Ordenación

Alberto de la Encina Vara

Ordenación

36	-23	8	26	-32	-9	38	-7	13	-43	-47	-15	42	23	9

-12	6	-5	18	-27	14	-40	25	-31	3	-16	48	-2	16	-33

-47	-43	-40	-33	-32	-31	-27	-23	-16	-15	-12	-9	-7	-5	-2

3	6	8	9	13	14	16	18	23	25	26	36	38	42	48

Ordenación

- **in-place**: sobre el propio array de entrada (sin arrays auxiliares)
- **adaptativo**: más eficiente cuanto más ordenado esté el array
- **estable**: respeta el orden de los elementos iguales
- **complejidad y eficiencia**: cantidad de operaciones respecto al tamaño del array de partida N. Casos de estudio:
  - Lista inicialmente ordenada
  - Lista inicialmente ordenada al revés
  - Lista con disposición inicial aleatoria

Ordenación: Selección con vector auxiliar

selecciono el mínimo/máximo lo quito

```
mientras NOestaVacio (elVector) hacer:
        posMin = calcularPosMinimo(elVector)
        valorMin = elementoEnPos(elVector, posMin)
        quitarPos(elVector, posMin)
        añadirAlFinal(elVectorAux, valorMin)
```

Ordenación: Selección inSitu

selecciono el mínimo/máximo lo intercambio

```
desde la pos = 0 hasta pos < tamaño(elVector)-1
        avanzando en 1 pos hacer:
            posMin = calcularPosMinimoDesde(elVector, pos)
            intercambiarValoresPos(elVector, posMin, pos)
```

- Invariante: El vector tras cada pasada está ordenado [0, pos]

Ordenación: Inserción con vector auxiliar

cojo de uno en uno y lo *inserto* ordenado

```
desde la pos = 0 hasta pos < tamaño(elVector)
    avanzando en 1 pos hacer:
        valor = elementoEnPos(elVector, pos)
        insertarOrdenadamente(elVectorAux, valor)
```

insertarOrdenadamente (elVector, valor) : 

- Invariante: El vector auxiliar contiene ordenados los elementos de elVector entre [0, pos]

Ordenación: InsertarOrdenadamente I

elVector

	0	1	2	3	4	5	6	7	8	9
datos	3	6	8	18	23	25				
nElem	6									

valor

13

```
insertarOrdenadamente (elVector, valor) :
        posIns = buscarPosIns(elVector, valor)
        añadirEnPos(elVector, valor, posIns)
```

- Invariante: Si *elVector* estaba ordenado continua ordenado y tiene un elemento más el *valor*

Ordenación: InsertarOrdenadamente I

elVector

	0	1	2	3	4	5	6	7	8	9
datos	3	6	8	18	23	25				

nElem	6

elVector

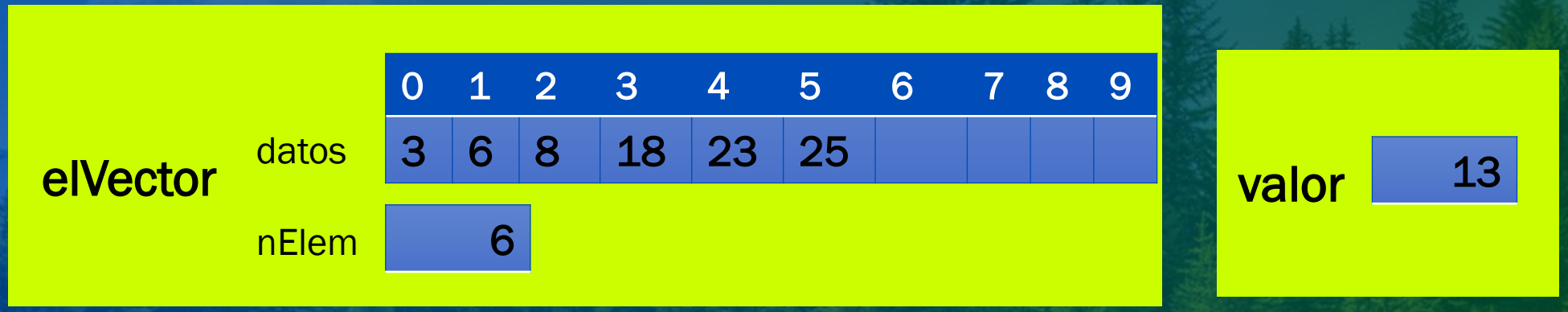
	0	1	2	3	4	5	6	7	8	9
datos	3	6	8	13	18	23	25			

nElem	7

```
insertarOrdenadamente (elVector, valor) :
        posIns = buscarPosIns(elVector, valor)
        añadirEnPos(elVector, valor, posIns)
```

