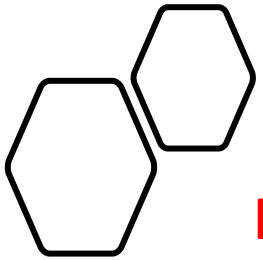
Sesión 07

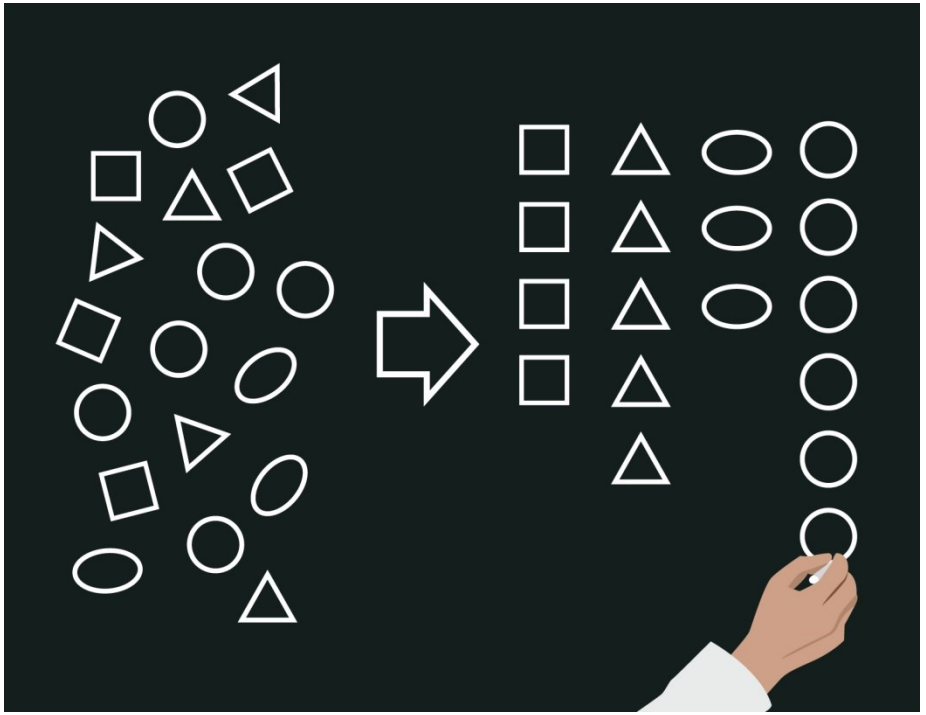


ESTRUCTURA DE DATOS

- Métodos de Ordenamientos
  - Simples
  - Logarítmicos

Métodos de Ordenamiento

El ordenar un grupo de datos significa mover los datos o sus referencias para que queden en una secuencia tal que represente un orden, el cual puede ser numérico, alfabético o incluso alfanumérico, ascendente o descendente.



Métodos de Ordenamiento

Los métodos simples son (Directos):

- Burbuja
- Selección
- Inserción
- Intercambio

Los métodos más complejos son (Logarítmicos)

- Shell sort
- Quick-sort
- Merge Sort
- Radix

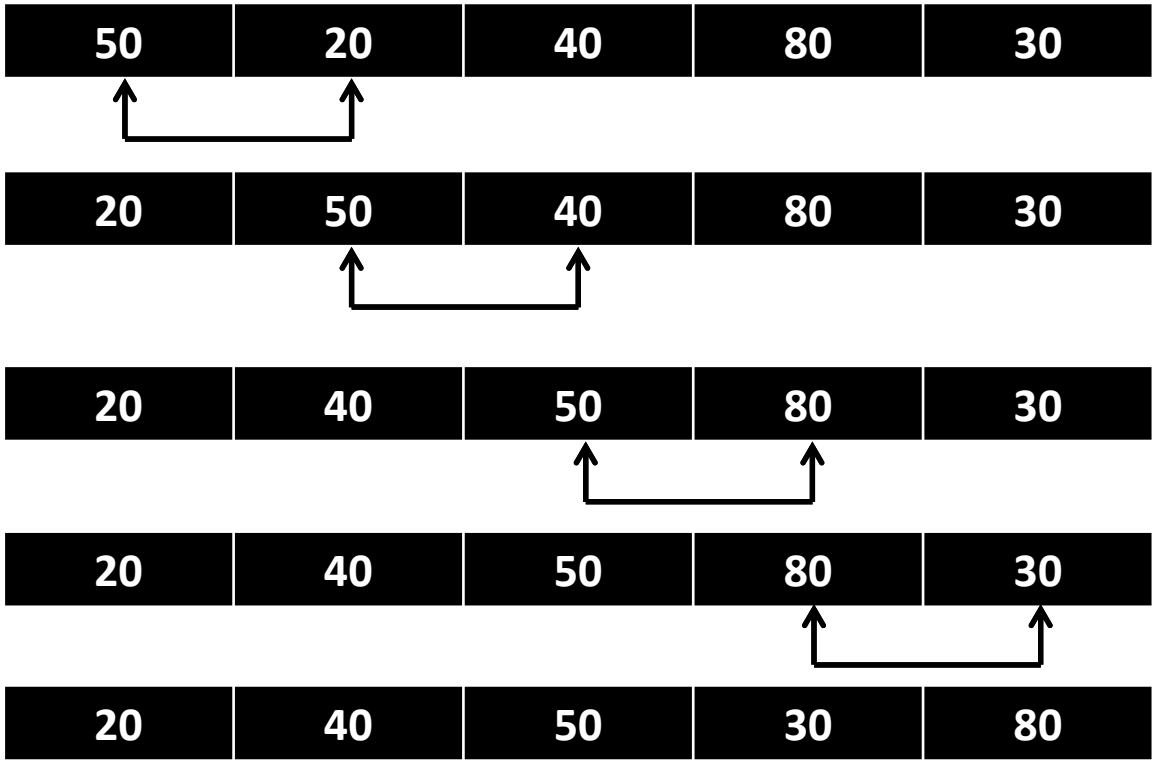
MÉTODO DE ORDENAMIENTO BURBUJA

Método Burbuja

- La **Ordenación de burbuja** (**Bubble Sort** en inglés) es un sencillo algoritmo de ordenamiento.
- Funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado.
- Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada.
- Este algoritmo obtiene su nombre de la forma con la que suben por la lista los elementos durante los intercambios, como si fueran pequeñas "burbujas".
- También es conocido como el **método del intercambio directo**. Dado que solo usa comparaciones para operar elementos
- Es considerado un algoritmo de comparación, siendo el más sencillo de implementar.

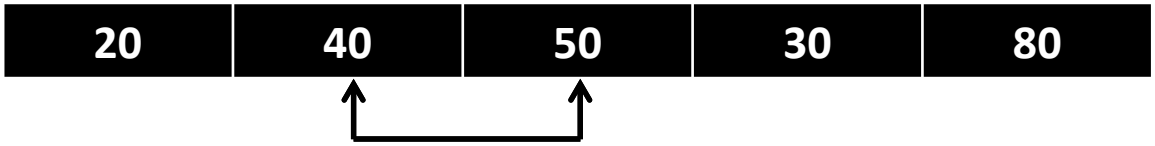
Ejemplo:

- Elementos (A = 50, 20, 40, 80, 30), donde se introduce una variable interruptor para detectar si se ha producido intercambio en la pasada.
- **Pasada 0**

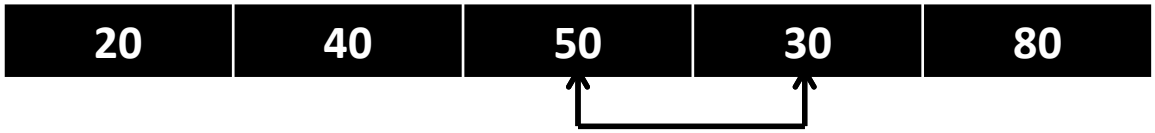


- **Pasada 1**

20 y 40 ordenados



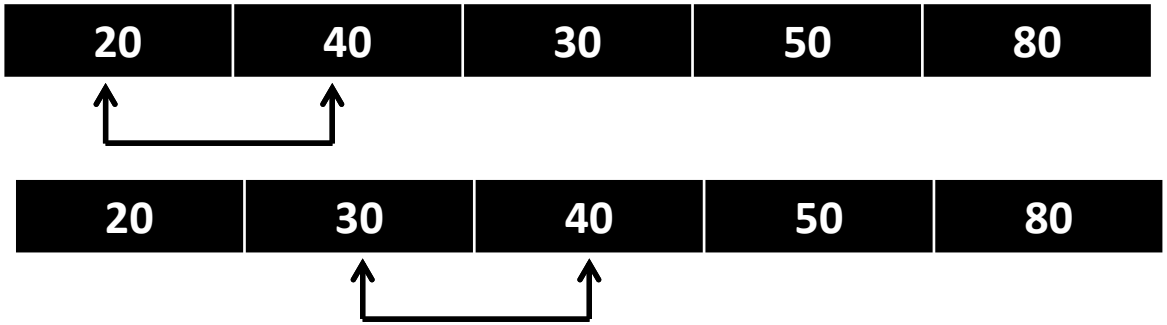
40 y 50 ordenados



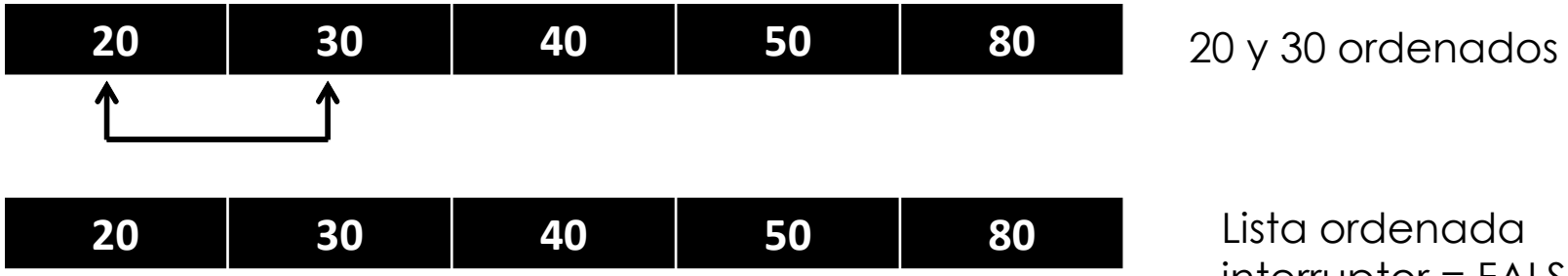
Se intercambian 50 y 30

- 50 y 80 elementos mayores y ordenados
- interruptor = TRUE

- **Pasada 2.- Solo se hacen dos comparaciones.**



- **Pasada 3.-** Se hace una única comparación de 20 y 30, y no se produce intercambio:

