

Programación

03 Aplicación de Estructuras de Almacenamiento

José Luis González Sánchez





Contenidos

1. Arrays unidimensionales.
2. Arrays multidimensionales.
3. Cadenas de caracteres.
4. Expresiones regulares.
5. Métodos de ordenación
6. Métodos de búsqueda

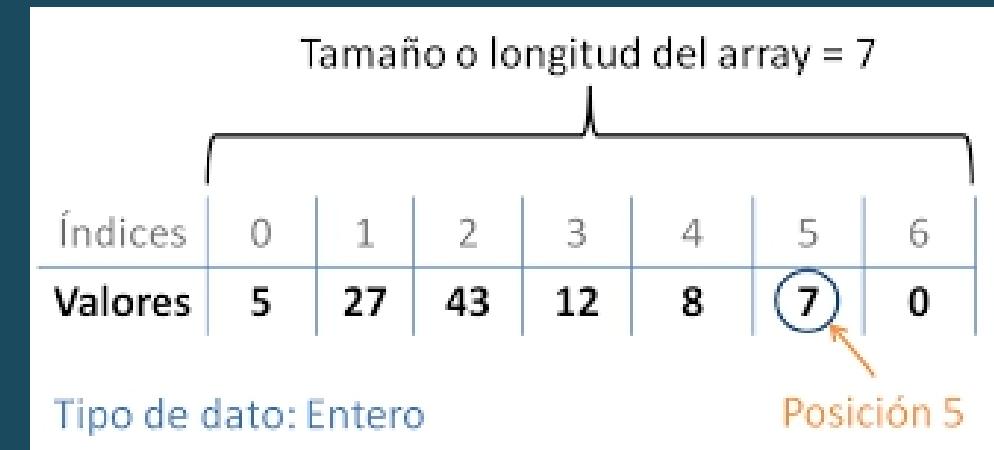
Arrays unidimensionales

Nuestras primeras colecciones



Arrays unidimensionales

- Un array es una estructura de datos que contiene una colección de datos del mismo tipo. Cada elemento de esta colección se puede acceder por un índice.
- Podemos imaginar que es un armario de cajones de un tipo determinado. Cada valor se almacena en un cajón. Para acceder a él, simplemente indicamos el índice del cajón y podemos leerlo o actualizar su valor.
- Ejemplos
 - Temperaturas mínimas de los últimos treinta días
 - Valor de las acciones de una empresa durante la última semana





Arrays unidimensionales

- **Propiedades de los arrays**

- Los arrays se utilizan como contenedores para almacenar datos relacionados (en vez de declarar variables por separado para cada uno de los elementos del array).
- Todos los datos incluidos en el array son del mismo tipo. Se pueden crear arrays de enteros de tipo int o de reales de tipo float, pero en un mismo array no se pueden mezclar datos de tipo int y datos de tipo float.
- El tamaño del array se establece cuando se crea el array (con el operador new, igual que cualquier otro objeto). No se puede cambiar.
- A los elementos del array se accederá a través de la posición que ocupan dentro del conjunto de elementos del array.
- Los arrays de una sola dimensión que se acceden con un índice se les conocen como vectores.
- En lenguajes que provienen de C el índice de la primera componente de un vector es siempre 0. Esto es debido a que se multiplica el índice por el tamaño del dato para acceder a la posición de memoria donde se almacena.
- Es importante conocer siempre la longitud para evitar salirse de los límites, lo cual produce errores.



Arrays unidimensionales

- **Definición**

- Entre corchetes se indica el tamaño del vector.
- tipo debe coincidir con el tipo con el que se haya declarado el vector.
- vector debe ser una variable declarada como tipo[]

- **Ejemplos**

- decimal[] notas = new decimal[NUM_ALUMNOS] ⚡ crea un vector vacío de tamaño NUM_ALUMNOS
- entero[] temperaturas = new entero[7] ⚡ crea un vector de 7 posiciones vacío
- entero vector[] = {1, 2, 3, 5, 7} ⚡ Crea un vector de 5 posiciones con los valores introducidos



Arrays unidimensionales

- **Manipulación**

- Accedemos a la posición mediante su índice.
 - Para recorrellos usamos bucles for
 - Podemos saber su tamaño con length/size
 - Como empezamos por el índice 0, el último es vector[vector.length-1] por este motivo es importante poner < al recorrello en bucles para no sobrepasarlo
-
- **IMPORTANTE:** Cuando se pasa un array como parámetro, se copia una referencia al array y no el conjunto de valores en sí.
 - Por tanto, tenemos que tener cuidado con los efectos colaterales que se producen si, dentro de un módulo, modificamos un vector que recibimos como parámetro.

Decimal media (decimal datos[])

INICIO

Variables

entero i

decimal suma = 0;

decimal media

Para (i=0 mientras
i<datos.length)

 suma = suma +
 datos[i]

fin_para

media = suma/datos.length

Devolver (media)

FIN



Arrays unidimensionales

- **Manipulación**
- **IMPORTANTE:** Cuando se pasa un array como parámetro, se copia una referencia al array y no el conjunto de valores en sí.
- Por tanto, tenemos que tener cuidado con los efectos colaterales que se producen si, dentro de un módulo, modificamos un vector que recibimos como parámetro.

```
leerVector (entero[] datos)
INICIO
Variables
    entero i;
    Para (i=0 mientras
    i<datos.length)
        datos[i] = leerValor()
    fin_para
FIN
```



Arrays unidimensionales

- **Copia de arrays**
- La siguiente asignación sólo copia las referencias, no crea un nuevo array:

```
entero[] pares = {2, 4, 6, 8}  
entero[] datos = pares;
```

- Debemos hacerlo:

```
entero[] datos = new entero[pares.length];  
Para (i=0 mientras i<pares.length)  
    datos[i] = pares[i]
```

Importante: el operador == no debe utilizarse para comparar arrays, pues son referencias.



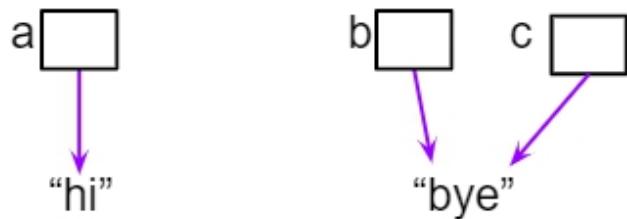
Arrays unidimensionales

- Comparaciones de arrays

Importante: el operador == no debe utilizarse para comparar arrays, pues son referencias.

