

Estructuras de Datos

Profesor
Sergio Gonzalez



UNIVERSIDAD
NACIONAL DE
HURLINGHAM

Unidad 3: Algoritmos de búsqueda y ordenamiento

Profesor

Sergio Gonzalez

Dividir y conquistar

- Diseño algorítmico
 1. Dividir el problema en k subproblemas del mismo tipo, pero mas chicos
 2. Resolver subproblemas (se puede seguir subdividiendo hasta casos base)
 3. Combinar las soluciones para resolver el problema original

Búsqueda

- Encontrar un elemento en un vector
- Si existe, obtener posición
- Vector ordenado???

Búsqueda secuencial

- Fuerza bruta (trivial)
- Recorrer vector desde el principio al final
- Comparar uno por uno
- Si se encuentra el valor, termina la búsqueda



algoritmo búsqueda_secuencial

A: vector donde se busca (contiene n elementos)

t: valor buscado

inicio

encontrado \leftarrow Falso

$i \leftarrow 1$

mientras ($i \leq n$) y ($\text{encontrado} = \text{Falso}$) **hacer**

si $t = A[i]$

entonces

escribir 'Se encontró el elemento buscado en la posición', i
 encontrado \leftarrow Verdadero

si_no

$i \leftarrow i + 1$

fin_si

fin_mientras

si $i = n + 1$ {también puede utilizarse si encontrado = Falso}

entonces

escribir 'No se encuentra el elemento'

si_no

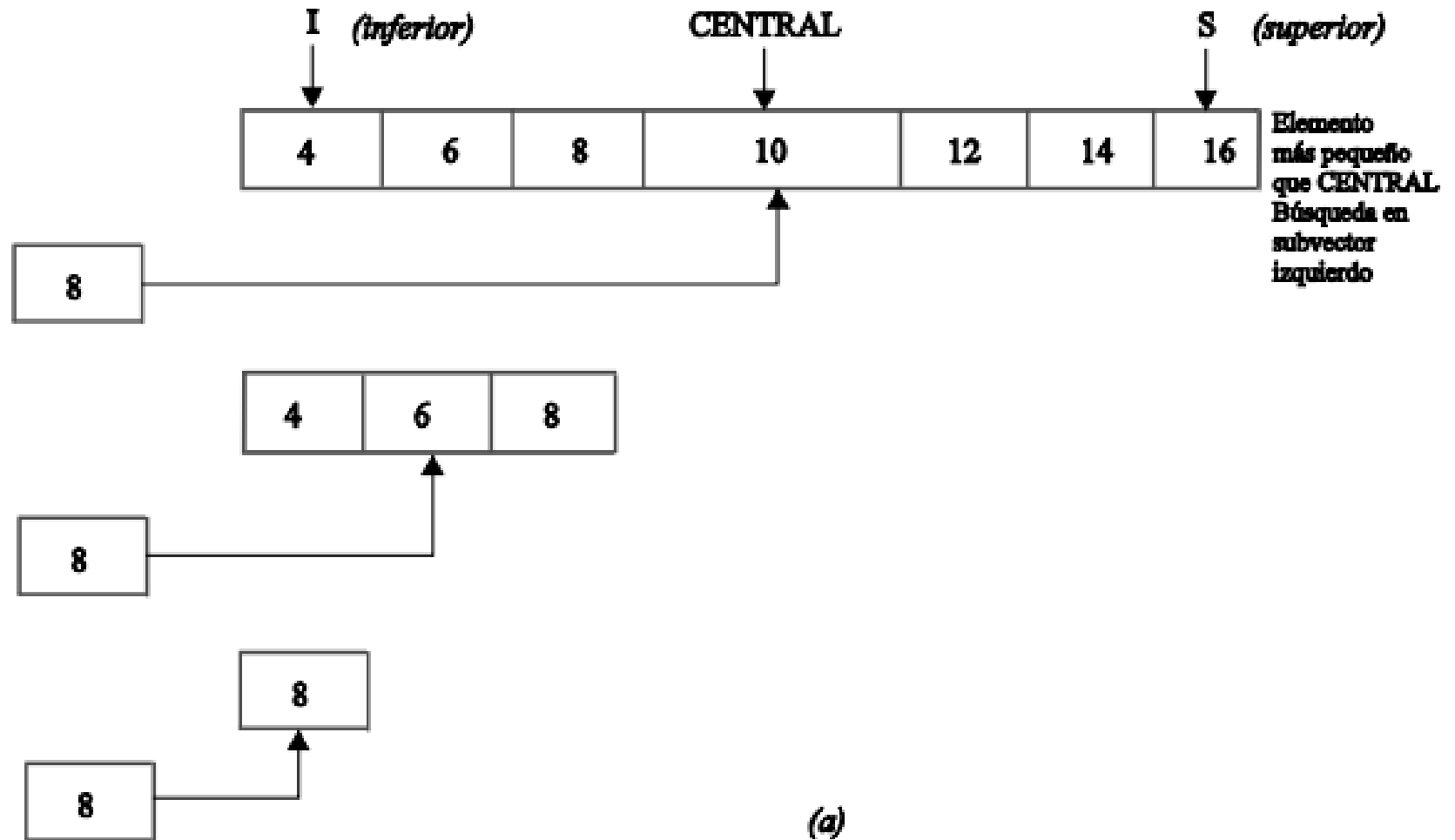
escribir 'El elemento se encuentra en la posición', i

