Tema 3. Recursividad.

Diseño de Algoritmos.

E.U. Politécnica
Departamento Lenguajes y Ciencias de la Computación.
Universidad de Málaga
José Luis Leiva Olivencia.
Despacho: I-326D (Edificio E.U.P)/ 3.2.41 (Teatinos-E.T.S.I.I.)

Introducción

- Definición de Recursividad: Técnica de programación muy potente que puede ser usada en lugar de la iteración, consistente en la invocación de un algoritmo a sí mismo.
- 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 más fáciles de resolver que con 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 subalgoritmo se invoca al propio subalgoritmo para resolver "una versión más pequeña" del problema original.
- Aspecto de un subalgoritmo recursivo.

```
tipo Recursivo(...);
{
   ...
   Recursivo(...);
   ...
}
```


Introducción

- Ejemplo: Factorial de un natural.

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

```
int Factorial(int n)
  {
    if (n==0) return 1;
    else
    return (n*Factorial(n-1));
  }
```

Introducción

- El método de definición de una función en términos de sí misma se llama en matemáticas a una definición **inductiva** y conduce naturalmente a una implementación recursiva.

- El caso base de 0!=1 es esencial dado que se detiene, potencialmente, una cadena de llamadas recursivas (*Caso Base o condición de salida)*

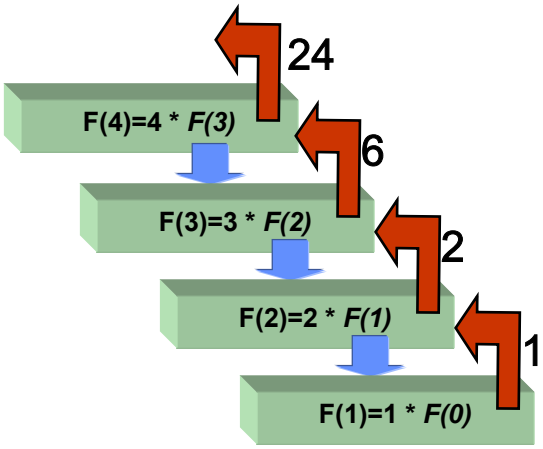
Diseño de Algoritmos.
E.U.Politécnica. I.T.Industrial (Electricidad)

Introducción

- Tipos de recursividad:
  - Si una función o procedimiento se invoca a sí misma, el proceso se denomina *recursión directa*; si una función o procedimiento puede invocar a una segunda función o procedimiento que a su vez invoca a la primera este proceso se conoce como *recursión indirecta o mutua*.

Departamento de Lenguajes y Ciencias de la Computación

Introducción



Introducción

Registro de Activación. bloque de memoria que contiene la información sobre las constantes y variables declaradas en el procedimiento, junto con una dirección de retorno, dirección en memoria de la sentencia que causó la llamada al subalgoritmo.

Dirección de Retorno.

Pila (Stack) Forma especial de organizar la memoria, en la que la información siempre se añade y se suprime por una posición llamada cima .

Vinculación(dirección memoria ←→ variable).

Asignación estática y dinámica de memoria

- Partimos del siguiente ejemplo

```
void uno (int x,int y)
   {
   int z;
      ...
      ...
   }
```


Diseño de Algoritmos.
E.U.Politécnica. I.T.Industrial (Electricidad)

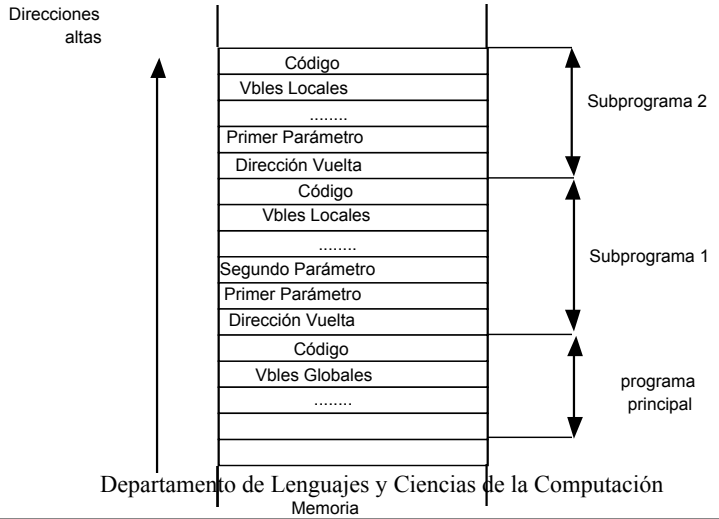
Asignación estática y dinámica de memoria

