

Programación

Recursividad & Ordenamiento

Dr. Wenceslao Palma
wenceslao.palma@ucv.cl



- Las funciones en lenguaje C se pueden emplear recursivamente, es decir, una función puede llamarse a sí misma ya sea directa o indirectamente.
- La recursividad es una técnica del tipo "dividir y conquistar" en la cual el problema a resolver se expresa (define) en términos de pequeños problemas (subproblemas) del mismo tipo.
- Ejemplo: el factorial de un número entero. Se sabe que:

$$\begin{aligned}0! &= 1 \\1! &= 1 \\2! &= 2 \times 1 \\3! &= 3 \times 2 \times 1 \\4! &= 4 \times 3 \times 2 \times 1 \\5! &= 5 \times 4 \times 3 \times 2 \times 1 \\&\dots\dots\dots\end{aligned}$$

De este modo podemos definir una función no recursiva de la siguiente manera:

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n - 1) \times (n - 2) \times \dots \times 3 \times 2 \times 1 & \text{if } n \geq 1 \end{cases}$$

y podemos escribir una función que nos permita calcular $n!$:

```
int factorial(int n){
    int i, fact=1;

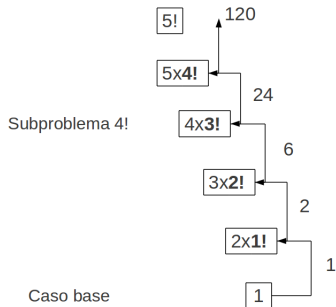
    for(i=1;i<=n;i++)
        fact *= i;

    return fact;
}
```

Sin embargo, $n!$ puede ser definido en forma recursiva:

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

$n!$ es expresado en función del subproblema $(n - 1)!$, y éste último en función del subproblema $(n - 2)!$



Ahora, podemos escribir una función recursiva en lenguaje C:

```
int factorial(int n){
    int fact;

    if (n==0)
        return 1;

    fact = n*factorial(n-1);
    return fact;
}
```

La recursividad no proporciona un ahorro en almacenamiento ya que se debe mantener una pila de los valores procesados. Ni será más rápida. Pero el código recursivo es más compacto y frecuentemente más fácil de escribir y de entender que su equivalente no recursivo.

En la secuencia de Fibonacci 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, cada elemento es la suma de los dos precedentes. Si definimos $fib(0)=0$, $fib(1)=1$ y así sucesivamente, entonces puede definirse la secuencia de Fibonacci en forma recursiva:

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

la función en lenguaje C:

```
int fibonacci(int n){
    int fib;

    if ((n==0) || (n==1))
        return n;

    fib = fibonacci(n-2)+fibonacci(n-1);

    return fib;
}
```

Proponga un método iterativo para calcular $fib(n)$ y compárelo con el método recursivo usando $fib(6)$.

- Insertion sort
- Quicksort

Insertion sort:

- va tomando cada uno de los elementos del conjunto de datos a ordenar y los deja (inserta) en la posición correcta.
- En un ordenamiento ascendente, para insertar un elemento k se realiza un corrimiento hacia la derecha de todos los elementos que son mayores o iguales a k .
- Si la secuencia es $1 - 9 - 4 - 8 - 7 - 6 - 12 - 13 - 15 - 11$:

1-9-4-8-7-6-12-13-15-11

1-4-9-8-7-6-12-13-15-11


1-4-8-9-7-6-12-13-15-11

1-4-7-8-9-6-12-13-15-11

1-4-6-7-8-9-12-13-15-11

1-4-6-7-8-9-11-12-13-15

INSERTION SORT		
Best	Average	Worst
$O(n)$	$O(n^2)$	$O(n^2)$

 Array

sort (A)
 1. **for** $i = 1$ **to** $n - 1$ **do**
 2. insert (A, i, A[i])
end

insert (A, pos, value)
 1. $i = pos - 1$
 2. **while** ($i \geq 0$ **and** $A[i] > value$) **do**
 3. $A[i + 1] = A[i]$
 4. $i = i - 1$
 5. $A[i + 1] = value$
end

insert (A, 6, "7")

Already sorted

value

Insert into proper spot

Elements compared and bumped up

Sorted region extended by one




Otro ejemplo:

15	09	08	01	04	11	07	12	13	06	05	03	16	02	10	14
09	15	08	01	04	11	07	12	13	06	05	03	16	02	10	14
08	09	15	01	04	11	07	12	13	06	05	03	16	02	10	14
01	08	09	15	04	11	07	12	13	06	05	03	16	02	10	14
01	04	08	09	15	11	07	12	13	06	05	03	16	02	10	14
01	04	08	09	11	15	07	12	13	06	05	03	16	02	10	14
01	04	07	08	09	11	15	12	13	06	05	03	16	02	10	14
01	04	07	08	09	11	12	15	13	06	05	03	16	02	10	14
01	04	07	08	09	11	12	13	15	06	05	03	16	02	10	14
01	04	06	07	08	09	11	12	13	15	05	03	16	02	10	14
01	04	05	06	07	08	09	11	12	13	15	03	16	02	10	14
01	03	04	05	06	07	08	09	11	12	13	15	16	02	10	14
01	03	04	05	06	07	08	09	11	12	13	15	16	02	10	14
01	02	03	04	05	06	07	08	09	11	12	13	15	16	10	14
01	02	03	04	05	06	07	08	09	10	11	12	13	15	16	14
01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16