- Invariante: Si *elVector* estaba ordenado continua ordenado y tiene un elemento más el *valor*

Ordenación: InsertarOrdenadamente 2



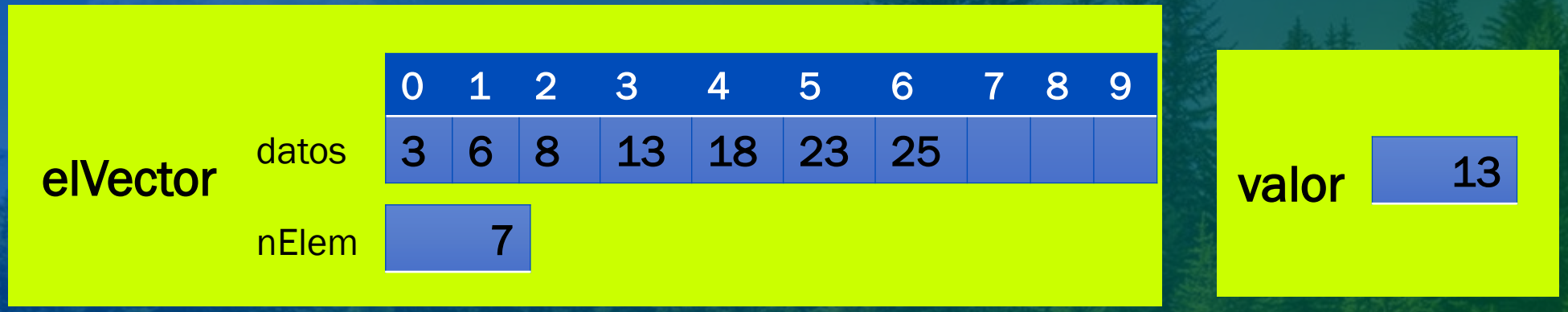
insertarOrdenadamente (elVector, valor) :

añadirAlFinal(elVector, valor)

hundirUltimoElemento(elVector)

- Invariante: Si *elVector* estaba ordenado continua ordenado y tiene un elemento más el *valor*

Ordenación: InsertarOrdenadamente 2



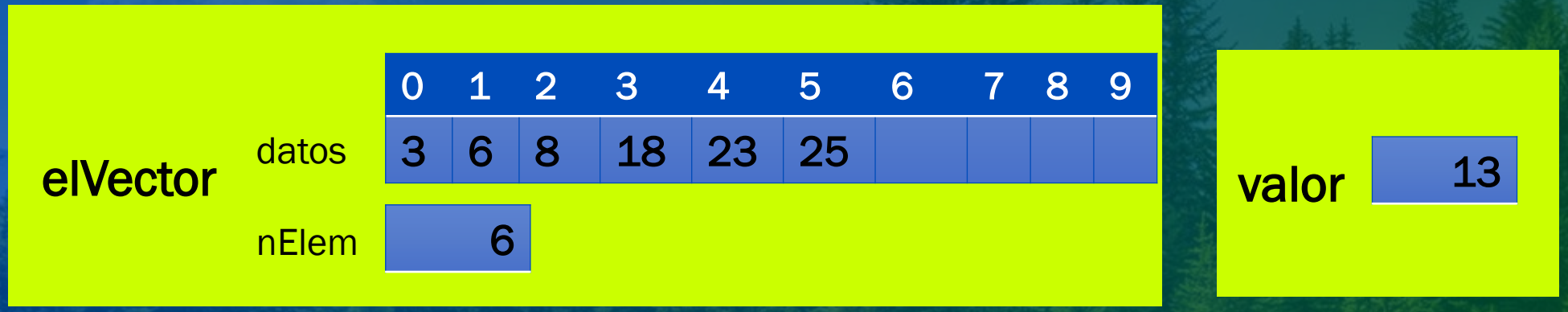
insertarOrdenadamente (elVector, valor) :

añadirAlFinal(elVector, valor)

hundirUltimoElemento(elVector)

- Invariante: Si *elVector* estaba ordenado continua ordenado y tiene un elemento más el *valor*