• 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.
- En este caso: x ⟷ 0100
  y ⟷ 0101
  z ⟷ 0110
- La zona reservada para variables locales y parámetros formales usualmente preceden al código del subprograma

Departamento de Lenguajes y Ciencias de la Computación

Asignación estática y dinámica de memoria



Diseño de Algoritmos.
E.U.Politécnica. I.T.Industrial (Electricidad)

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 (Sobreescritura)

Asignación estática y dinámica de memoria

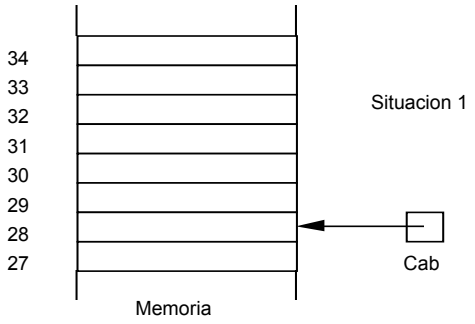
• Asignación dinámica

- Asignación de cada variable, parámetro relativa a una posición (CAB)
  - En este caso: x → 1
    y → 2
    z → 3
  - Dirección de retorno→0

Diseño de Algoritmos.
E.U.Politécnica. I.T.Industrial (Electricidad)

Asignación estática y dinámica de memoria

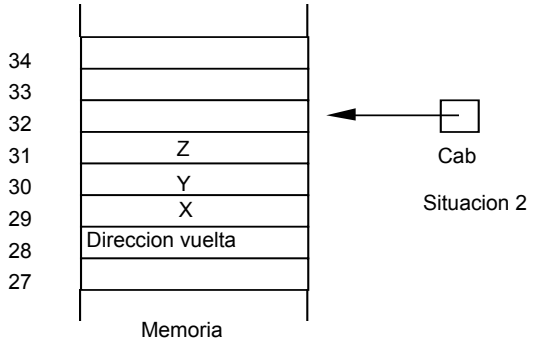
Tiempo de ejecución:Se reserva espacio para las variables y parámetros a partir de la situación actual de CAB



Departamento de Lenguajes y Ciencias de la Computación

Asignación estática y dinámica de memoria

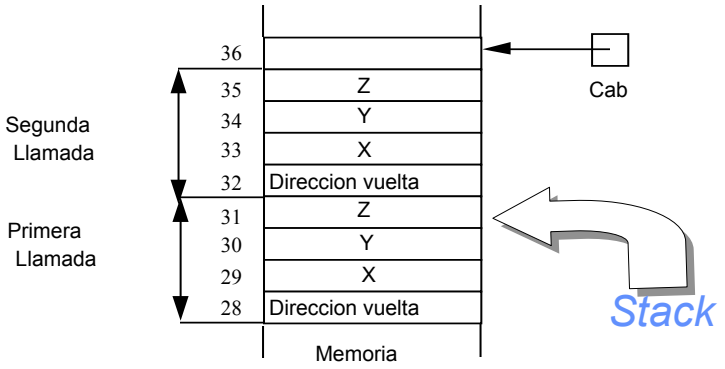
- Llamada al subalgoritmo



Diseño de Algoritmos.
E.U.Politécnica. I.T.Industrial (Electricidad)

Asignación estática y dinámica de memoria

- Llamada recursiva al algoritmo



Asignación estática y dinámica de memoria

- Ejemplo con la función factorial

```
int Factorial(int n)
{
    if (n==0) return 1;
    else return(n*Factorial(n-1);)
}
```

La invocación inicial es:

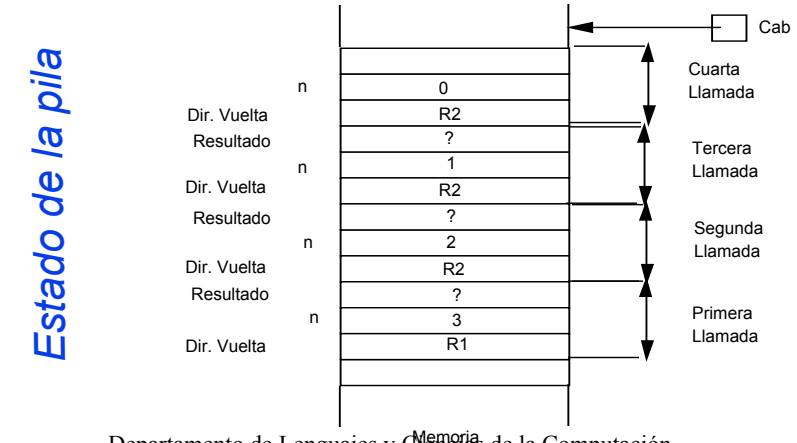
```
Resultado = Factorial(3)
```

R2

R1

Diseño de Algoritmos.
E.U.Politécnica. I.T.Industrial (Electricidad)