Debemos hacerlo componente a componente

```
String a = new String("hi");
String b = new String("bye");
String c = b;      // c is now an alias for b
```



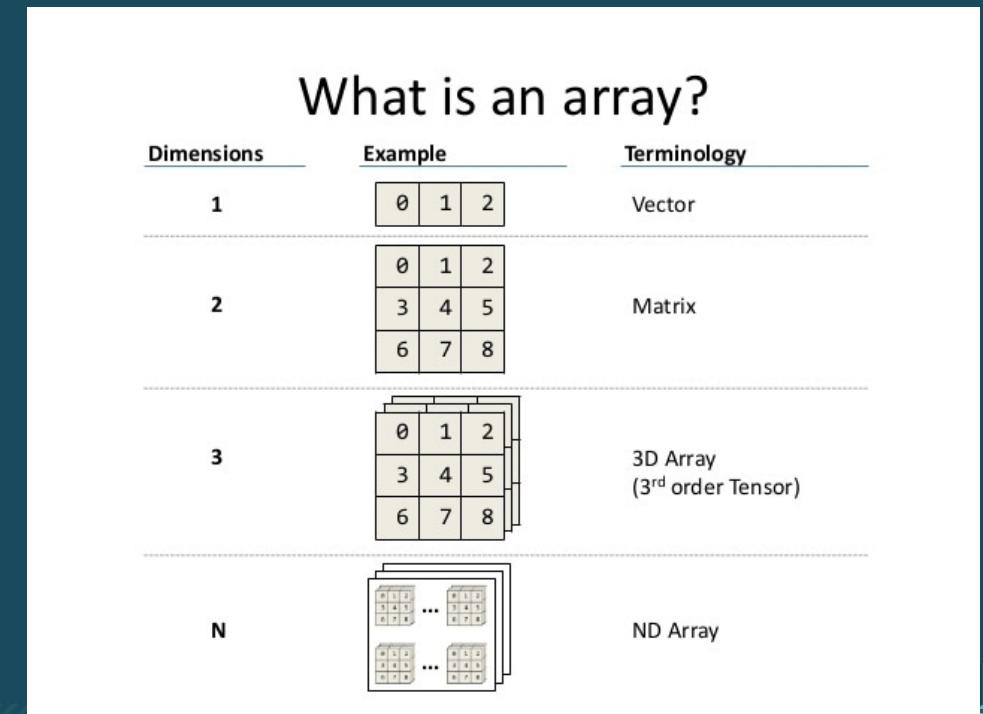
Arrays multidimensionales

Nos movemos en varias dimensiones

Arrays multidimensionales

- Un array multidimensional, es aquel que tiene dos dimensiones. Y por lo tanto necesitamos un índice por cada dimensión para recorrerlo
- Se suelen llamar Matrices si son de dos dimensiones, y los definimos en base a sus filas y columnas
- Podemos tener arrays con las dimensiones que queramos

		x =					
		0	1	2	3	4	
y =		0	1	0	0	0	0
		1	0	0	2	0	0
y =		2	1	1	0	0	1
		3	2	1	1	1	0
y =		4	1	2	1	2	2





Arrays unidimensionales

- Definición
 - tipo[][] identificador;
- Ejemplos
 - tipo[][] matriz = new tipo[filas][columnas];
 - decimal[][] temperaturas = new decimal[12][31] ↗ crea una matriz de 12 filas y 31 columnas, ideal para representar los 12 meses y por cada mes los 31 días para almacenar un dato decimal
 - Entero notas = {{4, 7, 9},{7, 2, 10},{5, 5, 7}} ↗ crea una matriz de tres filas (asignaturas) y tres columnas, exámenes realizados en cada asignatura



Arrays unidimensionales

• Manipulación

- Accedemos a la posición mediante cada uno de los índices de sus dimensiones.
- Para recorrelas usamos varios bucles for, uno por cada dimensión. Por ejemplo en una Matriz recorremos las filas y luego por cada fila las columnas
- Podemos saber el tamaño de cada fila y columna con length o size
- Como empezamos por el índice 0, el último es vector[vector.length-1] por este motivo es importante poner < al recorrela en bucles para no sobrepasarlo
- matriz.length nos da el número de filas
- matriz[0].length nos da el número de columnas
- Por tanto, el último elemento de la matriz es
 - matriz[matriz.length-1][matriz[0].length-1]

```
mostrarMatriz (decimal matriz[][])
INICIO
Variables
    entero i,j;
    entero filas = matriz.length;
    entero columnas = matriz[0].length;
    // Recorrido de las filas de la
    matriz
    para (i=0 mientras i<filas )
        // Recorrido de las celdas de una
        fila
            para (j=0 mientras j<columnas)
                Escribir( "matriz["+i+"]["+j+"]=" +
                    matriz[i][j] )
                fin_para
            fin_para
        FIN
```

Cadena de Caractéres

Un array vitaminado



Cadena de Caractéres

- Una cadena de caracteres no es un array de caracteres. Es decir
 - carácter[] nombre = {'j','o','s','e'}
 - cadena nombre = “Jose”
- Una cadena nos ofrece una serie de métodos para poder operar ágilmente con este tipo de estructuras de almacenamiento
 - El método subCadena(inicio, fin) nos permite obtener una subcadena entre dos índices: inicial y final.
 - El método caracterEn(n) nos devuelve el carácter que se encuentra en la posición n de la cadena.
 - El método indiceDe(s) nos devuelve la posición de una subcadena dentro de la cadena.
 - El método remplaza(old,new) reemplaza subcadenas old, por la new.
 - El método igual(s) se usa para comprobar si dos cadenas son iguales. No debemos usar el == como hemos visto con anterioridad.
 - El método comienzaCon(s) nos dice si una cadena empieza con un prefijo determinado.
 - El método terminaCon(s) nos dice si una cadena termina con un sufijo determinado.

Métodos de Ordenación

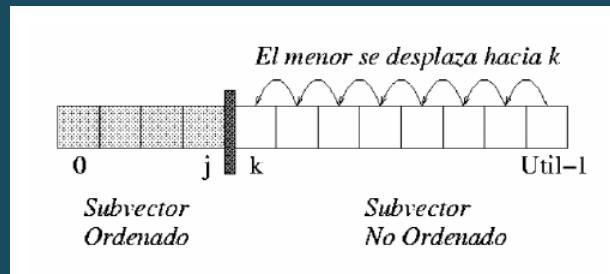
Vamos a ordenar nuestros elementos de un array



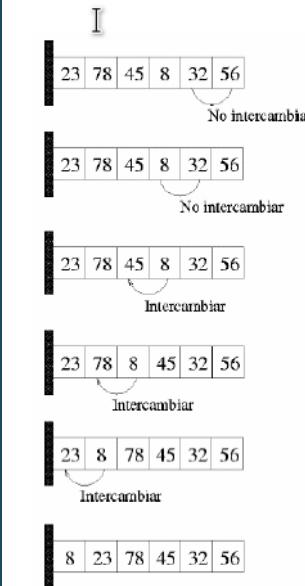
Métodos de Ordenación: Burbuja

- **Ordenación por Burbuja:** El ordenamiento burbuja hace múltiples pasadas a lo largo de una lista. Compara los ítems adyacentes e intercambia los que no están en orden. Cada pasada a lo largo de la lista ubica el siguiente valor más grande en su lugar apropiado. En esencia, cada ítem “burbujea” hasta el lugar al que pertenece..