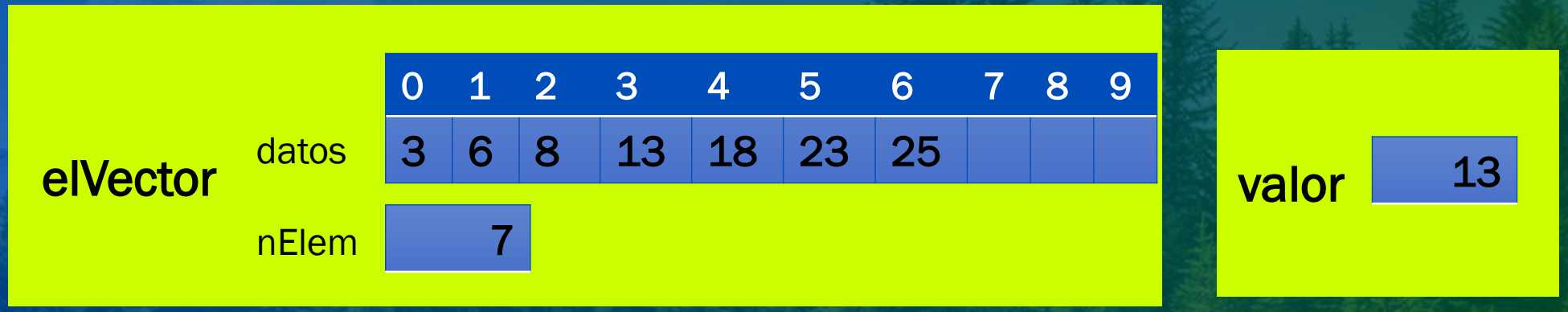
Ordenación: InsertarOrdenadamente 3




```
insertarOrdenadamente (elVector, valor) :
        posHueco = hacerHuecoDesplazandoMayores(elVector, valor)
        reemplazarValorEnPos(elVector, valor , posHueco)
```

- Invariante: Si *elVector* estaba ordenado continua ordenado y tiene un elemento más el *valor*

Ordenación: InsertarOrdenadamente 3




```
insertarOrdenadamente (elVector, valor) :
        posHueco = hacerHuecoDesplazandoMayores(elVector, valor)
        reemplazarValorEnPos(elVector, valor , posHueco)
```

- Invariante: Si *elVector* estaba ordenado continua ordenado y tiene un elemento más el *valor*

Ordenación: Inserción inSitu

cojo de uno en uno y lo *inserto* ordenado

desde la pos = 1 hasta pos < tamaño(elVector)
avanzando en 1 pos hacer:
hundirElementoPos(elVector, pos)

```
hundirElementoPos(elVector, pos):
        valorAOrd = elementoEnPos(elVector, pos)
        posHueco = desplazarMayoresQuePos(elVector, pos)
        reemplazarValorEnPos(elVector, valorAOrd , posHueco)
```

- Invariante: El vector tras cada pasada está ordenado [0, pos]

Ordenación: Burbuja inSitu

floto los elementos muchas veces

repetir tamanyo(elVector) veces:
recorrerBurbuja(elVector)

recorrerBurbuja(elVector):
desde la pos = 0 hasta pos < tamaño(elVector) -1 avanzando en 1 pos hacer:
si elementoEnPos(elVector, pos) > elementoEnPos(elVector, pos +1)
intercambiarValoresPos(elVector, pos, pos +1)

- Invariante: Tras cada pasada ordenado (posFin-nPasadas, posFin]
  - posFin = tamaño(elVector) -1

Ordenación: Burbuja inSitu

floto los elementos muchas veces

repetir tamanyo(elVector) veces:
recorrerBurbuja(elVector)

```
recorrerBurbuja(elVector):
    desde la pos = 1 hasta pos < tamaño(elVector) avanzando en 1 pos hacer:
        si elementoEnPos(elVector, pos-1 ) > elementoEnPos(elVector, pos)
            intercambiarValoresPos(elVector, pos-1 , pos)
```

- Invariante: Tras cada pasada ordenado (posFin-nPasadas, posFin]
  - posFin = tamaño(elVector) -1

Ordenación: Burbuja inSitu (mejora 1, idea)

floto los elementos muchas veces (hasta no hay cambios)

repetir:
recorrerBurbuja(elVector)
mientras hayCambios(elVector)

recorrerBurbuja(elVector):
desde la pos = 1 hasta pos < tamaño(elVector) avanzando en 1 pos hacer:
si elementoEnPos(elVector, pos-1) > elementoEnPos(elVector, pos)
intercambiarValoresPos(elVector, pos-1 , pos)

- Invariante: Tras cada pasada ordenado (posFin-nPasadas, posFin]
  - posFin = tamaño(elVector) -1

Ordenación: Burbuja inSitu (mejora 1)

floto los elementos muchas veces (hasta que NO hay cambios)

repetir

hayCambios = recorrerBurbuja(elVector)

mientras hayCambios hacer:

```
recorrerBurbuja(elVector):
        hayCambios = false;
        desde la pos = 1 hasta pos < tamaño(elVector) avanzando en 1 pos hacer:
                si elementoEnPos(elVector, pos-1) > elementoEnPos(elVector, pos)
                        hayCambios = true
                        intercambiarValoresPos(elVector, pos-1, pos )
        devolver hayCambios;
```