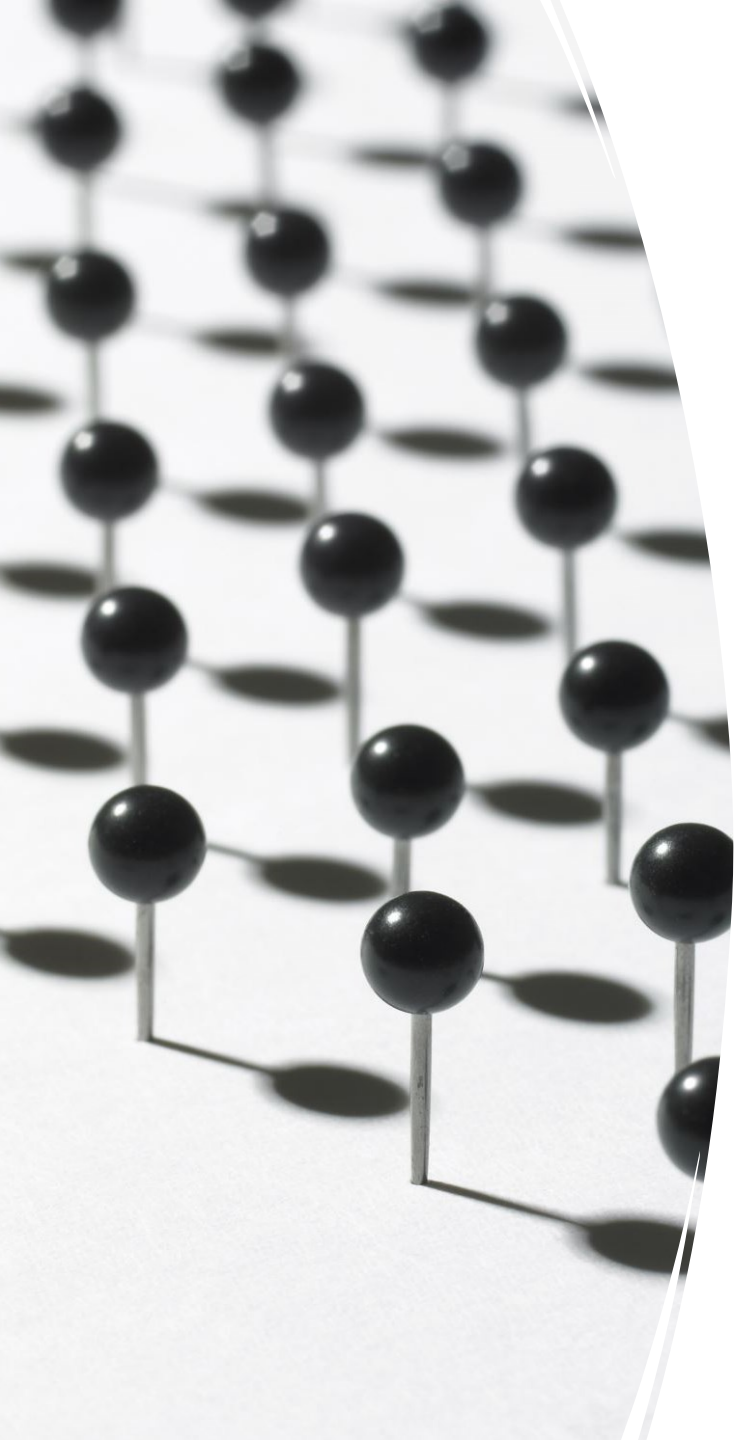


```
1  static void BurbujaEnteros(int[] A)
2          {
3          int n = A.Length;
4          int iaux;
5          for (int i=0; i < n-1;i++)
6              {
7                  for (int j = 0; j < n - i - 1;j++ )
8                  {
9                      if (A[j] > A[j + 1])
10                     {
11                         iaux = A[j];
12                         A[j] = A[j + 1];
13                         A[j + 1] = iaux;
14                     }
15                 }
16             }
17         }
```

```
1  static void BurbujaEnteros(int[] A)
2          {
3              int n = A.Length;
4              int iaux;
5              int i=1;
6              bool intercambios;
7              do
8              {
9                  intercambios = false;
10                 for (int j = 1; j <= n - i; j++)
11                 {
12                     if (A[j - 1] > A[j])
13                     {
14                         iaux = A[j - 1];
15                         A[j - 1] = A[j];
16                         A[j] = iaux;
17                         intercambios = true;
18                     }
19                 }
20                 i++;
21             } while (intercambios && i <= n - 1);
22         }
```

Ejercicio 6.1

- Elaborar un algoritmo (psudocodigo o lenguaje C++) usando estructuras dinámicas, el cual permita ordenar un conjunto de números enteros, utilizando el método Burbuja.



Método de Ordenamiento por Selección

Ordenamiento por Selección

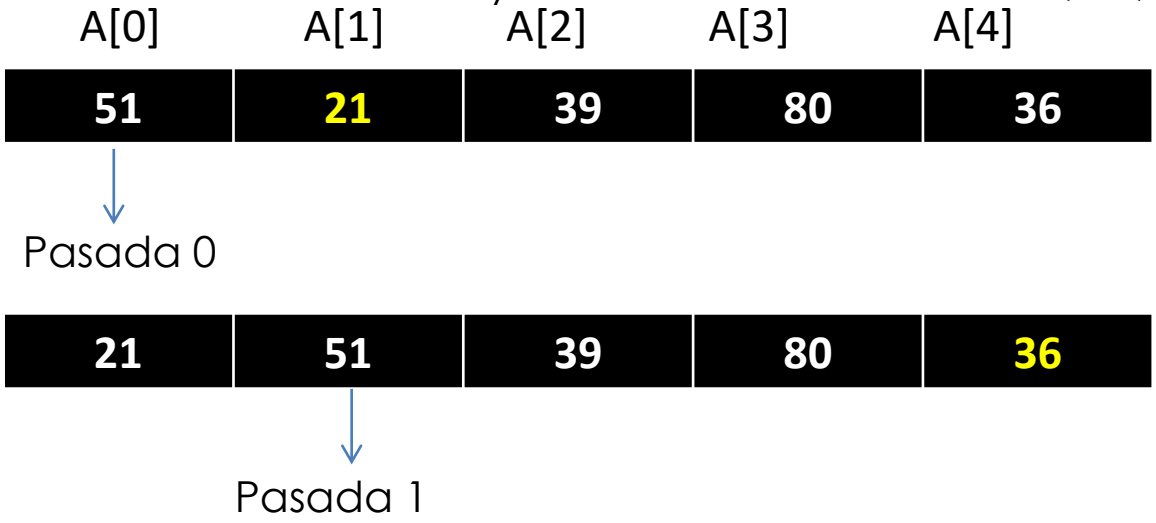
Consiste en lo siguiente:

- Buscar el elemento más pequeño de la lista.
- Se intercambia con el elemento ubicado en la primera posición de la lista.
- Buscar el segundo elemento más pequeño de la lista.
- Se intercambia con el elemento que ocupa la segunda posición en la lista.
- Se repite este proceso hasta que esté ordenada toda la lista.

Método Selección

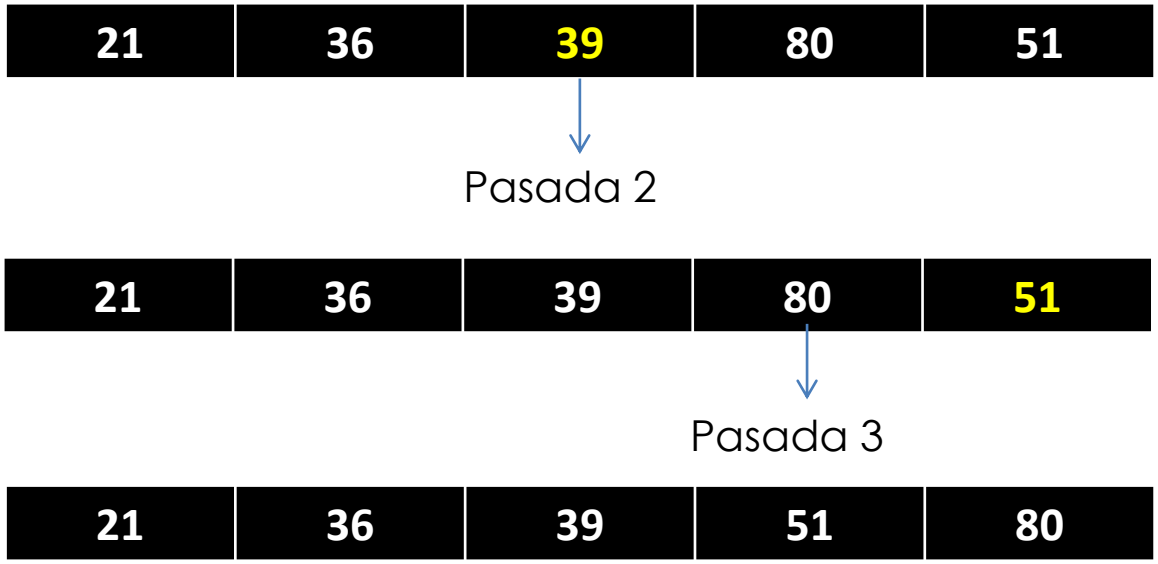
- Los métodos de ordenación por selección se basan en dos principios básicos: Seleccionar el elemento más pequeño (o más grande) del arreglo. Colocarlo en la posición más baja (o más alta) del arreglo. A diferencia del método de la burbuja, en este método el elemento más pequeño (o más grande) es el que se coloca en la posición final que le corresponde.

