Algoritmos y Estructura de datos

Clase 02
Métodos de Ordenación Directos

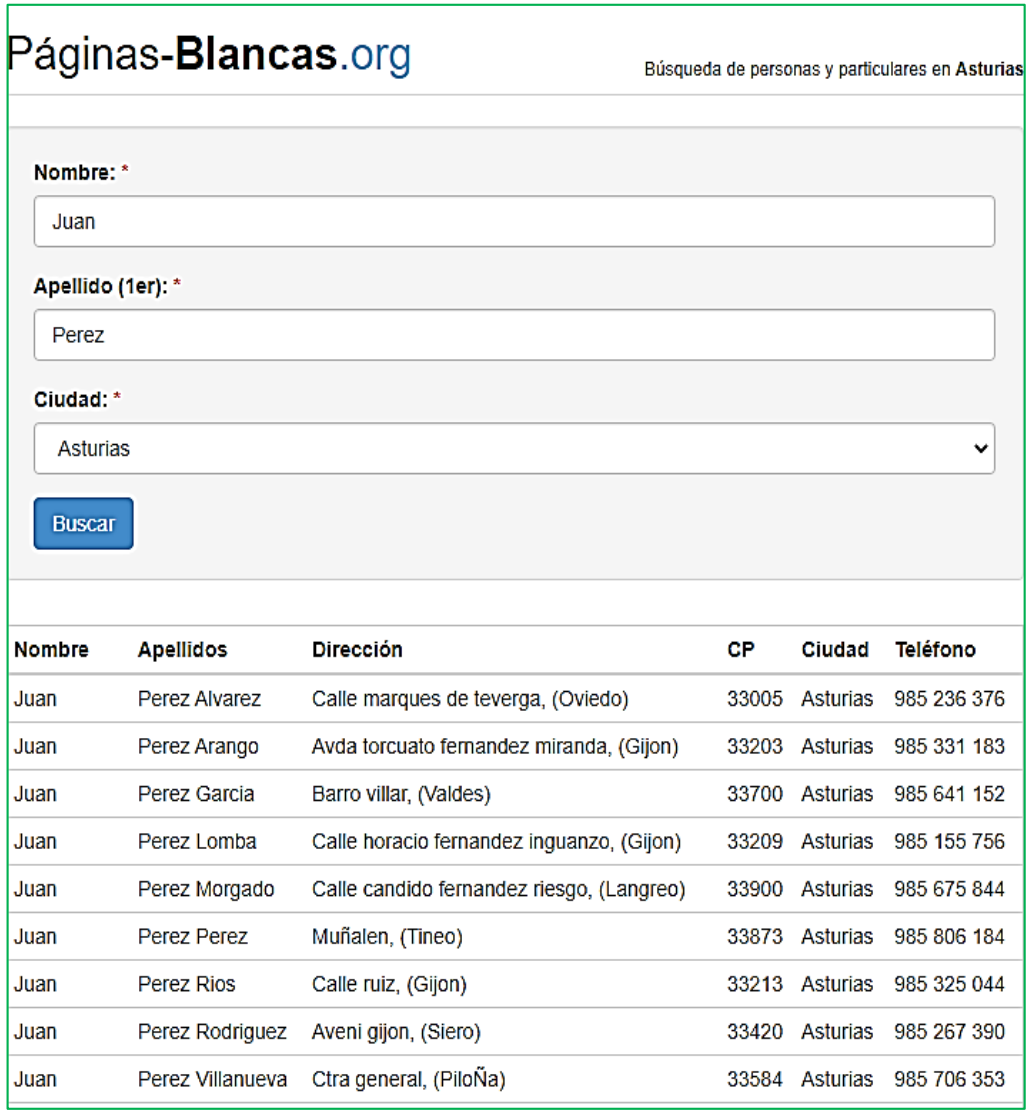
Profesor: Carlos Diaz

Contenido

- ¿Qué es un ordenamiento?
- Tipos de ordenación
- Ordenación por Intercambio
- Ordenación por Selección
- Ordenación por Inserción
- Ordenación por Burbuja
- Ejercicio

¿Qué es un ordenamiento?

- La ordenación de datos (sort en inglés) es una operación que consiste en reorganizar un conjunto de datos en una secuencia especifica respecto a uno de los campos.
  Ejemplo: La guía telefónica tiene los campos Nombre, Apellidos, Dirección, CP, Ciudad y Teléfono,
  La guía telefónica está organizada en orden alfabético ascendente por Apellidos.
- Clave: Es el campo respecto al cual esta ordenado el conjunto de datos.
  En el ejemplo el campo Apellidos es la clave.



Páginas-**Blancas**.org

Búsqueda de personas y particulares en **Asturias**

Nombre: *

Juan

Apellido (1er): *

Perez

Ciudad: *

Asturias

Buscar

Nombre	Apellidos	Dirección	CP	Ciudad	Teléfono
Juan	Perez Alvarez	Calle marques de teverga, (Oviedo)	33005	Asturias	985 236 376
Juan	Perez Arango	Avda torcuato fernandez miranda, (Gijon)	33203	Asturias	985 331 183
Juan	Perez Garcia	Barro villar, (Valdes)	33700	Asturias	985 641 152
Juan	Perez Lomba	Calle horacio fernandez inguanzo, (Gijon)	33209	Asturias	985 155 756
Juan	Perez Morgado	Calle candido fernandez riesgo, (Langreo)	33900	Asturias	985 675 844
Juan	Perez Perez	Muñalen, (Tineo)	33873	Asturias	985 806 184
Juan	Perez Rios	Calle ruiz, (Gijon)	33213	Asturias	985 325 044
Juan	Perez Rodriguez	Aveni gijon, (Siero)	33420	Asturias	985 267 390
Juan	Perez Villanueva	Ctra general, (PiloÑa)	33584	Asturias	985 706 353

