```
        if (a[i] > a[j])
        {
          intercambiar(a[i], a[j]);
        }
}
```

8.3.2. Complejidad del algoritmo de ordenación por intercambio

El algoritmo consta de dos bucles anidados, está *dominado* por los dos bucles. De ahí, que el análisis del algoritmo en relación a la complejidad sea inmediato, siendo `n` el número de elementos, el primer bucle hace `n-1` pasadas y el segundo `n-i-1` comparaciones en cada pasada (`i` es el índice del bucle externo, `i = 0 .. n-2`). El número total de comparaciones se obtiene desarrollando la sucesión matemática formada para los distintos valores de `i`:

```
n-1, n-2, n-3, ... , 1
```

El número de comparaciones se obtiene sumando los términos de la sucesión: $\frac{(n-1)\cdot n}{2}$

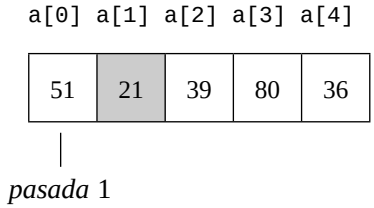
y un número similar de intercambios *en el peor de los casos*. Entonces, el número de comparaciones y de intercambios *en el peor de los casos* es $(n-1)\cdot n/2 = (n^2 - n)/2$. El término dominante es n^2, por tanto la complejidad es $0(n^2)$.

8.4. ORDENACIÓN POR SELECCIÓN

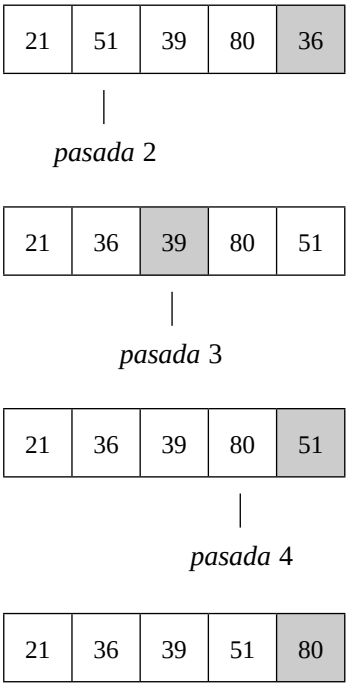
Considérese el algoritmo para ordenar un array `a[]` en orden ascendente; es decir, si el array tiene *n* elementos, se trata de ordenar los valores del array de modo que `a[0]` sea el valor más pequeño, el valor almacenado en `a[1]` el siguiente más pequeño, y así hasta `a[n-1]` que ha de contener el elemento 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* (hay que considerar el índice `0`).

La *pasada* inicial busca el elemento más pequeño de la lista y se intercambia con `a[0]`, primer elemento de la lista. 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, que 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*, en ese momento la lista desordenada se reduce a un elemento (el mayor de la lista) y el *array* completo ha quedado ordenado.

El siguiente ejemplo práctico ayudará a la comprensión del algoritmo:



Pasada 1*: Seleccionar* `21`
Intercambiar `21` y `a[0]`



Pasada 2: Seleccionar `36`
Intercambiar `36` y `a[1]`

Pasada 3: Seleccionar `39`
Intercambiar `39` y `a[2]`

Pasada 4: Seleccionar `51`
Intercambiar `51` y `a[3]`

Array ordenado

8.4.1. Codificación del algoritmo de *selección*

La función `ordSeleccion()` ordena un array de números reales de `n` elementos. El proceso de selección explora, en la pasada `i`, la sublista `a[i]` a `a[n-1]` y fija el índice del elemento más pequeño. Después de terminar la exploración, los elementos `a[i]` y `a[indiceMenor]` se intercambian.

```
  /*
      ordenar un array de n elementos de tipo double
      utilizando el algoritmo de ordenación por selección
  */

void ordSeleccion (double a[], int n)
{
  int indiceMenor, i, j
         // ordenar a[0]..a[n-2] y a[n-1] en cada pasada
  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]);
    }
  }

  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`

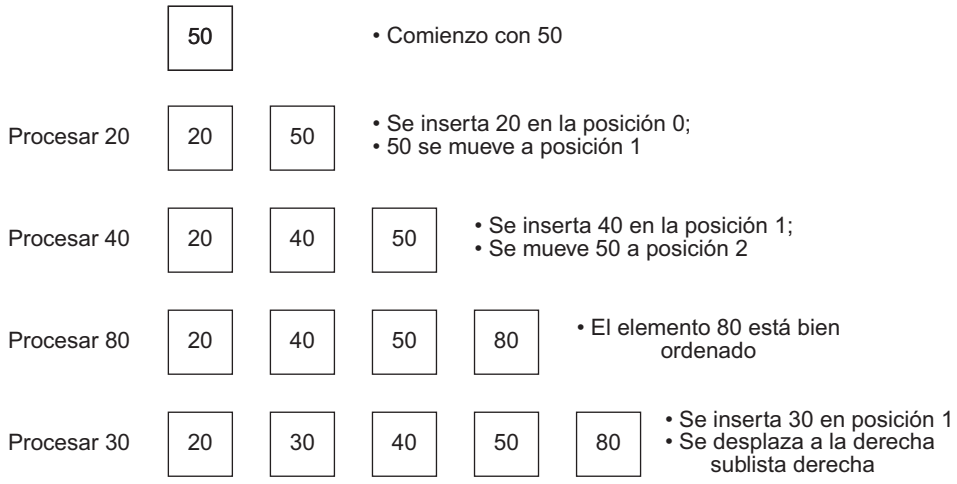


Figura 8.1. Método de ordenación por inserción