- Consideremos un array A con 5 valores enteros 51, 21, 39, 80, 36:



Pasada 0.
Seleccionar 21
Intercambiar 21 y A[0]

Pasada 1.
Seleccionar 36
Intercambiar 36 y A[1]



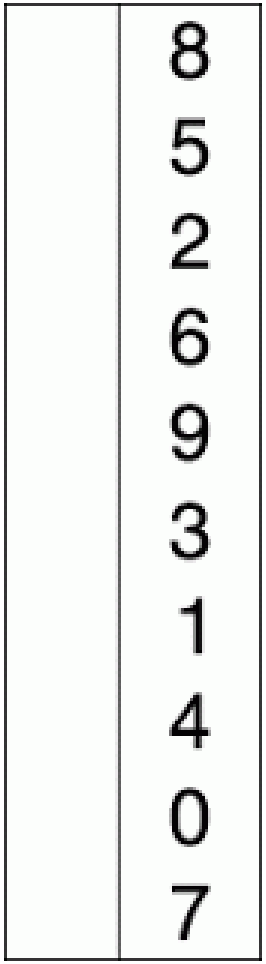
Pasada 2.
Seleccionar 39
Intercambiar 39 y A[2]

Pasada 3.
Seleccionar 51
Intercambiar 51 y A[3]

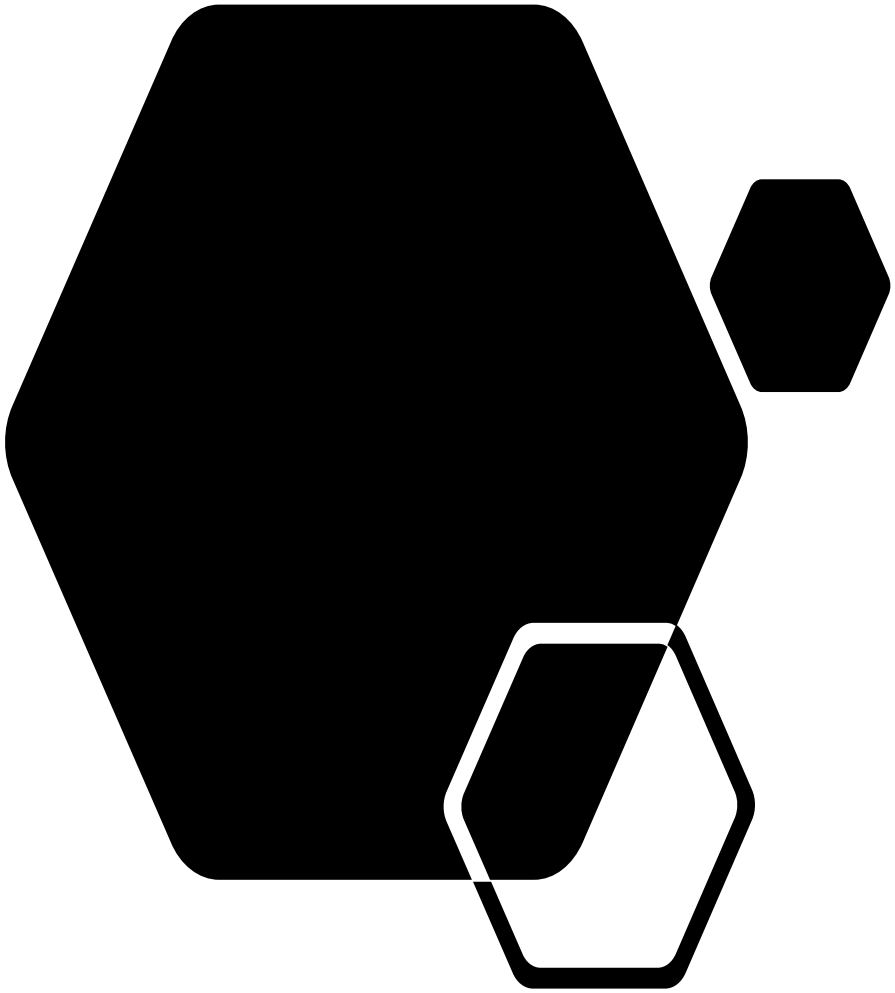
Lista ordenada

Pseudocodigo

```
para i=1 hasta n-1
    minimo = i;
    para j=i+1 hasta n
        si lista[j] < lista[minimo]
entonces
            minimo = j /* (!) */
        fin si
    fin para
    intercambiar(lista[i],
lista[minimo])
fin para
```



Método de Ordenamiento por Inserción

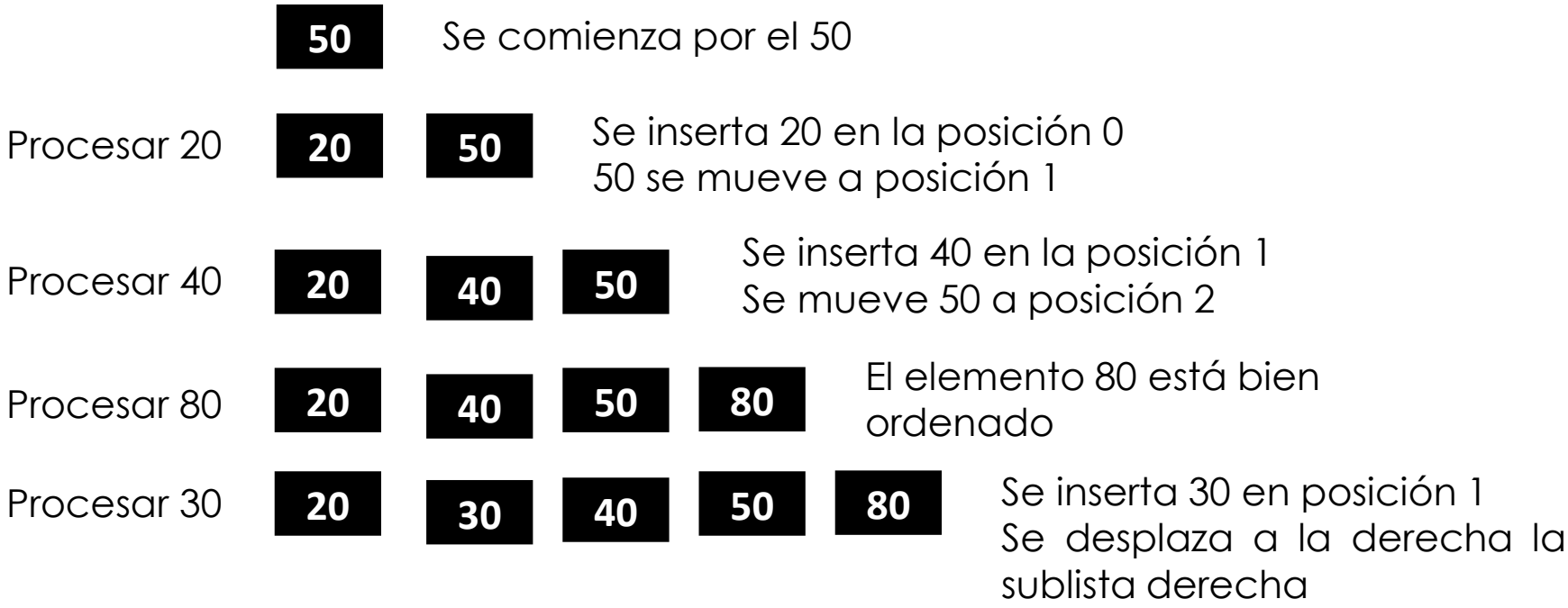


Ordenamiento por Inserción

- El algoritmo consiste en realizar varias pasadas sobre la lista de elementos.
- En cada pasada se analiza un elemento, y se intenta encontrar su orden relativo entre los analizados en pasadas anteriores.
- Cada elemento a analizar se desplaza por esa lista hasta encontrar su lugar.
- Cuando todos los elementos de la lista han sido analizados, la lista está completamente ordenada

Método Inserción

- El método de ordenación por inserción es similar al proceso típico de ordenar tarjetas de nombres (cartas de una baraja) por orden alfabético, que consiste en insertar un nombre en su posición correcta dentro de una lista o archivo que ya está ordenado.
- Así el proceso en el caso de la lista de enteros A = 50, 20, 40, 80, 30.



Ordenamiento por Inserción

6 5 3 1 8 7 2 4

Insercion

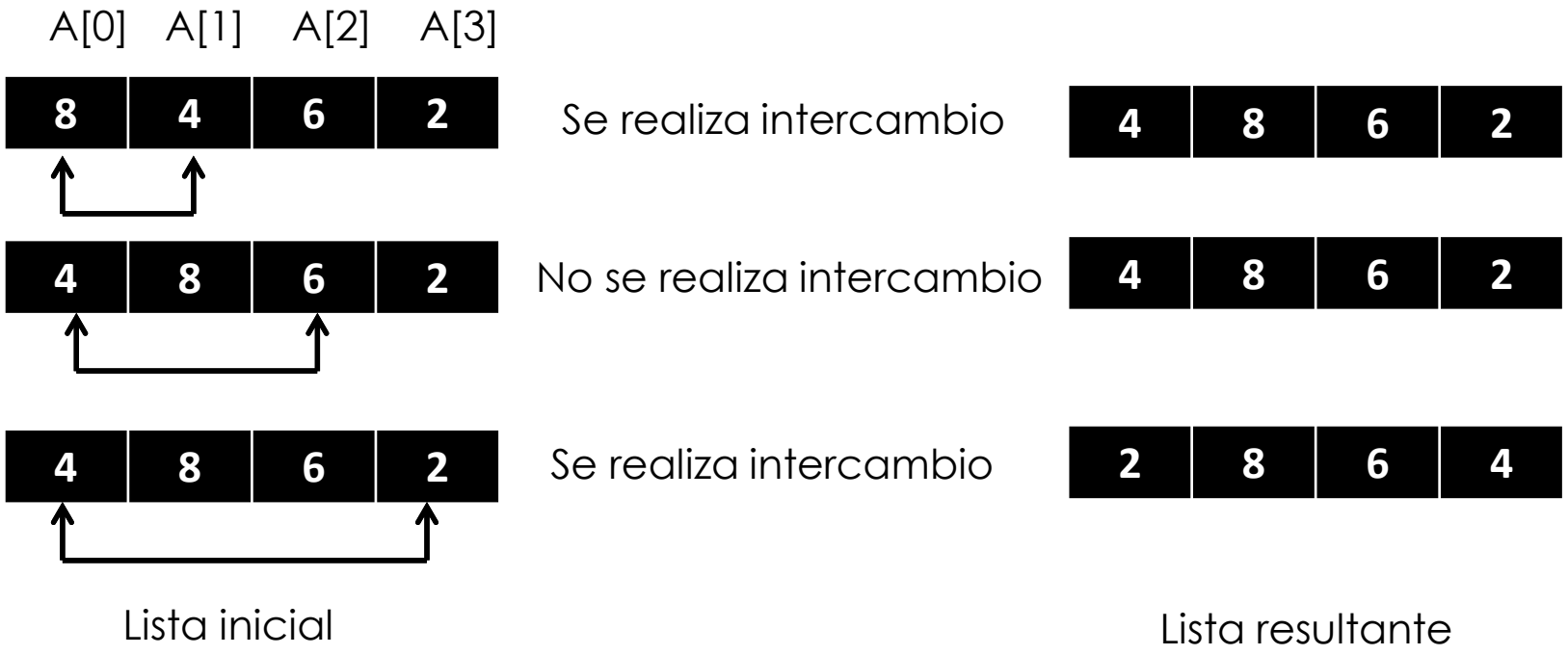
```
#include<stdio.h>
#include<conio.h>
int a[4]={4,1,7,2};
int n=4;
int i,j,aux;
void main(){
clrscr();
for(i=1;i<n;i++)
{
j=i;
aux=a;
while(j>0 && aux<a[j-1])
{
a[j]=a[j-1];
j--;
}
a[j]=aux;
}
for(i=0;i<4;i++)
{
printf("%d",a);
}
getch();
}
```

Método de Ordenamiento por Intercambio

Método Intercambio

- Se encarga de ordenar los elementos de una lista en orden ascendente. Este algoritmo se basa en la lectura sucesiva de la lista a ordenar, comparando el elemento inferior de la lista con los restantes y efectuando intercambio de posiciones cuando el orden resultante de la comparación no sea el correcto.