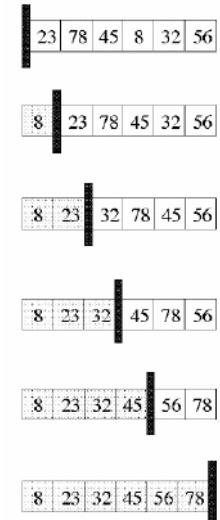
```
funcion ordenarBurbuja (numero v[]) {  
    numero tmp;  
    numero i, j;  
    numero N = v.length;  
  
    for (i=1; i<N; i++)  
        for (j=N-1; j>=i; j--)  
            if (v[j] < v[j-1]) {  
                tmp = v[j];  
                v[j] = v[j-1];  
                v[j-1] = tmp;  
            }  
}
```



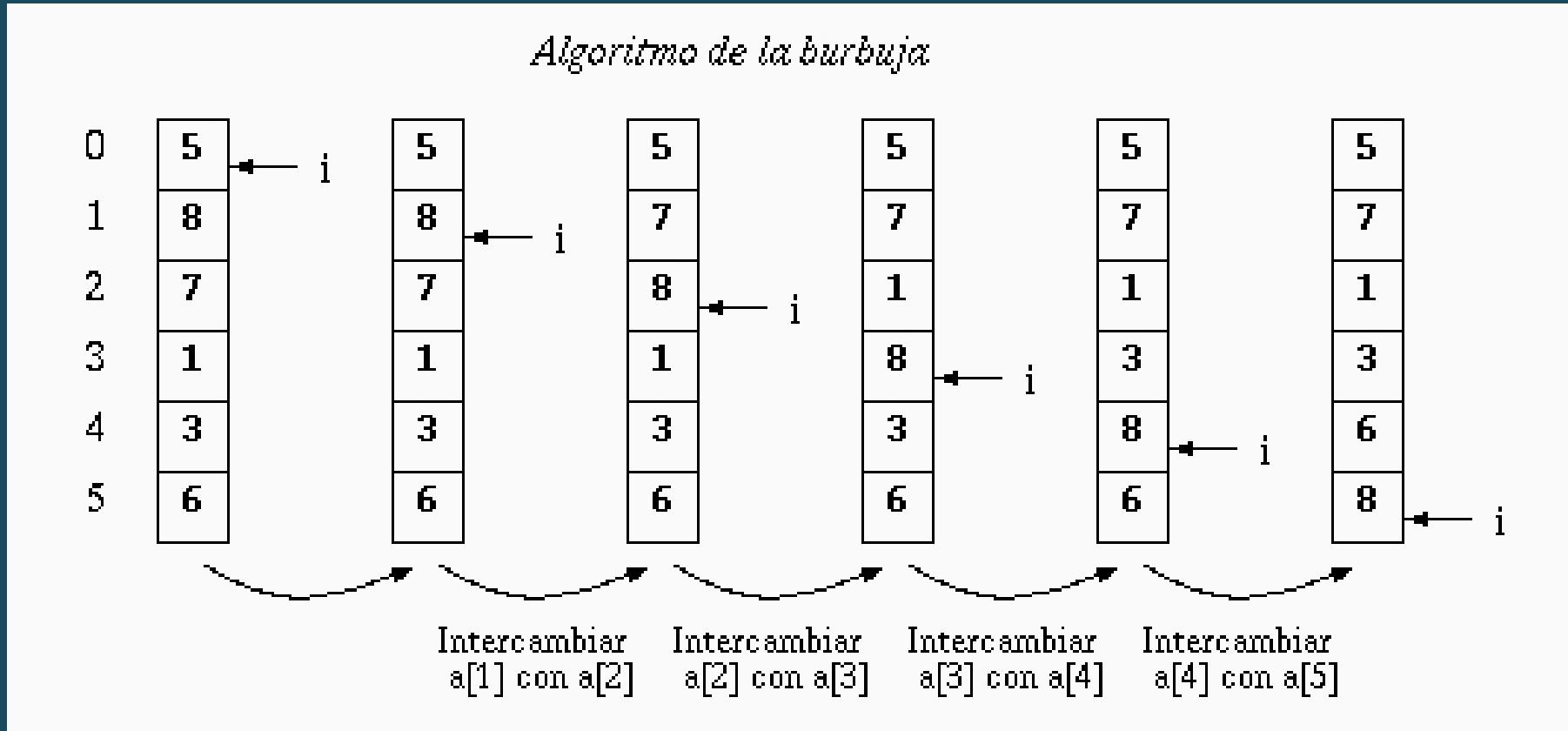
En cada iteración



Estado del vector
tras cada iteración:



Métodos de Ordenación: Burbuja



https://www.youtube.com/watch?v=EQMGabLO_M0

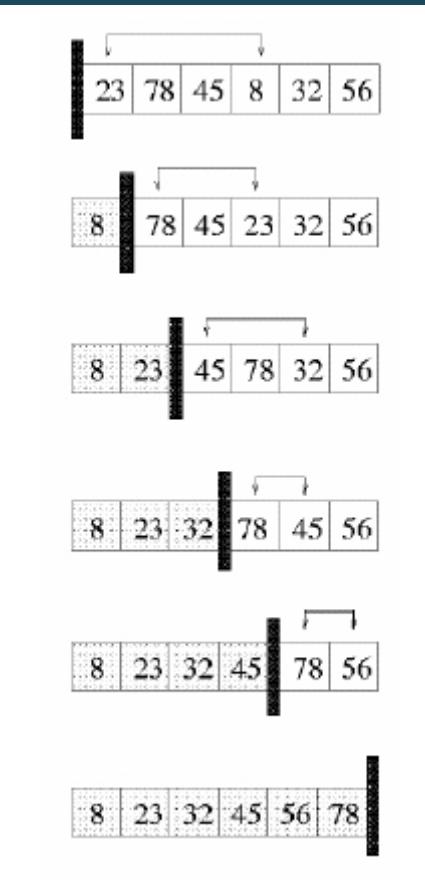
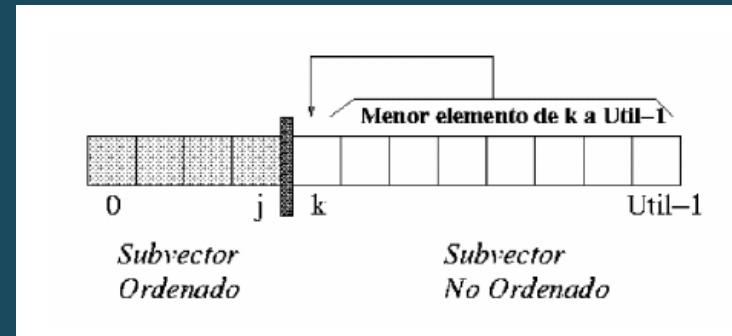
<https://www.youtube.com/watch?v=lyZQPjUT5B4>



