6.5. ORDENACIÓN POR 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`.

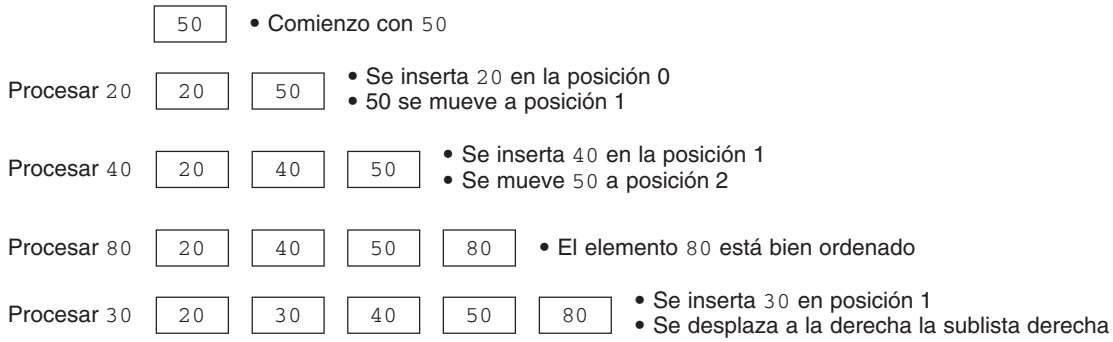

Figura 6.1. Método de ordenación por inserción.

6.5.1. Algoritmo de ordenación por inserción

El algoritmo correspondiente a la ordenación por inserción contempla los siguientes pasos:

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 bucle o iteración `i (desde i=1 hasta n-1)` se explora la sublista `A[i-1] . . A[0]` buscando la posición correcta de inserción; 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 la posición correcta.

6.5.2. Codificación en C del algoritmo de ordenación por inserción

La función `ordInsercion()` tiene dos argumentos, el array `a[]` que se va a ordenar crecientemente, y el número de elementos `n`. En la codificación se supone que los elementos son de tipo entero.

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

  for (i = 1; i < n; i++)
  {
    /* índice j explora la sublista a[i-1]..a[0] buscando la
      posición correcta del elemento destino, lo asigna a a[j] */
```

```
        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;
      }
    }
```

El análisis del algoritmo de inserción se realizó como ejemplo en el Apartado 2.6.2, se determinó que la complejidad del algoritmo es $0(n^2)$, *complejidad cuadrática*.

6.6. 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 comprensión y programación; 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. La técnica consiste en hacer varias pasadas a través del array. En cada pasada, se comparan parejas sucesivas de elementos. Si una pareja está en orden creciente (o los valores son idénticos), se dejan los valores como están. Si una pareja está en orden decreciente, sus valores se intercambian en el array.

6.6.1. Algoritmo de la burbuja

En el caso de un array (lista) 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 0 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 0 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 1 se realizan las mismas comparaciones e intercambios, terminando con el elemento 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]`.