Pasada 0



- **Pasada 1**

Pasada 2

La sublista a considerar ahora es 8, 6 ya que 2, 4 está ordenada. Una comparación única se produce entre los dos elementos de la sublista

Para simular esto en un programa necesitamos tener en cuenta algo: no podemos desplazar los elementos así como así o se perderá un elemento. Lo que hacemos es guardar una copia del elemento actual y desplazar todos los elementos mayores hacia la derecha. Luego copiamos el elemento guardado en la posición del último elemento que se desplazó.

```
Insercion(int lista[],n)
{
        int i, temp, j;
        for (i = 1; i < n; i++)
        {
        temp = lista[i];
        j = i - 1;
        while ( (lista[j] > temp) && (j >= 0) )
        {
                lista[j + 1] = lista[j];
                j--;
        } // while
        lista[j + 1] = temp;
        } // for
}
```

Nombre	Tipo	Uso
lista	Cualquiera	Lista a ordenar
TAM	Constante Entera	Tamaño de la lista
i	Entero	Contador
j	Entero	Contador
temp	El mismo que los elementos de la lista	Para realizar los intercambios

```
1. for (i=1; i<TAM; i++)
2.       temp = lista[i];
3.       j = i - 1;
4.       while ( (lista[j] > temp) && (j >= 0) )
5.            lista[j+1] = lista[j];
6.            j--;
7.       lista[j+1] = temp;
```

Nota: Observa que en cada iteración del ciclo externo los elementos 0 a i forman una lista ordenada.

Métodos de Búsquedas

- Búsqueda Secuencial
- Búsqueda Binaria
- Búsqueda por Funciones de HASH

Métodos de búsqueda.

- La *búsqueda* permite recuperar datos previamente almacenados.
- El resultado de una búsqueda puede ser un éxito, si se encuentra la información o un fracaso, si no la encuentra.
- La búsqueda se puede aplicar sobre elementos previamente ordenados o sobre elementos desordenados, en el primer caso la búsqueda es más fácil, en cambio en el segundo se dificulta un poco más el proceso, sobre todo cuando de se trata de encontrar una cantidad de elementos similares.
- Los métodos de búsqueda se clasifican en:
  - Búsqueda interna.
  - Búsqueda externa.

Búsqueda interna.

Donde todos los elementos de la estructura estática (arreglo) o dinámica (listas o árboles) se encuentran almacenados en la memoria principal de la computadora.

Los métodos de búsqueda interna más importantes son:

- Secuencial o lineal.
- Binaria.
- Hash (transformación de claves)

Método Secuencial.

- El método de *búsqueda secuencial* consiste en revisar la estructura de datos elemento por elemento hasta encontrar el dato que estamos buscando, o hasta llegar al final de la estructura de datos.
- Normalmente cuando una función de búsqueda concluye con éxito, lo que interesa es conocer en qué posición fue encontrado el elemento buscado.
- La búsqueda secuencial se puede aplicar a estructuras de datos ordenadas o desordenadas.
- Si se aplica a una estructura desordenada y el elemento que se está buscando existe más de una vez en la estructura, el proceso de búsqueda debe continuar hasta que se llegue al fin de la estructura.

Ejemplo

Ejemplo. Si tenemos una estructura con los elementos 5, 8, 3, 2, 9, 5, 7, 0, 5, 1 y estamos buscando el número 5, el resultado de la búsqueda nos mostraría las posiciones 0, 5 y 8 y el proceso terminaría al llegar al numero 1 que es el ultimo de la lista de elementos.

Elementos	5	8	3	2	9	5	7	0	5	1
Posiciones	0	1	2	3	4	5	6	7	8	9
Posiciones donde encontró el número 5	√	×	×	×	×	√	×	×	√	×

Con una estructura ordenada al encontrar el elemento por primera vez podemos suponer que una vez que el elemento ya no sea igual al que estamos buscando, ya no es necesario llegar hasta el fin de la estructura.

Ejemplo. Si tenemos la estructura anterior pero ordenada 0, 1, 2, 3, 5, 5, 5, 7, 8, 9 y estamos buscando el mismo número 5, el resultado de la búsqueda nos mostraría las posiciones 4, 5, y 6, y el proceso terminaría ya que el número 7 no es menor ni igual al que estamos buscando.

Elementos	0	1	2	3	5	5	5	7	8	9
Posiciones	0	1	2	3	4	5	6	7	8	9
Posiciones donde encontró el número 5	×	×	×	×	√	√	√	×		

Ejercicios de Búsqueda Secuencial

- *Ejercicio 1.* Crear un programa que aplique una búsqueda secuencial de un dato dentro de un arreglo de elementos desordenados y presente la o las posiciones donde encontró el dato.

Método Binario

- El método de *búsqueda binaria* divide el total de los elementos en dos, comparando el elemento buscado con el central, en caso de no ser iguales, se determina si el elemento buscado es menor o mayor al central, para determinar si la búsqueda continua del lado izquierdo (menor) o derecho (mayor) del central, repitiendo el mismo proceso de división y comparación, hasta encontrar el elemento buscado o que la división ya no sea posible.
- Debemos destacar que este método de búsqueda solo funciona con estructuras de datos previamente ordenadas, dividiendo cada vez a la mitad el proceso de búsqueda, lo que hace que el método sea más eficiente.

Ejemplo. Si tenemos una estructura ordenada 0, 1, 2, 3, 5, 5, 5, 7, 8, 9 y estamos buscando el número 5, el resultado de la búsqueda nos mostraría la posicion 4 y el proceso terminaría ya que el elemento buscado no es diferente al que esta en la posición central.

Este proceso debe sumar la posición inicial y la final, dividiendo el resultado de la suma entre dos para obtener la posición central generada por el cociente de la división, en este caso es (0+9)/2 = 4, esta posición se compara con el elemento que estamos buscando y como son iguales la búsqueda se detiene mostrando la posición donde lo encontró.

Elementos	0	1	2	3	5	5	5	7	8	9
Posiciones	0	1	2	3	4	5	6	7	8	9
Posiciones donde encontró el número 5	i				√					F

Ejemplo Búsqueda Binaria

Se desea buscar el elemento 225 y ver si se encuentra en el conjunto de datos siguiente:

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
13	44	75	100	120	275	325	510

El punto central de la lista es el elemento a[3] (100). El valor que se busca es 225, mayor que 100; por consiguiente, la búsqueda continúa en la mitad superior del conjunto de datos de la lista, es decir, en la sublista,

a[4]	a[5]	a[6]	a[7]
120	275	325	510

Ejemplo Búsqueda Binaria

Ahora el elemento mitad de esta sublista a[5] (275).

El valor buscado, 225, es menor que 275 y, por consiguiente, la búsqueda continúa en la mitad inferior del conjunto de datos de la lista actual; es decir, en la sublista de un único elemento:

a[4]
120

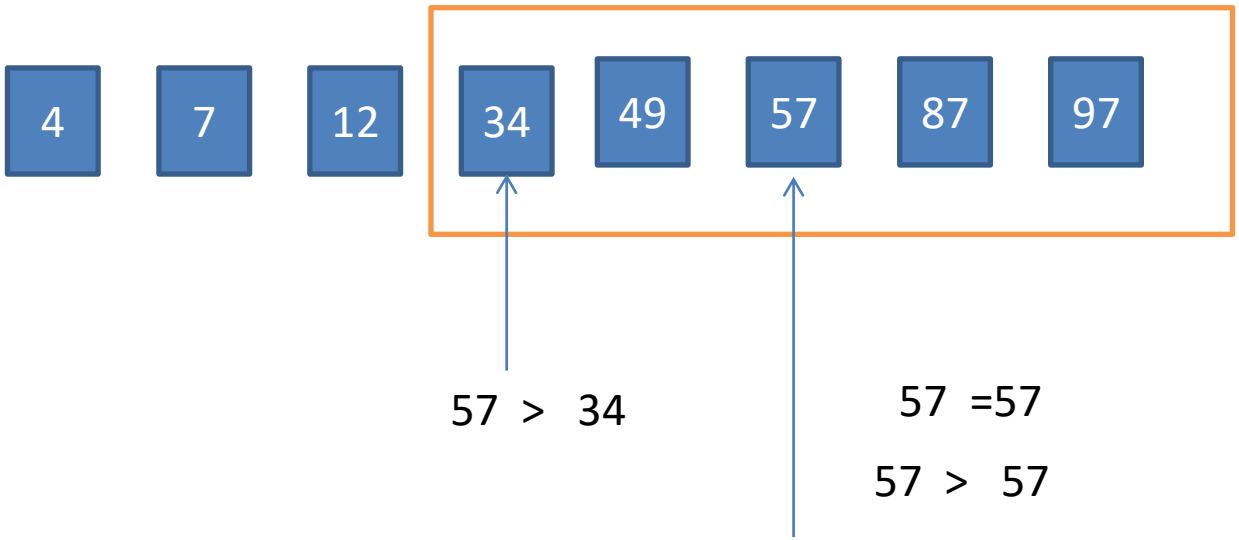
El elemento mitad de esta sublista es el propio elemento a[4] (120). Al ser 225 mayor que 120, la búsqueda debe continuar en una sublista vacía. Se concluye indicando que no se ha encontrado la clave en la lista.