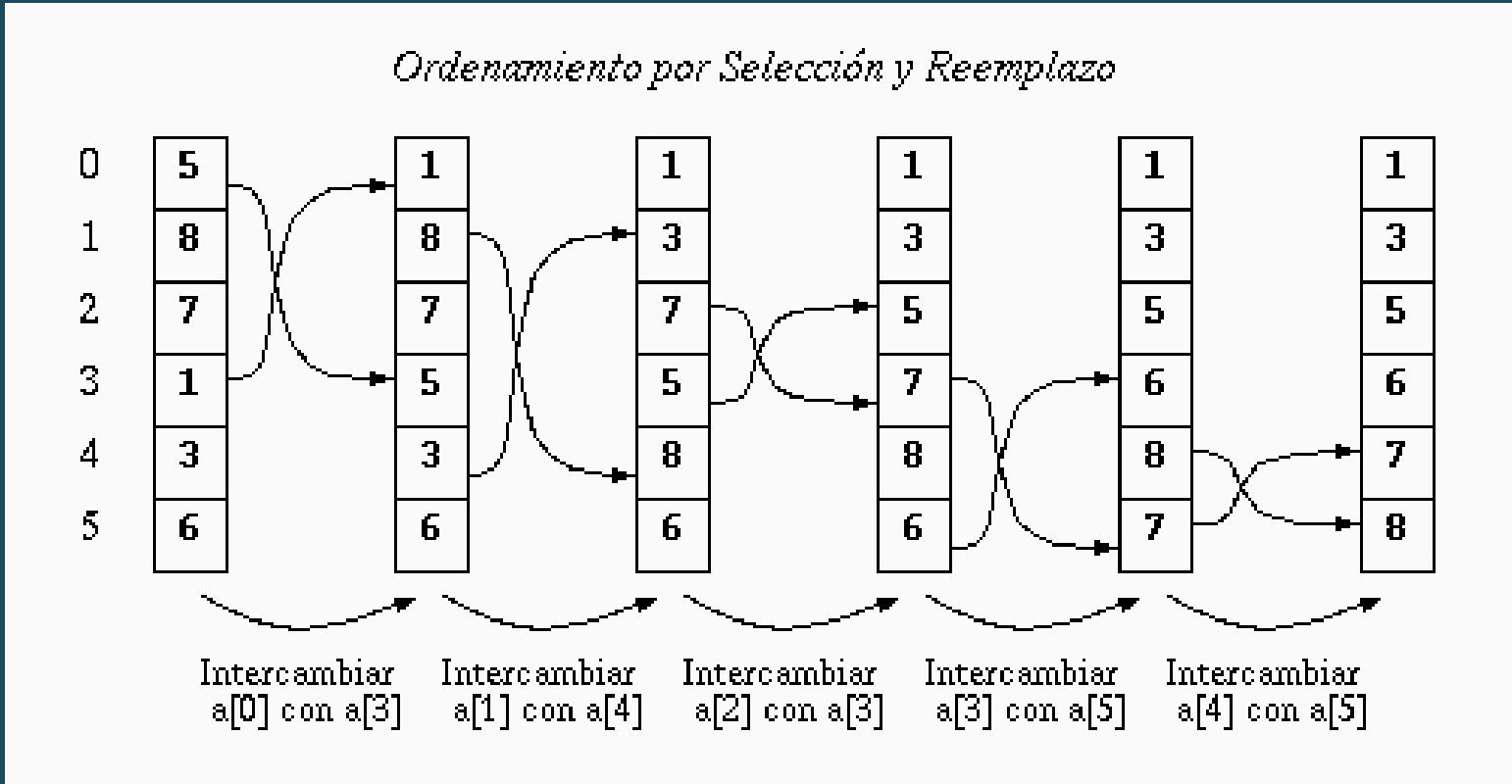
Métodos de Ordenación: Selección

- **Ordenación por selección:** En cada iteración, se selecciona el menor elemento del subvector no ordenado y se intercambia con el primer elemento de este subvector.

```
funcion ordenarSeleccion (numero v[]) {  
    numero tmp;  
    numero i, j, pos_min;  
    numero N = v.length;  
    for (i=0; i<N-1; i++) {  
  
        // Menor elemento del vector v[i..N-1]  
  
        pos_min = i;  
        for (j=i+1; j<N; j++)  
            if (v[j]<v[pos_min])  
                pos_min = j;  
  
        // Coloca el mínimo en v[i]  
        tmp = v[i];  
        v[i] = v[pos_min];  
        v[pos_min] = tmp;  
    }  
}
```



Métodos de Ordenación: Selección



<https://www.youtube.com/watch?v=ZM03Fow05tg>

<https://www.youtube.com/watch?v=Ns4TPTC8whw>



Métodos de Ordenación: Insercción

- **Ordenación por inserción:** En cada iteración, se selecciona el menor elemento del subvector no ordenado y se intercambia con el primer elemento de este subvector.

```
funcion insercionDirecta(int v[]){
    numero p, j;
    numero aux;
    numero TAM = v.length
    for (p = 1; p < TAM; p++){ // desde el segundo elemento hasta
        aux = v[p];           // el final, guardamos el elemento y
        j = p - 1;            // empezamos a comprobar con el anterior
        while ((j >= 0) && (aux < A[j])){ // mientras queden posiciones y el
                                            // valor de aux sea menor que los
            A[j + 1] = A[j];    // de la izquierda, se desplaza a
            j--;                // la derecha
        }
        A[j + 1] = aux;         // colocamos aux en su sitio
    }
}
```