Asignación estática y dinámica de memoria



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 el caso base.
- Ocultación de los detalles de gestión de la memoria en las llamadas recursivas **(Pila interna).**

Verificación de funciones y procedimientos recursivos

Método inductivo de las tres preguntas

- La pregunta Caso-Base: ¿Existe al menos una salida no recursiva o casos bases 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?.¿No hay recursión infinita?
- 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 resolver → Parámetros en la llamada inicial
- Resolver el caso general en términos de un caso más pequeño (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
  - Grupo>=Miembros

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

Ejemplos

- FUNCIÓN COMBINACIONES

- Definición:Calcular cuantas combinaciones de tamaño Miembros pueden hacerse del tamaño total Grupo
- Tamaño:Número de elementos en la llamada original
- Casos-base:1)Miembros=1 ⇨ Combinaciones=Grupo
  2)Miembros=Grupo ⇨ Combinaciones=1
- Caso General:Grupo>Miembros>1

⇩

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

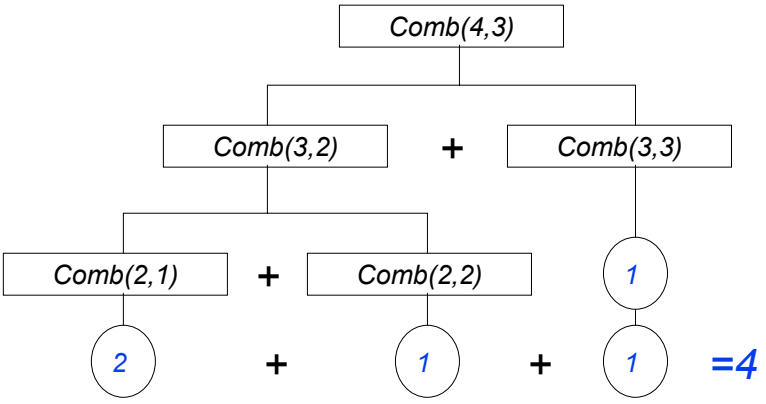
Ejemplos

```
int Comb(int Grupo,int Miembros)
{
   if (Miembros==1)
      return Grupo;/*Caso Base 1*/
   else
      if (Miembros==Grupo) ENTONCES
           return 1; /*Caso Base 2*/
      else (*Caso General*)
           return (Comb(Grupo-1,Miembros-1)
                     +Comb(Grupo-1,Miembros));
}
```

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

Ejemplos

- Seguimiento de Comb(4,3)



Ejemplos

FUNCIÓN FIBONACCI

- Definiciones:Calcular el valor de la función de Fibonacci para un número **n** dado.
- Tamaño:Número n de la llamada original
- Casos-base:n<=2 ⇒ fib=1
- Caso General:n>2 ⇒ fib(n)=fib(n-1)+fib(n-2)

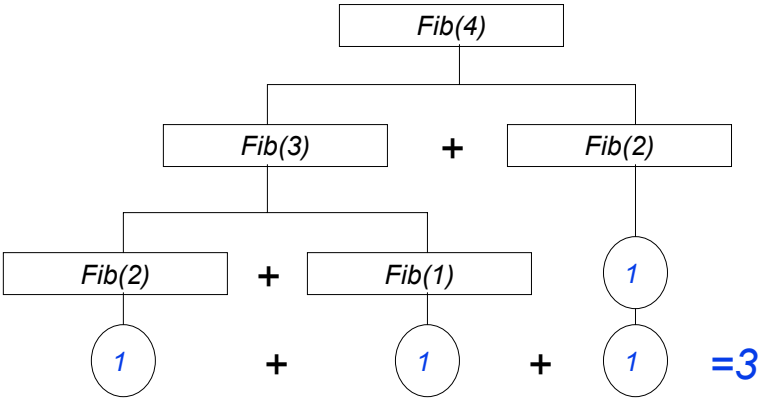
Ejemplos

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


Diseño de Algoritmos.
E.U.Politécnica. I.T.Industrial (Electricidad)