Codificación de Método de Búsqueda Binaria

```
int busq_bin(int a[], int n, int nx)
{
int central,bajo,alto,valorCentral;
bajo=1;
alto=n-1;
while(bajo<=alto)
    {
    central=(bajo+alto)/2;
    valorCentral=a[central];
    if(nx==valorCentral)
      return(central);
    else if(nx<valorCentral)
      alto=central-1;
    else
      bajo=central+1;
    }
    return-1;
}
```

Ejercicio con Búsqueda Binaria

1. Ingresar elementos aplicando un metodo de ordenamiento
2. Buscar un elemento de la lista aplicando el metodo Binario



Métodos de Ordenamiento Logarítmicos

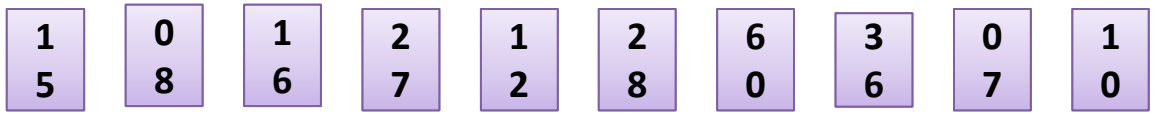
Otros Métodos de Ordenamiento(Logarítmicos)

- Shell Sort
- Quick Sort
- Merge Sort
- Radix

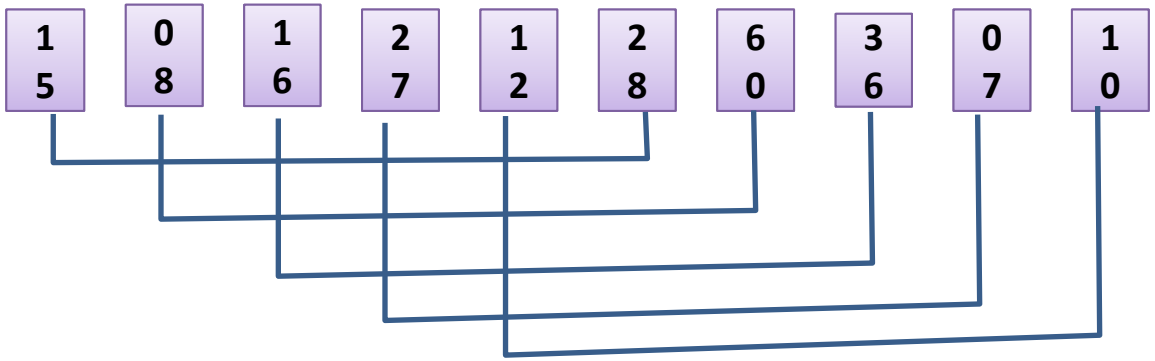
Ordenación Shell (Sort Shell)

- Llamado también Método de inserción con incrementos decrecientes
- Es una mejora del método de inserción directa que se utiliza cuando el número de elementos a ordenar es grande.
- Shell sort lleva este nombre en honor a su inventor, Donald Shell, que lo publicó en 1959.
- La idea básica de este método es distribuir el arreglo de manera que se genere una matriz de valores
- Cada elemento es comparado de manera adyacente empleando un mecanismo de inserción directa simple, dicho rango que genera grupos de manera matricial que es reducido gradualmente hasta estabilizarse en un valor uniforme de 1.

Ejemplo:
Se desean ordenarse las siguientes clave del arreglo



Se selecciona el nr de salto con lo siguiente:
10 elementos entre 2 igual a 5
Se compara cada elemento con su correspondiente salto de posición + 5 ejemplo:



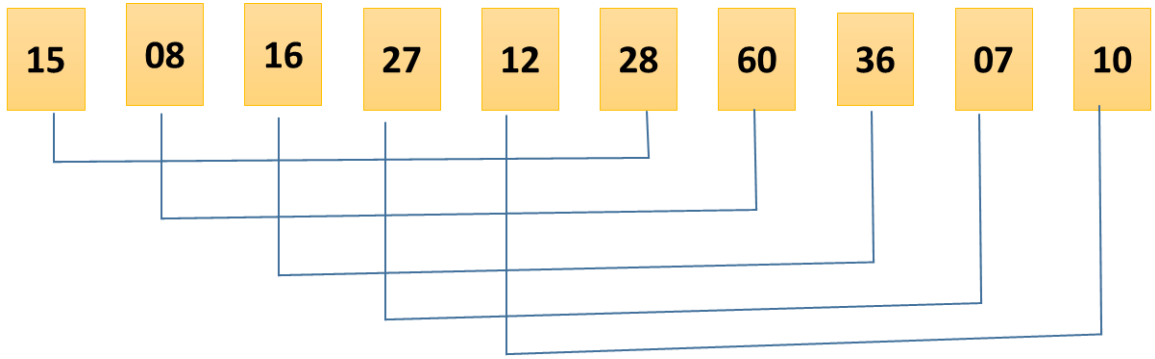
15 < 28 Verdadero

08 < 60 Verdadero

16 < 36 Verdadero

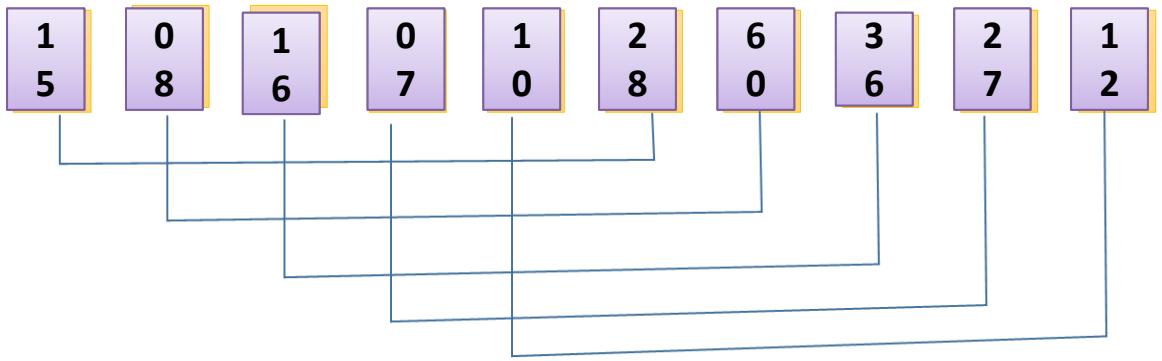
27 < 07 Falso Intercambia

12 < 10 Falso Intercambia



15 08 16 07 10 28 60 36 27 12

15	<	28	Verdadero	
08	<	60	Verdadero	
16	<	36	Verdadero	
07	<	27	Falso	Intercambia
10	<	12	Falso	Intercambia



Se Obtiene un nuevo intervalo de salto, tomando como referencia el primero 5 entre 2 y se toma el valor entero

5/2=2.5 => el salto será 2 numeros

15 < 16

SI

08 < 07

NO → Intercambia

10 < 16

SI

08 < 28

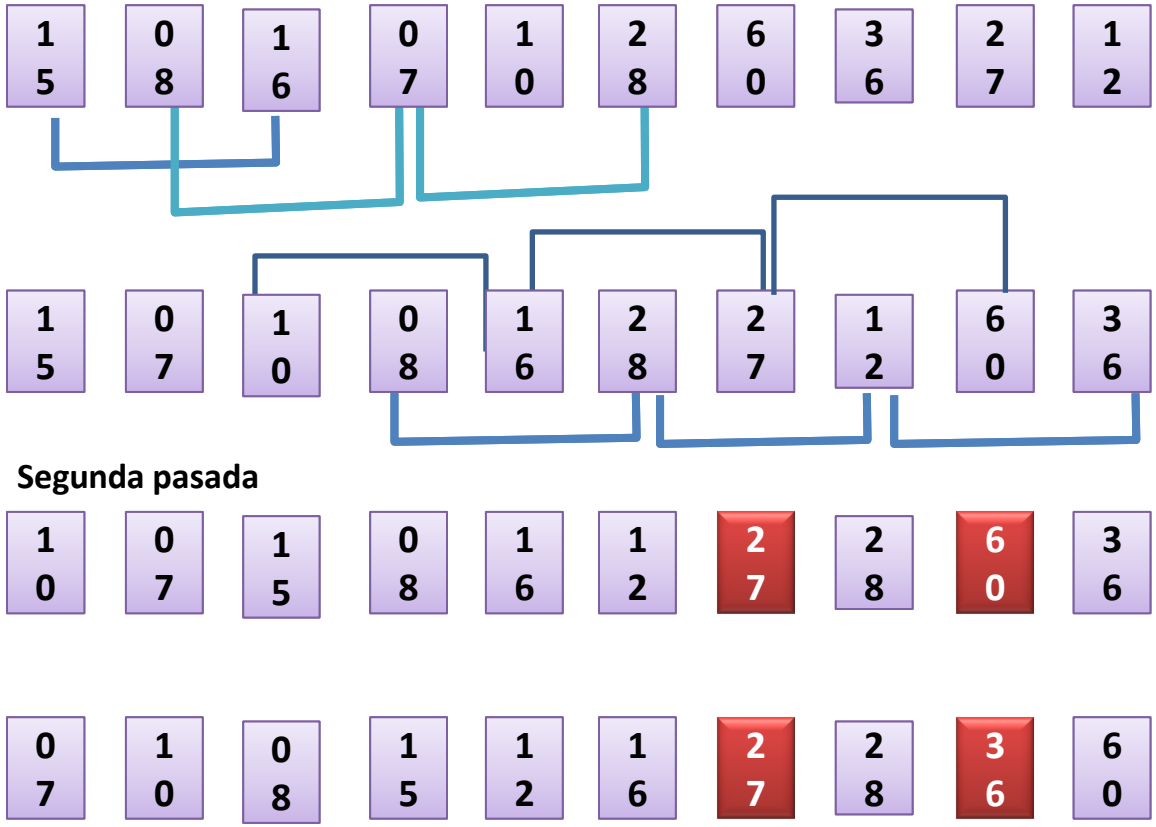
16 < 27

28 < 12

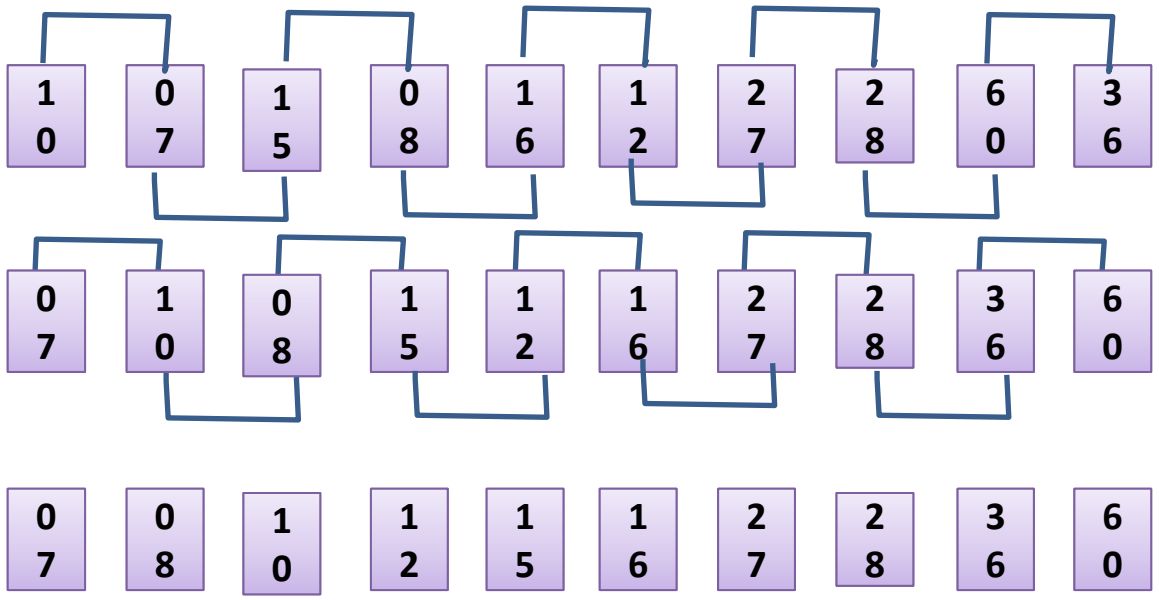
NO → Intercambia

27 < 60

28 < 36



Se hacen comparaciones con saltos de uno, se continua haciendo hasta que no se realice ningún cambio