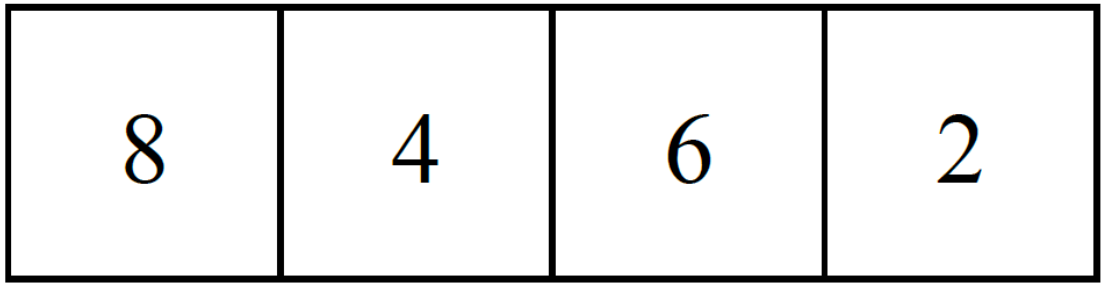
Tipos de ordenación

- Se suelen clasificar en tres tipos:

1. Según el orden,
   - Ascendentes: $i < j \implies K[i] <= K[j]$
   - Descendentes: $i > j \implies K[i] >= K[j]$
2. Según el lugar donde se almacenen.
   - Interna: Cuando los datos se encuentran en la memoria de la computadora.
   - Externa: Cuando los datos se encuentran en unidades de almacenamiento externo.
3. Los métodos de ordenación interna, a su vez, se clasifican según la cantidad de datos.
   - Directos: Son eficientes cuando se trata de una pequeña cantidad de datos.
   - Indirectos: Son eficientes en grandes cantidades de datos.

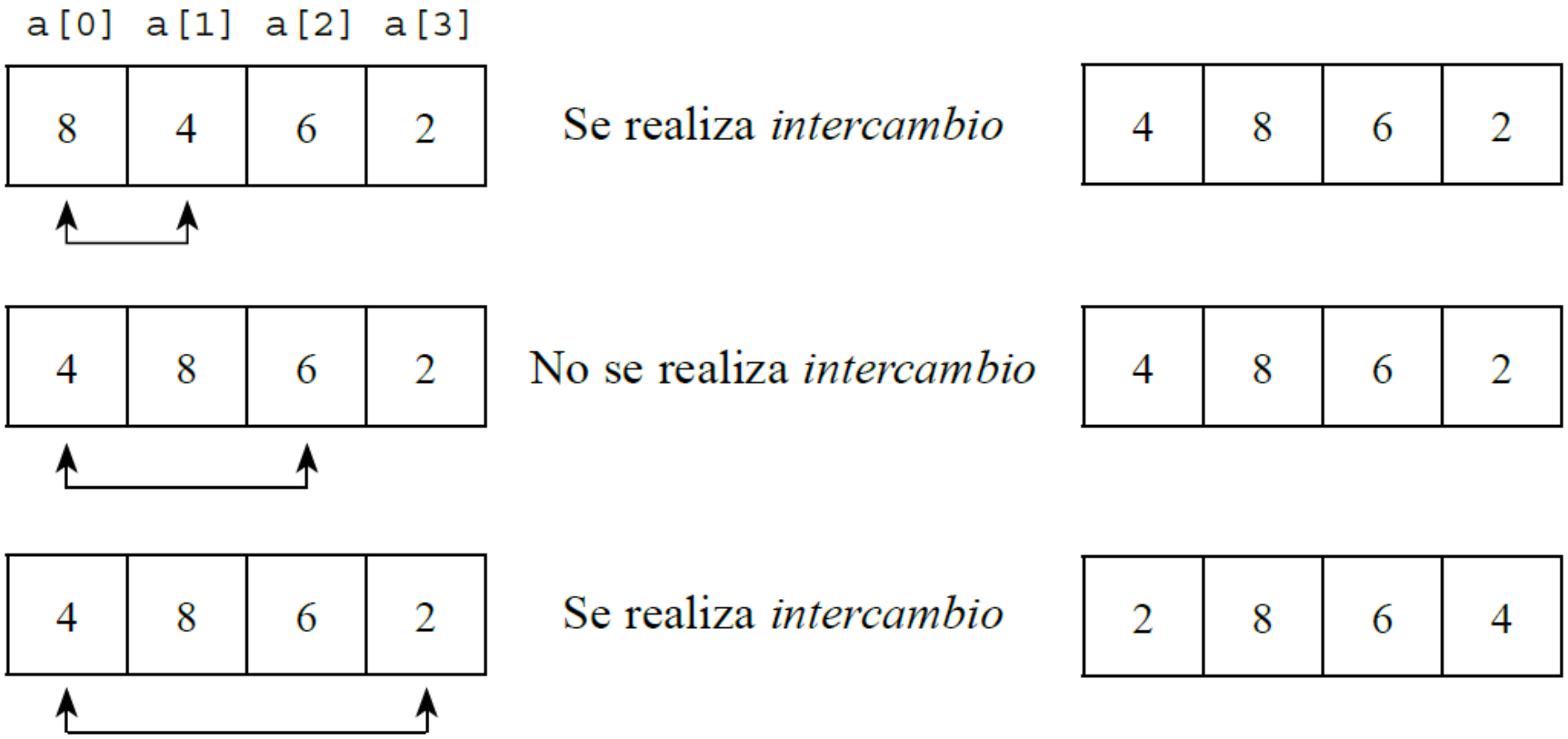
Ordenación por Intercambio

- El 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.
- El algoritmo efectúa $n - 1$ pasadas, siendo n el número de elementos.
- Ejemplo: Ordenar la lista utilizando el método de ordenación por intercambio.