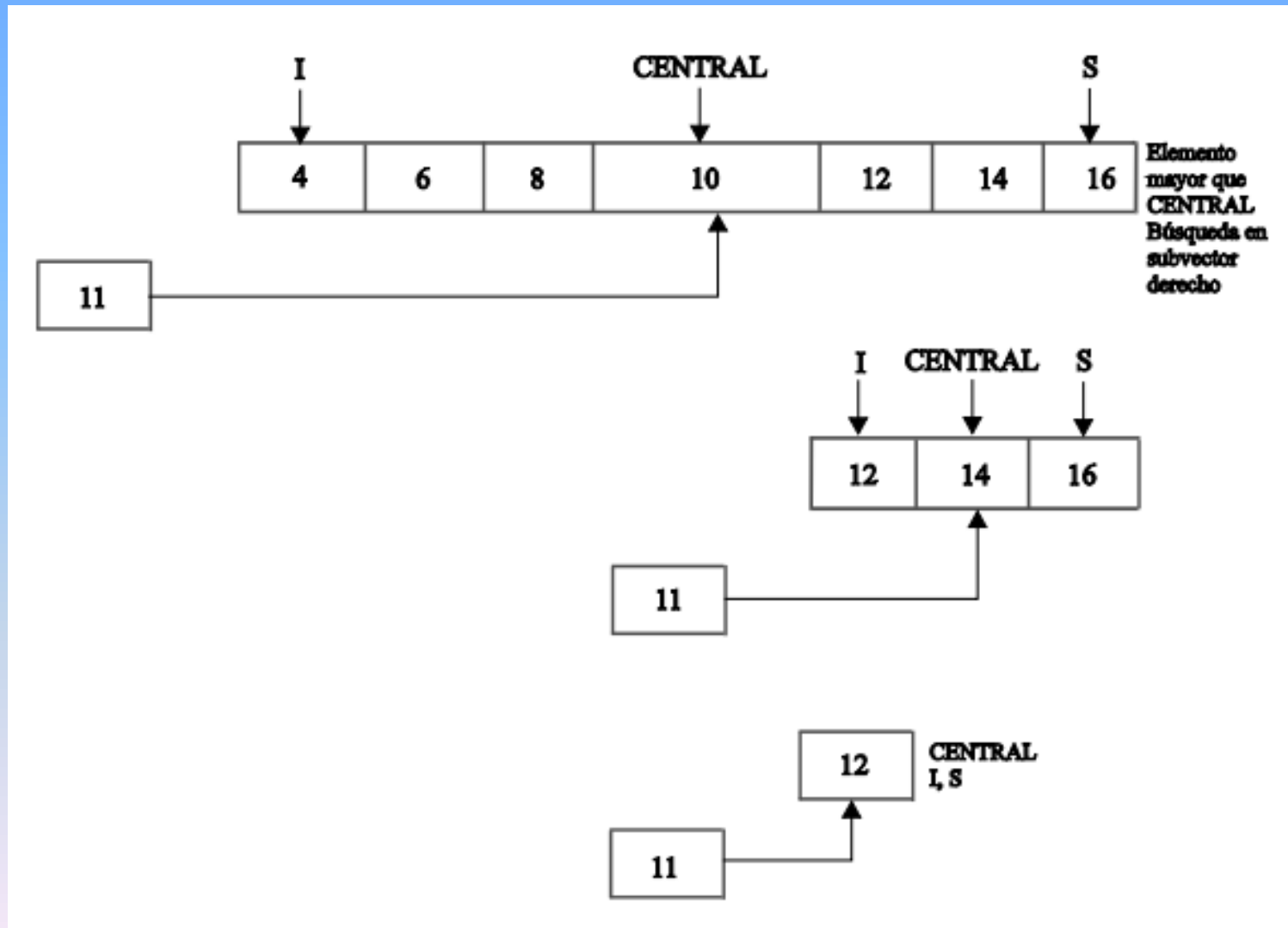
fin_si

fin

Búsqueda binaria

- Aplicable solo a vectores ordenados
- Comparar con el elemento central del vector:
 - Si es el buscado, termina
- Si no es el buscado:
 - Se determina en que mitad puede estar el elemento
 - Se repite el proceso con esa mitad







```
algoritmo búsqueda_binaria
X: vector de N elementos
K: valor buscado
inicio
    BAJO  $\leftarrow$  1
    ALTO  $\leftarrow$  N
    CENTRAL  $\leftarrow$  ent((BAJO + ALTO) / 2)
    mientras BAJO < ALTO y X[CENTRAL]  $\neq$  K hacer
        si K < X[CENTRAL]
            entonces
                ALTO  $\leftarrow$  CENTRAL - 1
            si_no
                BAJO  $\leftarrow$  CENTRAL + 1
        fin_si
        CENTRAL  $\leftarrow$  ent((BAJO + ALTO) / 2)
    fin_mientras
    si K = X(CENTRAL)
        entonces escribir 'valor encontrado en', CENTRAL
        si_no escribir 'valor no encontrado'
    fin_si
fin
```

Búsqueda binaria

- Hagamos en el pizarrón entre todos una implementación recursiva de búsqueda binaria

Ordenamiento