Se hace un ultimo recorrido, si no hubieron cambios entonces la lista esta ordenada

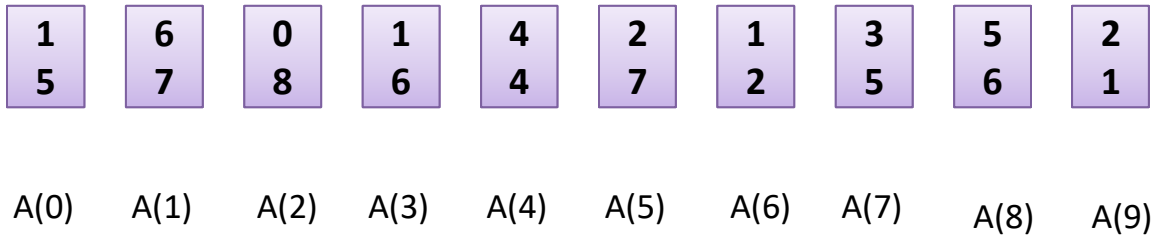
QUICK SORT

El método de ordenamiento Quick Sort es actualmente el más eficiente y veloz de los métodos de ordenación interna. Es también conocido con el nombre del método rápido y de ordenamiento por partición

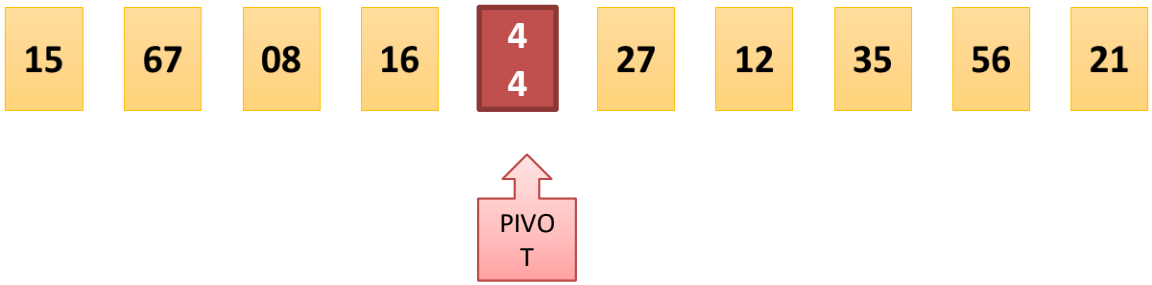
Funcionamiento:

PASO 1

- Se toma un elemento x de una posición (izquierda, medio o derecha) del arreglo, el cual se denominará PIVOTE
- Se harán comparaciones con el PIVOTE alternadamente de izquierda y derecha, haciendo cambios de posición donde corresponda, de tal forma que todos los elementos que se encuentran a su izquierda sean menores o iguales al PIVOTE y todos los elementos que se encuentren a su derecha sean mayores o iguales al PIVOTE

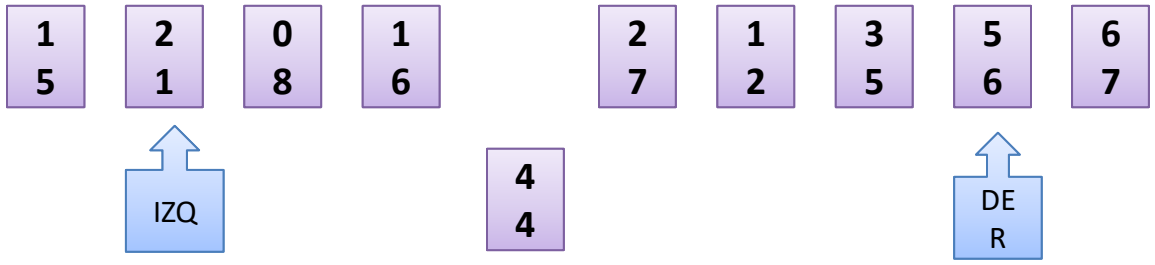


QUICK SORT

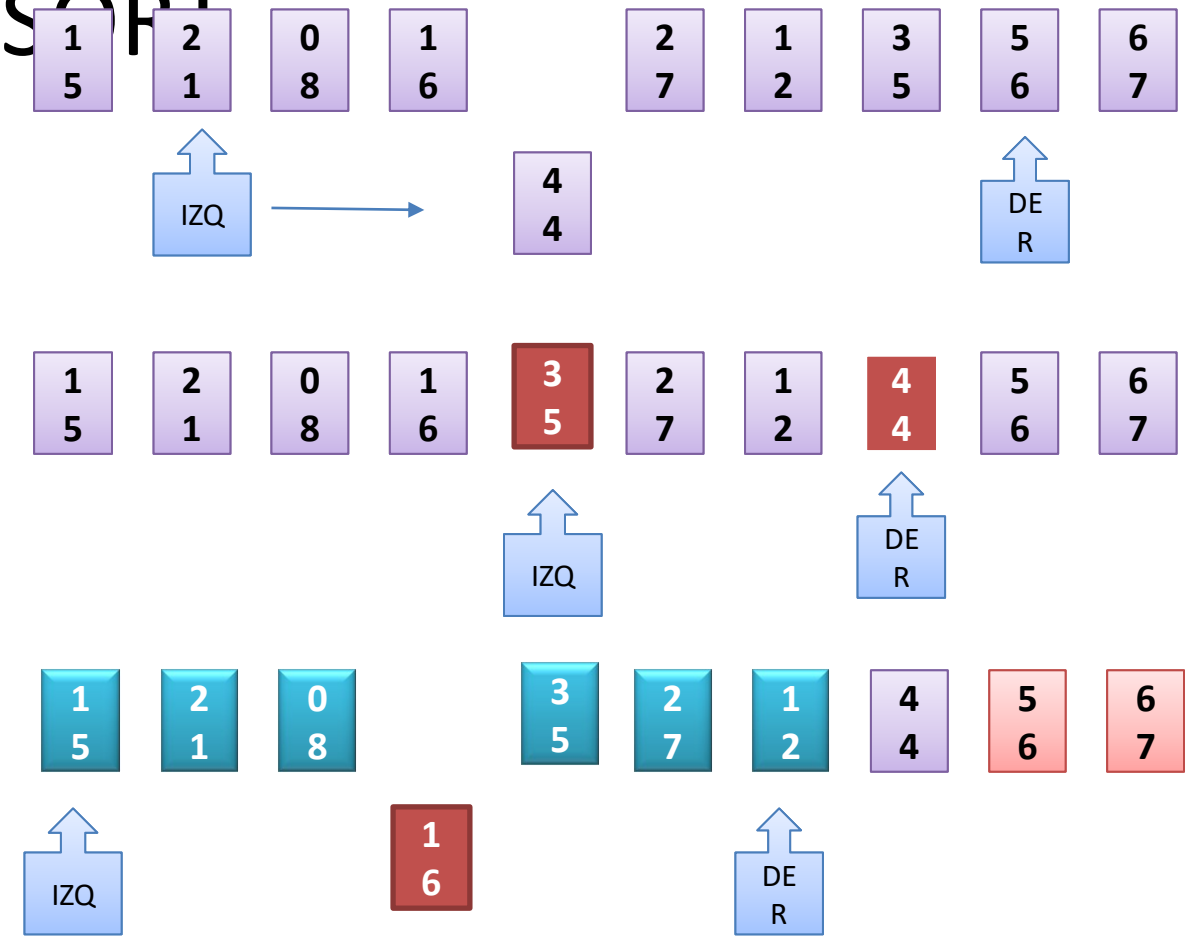


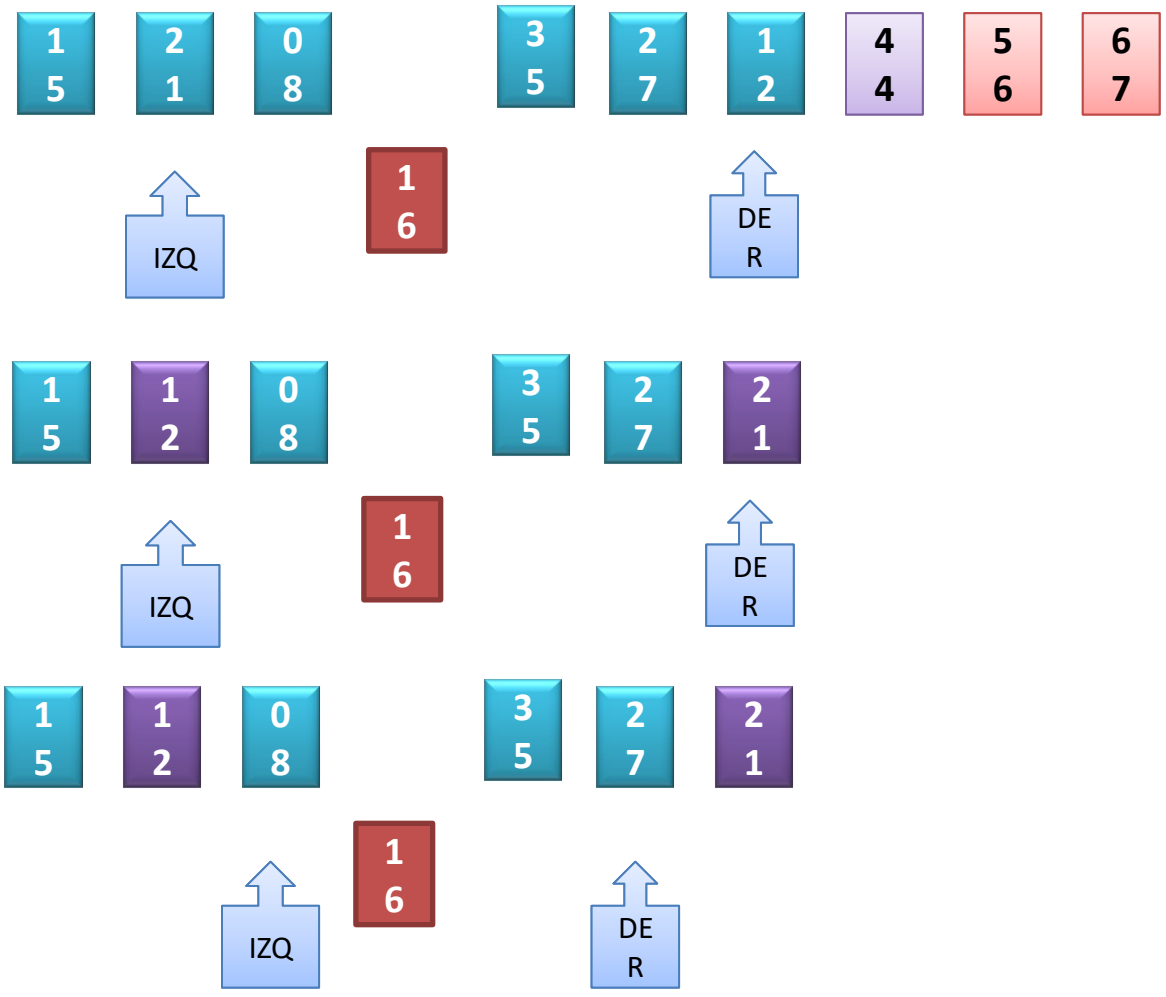
15 < 44 No se mueve
67 > 44 y 21 < 44 Se intercambian