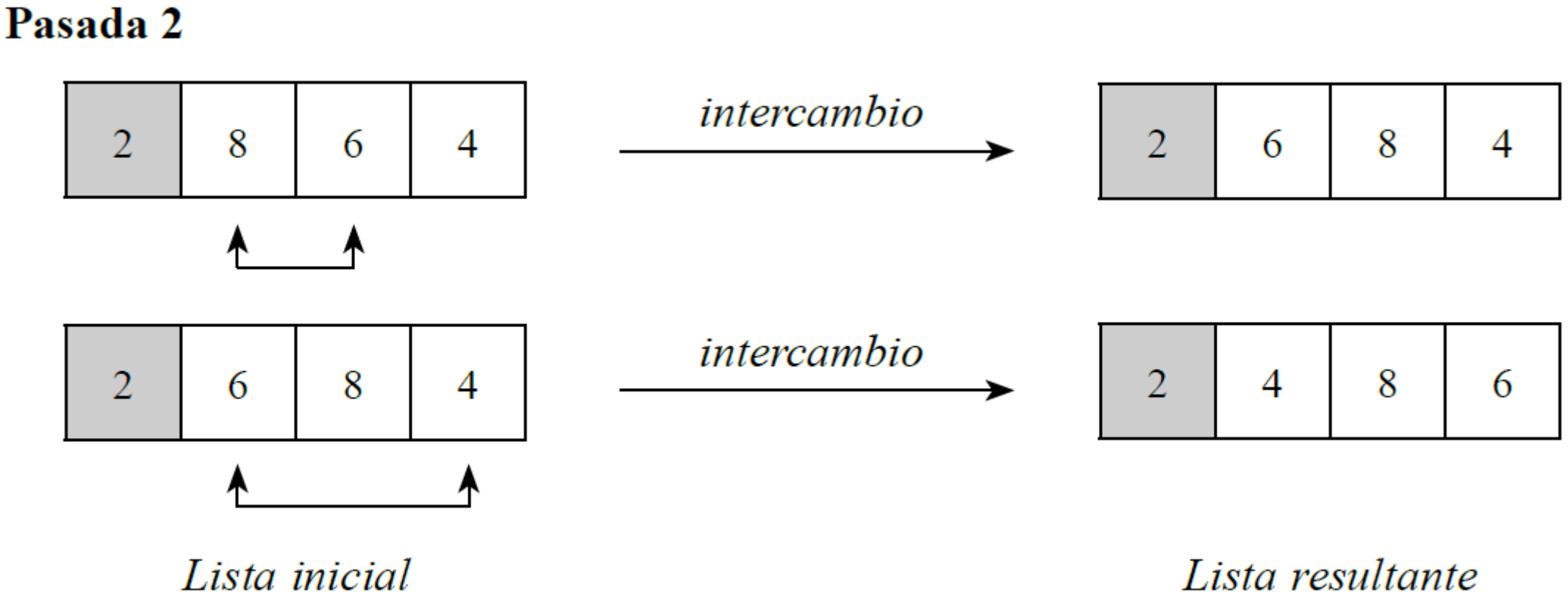


Ordenación por Intercambio

Pasada 1

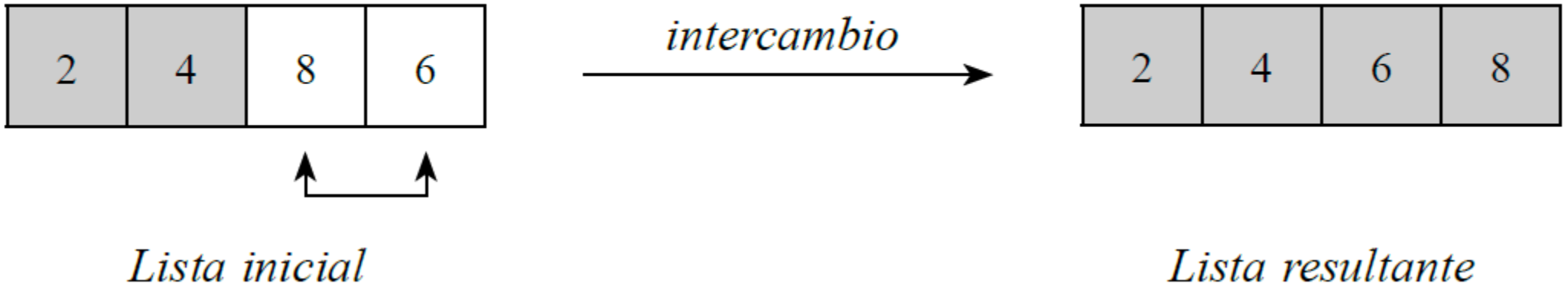


Ordenación por Intercambio



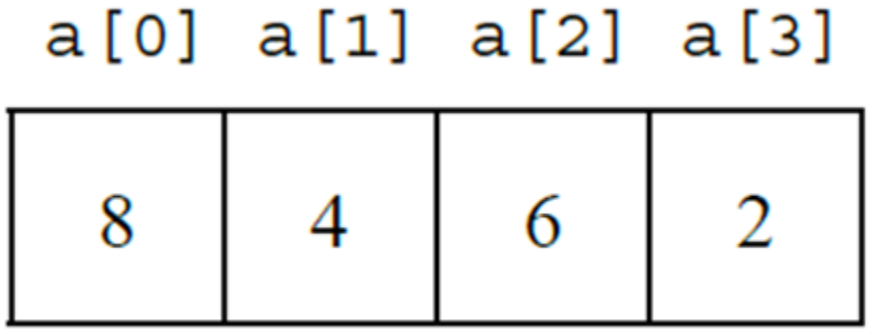
Ordenación por Intercambio

Pasada 3



Ordenación por Intercambio

- El método ordIntercambio() implementa el algoritmo, para ello utiliza dos bucles anidados.
- El bucle externo va desde *i=0* a *n – 2*. Lo que asegura *n-1* pasadas. Donde n es la cantidad de elementos.
- En el bucle interno, por cada índice *i,* se comparan los elementos posteriores de índices *j = i + 1, i + 2 ,..., n – 1*.
- El intercambio *(swap)* de dos elementos *a[i]*, *a[j]* lo realiza la función intercambiar().



