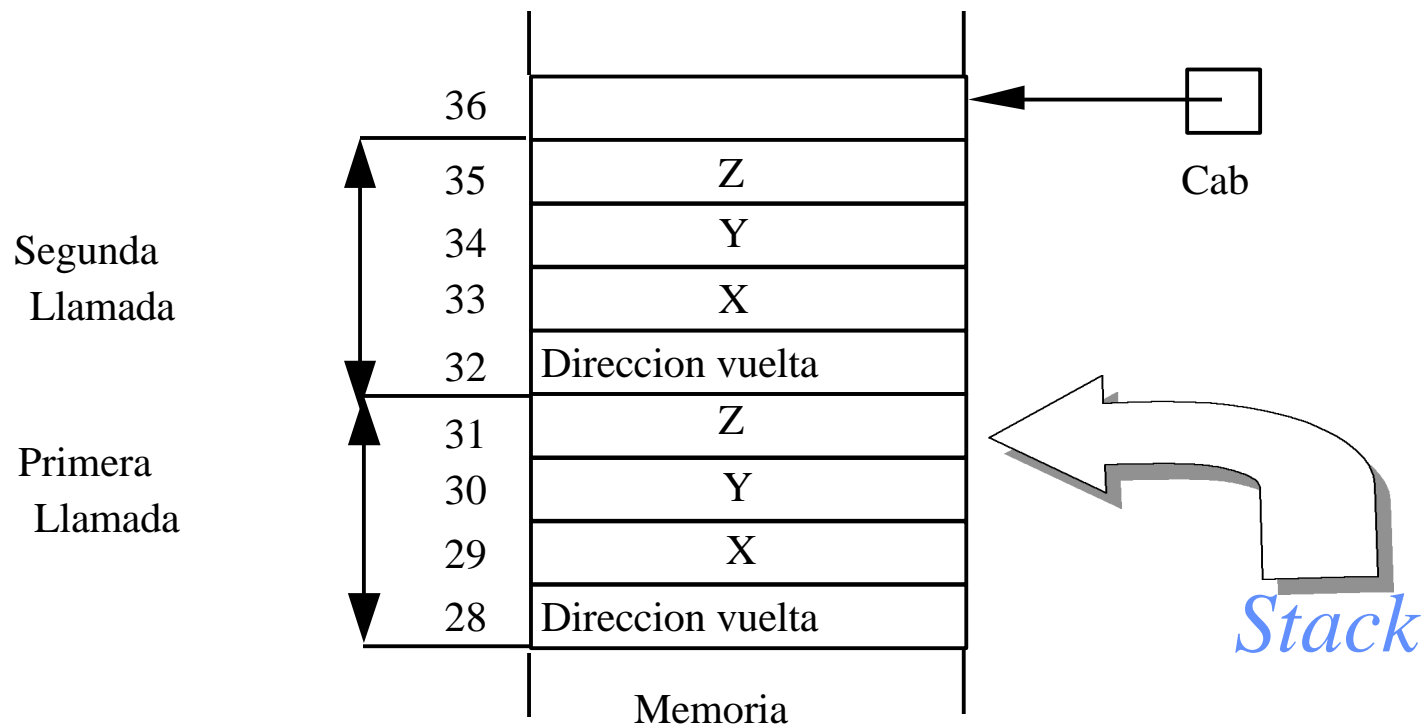
- Llamada al subalgoritmo





Asignación estática y dinámica de memoria

- Llamada recursiva al algoritmo





Asignación estática y dinámica de memoria

- Ejemplo con la función factorial

Algoritmo Factorial($n:N$): N

Inicio

SI $n=0$ **ENTONCES** DEVOLVER 1

ENOTROCASO DEVOLVER $n * \text{Factorial}(n-1)$

FINSI

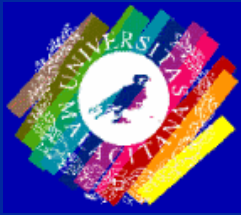
Fin

R2

La invocación inicial es:

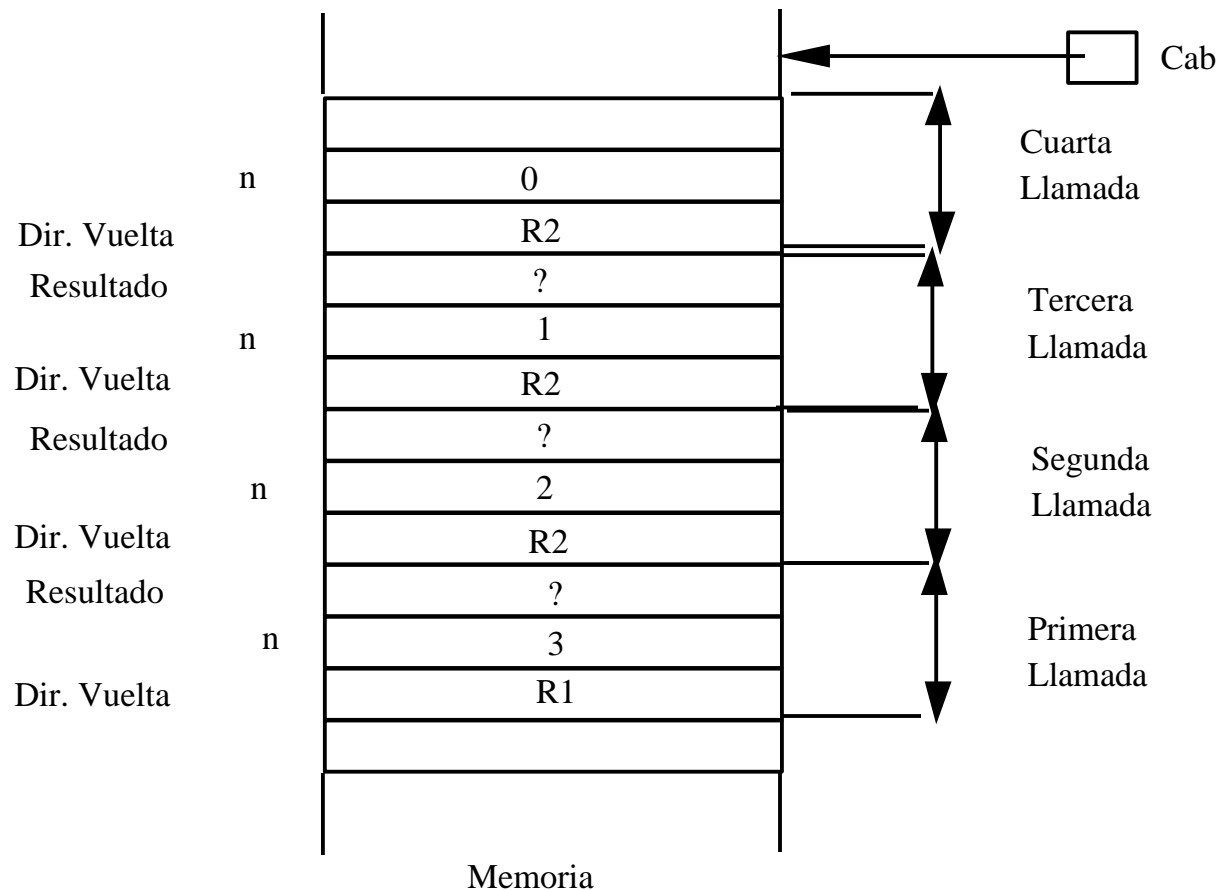
Resultado := Factorial(3)