Métodos de Ordenación: Insercción

54 26 93 17 77 31 44 55 20	Se asume que 54 es una lista ordenada de 1 ítem
26 54 93 17 77 31 44 55 20	Se inserta 26
26 54 93 17 77 31 44 55 20	Se inserta 93
17 26 54 93 77 31 44 55 20	Se inserta 17
17 26 54 77 93 31 44 55 20	Se inserta 77
17 26 31 54 77 93 44 55 20	Se inserta 31
17 26 31 44 54 77 93 55 20	Se inserta 44
17 26 31 44 54 55 77 93 20	Se inserta 55
17 20 26 31 44 54 55 77 93	Se inserta 20

30 15 2 21 44 8	Array original
30 15 2 21 44 8	Se empieza por el segundo elemento. Se compara con el primero. Como $15 < 30$ se desplaza el 30 hacia la derecha y se coloca el 15 en su lugar
15 30 2 21 44 8	Seguimos por el tercer elemento. Se compara con los anteriores y se van desplazando hasta que el 2 queda en su lugar.
2 15 30 21 44 8	Continuamos por el cuarto elemento. Se compara con los anteriores y se van desplazando hasta que el 21 queda en su lugar.
2 15 21 30 44 8	Lo mismo para el quinto elemento En este caso ya está en su posición correcta respecto a los anteriores.
2 15 21 30 44 8	Y finalmente se coloca el último elemento El array queda ordenado
2 8 15 21 30 44	

<https://www.youtube.com/watch?v=C-fPGWgeYzY>

<https://www.youtube.com/watch?v=ROalU379l3U>



Métodos de Ordenación: Quicksort

- **Ordenación por Quicksort:** El más rápido de los que veremos.
 1. Se toma un elemento arbitrario del vector, al que denominaremos **pivote** (**p**).
 2. Se **divide** el vector de tal forma que todos los elementos a la izquierda del pivote sean menores que él, mientras que los que quedan a la derecha son mayores que él.
 3. Ordenamos, por separado, las dos zonas delimitadas por el pivote.
 4. Es recursivo, de ahí su gran ventaja. Repetimos el proceso recursivamente conc ad aparte, hasta que al salir todas las llamadas tenemos el vector completo.

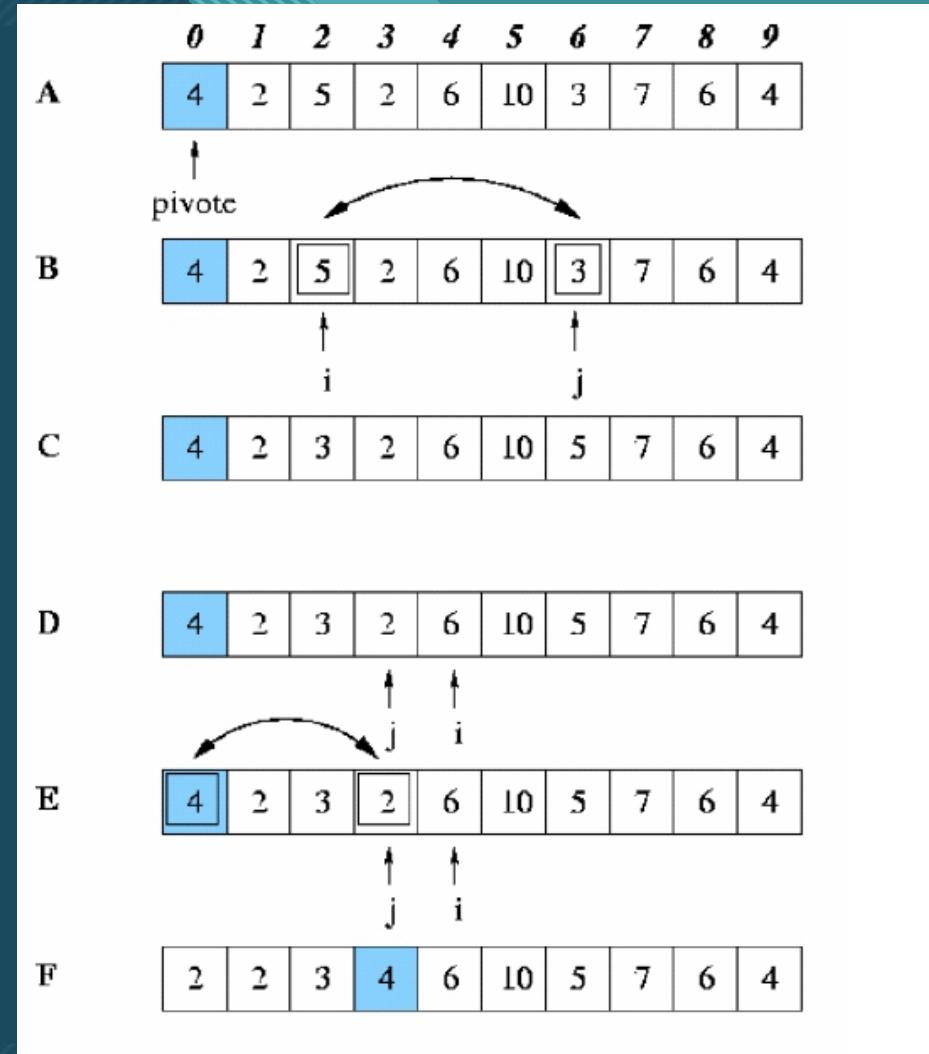
```
funcion quicksort (numero v[], numero izda, numero dcha) {  
    numero pivote; // Posición del pivote  
  
    if (izda<dcha) {  
        pivote = partir (v, izda, dcha);  
        quicksort (v, izda, pivote-1);  
        quicksort (v, pivote+1, dcha);  
    }  
}
```

Uso: quicksort (vector, 0, vector.length-1);



Métodos de Ordenación: Quicksort

- Obtención del pivote
 - Mientras queden elementos mal colocados respecto al pivote:
 1. Se recorre el vector, de izquierda a derecha, hasta encontrar un elemento situado en una posición i tal que $v[i] > p$.
 2. Se recorre el vector, de derecha a izquierda, hasta encontrar otro elemento situado en una posición j tal que $v[j] < p$.
 3. Se intercambian los elementos situados en las casillas i y j (de modo que, ahora, $v[i] < p < v[j]$).