Codificación por Intercambio

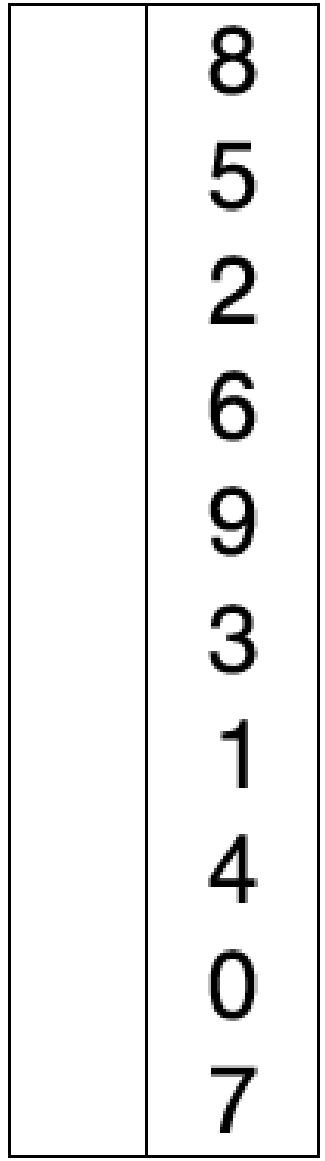
```
void intercambiar(int& x, int& y)
{
   int aux = x;
   x = y;
   y = aux;
}
```

```
void ordIntercambio (int a[], int n)
{
   int i, j;

   for (i = 0; i < n - 1; i++)
      for (j = i + 1; j < n; j++)
         if (a[i] > a[j])
         {
            intercambiar(a[i], a[j]);
         }
}
```

Ordenación por Selección

- Su funcionamiento es el siguiente:
  1. Buscar el mínimo elemento de la lista
  2. Intercambiarlo con el primero
  3. Buscar el siguiente mínimo en el resto de la lista
  4. Intercambiarlo con el segundo
- Y en general:
  1. Buscar el mínimo elemento entre una posición *i* y el final de la lista
  2. Intercambiar el mínimo con el elemento de la posición *i*