- Organizar los datos de un vector con un criterio y de una forma específica
 - Numérico
 - Alfabético
 - Ascendente
 - Descendente
- Importancia de trabajar con vectores ordenados

Ordenamiento por inserción

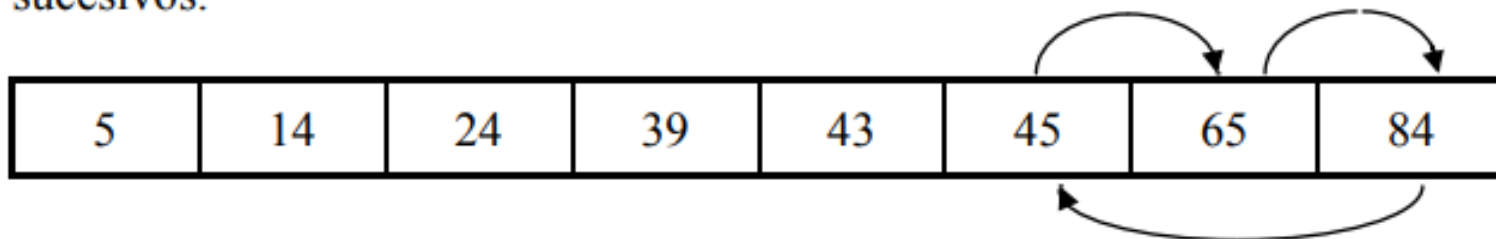
- Insertar de forma ordenada un elemento en una parte de un vector previamente ordenado
- Recorrer el vector v desde el segundo elemento hasta N
- Se inserta el elemento i en el lugar adecuado del subvector ordenado $sv[0:i-1]$
- Se soluciona con comparaciones y desplazamientos

Ordenamiento por inserción

Consideremos el siguiente vector, ya parcialmente ordenado en sus primeros siete elementos y hagamos la inserción del octavo (45) en su lugar correspondiente:

5	14	24	39	43	65	84	45
---	----	----	----	----	----	----	----

Para insertar el elemento 45, que es el siguiente en la ordenación, habrá que insertarlo entre 43 y 65, lo que supone desplazar a la derecha todas aquellas números de valor superior a 45, es decir, saltar sobre 65 y 84. Como puede observarse, el algoritmo se ejecutara en base a comparaciones y desplazamientos sucesivos.





