```
        intercambiar(a[i], a[indiceMenor]);
    }
  }

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

8.4.2. Complejidad del algoritmo de *selección*

El análisis del algoritmo, con el fin de determinar la función *tiempo de ejecución* `t(n)`, es sencillo y claro, ya que requiere un número fijo de comparaciones que sólo dependen del tamaño del *array* y no de la distribución inicial de los datos. El término *dominante* del algoritmo es el bucle externo que anida a un bucle interno. Por ello, el número de comparaciones que realiza el algoritmo es el número decreciente de iteraciones del bucle interno: `n-1, n-2, ... 2, 1` (`n` es el número de elementos). La suma de los términos de la sucesión se ha obtenido en el apartado anterior, 8.3.2, y se ha comprobado que depende de n^2. Como conclusión, la complejidad del algoritmo de selección es $0(n^2)$.

8.5. ORDENACIÓN POR INSERCIÓN

Este método de ordenación 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 que ya está ordenada. Así el proceso en el caso de la lista de enteros `a[] = 50, 20, 40, 80, 30`

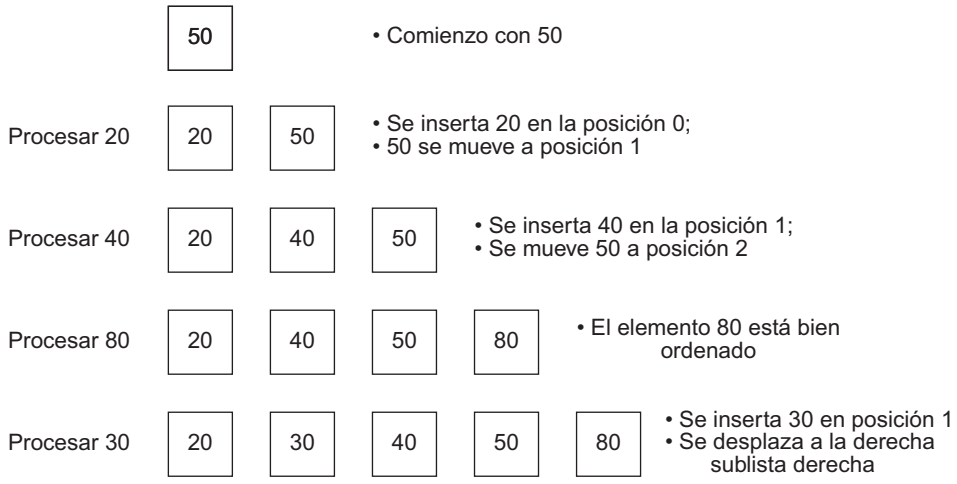

Figura 8.1. Método de ordenación por inserción

8.5.1. 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 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.

8.5.2. Codificación del algoritmo de ordenación por *inserción*

La codificación del algoritmo se realiza en la función `ordInsercion()`. Los elementos del *array* son de tipo entero, en realidad puede ser cualquier tipo básico y ordinal.

```
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 po-
       sicion 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;
  }
}
```

8.5.3. Complejidad del algoritmo de *inserción*

A la hora de analizar este algoritmo se observa que el número de instrucciones que realiza depende del bucle externo que anida al bucle condicional `while`. Siendo `n` el número de elementos, el bucle externo realiza `n - 1` *pasadas*, por cada una de ellas y *en el peor de los casos* (`aux` siempre menor que `a[j-1]`), el bucle interno `while` itera un número creciente de veces que da lugar a la sucesión: `1, 2, 3, ... n-1` (para `i == n-1`). La suma de los términos de la sucesión se ha obtenido en el Apartado 8.3.2, y se ha comprobado que el término dominante es n^2. Como conclusión, la complejidad del algoritmo de inserción es $0(n^2)$.