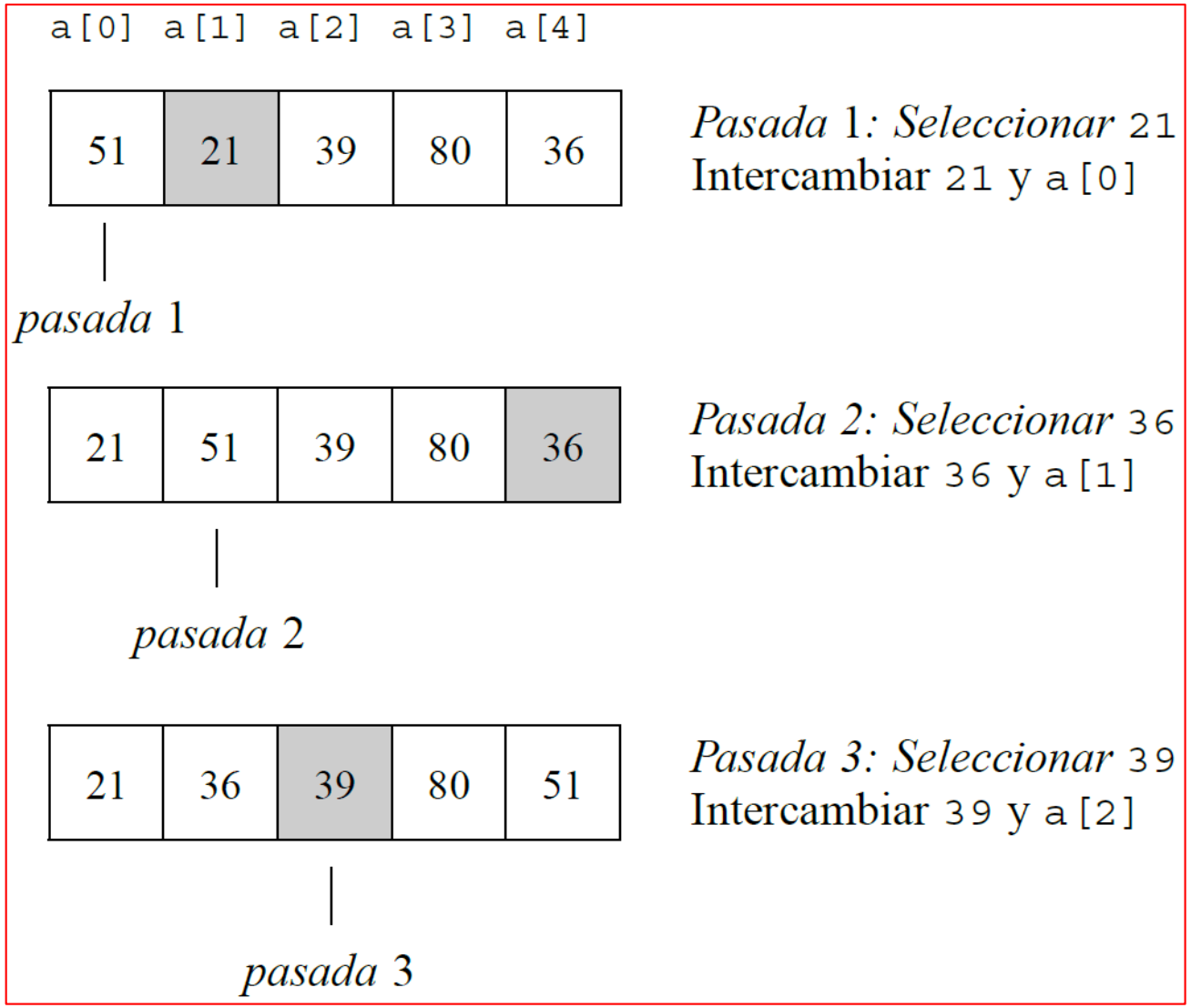


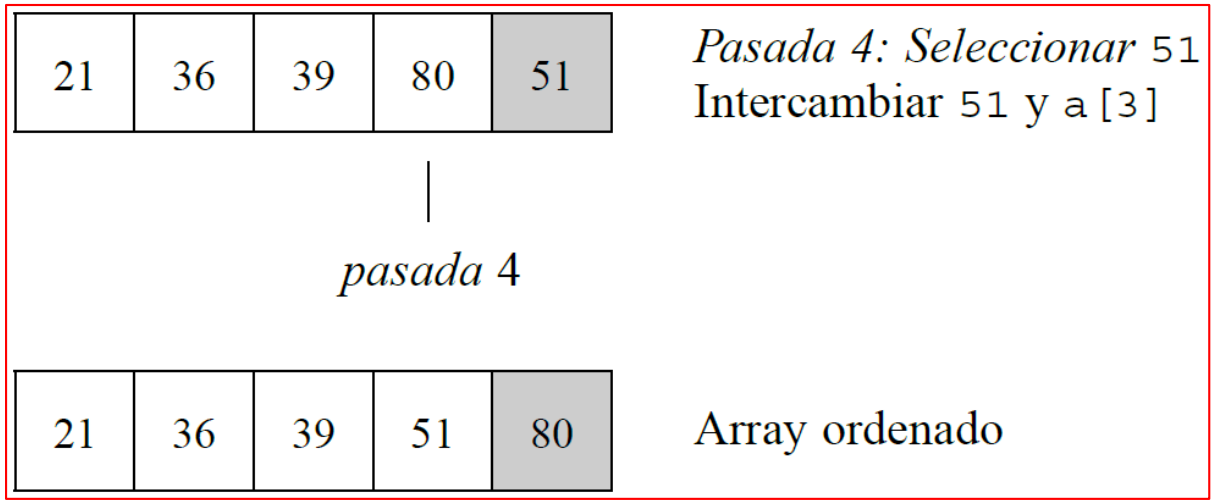
Ordenación por Selección

- Consiste en ordenar los valores del array de modo que a[0] sea el valor más pequeño de la lista, luego en a[1] el siguiente valor más pequeño, y así hasta que a[n-1] contenga al mayor.
- El algoritmo de selección realiza *pasadas* que intercambian el elemento más pequeño sucesivamente, con el elemento del array que ocupa la posición igual al orden de *pasada*.
- La *pasada* inicial busca el elemento más pequeño de la lista y se intercambia con a[0].
- Después de terminar esta primera pasada, el frente de la lista está ordenado y el resto de la lista a[1], a[2] ... a[n-1] permanece desordenada.
- La siguiente *pasada* busca en esta lista desordenada y *selecciona* el elemento más pequeño y se almacena en la posición a[1].
- De este modo los elementos a[0] y a[1] están ordenados y la sublista a[2], a[3]...a[n-1] desordenada. El proceso continúa hasta realizar n-1 *pasadas*.
- Ejemplo: Ordenar la lista utilizando el método de ordenación por selección.

51	21	39	80	36

Ordenación por Selección





Codificación por Selección

```
void intercambiar(double& x, double& y)
{
  double aux = x;
  x = y;
  y = aux;
}
```

```
void ordSeleccion (double a[], int n)
{
  int indiceMenor, i, j

  for (i = 0; i < n - 1; i++)
  {
     // comienzo de la exploración en índice i
     indiceMenor = i;
     // j explora la sublista a[i+1]..a[n-1]
     for (j = i + 1; j < n; j++)
       if (a[j] < a[indiceMenor])
         indiceMenor = j;
     // sitúa el elemento mas pequeño en a[i]
     if (i != indiceMenor)
        intercambiar(a[i], a[indiceMenor]);
  }
}
```

Ordenación por Inserción

- Este método de ordenación es similar al proceso típico de ordenar las cartas de una baraja, que consiste en insertar una carta en su posición correcta dentro de una lista que ya está ordenada.