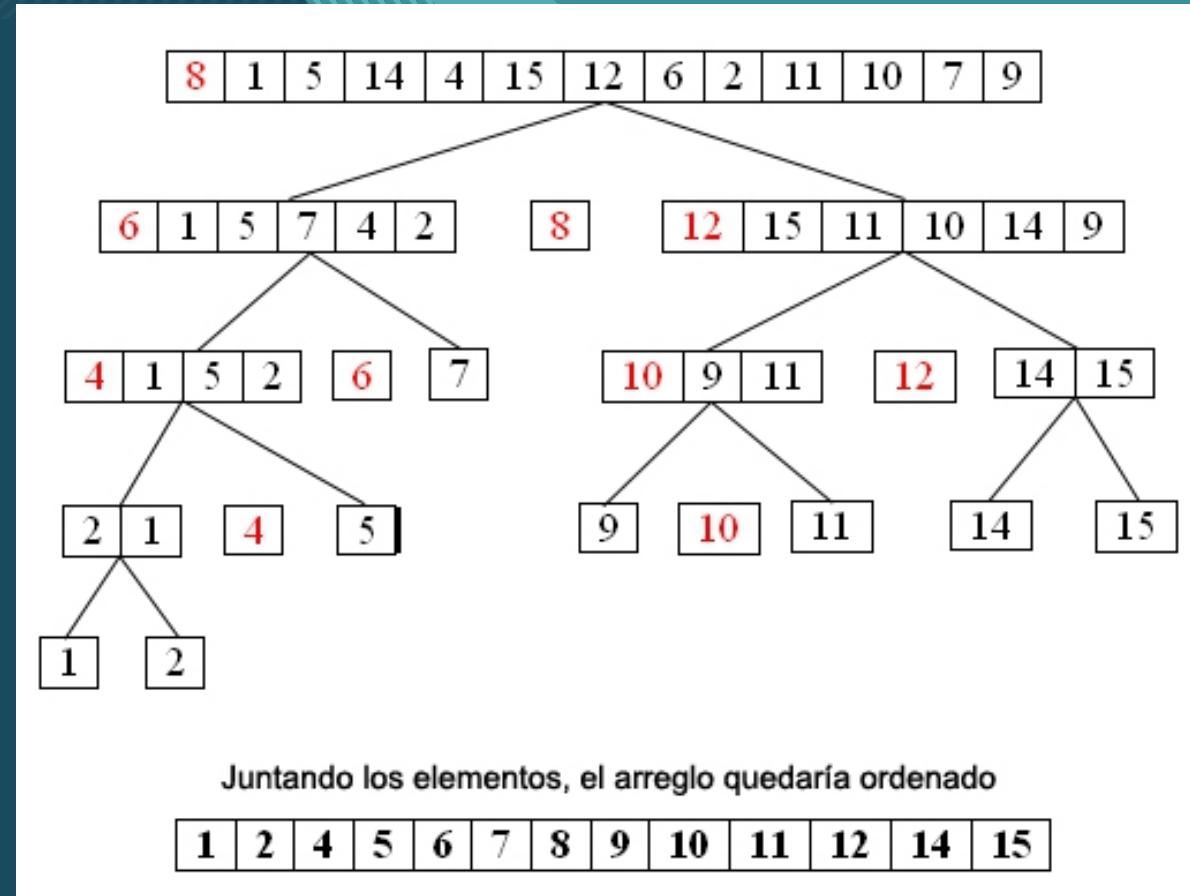
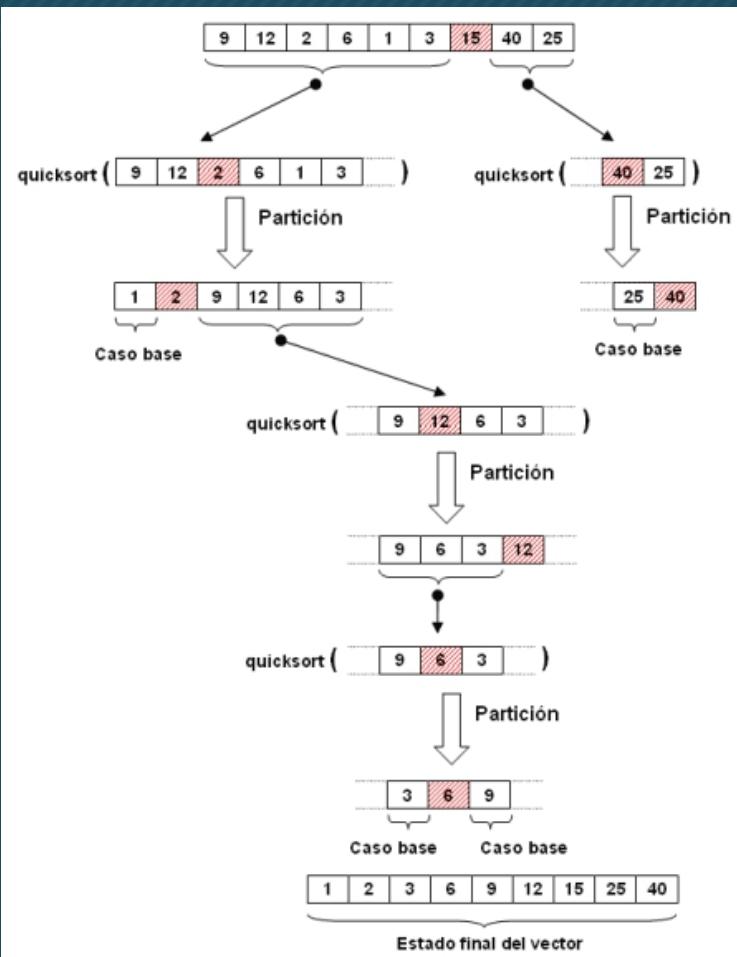
Métodos de Ordenación: Quicksort

- Funcion Partir

```
funcion partir (numero v[], numero primero, numero ultimo): numero {
    numero pivot = v[primero]; // Valor del pivote
    numero temporal;
    // Variable auxiliar
    numero izda = primero+1;
    numero dcha = ultimo;
    do {
        // Pivotear...
        while ((izda<=dcha) && (v[izda]<=pivot))
            izda++;
        while ((izda<=dcha) && (v[dcha]>pivot))
            dcha--;
        if (izda < dcha) {
            temporal = v[izda];
            v[izda] = v[dcha];
            v[dcha] = temporal;
            dcha--;
            izda++;
        }
    } while (izda <= dcha);

    // Colocar el pivote en su sitio
    temporal = v[primero];
    v[primero] = v[dcha];
    v[dcha] = temporal;
    return dcha; // Posición del pivote
}
```