R1



Asignación estática y dinámica de memoria

Estado





Asignación estática y dinámica de memoria

Observaciones

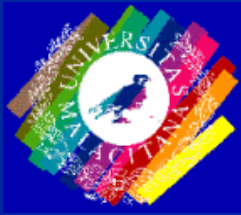
- Invocación del subalgoritmo **a sí mismo**.
- Cada llamada al subalgoritmo se realiza con un valor de parámetro que hace el problema **“de menor tamaño”**.
- La llamada al subalgoritmo se realiza siempre en una sentencia de selección.
- En dicha sentencia de selección, al menos debe de haber un caso donde se actúa de forma diferente (no recursiva). Este es
- Ocultación de los detalles de gestión de la memoria en las llamadas recursivas **(Pila interna)**.



Verificación de funciones y procedimientos recursivos

Método de las tres preguntas

- **La pregunta Caso-Base:** ¿Existe una salida no recursiva o caso base del subalgoritmo? Además, ¿el subalgoritmo funciona correctamente para ella?
- **La pregunta Más-pequeño:** ¿Cada llamada recursiva se refiere a un caso más pequeño del problema original?
- **La pregunta Caso-General:** ¿es correcta la solución en aquellos casos no base?



Escritura de programas recursivos

- Obtención de una definición exacta del problema
- Resolver el(los) casos bases o triviales (no recursivos).
- Determinar el tamaño del problema completo que hay que
→ **Parámetros en la llamada inicial**
- Resolver el caso general en términos de
(llamada recursiva).



Distintos parámetros

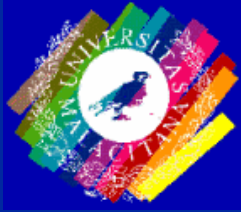


Ejemplos

- **Combinaciones:** ¿cuántas combinaciones de cierto tamaño pueden hacerse de un grupo total de elementos?

- C: número total de combinaciones
- Grupo: tamaño total del grupo del que elegir
- Miembros: tamaño de cada subgrupo
-

$$C(\text{Grupo}, \text{Miembros}) \begin{cases} -\text{Grupo} & \text{si } \text{Miembros}=1 \\ -1 & \text{si } \text{Miembros}=\text{Grupo} \\ -C(\text{Grupo}-1, \text{Miembros}-1)+C(\text{Grupo}-1, \text{Miembros}) & \text{si } \text{Grupo}>\text{Miembros}>1 \end{cases}$$



Ejemplos

- FUNCIÓN COMBINACIONES

- **Definición:** Calcular cuantas combinaciones de tamaño Miembros pueden hacerse del tamaño total Grupo

- :Número de elementos en la llamada original

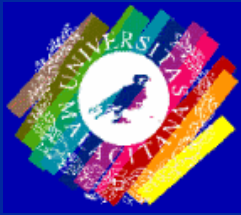
- :1) Miembros=1 → Combinaciones=Grupo

2) Miembros=Grupo → Combinaciones=1

- :Grupo>Miembros>1



Combinaciones = Combinaciones(Grupo-1 , Miembros -1)+Combinaciones(Grupo-1,



Ejemplos

Algoritmo Comb(Grupo,Miembros:N):N

Inicio

SI Miembros=1 **ENTONCES**

DEVOLVER Grupo (*Caso Base 1*)

ENOTROCASO

SI Miembros=Grupo **ENTONCES**

DEVOLVER 1 (*Caso Base 2*)

ENOTROCASO (*Caso General*)

DEVOLVER Comb(Grupo-1,Miembros-1)

+Comb(Grupo-1,Miembros)