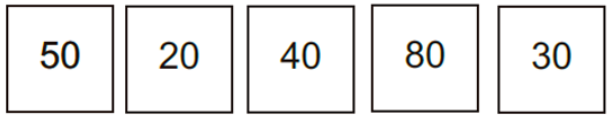
6 5 3 1 8 7 2 4

Ordenación por Inserción

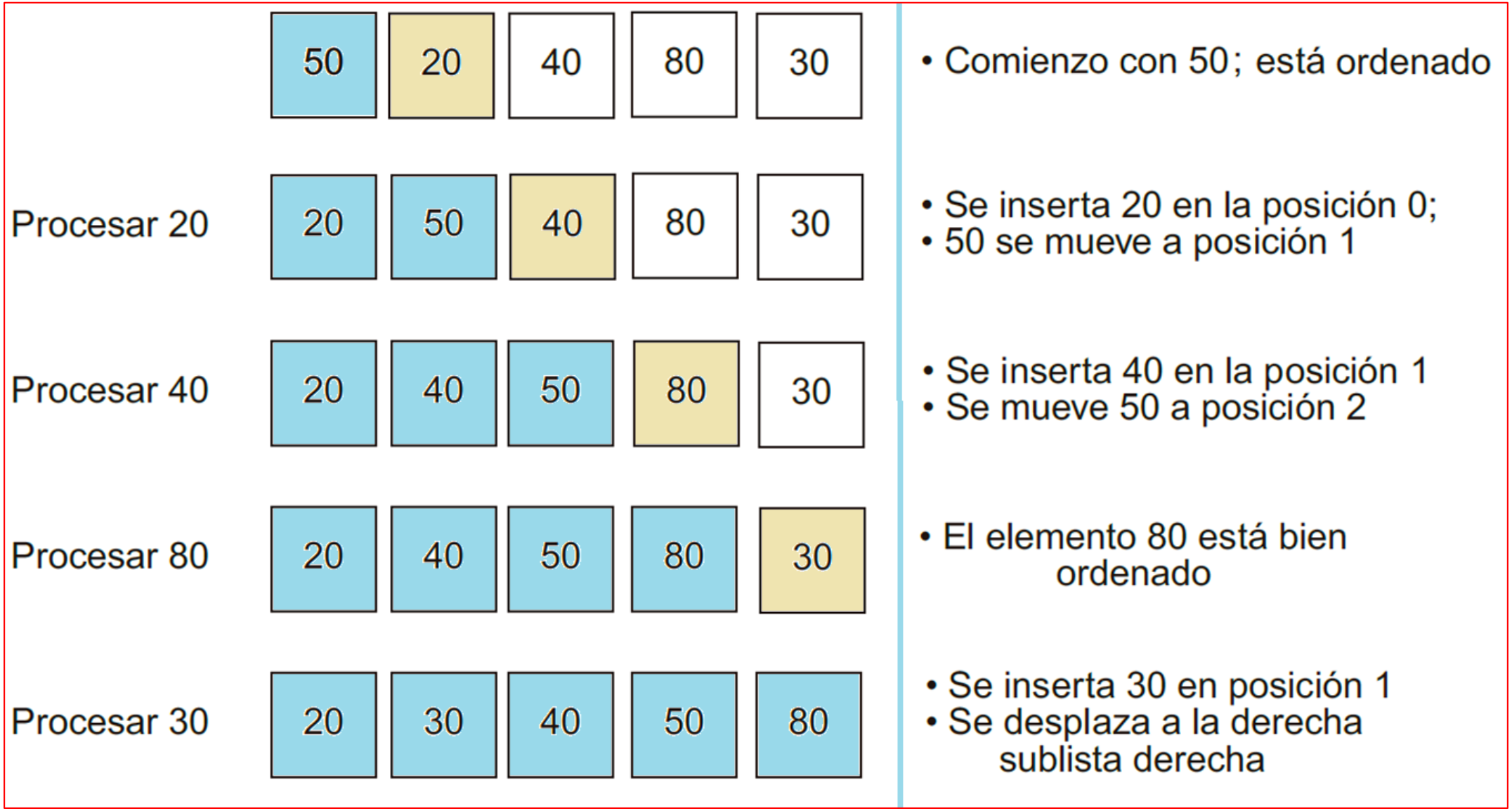
- El algoritmo es el siguiente:

1. El primer elemento a[0] se considera ordenado; es decir, la lista inicial consta de un elemento.
2. Se inserta a[1] en la posición correcta; delante o detrás de a[0], dependiendo de que sea menor o mayor.
3. Por cada iteración i (desde i = 1 hasta n - 1) se explora la sublista a[i-1]... a[0] buscando la posición correcta de inserción de a[i]; a la vez se mueve hacia abajo (a la derecha en la sublista) una posición todos los elementos mayores que el elemento a insertar a[i], para dejar vacía esa posición.
4. Insertar el elemento a[i] en la posición correcta.

- Ejemplo: Ordenar la lista utilizando el método de ordenación por inserción.



Ordenación por Inserción



Codificación por Inserción

```
void ordInsercion (int a[], int n)
{
  int i, j, aux;

  for (i = 1; i < n; i++)
  {
    /* indice j es para explorar la sublista a[i-1]..a[0] buscando la
       posición correcta del elemento destino */
    j = i;
    aux = a[i];
      // se localiza el punto de inserción explorando hacia abajo
    while (j > 0 && aux < a[j-1])
    {
      // desplazar elementos hacia arriba para hacer espacio
      a[j] = a[j-1];
      j--;
    }
    a[j] = aux;
  }
}
```

Ordenación por Burbuja

- El método de ordenación por burbuja es el más conocido y popular entre estudiantes y aprendices de programación, por su facilidad de comprender y programar; por el contrario, es el menos eficiente y por ello, normalmente, se aprende su técnica pero no suele utilizarse.
- La técnica utilizada se denomina ordenación por burbuja debido a que los valores más grandes (pequeños) “burbujean” suben gradualmente hacia la parte superior del array de modo similar a como suben las burbujas en el agua, mientras que los valores menores (mayores) se hunden en la parte inferior del array.

Ordenación por Burbuja

6 5 3 1 8 7 2 4

Ordenación por Burbuja

- Las etapas del algoritmo son:
- En la pasada 1 se comparan elementos adyacentes.

  (a[0],a[1]),(a[1],a[2]),(a[2],a[3]),...(a[n-2],a[n-1])