Ejemplos

- Seguimiento de Fib(4)



Ejemplos

- Imprimir el equivalente binario de un número decimal

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

Ejemplos

Bin de N= { N Si N<2 ; Binaria de (N / 2)||(N MOD 2)

con || la concatenación

- Ventaja: no requiere arrays

Ejemplos

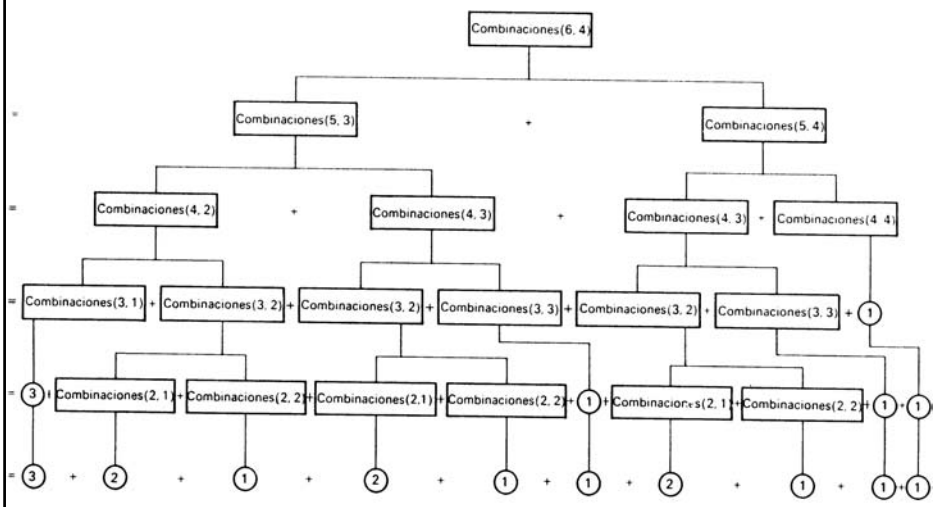
```
void DecimalAbinario(int num)
{
  if (num>=2)
  {
     DecimalABinario(num/2);
     printf("%d",num%2);
  }
  else
     printf("%d",num);
}
```


¿Recursión o iteración?

- Se puede utilizar la recursividad como una alternativa a la iteración.
- Una solución recursiva es normalmente menos eficiente en términos de tiempo de computadora que una solución iterativa debido a las operaciones auxiliares que llevan consigo las llamadas suplementarias a las funciones.
- La solución recursiva muchas veces permite a los programadores especificar soluciones naturales, que en otros casos sería difícil de resolver.

¿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.

```
int Fib(int n)
{
   int R, R1, R2, i;
            R1 ← 1;
            R2 ← 1;
            R ← 1
            for (i=3;i<=n;i++)
              {   R ← R1 + R2
                  R2 ← R1
                  R1 ← R
               }
                  return r;
}
```


¿Recursión o iteración?

LA RECURSIVIDAD SE DEBE USAR CUANDO SEA REALMENTE NECESARIA, ES DECIR, CUANDO NO EXISTA UNA SOLUCIÓN ITERATIVA SIMPLE. CUANDO LAS DOS SOLUCIONES SON FACILMENTE EXPRESABLES, SIEMPRE SOLUCIÓN ITERATIVA.

¿Recursión o iteración?

Evitar la utilización de la recursividad en situaciones de rendimiento crítico o exigencia de altas prestaciones en tiempo y memoria, ya que las llamadas recursivas emplean tiempo y consumen memoria adicional.

Depuración

- ERRORES COMUNES

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

Comprobar el uso de SI o CASO

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

Seguir el método de las 3 preguntas

Ejemplos

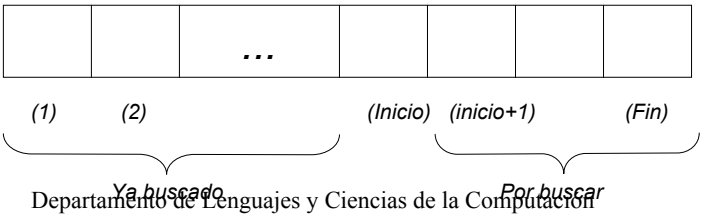
BUSQUEDA EN UN ARRAY

Función ValorEnLista:Buscar el valor Val en un arrayLista:TLista

Solución recursiva

RESULTADO ← (Val en 1ª posición) ∨ (Val en resto del ARRAY)