FINSI

FINSI

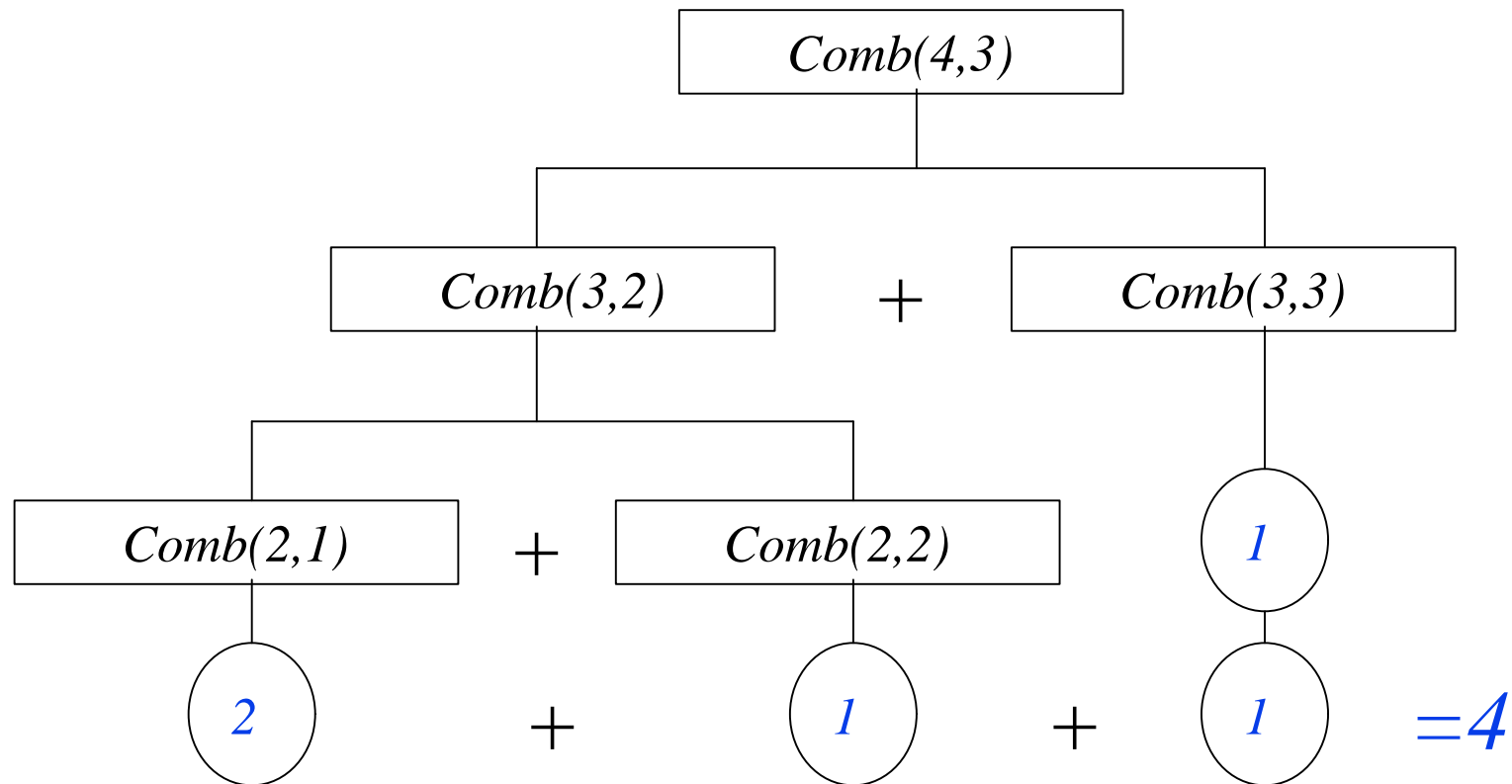
FIN

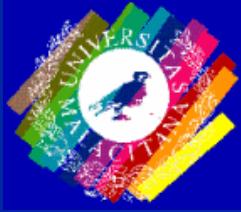
Llamada: Escribir("Número de combinaciones=", Comb(20,5))



Ejemplos

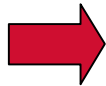
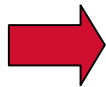
- Seguimiento de $\text{Comb}(4,3)$

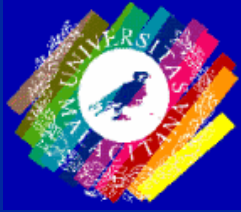




Ejemplos

FUNCIÓN FIBONACCI

- **Definiciones:** Calcular el valor de la función de Fibonacci **n** dado.
- **Tamaño:** Número **n** de la llamada original
- 
-  $\text{fib}(n) := \text{fib}(n-1) + \text{fib}(n-2)$



Ejemplos

Algoritmo Fib($n:N$):N

Inicio

SI ($n \leq 2$) **ENTONCES**

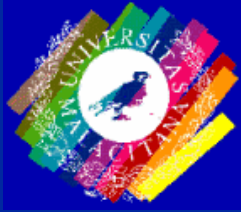
DEVOLVER 1

ENOTROCASO

DEVOLVER (Fib($n-1$)+Fib($n-2$))