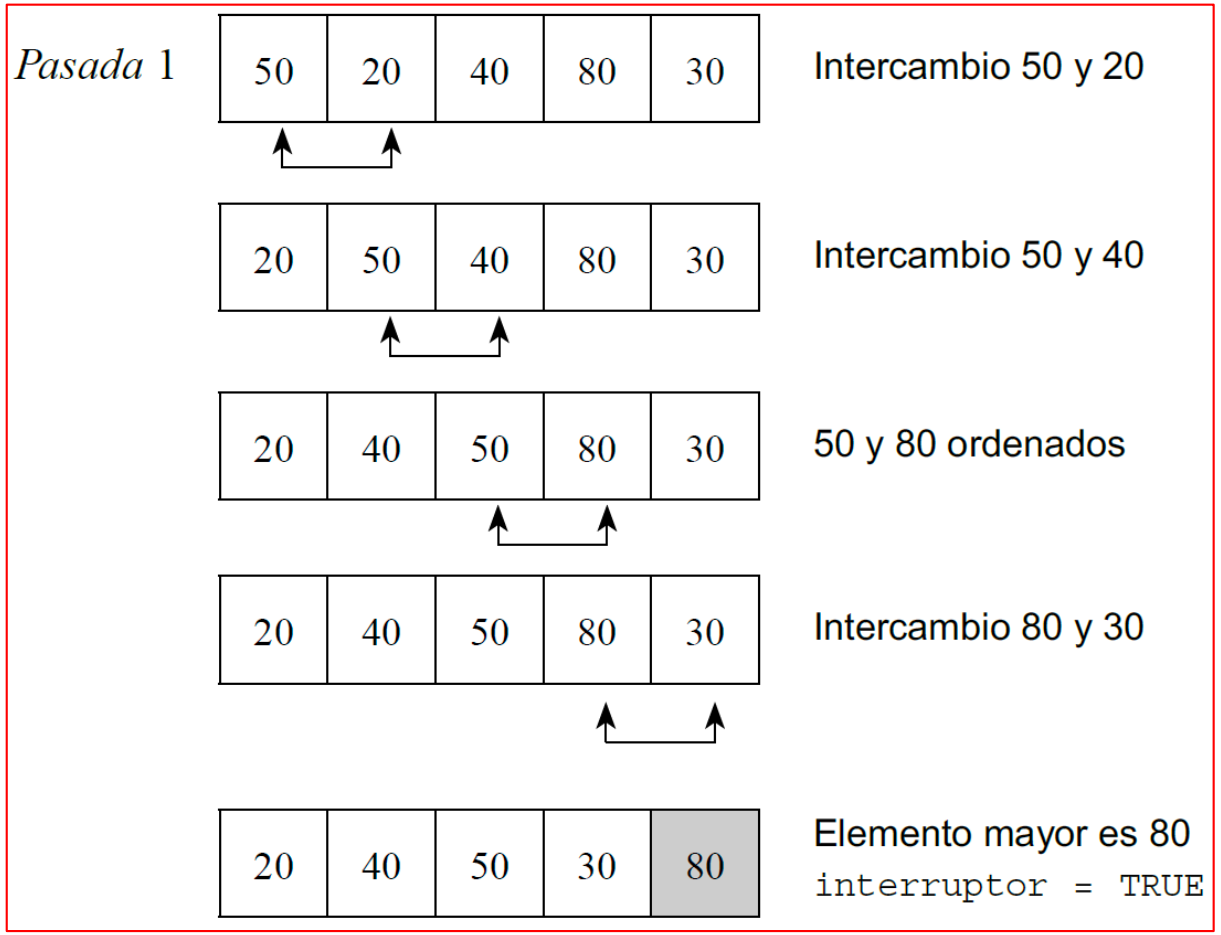
  Se realizan n – 1 comparaciones, por cada pareja (a[i],a[i+1])

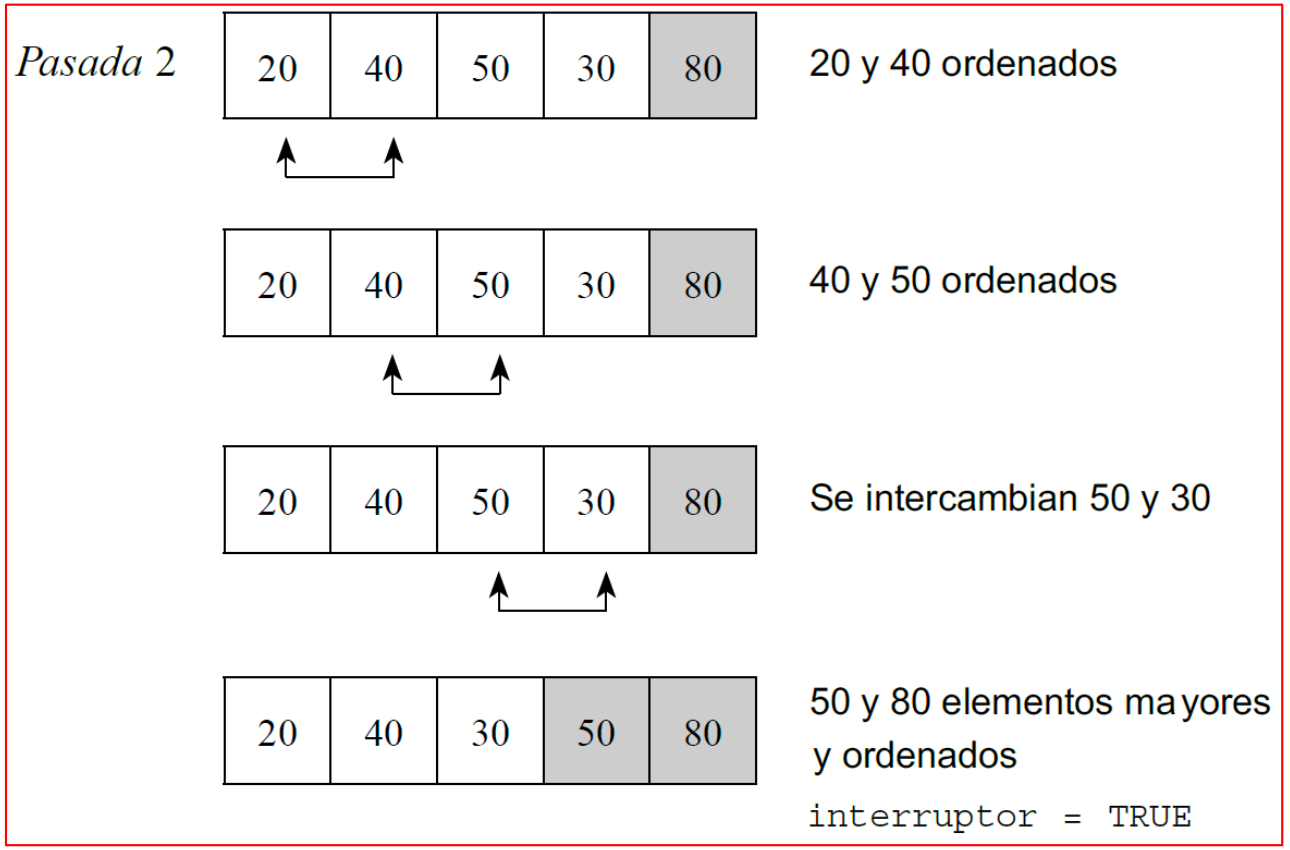
  Se intercambian los valores si a[i+1] < a[i].