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



Diseño de Algoritmos.
E.U.Politécnica. I.T.Industrial (Electricidad)

Ejemplos

Función /*1 si lo encuentra 0 en caso contrario*/

```
int ValorEnLista(TLista l,Tvalor valor,int Inicio, int Fin);
```

Invocación:

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

- Casos Base:

```
   Lista[Inicio]==Val  ➡   return 1;
(Inicio==Fin) && (Lista[Inicio] != Val ➡ return 0;
```

- Caso General:buscar en el resto del ARRAY

```
   return (ValorEnLista(Lista,Val,Inicio+1,Fin));
```

Ejemplos

```
int ValorEnLista(TLista l,Tvalor valor,int Inicio, int
  Fin) (*Busca recursiva en lista de Valor
  dentro del rango del indice del ARRAY*)
{
  if (l[Inicio]==valor)
    return 1;
  else
    if(Inicio==Fin)
        return 0;
    else
        return (ValorEnLista(l,valor,Inicio+1,Fin));
}
```

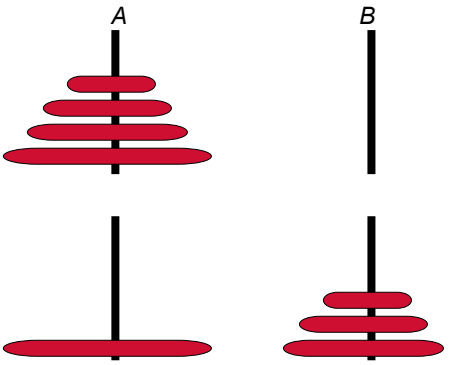

Ejemplos

Torres de Hanoi

- Algoritmo clásico cuyo origen oriental versa sobre una leyenda sobre el Templo de Brahma.
- 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 arriba.
- 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 de los tres palos.

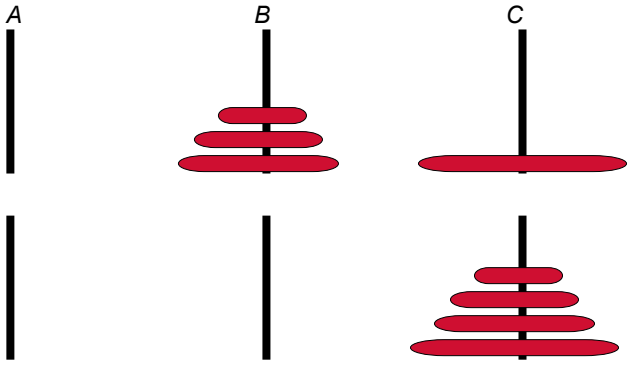
Leer por teclado un valor N, e imprimir la secuencia de pasos para resolver el problema.

Ejemplos



Diseño de Algoritmos.
E.U.Politécnica. I.T.Industrial (Electricidad)

Ejemplos



Departamento de Lenguajes y Ciencias de la Computación

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 auxiliar
  - Mueva el disco restante de A a C
  - Mueva los n-1 discos de B a C, usando A como auxiliar

Diseño de Algoritmos.
E.U.Politécnica. I.T.Industrial (Electricidad)

Ejemplos

Planteamos un procedimiento recursivo con cuatro parámetros:

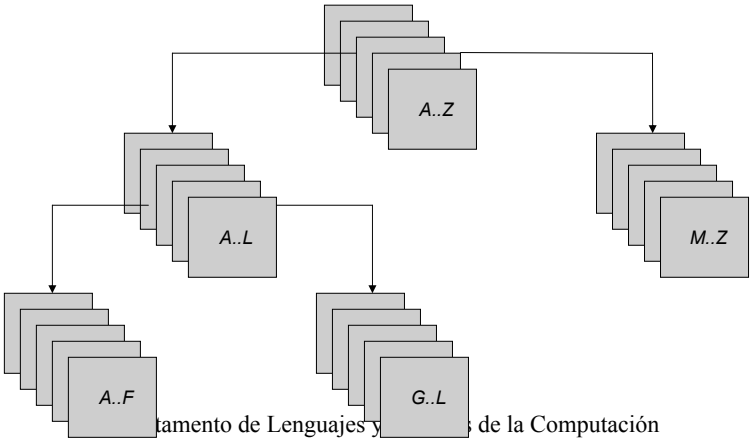
- El número de discos a mover.
- El palo origen desde donde moverlos.
- El palo destino hacia el que moverlos.
- El palo auxiliar.

```
void Mueve(int N,int origen,int auxiliar,int destino)
{
    if (N==1)
          printf("Palo %d a palo %d\n",origen,destino)
    else
    {
          Mueve(N-1,origen,destino,auxiliar)
          printf("Palo %d a palo %d\n",origen,destino)
          Mueve(N-1,auxiliar,origen,destino)
    }
}
```