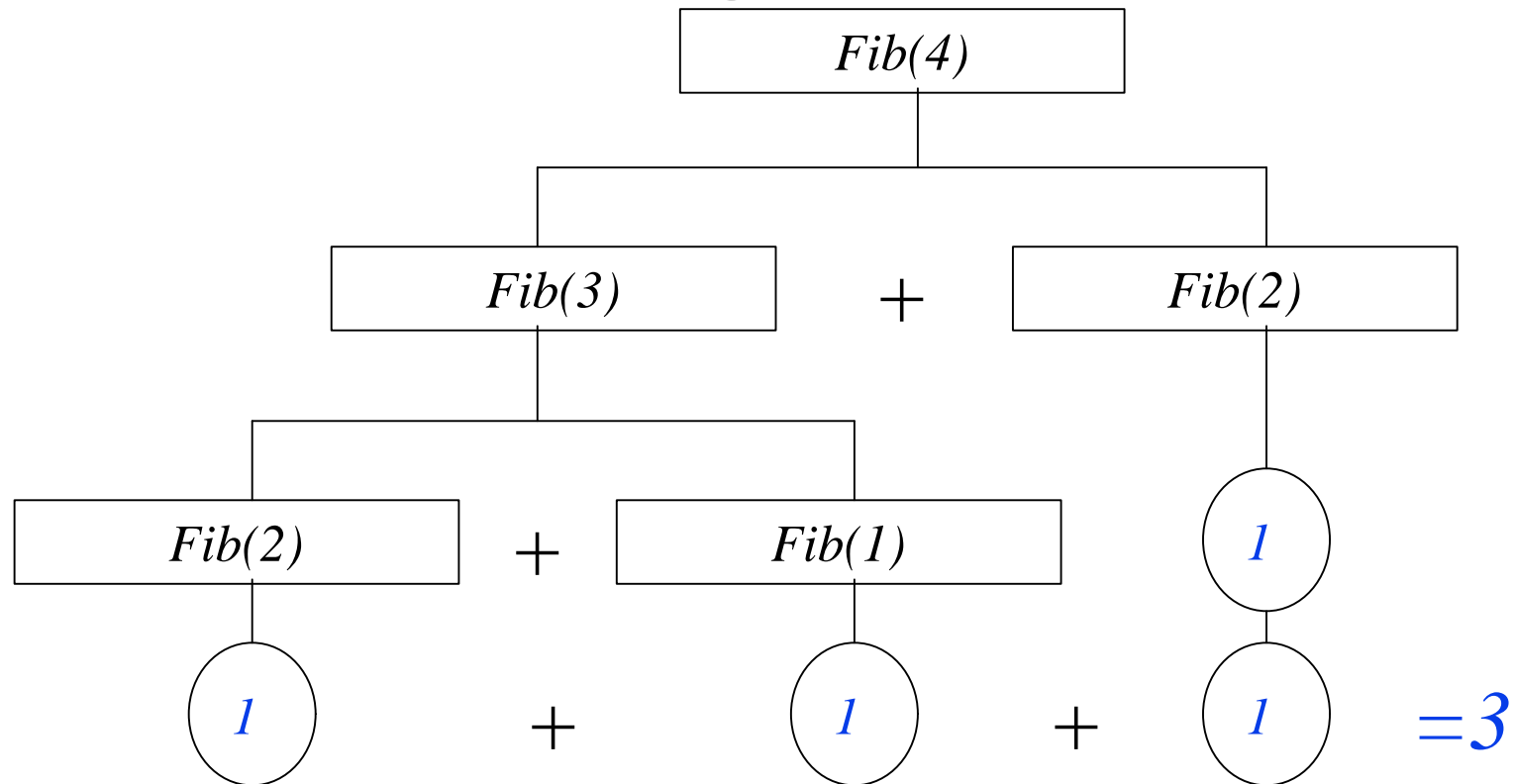
FINSI

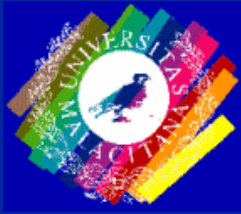
Fin



Ejemplos

- Seguimiento de $Fib(4)$

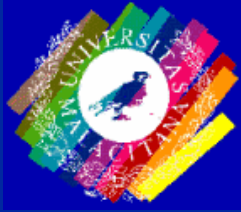




Ejemplos

- Imprimir el equivalente binario de un número decimal

N	N MOD 2	N DIV 2
23	1	11
11	1	5
5	1	2
2	0	1
1	1	0
0		

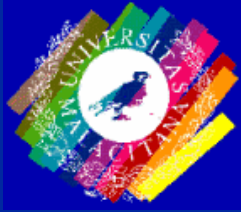


Ejemplos

$$\text{Bin de } N = \begin{cases} N & \text{Si } N < 2 \\ \text{Binaria de } (N \text{ DIV } 2) \parallel (N \text{ MOD } 2) & \end{cases}$$

con \parallel la concatenación

- **Ventaja:** no requiere arrays



Ejemplos

Algoritmo DecimalABinario(num:N)

Inicio

SI num \geq 2 **ENTONCES**

DecimalABinario(num DIV 2)

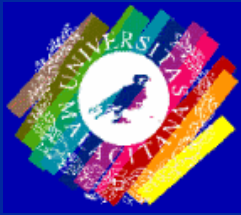
Escribir(num MOD 2)

ENOTROCASO

Escribir (num)

