8.6. 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*** u ***ordenación por hundimiento*** debido a que los valores más pequeños "*burbujean*" gradualmente (suben) hacia la cima o parte superior del *array* de modo similar a como suben las burbujas en el agua, mientras que los valores mayores se hunden en la parte inferior del *array*.

8.6.1. Algoritmo de la burbuja

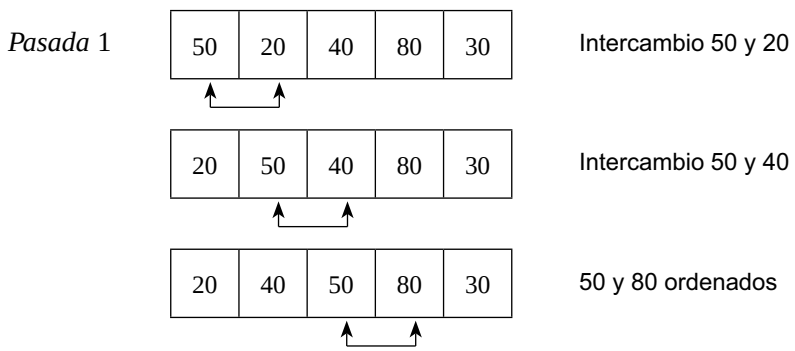
Para un *array* con *n* elementos, la ordenación por burbuja requiere hasta *n – 1* pasadas. Por cada pasada se comparan elementos adyacentes y se intercambian sus valores cuando el primer elemento es mayor que el segundo elemento. Al final de cada pasada, el elemento mayor ha "*burbujeado*" hasta la cima de la sublista actual. Por ejemplo, después que la pasada `1` está completa, la cola de la lista *a[n – 1]* está ordenada y el frente de la lista permanece desordenado. 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])
  ```

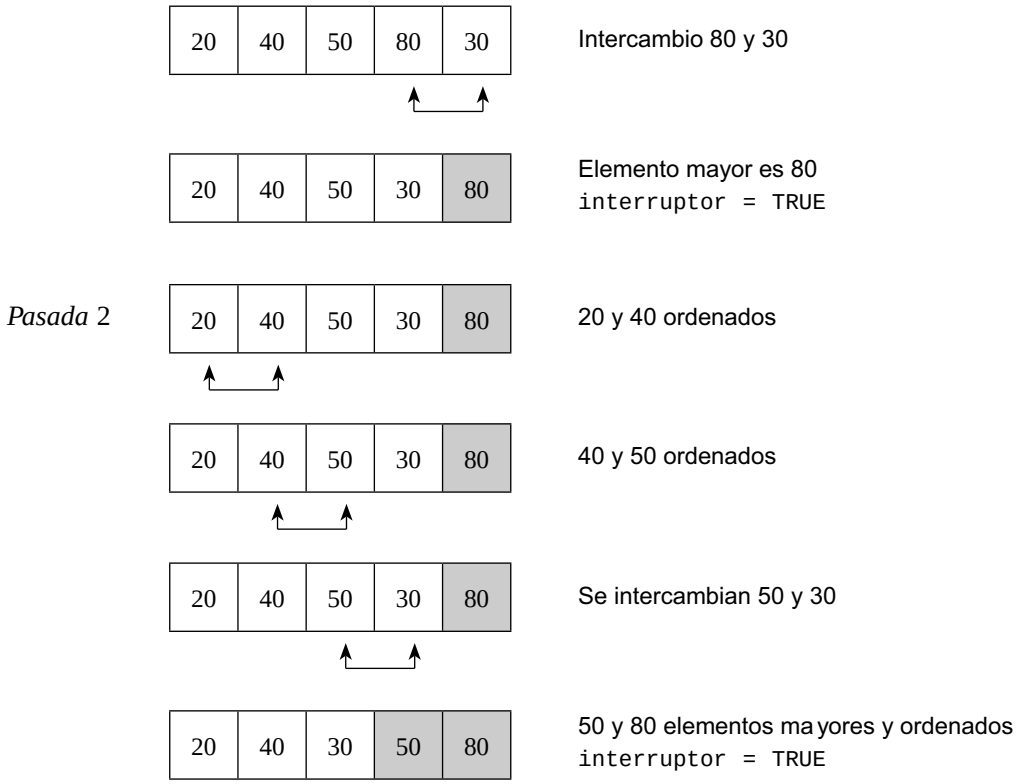
  Se realizan *n* – 1 comparaciones, por cada pareja `(a[i],a[i+1])`, se intercambian los valores si `a[i+1] < a[i]`.
  Al final de la pasada, el elemento mayor de la lista está situado en `a[n-1]`.
- En la pasada `2` se realizan las mismas comparaciones e intercambios, terminando con el elemento de segundo 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. A continuación, se ilustra el funcionamiento del algoritmo realizando las dos primeras pasadas en un array de 5 elementos; se introduce la variable `interruptor` para detectar si se ha producido intercambio en la pasada.



Copyright © 2007. McGraw-Hill España. All rights reserved.

Joyanes, Luis, and Ignacio Zahonero. Estructura de datos en C++, McGraw-Hill España, 2007. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/unadsp/detail.action?docID=3194764.
Created from unadsp on 2019-04-05 13:23:12.



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.

8.6.2. Codificación del algoritmo de la burbuja

El algoritmo de ordenación de burbuja *mejorado* contempla dos bucles anidados: el *bucle externo* controla la cantidad de pasadas, el *bucle interno* controla cada pasada individualmente y cuando se produce un intercambio, cambia el valor de `interruptor` a *verdadero* (`true`).