- Invariante: Tras cada pasada ordenado (posFin-nPasadas, posFin]

Ordenación: Burbuja inSitu (mejora 2)

floto los elementos muchas veces (bubuja más pequeña)

```
numBurbuja = 0
repetir:
        recorrerBurbujaEvaporada(elVector, numBurbuja )
        numBurbuja++
mientras hayCambios(elVector):
```

```
recorrerBurbujaEvaporada(elVector, i):
    desde la pos = 1 hasta pos < tamaño(elVector) –i avanzando en 1 pos hacer:
        si elementoEnPos(elVector, pos-1 ) > elementoEnPos(elVector, pos )
            intercambiarValoresPos(elVector, pos-1, pos )
```

- Invariante: Tras cada pasada ordenado (posFin-nPasadas, posFin]

Ordenación: Burbuja inSitu (mejora 1+2)

floto los elementos muchas veces (cambios y bubuja más pequeña)

```
numBurbuja = 0; hayCambios = true;
mientras (numBurbuja < tamaño(elVector) y  hayCambios) hacer:
        hayCambios = recorrerBurbujaEvaporada(elVector, numBurbuja )
        numBurbuja++

recorrerBurbujaEvaporada(elVector, i):
        hayCambios = false
        desde la pos = 1 hasta pos < tamaño(elVector) –i avanzando en 1 pos hacer:
                si elementoEnPos(elVector, pos-1 ) > elementoEnPos(elVector, pos )
                        hayCambios = true
                        intercambiarValoresPos(elVector, pos-1, pos )
        return hayCambios;
```

- Invariante: Tras cada pasada ordenado (posFin-nPasadas, posFin]

Ordenación: Mergesort inSitu

divido en dos el vector, los ordeno y *mezclo* los trozos ordenandos

```
si tamaño(elVector) >1
        posMedia = (posFin-posIni)/2+ posIni
        mergesort(elVector, posIni, posMedia)
        mergesort(elVector, posMedia+1, posFin)
        mezclaOrd(elVector, posIni, posMedia, posFin)
```

mezclaOrd:

coger de uno u otro según sean menores hasta que se vacíe uno

finalmente coger los que queden del que quede sin elementos

Ordenación: Quicksort inSitu

menores al inicio, mayores al final y ordenar inicio y fin

colocar en base a valorPivote

quicksort(elVector, posIni1, porFin1)

quicksort(elVector, posIni2, porFin2)

Algoritmos de ordenación

- **selección:** *selecciono* el mínimo/máximo lo quito y así hasta que no queden elementos.
- **inserción:** cojo de uno en uno y lo *inserto* ordenado
- **burbuja:** intercambios locales, tipo *burbujas*, hasta que queda ordenado
- **mergesort:** parto el vector en dos los ordeno y *mezclo* los vectores *ordenados*
- **quicksort:** separo menores a un lado, mayores a otro y ordeno
- **heapsort, radix, etc...**

Búsqueda de elemento/posición:

busco un elemento/posición en un array

- Desordenado: Búsqueda simple en el array:
  - mientras haya elementos y no lo encuentre pasar al siguiente
- Ordenado:
  - Búsqueda simple en el array:
    - mientras haya elementos y no lo encuentre pasar al siguiente
  - Búsqueda binaria en el array: (posIni, y posFin en la que buscar)
    - Ir al medio y mirar:
      - si igual terminado,
      - si menor buscar en la mitad superior
      - si mayor buscar en la mitad inferior

Calcular mínimo/máximo:

- Desordenado: Recorrido simple en el array:
  - Recorrer el array actualizando el menor si es mejor que el calculado hasta el momento.

- Ordenado:
  - Acceso directo o primero(mínimo) o último(máximo)

Mezcla ordenada: con vector auxiliar

mezclar dos arrays ordenados sobre un tercero

mezclaOrdenada (elVector1, elVector2, elVectorSol) :

- coger de uno u otro según sea el menor y añadirlo al final de elVectorSol hasta que se vacíe uno
- finalmente coger los que queden del que quede sin elementos

➢ Invariante: elVectorSol queda ordenado y con los elementos de elVector1 y elVector2

Mezcla ordenada: inSitu

mezclar dos partes ordenadas continuas de un array

mezclaOrdenada (elVector, posIni1, posIni2, posFin2) :

- Invariante: elVector queda ordenado entre posIni1 y posFin2