Ordenamiento por inserción

```
algoritmo ord_inserción  
inicio  
  desde J = 2 hasta N hacer  
    AUX  $\leftarrow$  X(J)  
    K  $\leftarrow$  J - 1  
    mientras AUX < X(K) hacer  
      X(K + 1)  $\leftarrow$  X(K)  
      K  $\leftarrow$  K - 1  
    fin_mientras  
    X(K + 1)  $\leftarrow$  AUX  
  fin_desde  
fin
```

Ordenamiento por inserción

- Ejemplo en pizarrón con el vector
 - $X = [1 \ 5 \ 3 \ 8 \ 2]$

Ordenamiento por intercambio (bubble sort)

- Comparar pares de elementos adyacentes
 - Si están ordenados, me corro a la derecha
 - Si no están ordenados, los intercambio
- En cada iteración, se compara un elemento menos (el ultimo)

Ordenamiento por intercambio (bubble sort)

- Ejemplo con vector $X = [3 \ 1 \ 5 \ 2 \ 4]$
- Idea de burbuja!!!!!!!



algoritmo burbuja

inicio

desde $l=1$ **hasta** $N-1$ **hacer**

desde $J=1$ **hasta** $N-l$ **hacer** {no utilizamos los elementos ya ordenados}

si $X(J) > X(J+1)$ **entonces** {intercambiar}

$AUX \leftarrow X(J)$

$X(J) \leftarrow X(J+1)$

$X(J+1) \leftarrow AUX$

fin-si

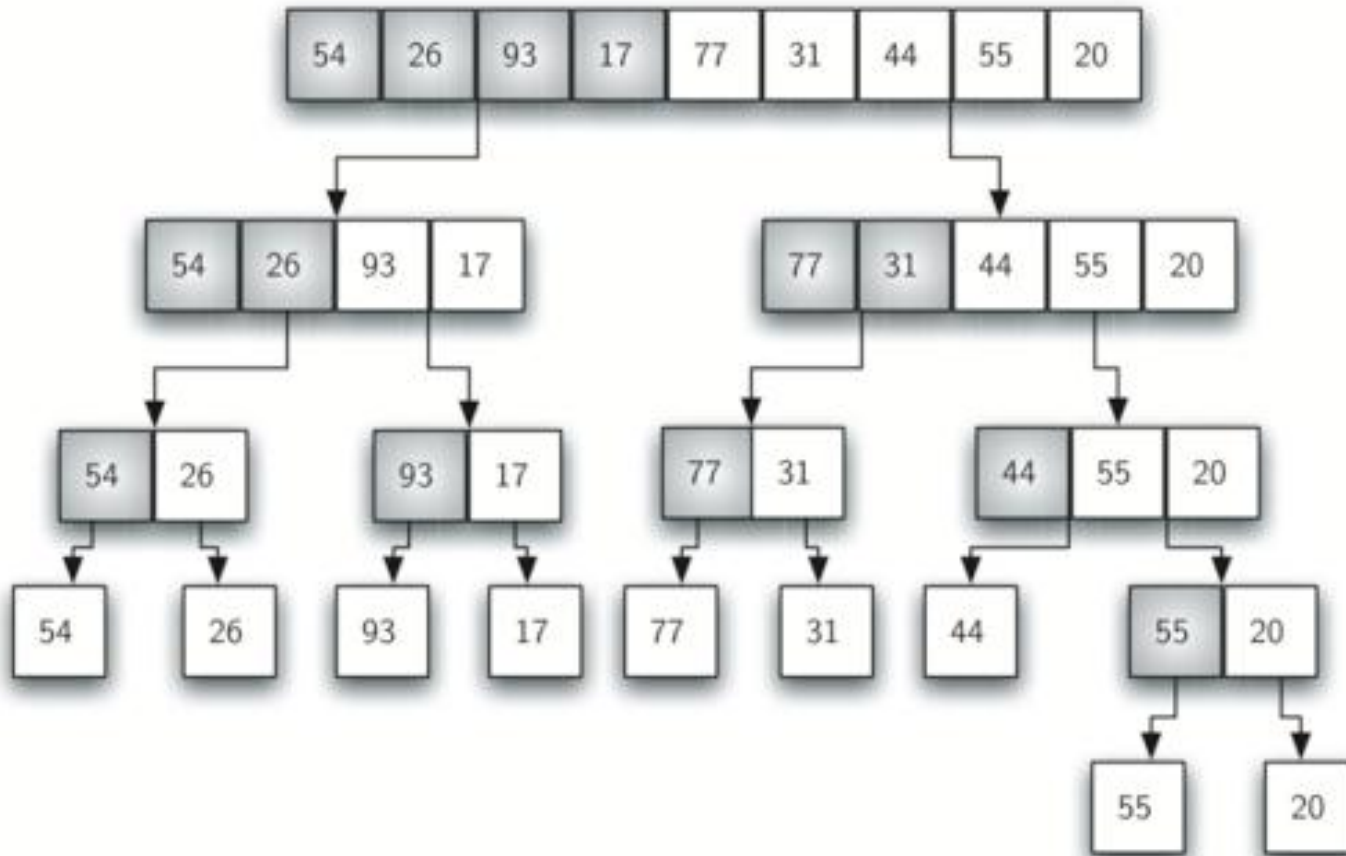
fin-desde

fin-desde

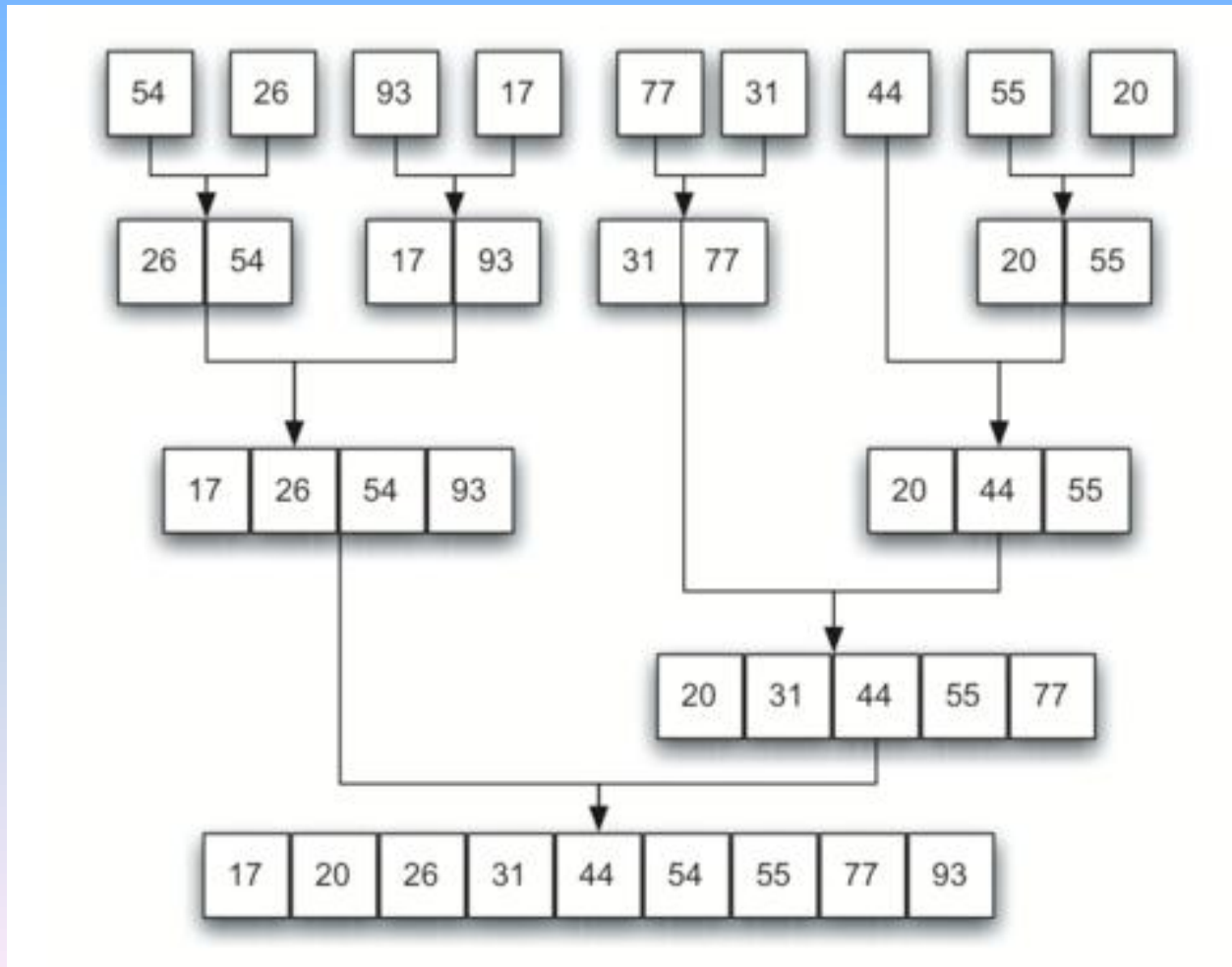
Ordenamiento por mezcla (Merge sort)