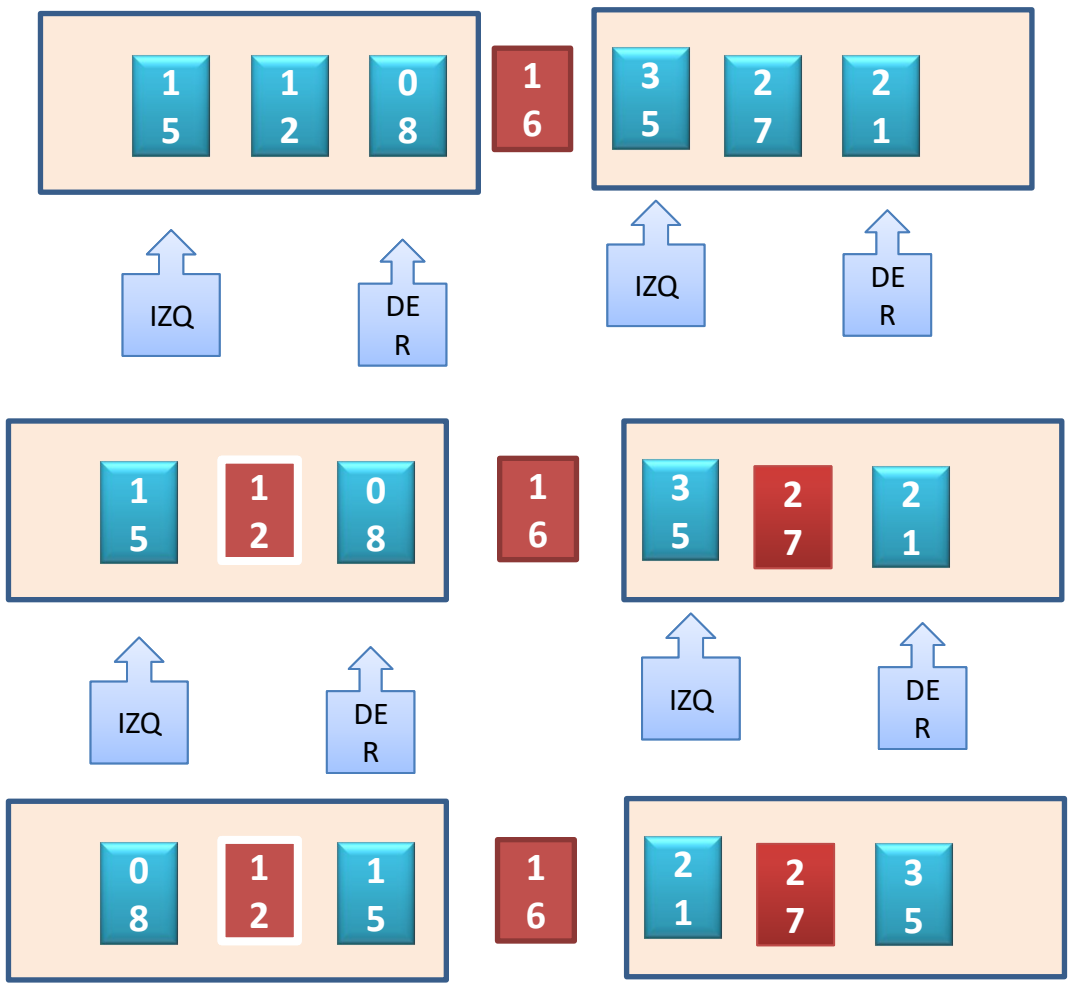
Cambiar los numeros menores al pivot a la izquierda por los numeros mayores a la derecha



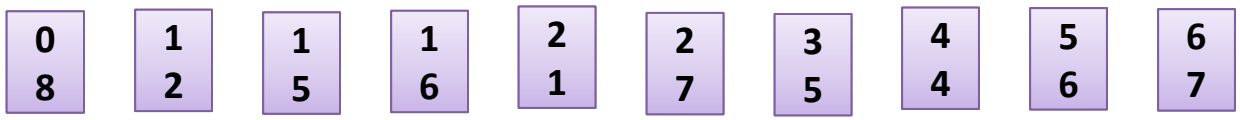
QUICK SORT


15
21
08
16
44
27
12
35
56
67
IZQ
DER
15
21
08
16
35
27
12
44
56
67
IZQ
DER
15
21
08
35
27
12
44
56
67
16
IZQ
DER


15
21
08
35
27
12
44
56
67
IZQ
16
DER
15
12
08
35
27
21
IZQ
16
DER
15
12
08
35
27
21
IZQ
16
DER


15
12
08
16
35
27
21
IZQ
DER
IZQ
DER
15
12
08
16
35
27
21
IZQ
DER
IZQ
DER
08
12
15
16
21
27
35


08
12
15
16
21
27
35
44
56
67
IZQ
DER


08
12
15
16
21
27
35
44
56
67

Metodo de Ordenamiento RADIX

¿Qué es?

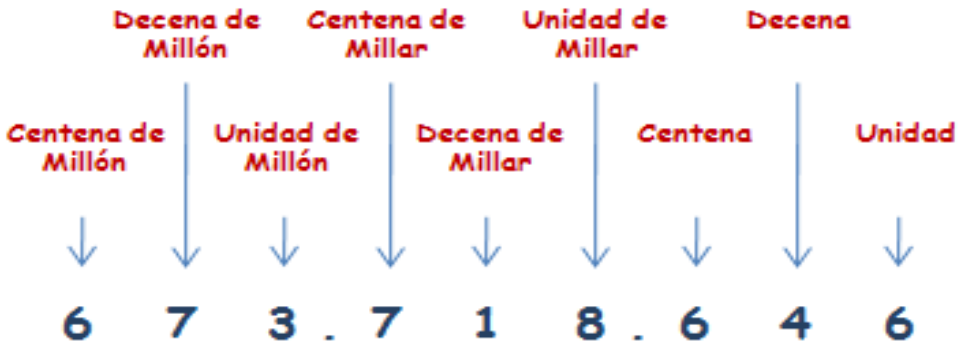
- El método de ordenación *radix* es un algoritmo que ordena datos procesando sus elementos de forma individual, según la posición que ocupan dentro del dato. Los datos numéricos los por dígitos y los datos alfabéticos por letras.

El método radix se clasifica en dos tipos según el orden en el que procesan los datos:

- De derecha a izquierda y
- De izquierda a derecha.

Si aplicamos este método solo a enteros, el método se clasificaría de la siguiente manera:

- **El digito menos significativo (LSD, Least Significat Digit) y**
- **El digito más significativo (MSD, More Significat Digit).**



RADIX

- El radix LSD procesa los enteros iniciando por el digito menos significativo y moviéndose al digito más significativo (de derecha a izquierda).
- El radix MSD procesa los enteros iniciando por el digito más significativo y moviéndose al digito menos significativo (de izquierda a derecha).
- El método más aplicado de radix, es el LSD, y se encarga de colocar los números en una de las 10 colas que representan un digito cada una de ella, iniciando desde la cola que controla el digito 0 hasta la cola que controla el digito 9, en estas colas se colocan los números dependiendo del digito que se este analizando en ese momento, hasta que termine con el número que contenga la mayor cantidad de dígitos, en cada cambio de digito los elementos se integran al arreglo nuevamente desde la cola 0 hasta la cola 9, para elegir el siguiente digito de ordenamiento. Cuando se efectúa este proceso para cada dígito al arreglo está ordenado.

RADIX Ejemplo

pos	0	1	2	3	4	5	6	7
valores	125	7	58	17	5	328	168	218

Dígitos	d0	d1	d2	d3	d4	d5	d6	d7	d8	d9
						125		7	58	
						5		17	328	
									168	
									218	

RADIX Ejemplo

Se sacan cada uno de los valores de las colas y se almacenan en el vector a partir de la cola cero.

pos	0	1	2	3	4	5	6	7
valores	125	5	7	17	58	328	168	218

Se revisa el segundo digito mas a la izquierda de los números y se almacenan en su cola respectiva a su valor, los que no tienen digito se almacenan en la CERO

Dígitos	d0	d1	d2	d3	d4	d5	d6	d7	d8	d9
	5	17	125			58	168			
	7	218	328							

RADIX Ejemplo

Paso 3: Se sacan cada uno de los valores de las colas y se almacenan en el vector a partir de la cola cero.

Pos	0	1	2	3	4	5	6	7
valores	5	7	17	218	125	328	58	168

Se revisa el primer digito mas a la izquierda de los números y se almacenan en su cola respectiva a su valor, los que no tienen digito se almacenan en la posición CERO

Dígitos	d0	d1	d2	d3	d4	d5	d6	d7	d8	d9
	5	125	218	328						
	7	168								
	17									
	58									

RADIX Ejemplo

Paso 4: Se sacan cada uno de los valores de las colas y se almacenan en el vector a partir de la cola cero.

Pos	0	1	2	3	4	5	6	7
valores	5	7	17	58	125	168	218	328

Como ya no se tienen más dígitos se termina con el proceso y ya quedan ordenados.

MERGE SORT ORDENAMIENTO POR MEZCLA

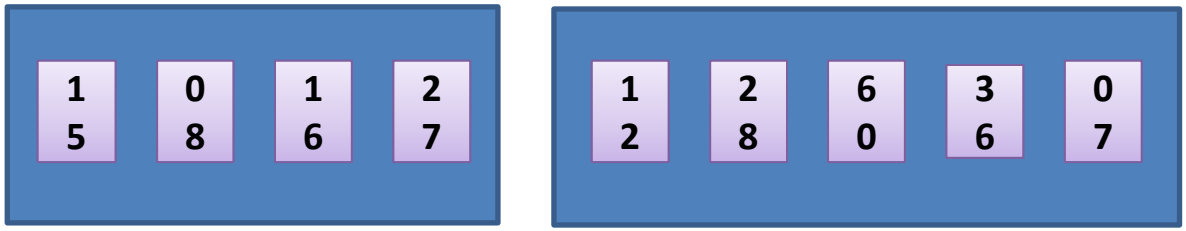
Este método se basa en la siguiente idea:

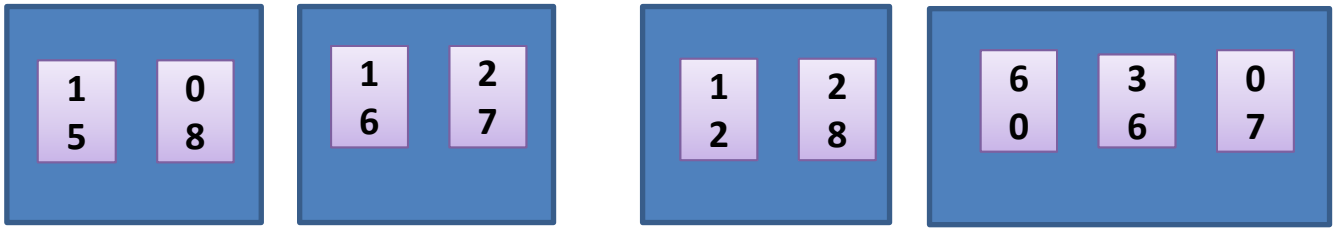
1. Si la lista es pequeña (vacía o de tamaño 1) ya está ordenada y no hay nada que hacer. De lo contrario hacer lo siguiente:
2. Dividir la lista al medio, formando dos sublistas de (aproximadamente) el mismo tamaño cada una.
3. Ordenar cada una de esas dos sublistas (usando este mismo método).
4. Una vez que se ordenaron ambas sublistas, intercalarlas de manera ordenada.