Departamento de Lenguajes y Ciencias de la Computación

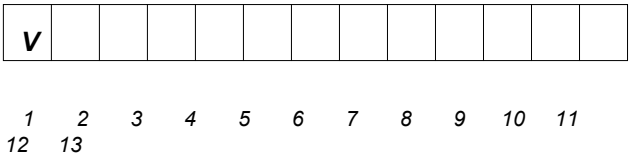
Ejemplos

ORDENACIÓN RÁPIDA (QUICKSORT)



Ejemplos

- Solución recursiva a la ordenación rápida.



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

Ejemplos

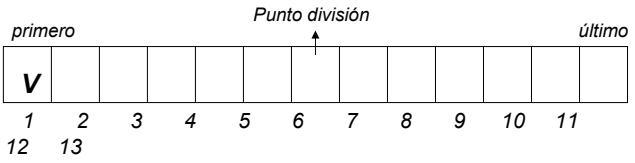
- El algoritmo básico OrdRápida es:

```
SI ¬ terminado ENTONCES
   Dividir el array por un valor V (Pivote)
   OrdRápida los elementos menores ó iguales que V
   OrdRápida los elementos mayores que V
```

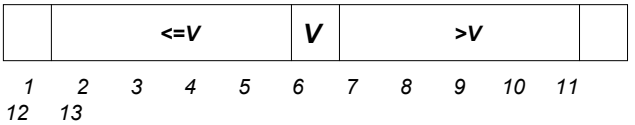
- **PROC**
  `OrdRápida(↓↑Datos:Tarray,↓Primero,↓Ultimo:NATURAL)`

  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>=Ultimo
  - **PROC** OrdRápida(↓↑Datos:Tarray;↓Primero,↓Ultimo:**NATURAL**)

```
Variables PuntoDivision:NATURAL
Inicio
SI Primero<Ultimo ENTONCES
  Dividir(Datos,Primero,Ultimo,PuntoDivision)
  OrdRápida(Datos,Primero,PuntoDivision-1)
  OrdRápida(Datos,PuntoDivision+1,Ultimo)
FINSI
Fin
```


Ejemplos

a) Inicialización `V=Datos[1] =9`

9	20	6	10	14	8	60	11

Izq *Dcha*

b) Mover `Izq` a la derecha hasta que `Datos[Izq]>V`

9	20	6	10	14	8	60	11

Izq *Dcha*

c) Mover `Dcha` a la izquierda hasta que `Datos[Dcha]<=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

Izq *Dcha*

e) Mover `Izq` hasta que `Datos[Izq]>V` ó `Dcha<Izq`
Mover `Dcha` hasta que `Datos[Dcha]<=V` ó `Dcha<Izq`

9	8	6	10	14	20	60	11

Dcha *Izq*

f) `Izq>Dcha`, por tanto no ocurre ningún intercambio dentro del bucle . Intercambiamos `Datos[1]` con Datos[Dcha]

6	8	9	10	14	20	60	11

PuntoDivisión


```
PROC Dividir(↓↑ Datos:Tarray;↓Primero,↓Ultimo:ENTERO;
               ↓↑Pdivision:ENTERO)
VARIABLES Izq,Dcha,V:ENTERO
Inicio
        V←Datos[Primero]
        Izq←Primero+1
        Dcha←Ultimo
        REPETIR
                MIENTRAS(Izq<=Dcha) ∧ (Datos[Izq]<=V) HACER
                        Izq←Izq+1
                FINMIENTRAS
                MIENTRAS (Izq<=Dcha) ∧ (Datos[Dcha]>V) HACER
                        Dcha←Dcha-1
                FINMIENTRAS
                SI Izq<Dcha ENTONCES
                        Intercambiar(Datos[Izq],Datos[Dcha])
                        Izq←Izq+1
                        Dcha←Dcha-1
                FINSI
        HASTA (Izq>Dcha)
        Intercambiar(Datos[Primero],Datos[Dcha])
        Pdivision←Dcha
Fin
```