- Algoritmo recursivo
 1. Si la longitud es 1 o 0, ya esta ordenado
 2. Si no, dividir el vector en dos y ordenar
 3. Mezclar los dos subvectores ordenados

Ordenamiento por mezcla (Merge sort)



Ordenamiento por mezcla (Merge sort)



Ordenamiento por mezcla (Merge sort)

- Hagamos en el pizarrón entre todos una implementación recursiva de «merge sort»
- Ejemplo con vector $X = [1\ 5\ 3\ 2\ 4\ 8\ 6]$

Ordenamiento por mezcla (Merge sort)

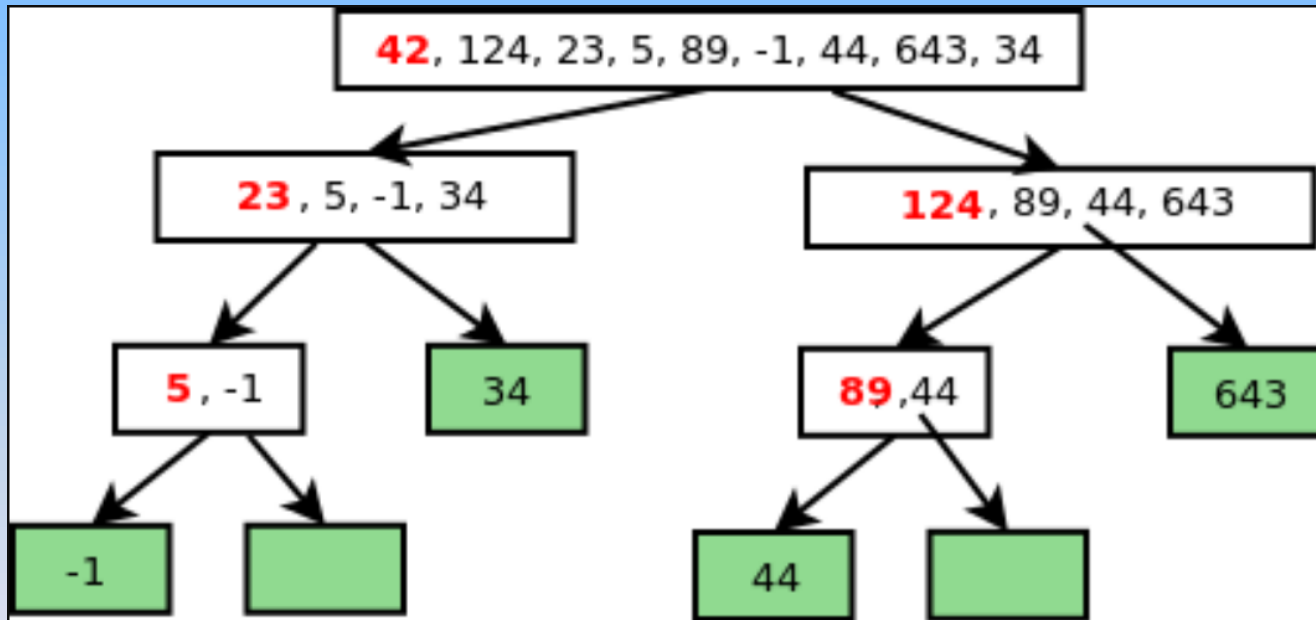
- Mezcla???
 - Agarrar dos vectores ordenados y combinarlos en uno nuevo y ordenado
 - Comparar solo primeros elementos e ir eliminando

Ordenamiento rápido (Quick sort)