```
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
```

Copyright © 2007. McGraw-Hill España. All rights reserved.

Joyanes, Luis, and Ignacio Zahonero. Estructura de datos en C++, McGraw-Hill España, 2007. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/unadsp/detail.action?docID=3194764.
Created from unadsp on 2019-04-05 13:23:12.

```
            interruptor = true;
            intercambiar(a[j], a[j + 1]);
          }
      }
  }
```

8.6.3. Análisis del algoritmo de la burbuja

¿Cuál es la eficiencia del algoritmo de ordenación de la burbuja?

La ordenación de burbuja hace una sola pasada en el caso de una lista que ya está ordenada en orden ascendente y, por tanto, su complejidad es *0(n)*. En el *caso peor* se requieren *(n – i – 1)* comparaciones y *(n – i – 1)* intercambios. La ordenación completa requiere $\frac{n(n-1)}{2}$ comparaciones y un número similar de intercambios. La complejidad para el caso peor es $0(n^2)$ comparaciones y $0(n^2)$ intercambios.

De cualquier forma, el análisis del caso general es complicado dado que alguna de las pasadas pueden no realizarse. Se podría señalar que el número medio de pasadas *k* es *0(n)* y el número total de comparaciones es $0(n^2)$. En el mejor de los casos, la ordenación por burbuja puede terminar en menos de *n – 1* pasadas pero requiere, normalmente, muchos más intercambios que la ordenación por selección y su prestación media es mucho más lenta, sobre todo cuando los arrays a ordenar son grandes.

Consejo de programación

Los algoritmos de ordenación interna: intercambio, selección, inserción y burbuja son fáciles de entender y de codificar, sin embargo poco eficientes y no recomendables para ordenar listas de muchos elementos. La complejidad de todos ellos es cuadrática, $0(n^2)$.

8.7. ORDENACIÓN SHELL

La ordenación Shell debe el nombre a su inventor, *D. L. Shell*. Se suele denominar también *ordenación por inserción con incrementos decrecientes*. Se considera que es una mejora del método de inserción directa.

En el algoritmo de inserción, cada elemento se compara con los elementos contiguos de su izquierda, uno tras otro. Si el elemento a insertar es el más pequeño hay que realizar muchas comparaciones antes de colocarlo en su lugar definitivo. El algoritmo de Shell modifica los saltos contiguos por saltos de mayor tamaño y con ello consigue que la ordenación sea más rápida. Generalmente, se toma como salto inicial $n/2$ (siendo *n* el número de elementos), luego en cada iteración se reduce el salto a la mitad, hasta que el salto es de tamaño 1. El Ejemplo 8.1 muestra paso a paso el método de Shell.

EJEMPLO 8.1. Aplicar el método Shell para ordenar en orden creciente la lista: 6 1 5 2 3 4 0

El número de elementos que tiene la lista es 6, por lo que el salto inicial es 6/2 = 3. La siguiente tabla muestra el número de recorridos realizados en la lista con los saltos correspondiente.

Copyright © 2007. McGraw-Hill España. All rights reserved.

Joyanes, Luis, and Ignacio Zahonero. Estructura de datos en C++, McGraw-Hill España, 2007. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/unadsp/detail.action?docID=3194764.
Created from unadsp on 2019-04-05 13:23:12.