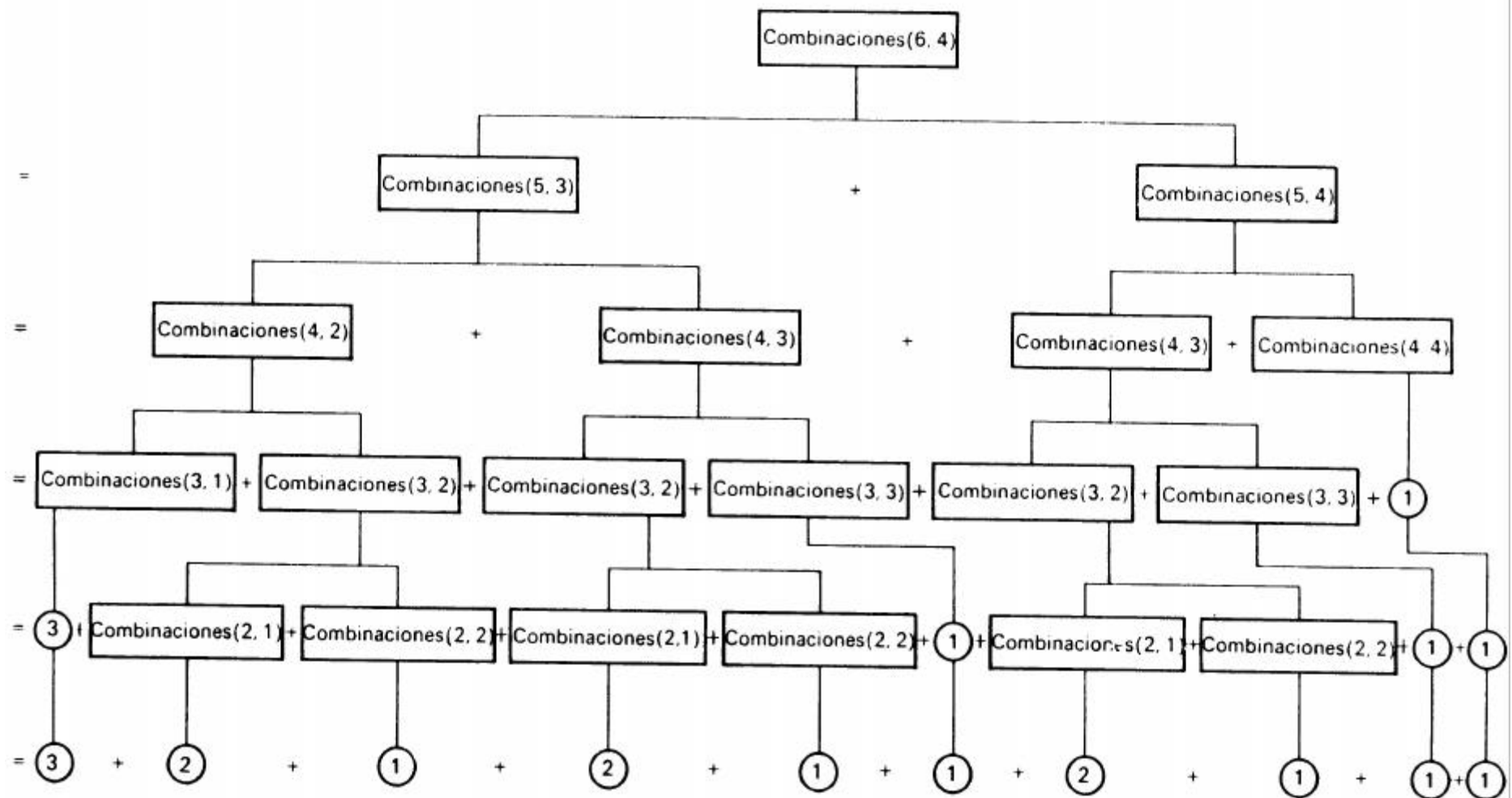
FINSI

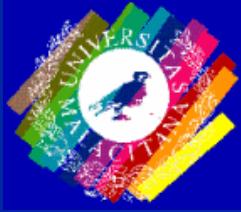
Fin



¿Recursión o iteración?

- Ventajas de la Recursión ya conocidas
 - Soluciones simples, claras.
 - Soluciones elegantes.
 - Soluciones a problemas complejos.
- Desventajas de la Recursión: **EFICIENCIA**
 - Sobrecarga asociada con las llamadas a subalgoritmos
 - Una simple llamada puede generar un gran numero de llamadas recursivas. (Fact(n) genera n llamadas recursivas)
 - ¿La claridad compensa la sobrecarga?
 - El valor de la recursividad reside en el hecho de que se puede usar para resolver problemas sin fácil solución iterativa.
 - La ineficiencia inherente de algunos algoritmos recursivos.





¿Recursión o iteración ?

- A veces, podemos encontrar una solución iterativa simple, que haga que el algoritmo sea más eficiente.

Algoritmo Fib(n:N): N

Variables

R, R1, R2, i : N

Inicio

R1 := 1; R2 := 1; R := 1

PARA i := 3 **HASTA** n **HACER**

 R := R1 + R2

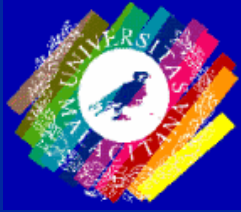
 R2 := R1

 R1 := R

FINPARA

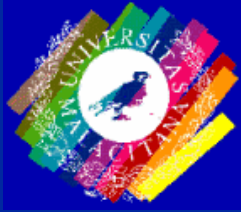
DEVOLVER R

Fin



¿Recursión o iteración?

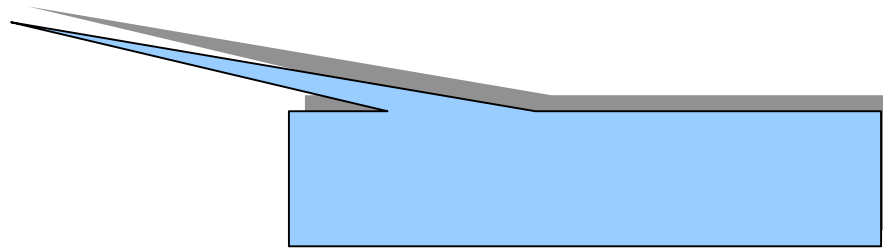
LA RECURSIVIDAD SE DEBE USAR
CUANDO SEA REALMENTE
NECESARIA, ES DECIR, CUANDO NO
EXISTA UNA SOLUCIÓN ITERATIVA



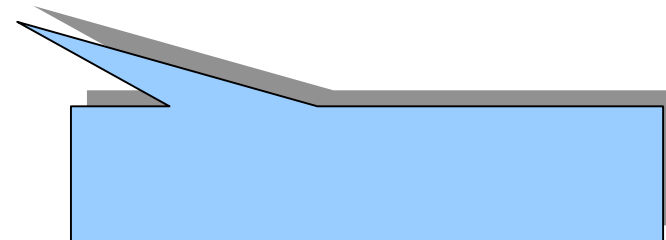
Depuración

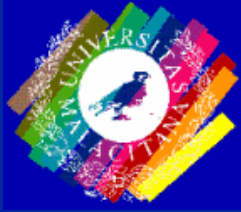
- ERRORES COMUNES

- Tendencia a usar estructuras iterativas en lugar de estructuras selectivas. El algoritmo no se detiene.



- Ausencia de ramas donde el algoritmo trate el caso-base.
- Solución al problema incorrecta





Ejemplos

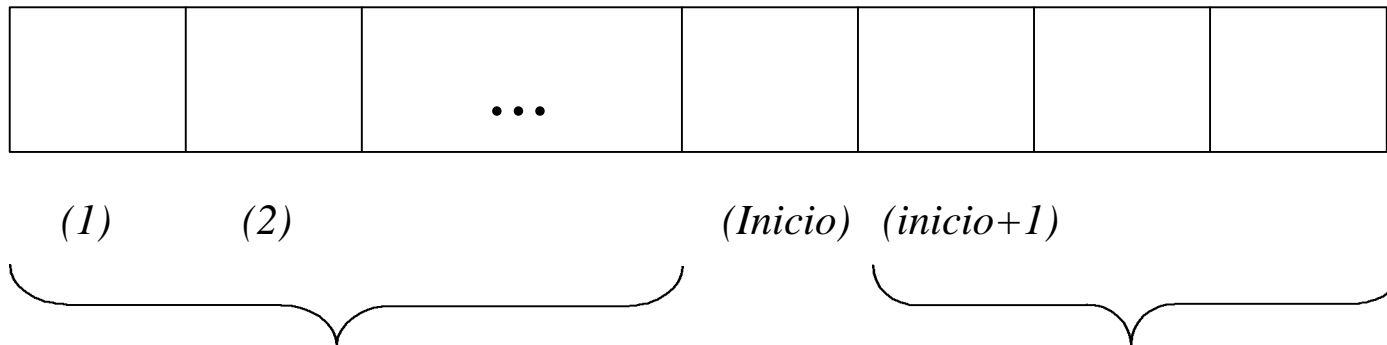
BUSQUEDA EN UN ARRAY

Función ValorEnLista: Buscar el valor **Val** en un array Lista: TLista

Solución recursiva

DEVOLVER (Val en 1ª posición) OR (Val en resto del ARRAY)