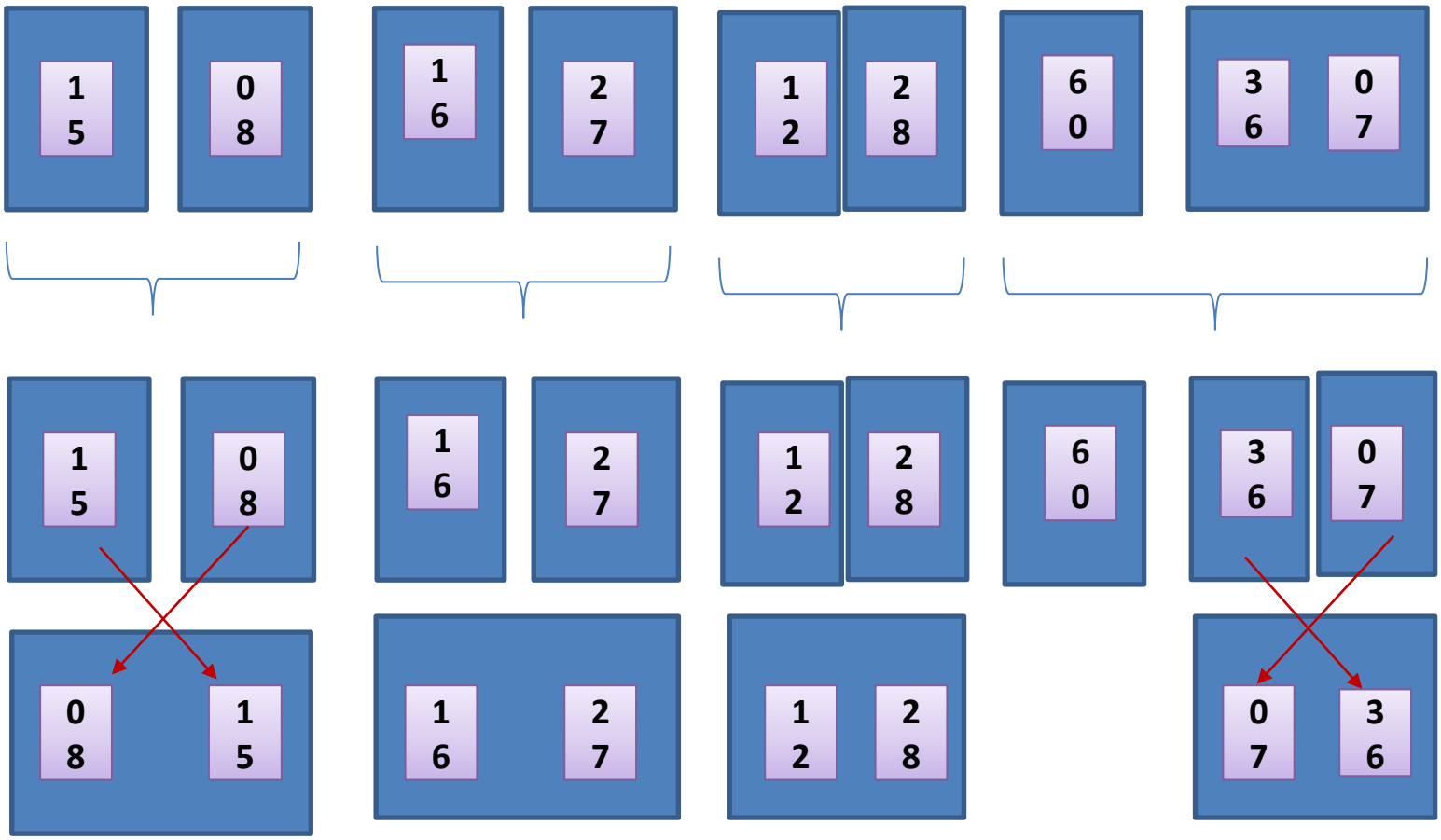
Ejemplo:
Se desean ordenarse las siguientes clave del arreglo

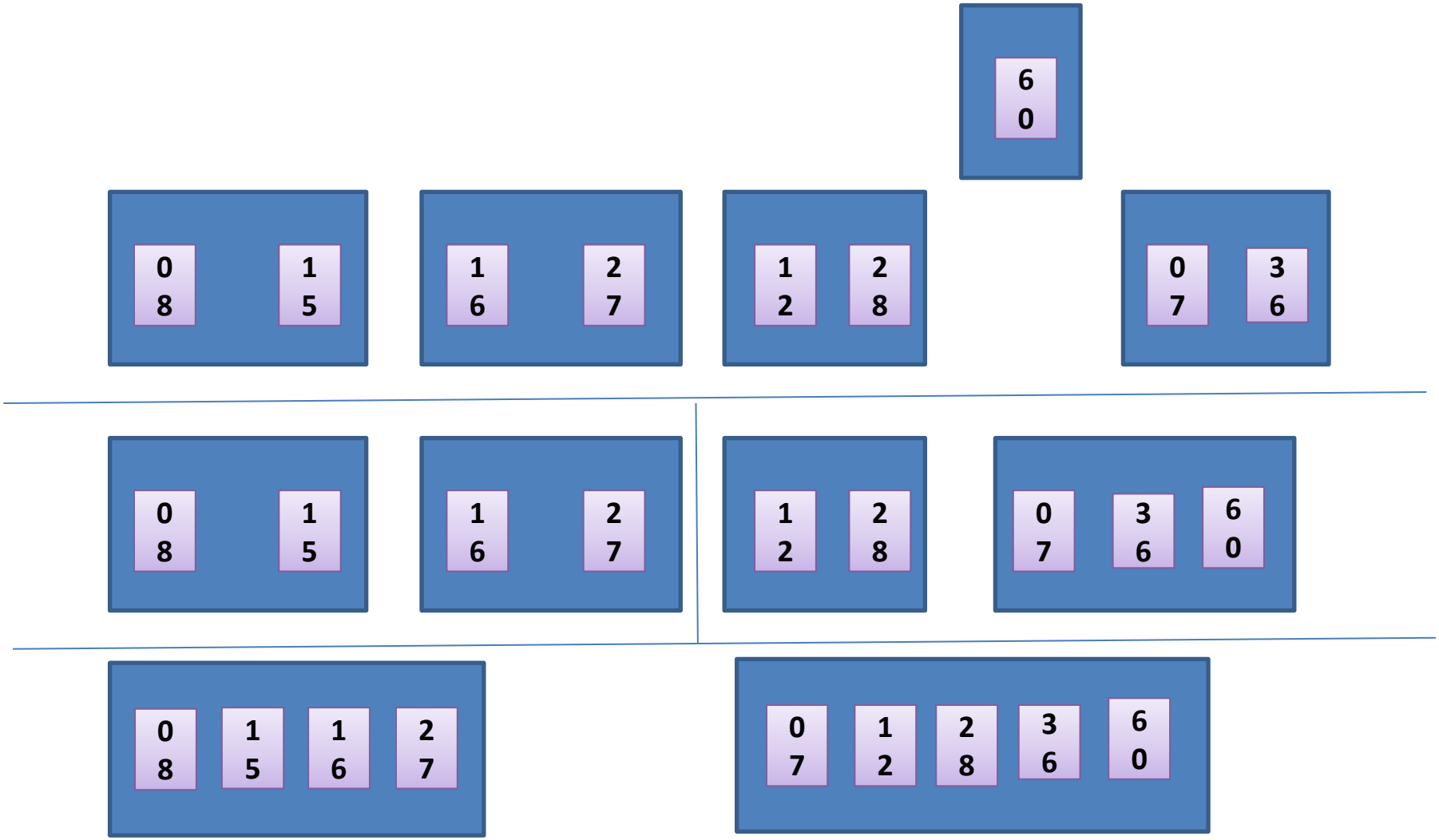
n=9

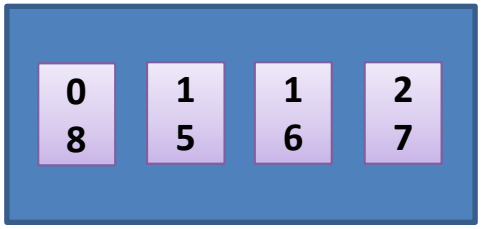
n=9/2

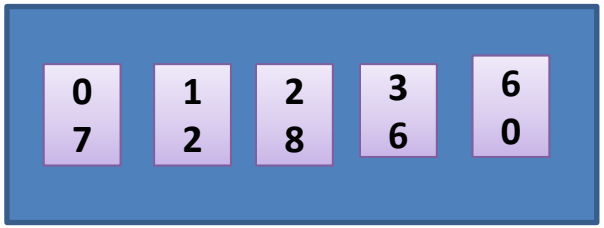


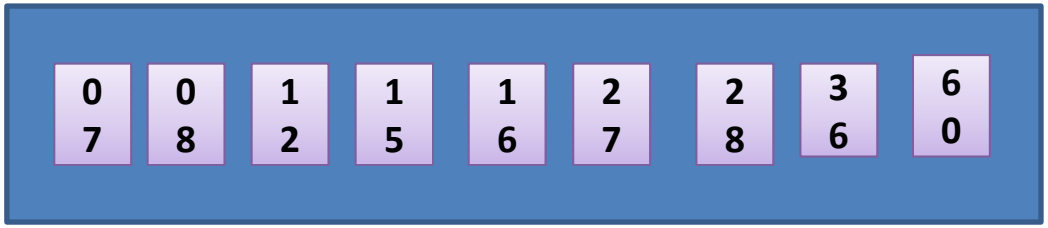



15
08
16
27
12
28
60
36
07
15
08
16
27
12
28
60
36
07
08
15
16
27
12
28
07
36


60
08
15
16
27
12
28
07
36
08
15
16
27
12
28
07
36
60
08
15
16
27
07
12
28
36
60







LISTA ORDENADA