- El mejor caso se presenta cuando el conjunto de datos ya se encuentra ordenado. $O(n)$
- El peor caso se presenta cuando el conjunto de datos se encuentra ordenado en orden inverso. $O(n^2)$

Quicksort:

- utiliza un elemento (pivote) para particionar el conjunto de elementos a ordenar en dos subconjuntos.
- un subconjunto contiene los elementos menores al pivote y el otro contiene los elementos mayores al pivote.
- luego cada subconjunto es ordenado en forma recursiva.

QUICKSORT			 Recursion  Divide and Conquer
Best	Average	Worst	
$O(n \log n)$	$O(n \log n)$	$O(n^2)$	 Array

sort (A)

- quickSort (A, 0, n - 1)

end

quickSort (A, left, right)

- if** (left < right) **then**
- pi = partition (A, left, right)
- quickSort (A, left, pi - 1)
- quickSort (A, pi + 1, right)

end

partition (A, left, right)

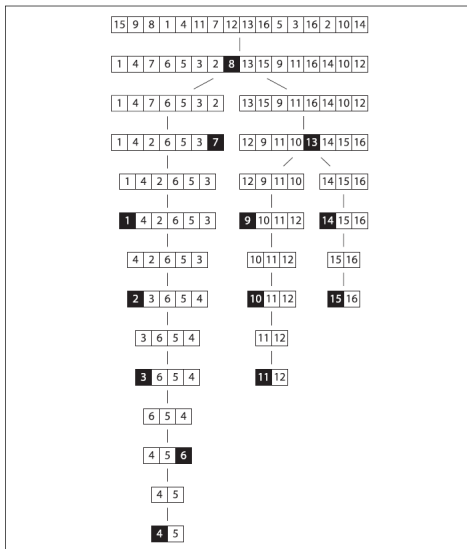
- p = select pivot in A[left, right]
- swap A[p] and A[right]
- store = left
- for** i = left **to** right - 1 **do**
- if** (A[i] ≤ A[right]) **then**
- swap A[i] and A[store]
- store++
- swap A[store] and A[right]
- return** store

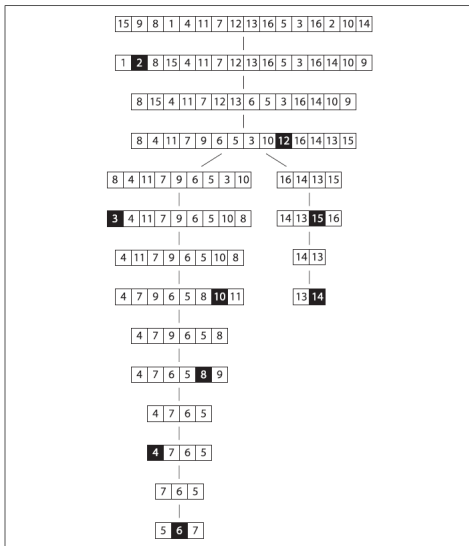
end

The diagram illustrates the partitioning process in quicksort. It shows an array A with elements [6, 5, 3, 1, 4, 2, 7]. The pivot 'pi' is 3. Elements less than or equal to 3 are moved to the left, and elements greater than 3 are moved to the right. The final sorted array is [1, 2, 3, 6, 4, 5, 7].

The partitioning process is shown in several steps:

- Initial array: [6, 5, 3, 1, 4, 2, 7]. Pivot 'p' is 3. 'store' is at index 0.
- Swap A[p] and A[right]: [6, 5, 7, 1, 4, 2, 3].
- store = left = 0.
- for i = left to right - 1 do
- if (A[i] ≤ A[right]) then
- swap A[i] and A[store]: i = 3, swap A[3] and A[0] → [1, 5, 7, 6, 4, 2, 3].
- swap A[i] and A[store]: i = 5, swap A[5] and A[0] → [1, 2, 7, 6, 4, 5, 3].
- store++ → 1.
- swap A[store] and A[right]: swap A[1] and A[6] → [1, 2, 3, 6, 4, 5, 7].
- return store = 1.





Quicksort en lenguaje C:

```
#include <stdlib.h>
void qsort( void *buf, size_t num, size_t size, int (*cmpInteger)(const void *, const void *) )
```

Un ejemplo:

```
#include <stdlib.h>

main(){
    int a[20]={2,1,0,23,12,45,30,7,10,29,9,5,40,38,33,20,11,60,55,4};

    int i=0;

    qsort(a,20,sizeof(int),(void *)cmpInteger);

    for(i=0;i<20;i++)
        printf("%d ",a[i]);
    printf("\n");
}

int cmpInteger(const int *v1, const int *v2){
    return (*v1 - *v2);
}
```