Métodos de Ordenación: Quicksort



<https://www.youtube.com/watch?v=ywWBy6J5gz8>

Métodos de Búsqueda

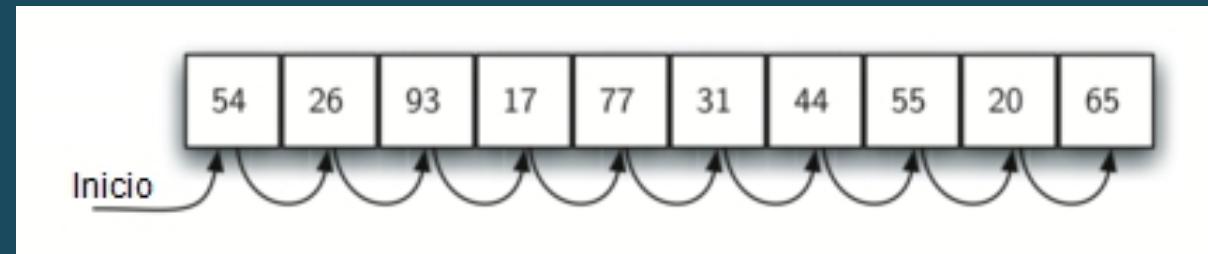
Vamos a buscar elementos en el array



Métodos de Ordenación: Búsqueda secuencial

- Búsqueda lineal de un elemento en un vector
 - Devuelve la posición de “dato” en el vector
 - Si “dato” no está en el vector, devuelve -1

```
funcion buscar (numero vector[], numero dato): numero {  
    int i;  
    int N = vector.length;  
    int pos = -1;  
    for (i=0; i<N; i++)  
        if (vector[i]==dato)  
            pos = i;  
    }  
    return pos;  
}
```

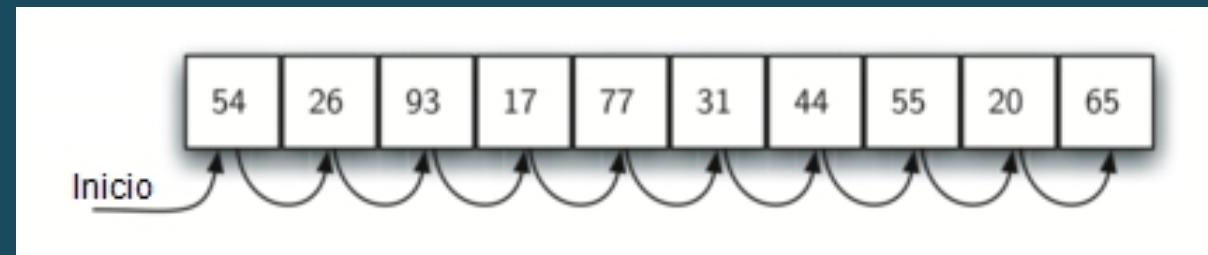




Métodos de Ordenación: Búsqueda secuencial

- Búsqueda lineal de un elemento en un vector
 - Devuelve la posición de “dato” en el vector
 - Si “dato” no está en el vector, devuelve -1

```
funcion buscar (numero vector[], numero dato): numero {  
    int i;  
    int N = vector.length;  
    int pos = -1;  
    for (i=0; (i<N) && (pos== -1); i++)  
        if (vector[i]==dato)  
            pos = i;  
    }  
    return pos;  
}
```

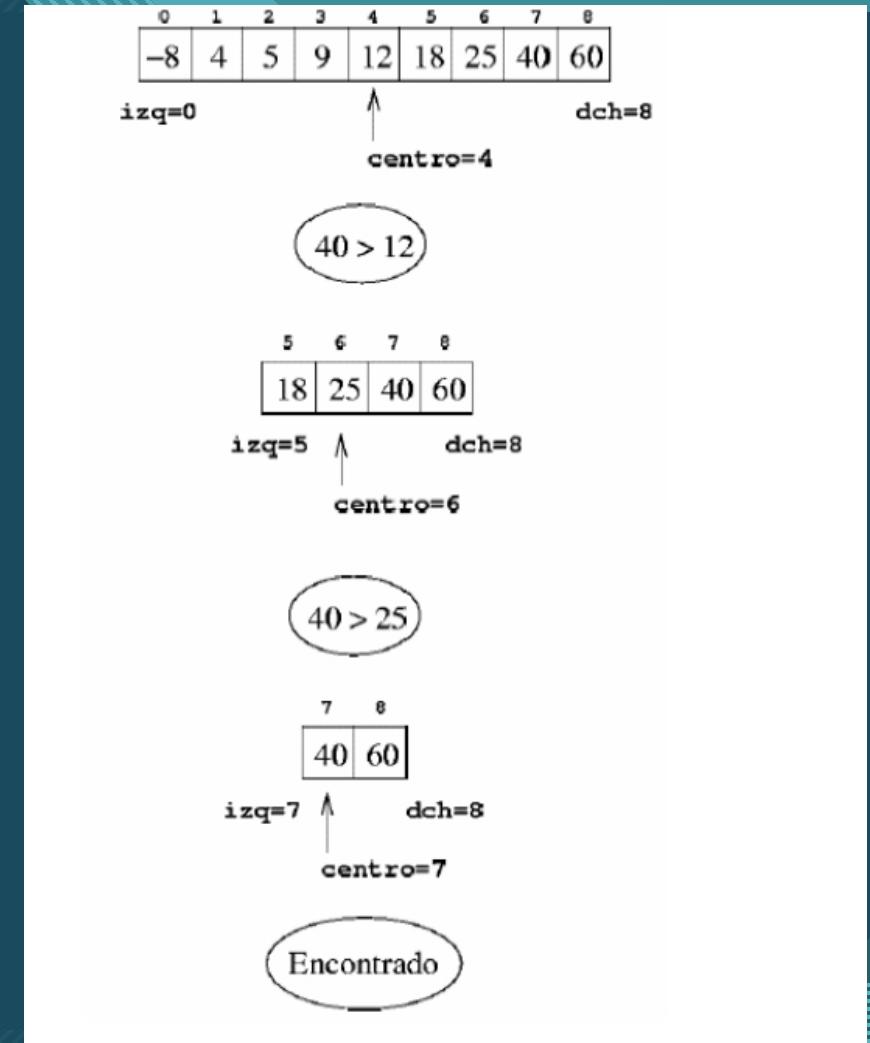


<https://www.youtube.com/watch?v=-PuqKbu9K3U>



Métodos de Ordenación: Búsqueda binaria

- Búsqueda Binaria
 - Precondición
 - El vector ha de estar ordenado
 - Algoritmo
 - Se compara el dato buscado con el elemento en el centro del vector:
 - Si coinciden, hemos encontrado el dato buscado.
 - Si el dato es mayor que el elemento central del vector, tenemos que buscar el dato en segunda mitad del vector.
 - Si el dato es menor que el elemento central del vector, tenemos que buscar el dato en la primera mitad del vector.