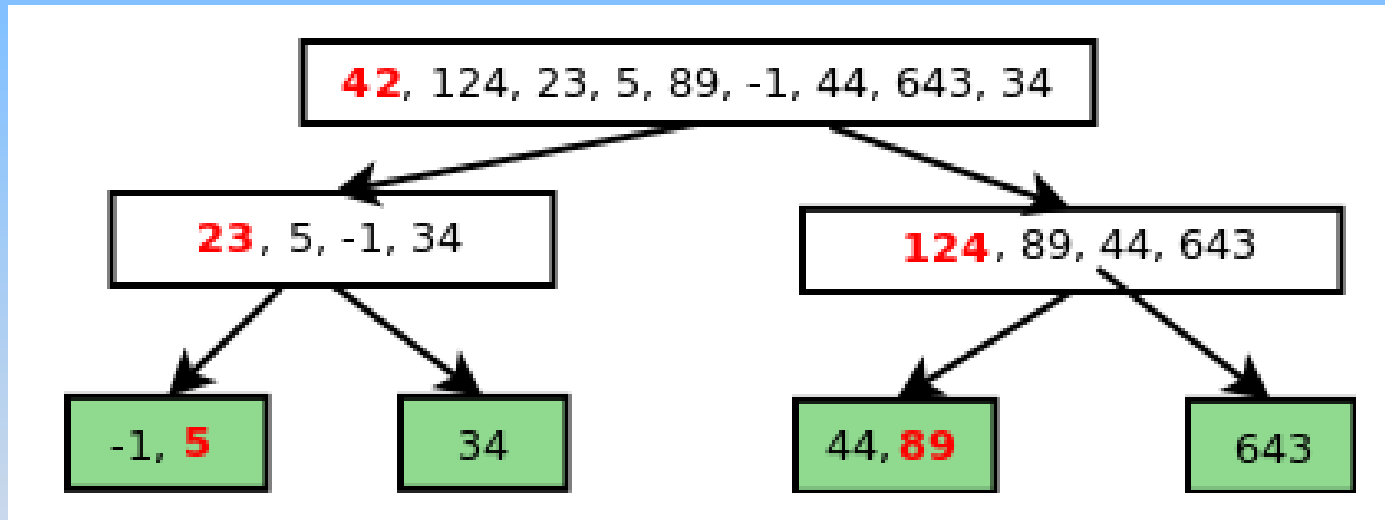
- Si la longitud es 1 o 0
 - Ya esta ordenado
- Si no es uno
 - Elegir el pivote
 - Reacomodar los elementos del vector a los lados del pivote
 - Repetir el proceso para cada subvector por separado



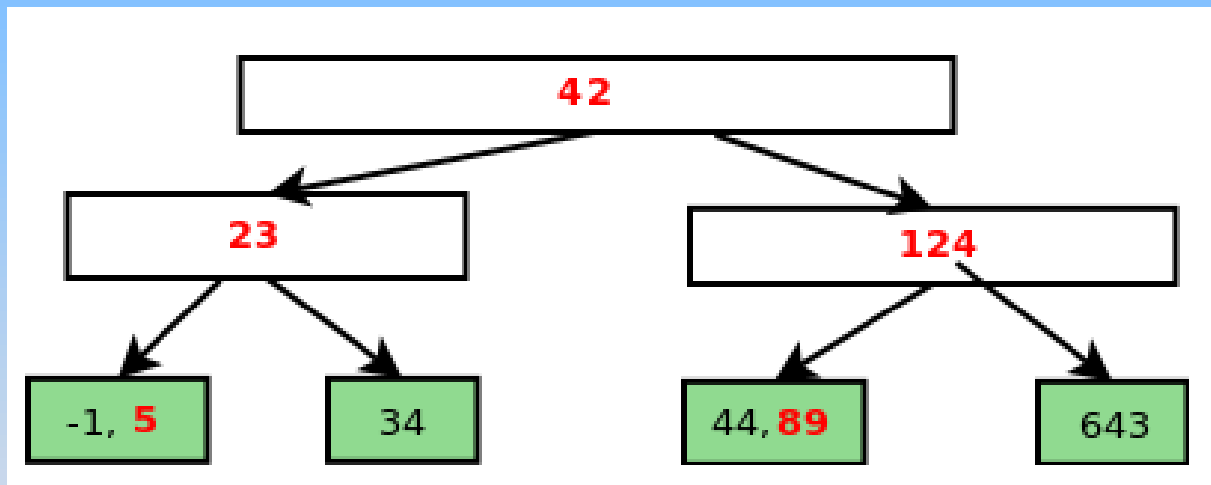
Ordenamiento rápido (Quick sort)