Para buscar en el resto del ARRAY, uso la misma función ValorEnLista





Ejemplos

Función ValorEnLista($l:TLista, valor:Tvalor, Inicio, Fin:Z$):B

- **Invocación:**

SI ValorEnLista(Lista,Val,1,MaxLista) ENTONCES....

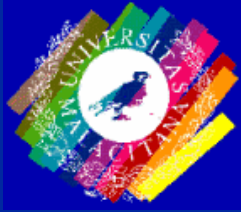
- **Casos Base:**

Lista[Inicio]=Val \Rightarrow DEVOLVER TRUE

Inicio=Fin y Lista[Inicio]<>Val \Rightarrow DEVOLVER FALSE

- **Caso General: buscar en el resto del ARRAY**

DEVOLVER ValorEnLista(Lista,Val,Inicio+1,Fin)



Ejemplos

Algoritmo ValorEnLista(l:TLista,valor:Tvalor,Inicio,Fin:Z):B
(*Busca recursiva en lista de Val dentro del rango del
indice del ARRAY*)

Inicio

SI l[Inicio]=valor **ENTONCES**
DEVOLVER TRUE

ENOTROCASO

SI Inicio=Fin **ENTONCES**
DEVOLVER FALSE

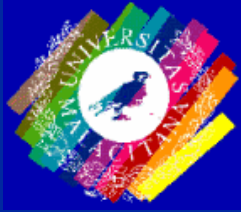
ENOTROCASO

DEVOLVER ValorEnLista(l,valor,Inicio+1,Fin)

FINSI

FINSI

Fin

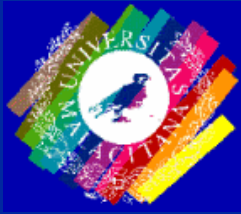


Ejemplos

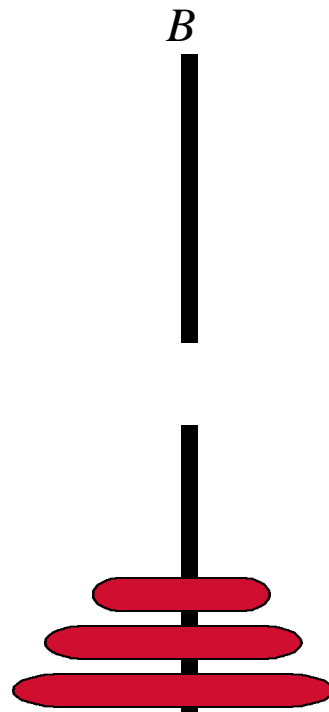
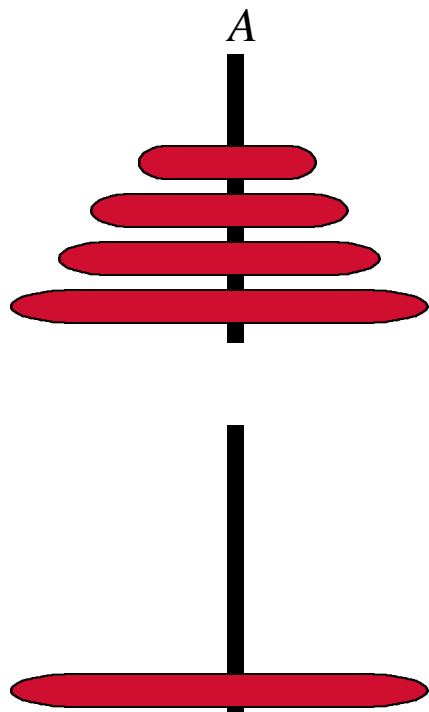
Torres de Hanoi

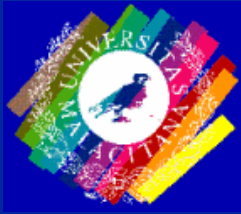
- Se tienen 3 palos de madera, que llamaremos palo **izquierdo, central y derecho**. El palo izquierdo tiene ensartados un montón de discos concéntricos de tamaño decreciente, de manera que el disco mayor está abajo y el menor
- El problema consiste en mover los discos del palo izquierdo al derecho respetando las siguientes reglas:
 - - Sólo se puede mover un disco cada vez.
 - - No se puede poner un disco encima de otro más pequeño.
 - - Después de un movimiento todos los discos han de estar en alguno

N, e imprimir la secuencia de pasos para resolver el



Ejemplos



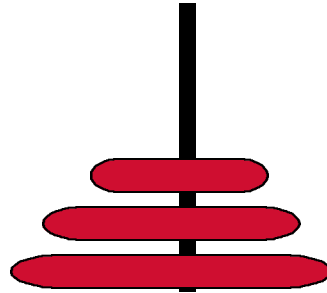


Ejemplos

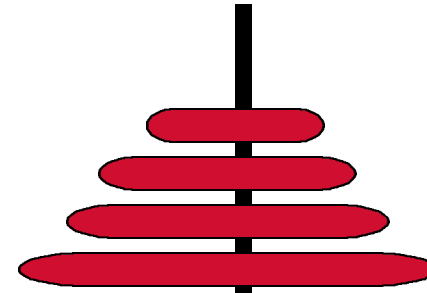
A

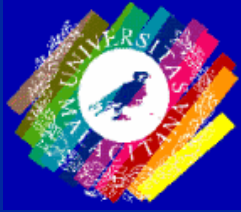


B



C





Ejemplos

- Solución recursiva a las Torres de Hanoi
 - Si $n=1$ mueva el disco de A a C y pare
 - Mueva los $n-1$ discos superiores de A a B, con C
 - Mueva el disco restante de A a C
 - Mueva los $n-1$ discos de B a C, usando A como



Ejemplos

Planteamos un procedimiento recursivo con cuatro parámetros:

- El número de discos a mover.
- El palo origen desde donde moverlos.
- El palo destino hacia el que moverlos.
- El palo auxiliar.