Métodos de Ordenación: Búsqueda binaria

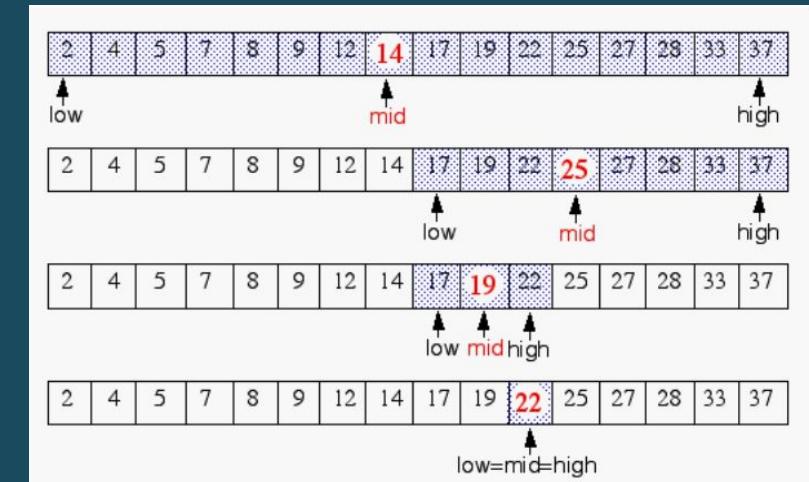
```
// Implementación iterativa
// Uso: binSearch (vector, dato)
funcion binSearch (numero v[], numero buscado): numero {
    numero izq = 0;
    numero der = v.length-1;
    numero centro = (izq+der)/2;
    while ((izq<=der) && (v[centro]!=buscado)) {
        if (buscado<v[centro])
            der = centro - 1;
        else
            izq = centro + 1;
        centro = (izq+der)/2;
    }
    if (izq>der)
        return -1;
    else
        return centro;
}
```



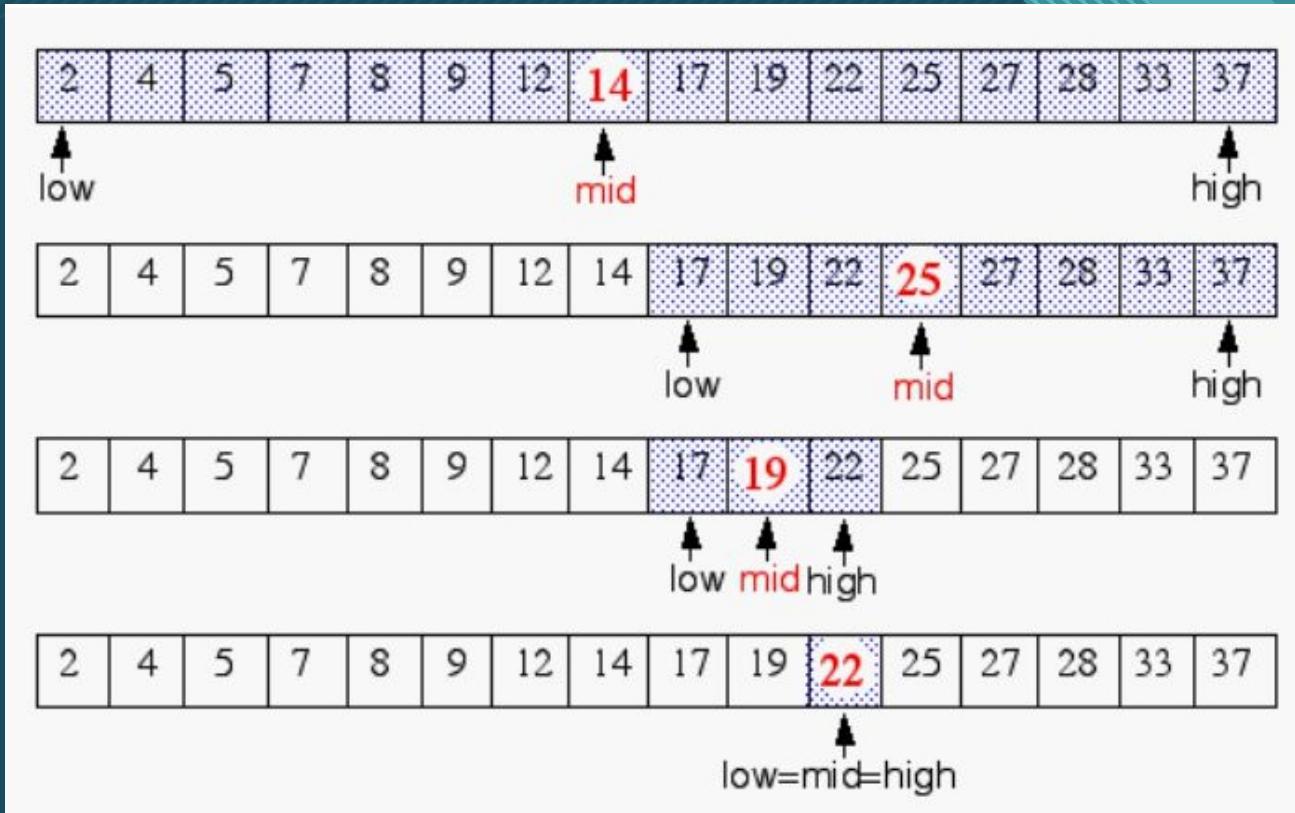
Métodos de Ordenación: Búsqueda binaria

```
// Implementación recursiva
// Uso: binSearch(vector,0,vector.length-1,dato)
funcion binSearch (numero v[], numero izq, numero der, numero buscado): numero {
    numero centro = (izq+der)/2;

    if (izq>der)
        return -1;
    else if (buscado==v[centro])
        return centro;
    else if (buscado<v[centro])
        return binSearch(v, izq, centro-1, buscado);
    else
        return binSearch(v, centro+1, der, buscado);
}
```



Métodos de Ordenación: Búsqueda binaria



<https://www.youtube.com/watch?v=iP897Z5Nerk>

“

"Cualquier código tuyo que no hayas
mirado en los últimos seis meses o más
es como si lo hubiese escrito otro"

- *Eagleson's Law*

”



Recursos

- Twitter: <https://twitter.com/joseluisgonsan>
- GitHub: <https://github.com/joseluisgs>
- Web: <https://joseluisgs.github.io>
- Discord: <https://discord.gg/uv7GcytM>
- Aula Virtual: <https://aulavirtual33.educa.madrid.org/ies.luisvives.leganes/course/view.php?id=245>



Gracias

José Luis González Sánchez