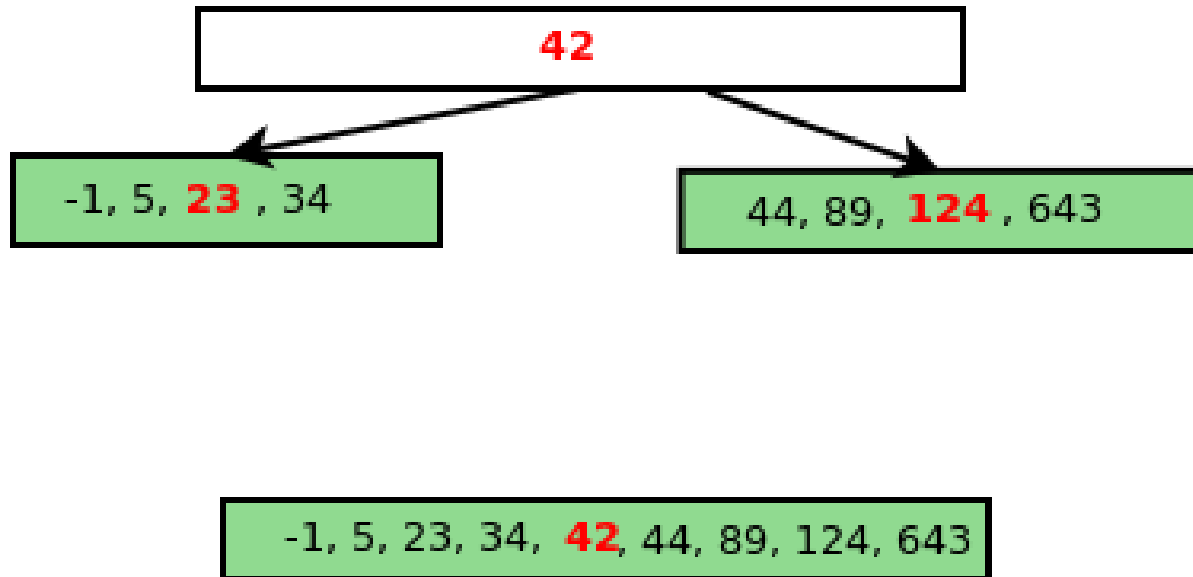
Ordenamiento rápido (Quick sort)



Ordenamiento rápido (Quick sort)



Ordenamiento rápido (Quick sort)



Ordenamiento rápido (Quick sort)

- Hagamos en el pizarrón entre todos una implementación recursiva de «Quick sort»
- Ejemplo con el vector $X = [5 \ 3 \ 7 \ 6 \ 2 \ 1 \ 4]$

Ordenamiento rápido (Quick sort)

- Reacomodo????
- Tomar elementos desde la izquierda y desde la derecha, comparar e intercambiar

Ordenamiento rápido (Quick sort)

De forma gráfica el proceso sería el siguiente:

10	40	7	9	15	27
----	----	---	---	----	----

Array original a ordenar

10	40	7	9	15	27
-----------	----	---	---	----	----

Se toma como pivote el primer elemento

10	40	7	9	15	27
-----------	-----------	---	----------	----	----

i

j

La búsqueda de izquierda a derecha encuentra el 40, mayor que pivote y la búsqueda de derecha a izquierda encuentra el 9, menor que pivote

10	9	7	40	15	27
-----------	----------	---	-----------	----	----

Se intercambian

10	9	7	40	15	27
-----------	---	----------	-----------	----	----

j

i

Continúa la búsqueda, se encuentra el valor 40 mayor que pivote y el valor 7 menor que pivote, pero ya se han cruzado. Paramos.

7	9	10	40	15	27
---	---	-----------	----	----	----

subarray 1

subarray 2

Finalmente colocamos el pivote en su lugar. (Posición j), quedando el array dividido en dos subarrays a los que les aplicará el mismo proceso