Algoritmo Mueve(N, origen, auxiliar, destino)

Inicio

SI N=1 **ENTONCES**

Mueve un disco del palo origen al destino

ENOTROCASO

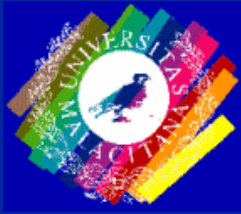
Mueve(N-1, origen, destino, auxiliar)

Mueve un disco del palo origen al destino

Mueve(N-1, auxiliar, origen, destino)

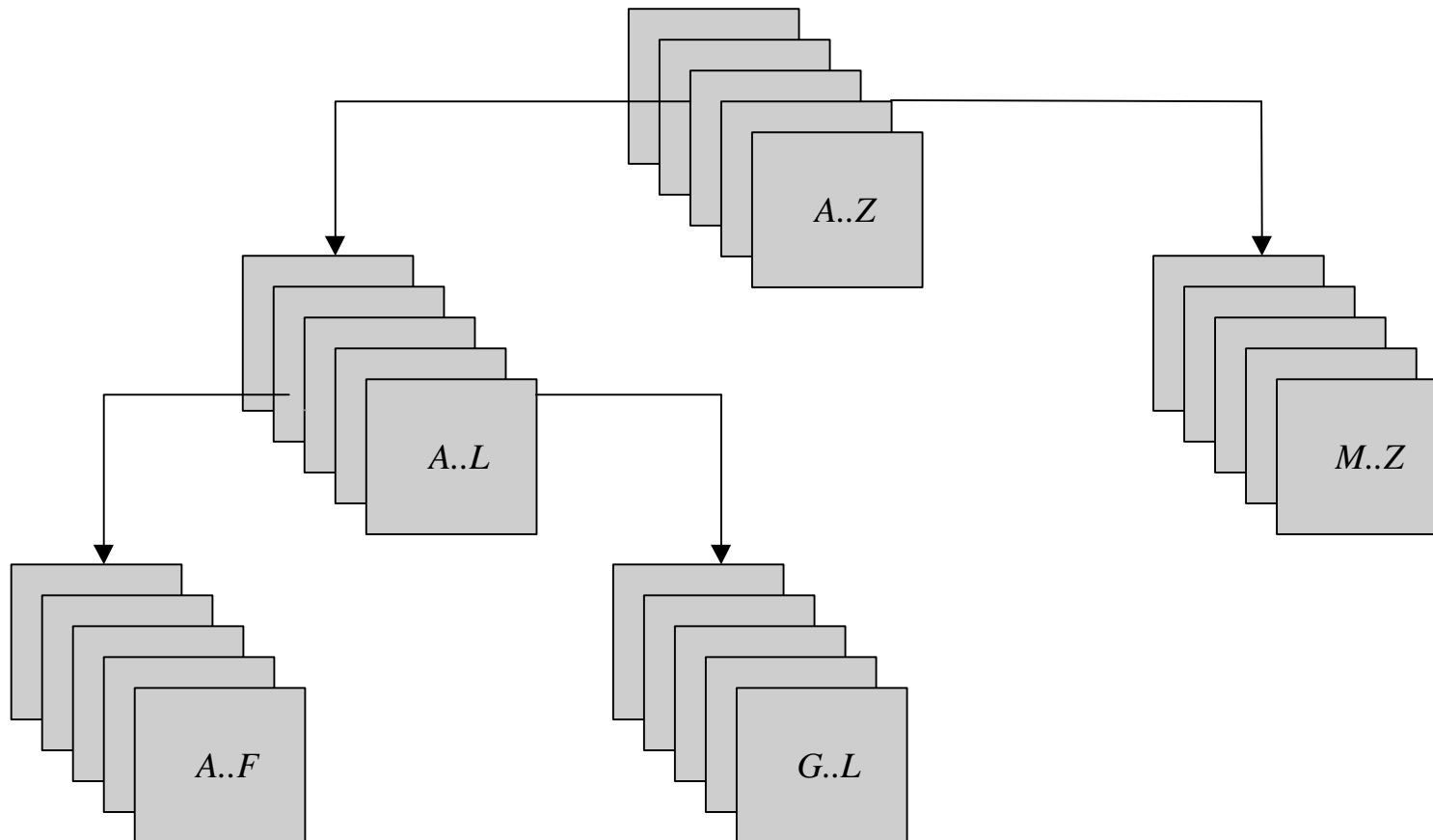
FINSI

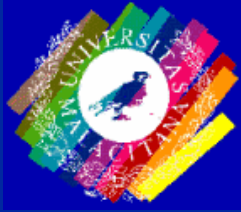
Fin



Ejemplos

ORDENACIÓN RÁPIDA (QUICKSORT)





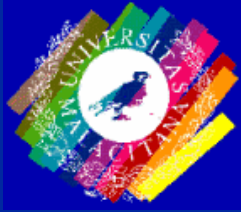
Ejemplos

- Solución recursiva a la ordenación rápida.