  Al final de la pasada, el elemento mayor de la lista está situado en a[n-1].
- En la pasada 2 se realizan las mismas comparaciones e intercambios, terminando con el elemento de segundo mayor valor en a[n-2] .
- El proceso termina con la pasada n – 1, en la que el elemento más pequeño se almacena en a[0].
- El algoritmo tiene una mejora inmediata, el proceso de ordenación puede terminar en la pasada n – 1, o bien antes, si en un una pasada no se produce intercambio alguno entre elementos del array es porque ya está ordenado, entonces no es necesario más pasadas.
- Ejemplo: Ordenar la lista utilizando el método de ordenación por burbuja.

50	20	40	80	30

Ordenación por Burbuja





El algoritmo terminará cuando se termine la última pasada (n – 1), o bien cuando el valor del interruptor sea falso, es decir no se haya hecho ningún intercambio.

Codificación por Burbuja

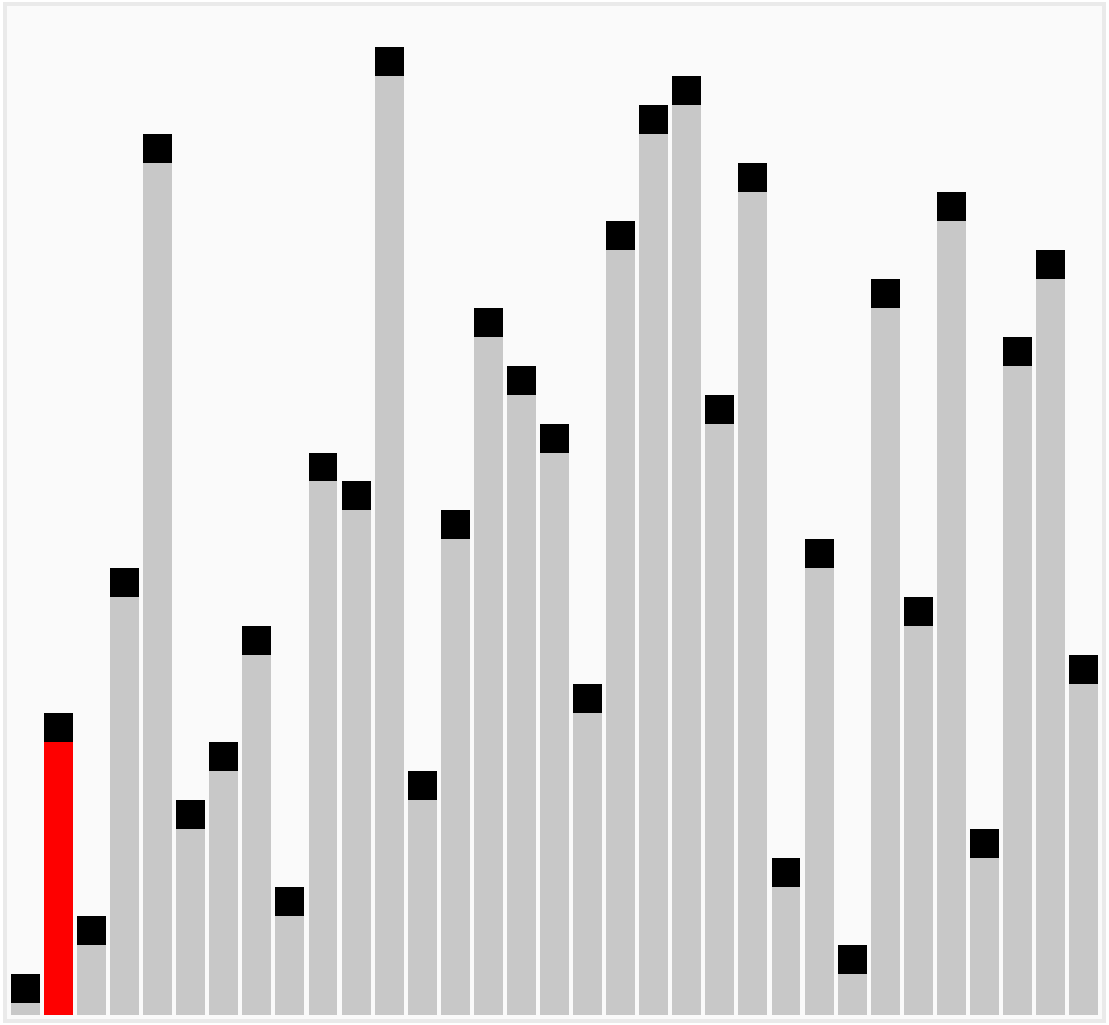
```
void ordBurbuja (long a[], int n)
{
  bool interruptor = true;
  int pasada, j;
  // bucle externo controla la cantidad de pasadas
  for (pasada = 0; pasada < n - 1 && interruptor; pasada++)
  {
    interruptor = false;
    for (j = 0; j < n - pasada - 1; j++)
      if (a[j] > a[j + 1])
      {
        // elementos desordenados, se intercambian
        interruptor = true;
        intercambiar(a[j], a[j + 1]);
      }
  }
}
```

Ejercicio 1

- Implemente los métodos de ordenación directos, tal como muestra la imagen.

Ejercicio 2

- Ordenamiento de burbuja bidireccional: es un algoritmo de ordenamiento que surge como una mejora del algoritmo ordenamiento de burbuja.
- La manera de trabajar de este algoritmo es ir ordenando al mismo tiempo por los dos extremos del vector.
- De manera que tras la primera iteración, tanto el menor como el mayor elemento estarán en sus posiciones finales.



Ejercicio 2

- Hacemos un recorrido ascendente (del primer elemento al último), tomamos el primer elemento y lo comparamos con el siguiente, si el siguiente es menor lo pasamos al puesto anterior, de esta forma al final de la lista nos queda el mayor.
- Una vez terminada la serie ascendente, hacemos un recorrido descendente (del último elemento al primero) pero esta vez nos quedamos con los menores a los que vamos adelantando posiciones en vez de retrasarlas como hicimos en la serie ascendente.
- Repetimos las series alternativamente pero reduciendo el ámbito en sus extremos pues ya tendremos allí los valores más bajos y más altos de la lista, hasta que no queden elementos en la serie.

FIN