V												
---	--	--	--	--	--	--	--	--	--	--	--	--

1 2 3 4 5 6 7 8 9 10 11 12 13

- Qué información es necesaria para abastecer a OrdRápida?
 - Nombre del array
 - su tamaño (primer y último índice)



Ejemplos

- El algoritmo básico **OrdRápida** es:

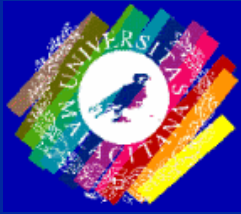
SI NOT terminado **ENTONCES**

Dividir el array por un valor V (Pivote)

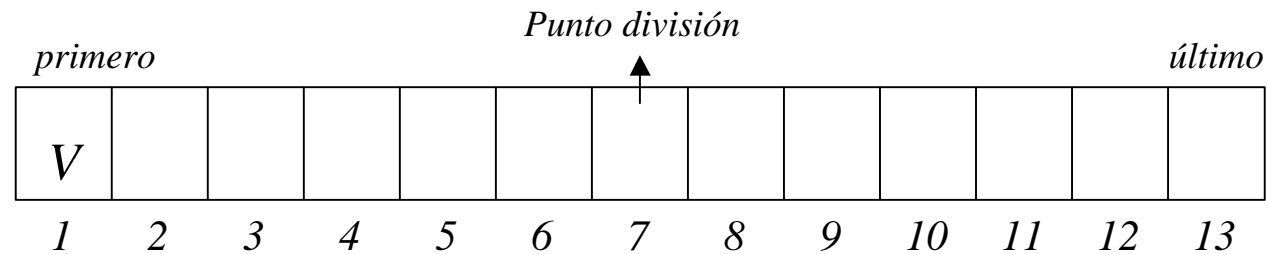
OrdRápida los elementos menores ó iguales

OrdRápida los elementos mayores que V

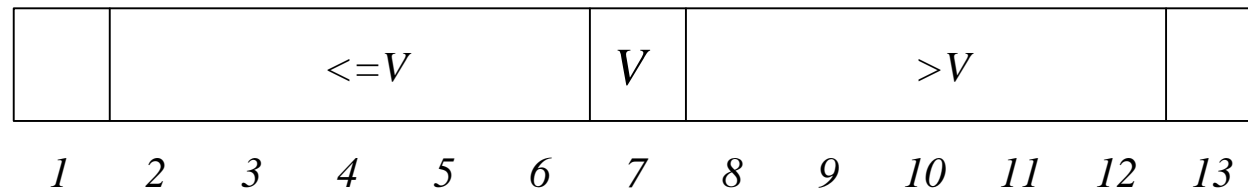
- **Algoritmo** OrdRápida(**VAR** Datos : Tarray ,
Primero, Ultimo: **N**)
- La llamada sería OrdRápida (Datos, 1, n)

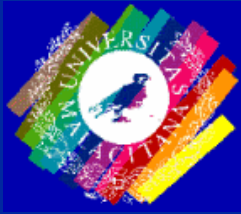


Ejemplos



- Usamos el valor de **Datos[1]** como pivote.
- **Subalgoritmo Dividir.**





Ejemplos

OrdRápida(Datos, Primero, PuntoDivisión-1)

OrdRápida(Datos, PuntoDivisión+1, Ultimo)

- ¿Cual es el caso base?
 - Si el segmento de array tiene menos de dos elementos: **SI** Primero \geq Ultimo

Algoritmo OrdRápida(**VAR** Datos:Tarray; Primero, Ultimo:N);

Variables PuntoDivision:N;

Inicio

SI Primero < Ultimo **ENTONCES**

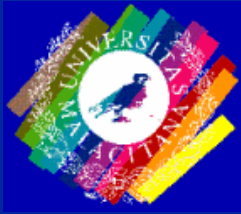
Dividir(Datos, Primero, Ultimo, PuntoDivision)

OrdRápida(Datos, Primero, -1)

OrdRápida(Datos, PuntoDivision+1, Ultimo)

FINSI

Fin



Ejemplos

a) Inicialización $V = \text{Datos}[1] = 9$

9	20	6	10	14	8	60	11
---	----	---	----	----	---	----	----

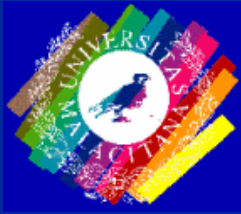
b) Mover Izq a la derecha hasta que $\text{Datos}[\text{Izq}] > V$

9	20	6	10	14	8	60	11
---	----	---	----	----	---	----	----

Izq Dcha
a la izquierda hasta que $\text{Datos}[\text{Izq}] > V$

9	20	6	10	14	8	60	11
---	----	---	----	----	---	----	----

Izq Dcha



Ejemplos

d) Intercambiar Datos[Izq] y Datos[Dcha], y mover Izq y Dcha

9	8	6	10	14	20	60	11
<i>Izq</i>			<i>Dcha</i>				

e) Mover Izq hasta que Datos[Izq]>V o Dcha<Izq

Mover Dcha hasta que Datos[Dcha]<=V o Dcha<Izq

9	8	6	10	14	20	60	11
<i>Dcha</i>			<i>Izq</i>				

f) Izq>Dcha, por tanto no ocurre ningún intercambio dentro del bucle . Intercambiamos
]

6	8	9	10	14	20	60	11
---	---	---	----	----	----	----	----

PuntoDivisión

```
Algoritmo Dividir(VAR Datos:Tarray,Primero,Ultimo:Z;VAR Pdivision:Z)
    Izq,
```

```
    V:=Datos[Primero]
    Izq
```

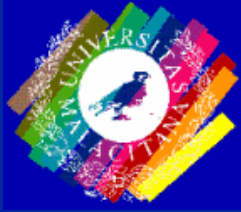
```
        Izq<=                                Izq
        Izq:=Izq+1
```

```
        Izq<=                                )and
        :=                                -1
```

```
    SI Izq<
        Intercambiar(Datos[Izq                                ])
        Izq:=Izq+1
        :=                                -1
```

```
    Izq>
    Intercambiar(Datos[Primero],Datos[                                ])
```

```
Fin
```



Bibliografía

- **Pascal.** Dale/Orshalick. Ed McGraw Hill 1986.
- **Pascal y Estructuras de Datos.** Nell Dale, Susan C. Lilly. McGraw Hill. 1989.
- **Fundamentos de programación.** Joyanes Aguilar. McGraw Hill. 1988
- **Introduction to programming with** . Saim Ural/Suzan Ural. Wiley. 1987.
- **Estructuras de datos en Pascal.** Aaron Tenenbaum. Prentice Hall. 1983