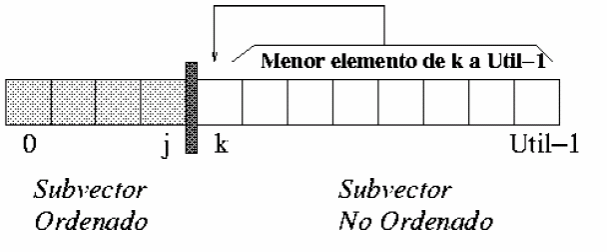
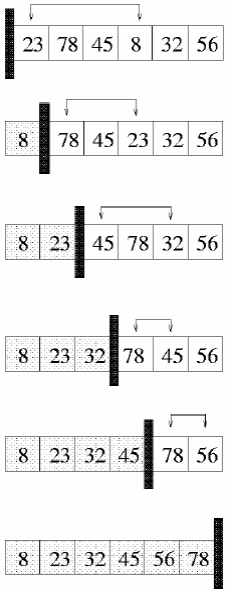
Algoritmos de ordenación

Ordenación por selección

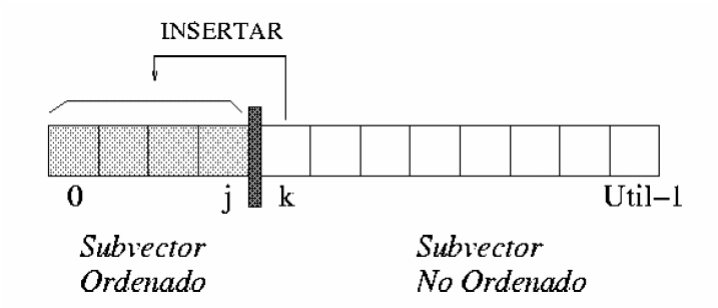



```
static void ordenarSeleccion (double v[])
{
  double tmp;
  int i, j, pos_min;
  int N = v.length;

  for (i=0; i<N-1; i++) {

      // Menor elemento del vector v[i..N-1]

      pos_min = i;

      for (j=i+1; j<N; j++)
          if (v[j]<v[pos_min])
             pos_min = j;

      // Coloca el mínimo en v[i]

      tmp = v[i];
      v[i] = v[pos_min];
      v[pos_min] = tmp; =
  }
}
```

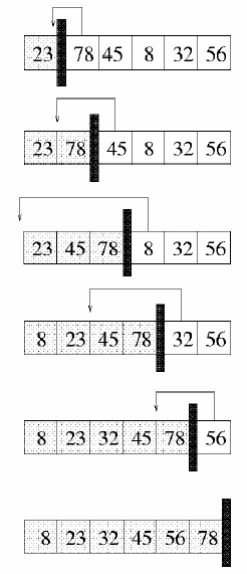


En cada iteración, se selecciona el menor elemento del subvector no ordenado y se intercambia con el primer elemento de este subvector.

Ordenación por inserción

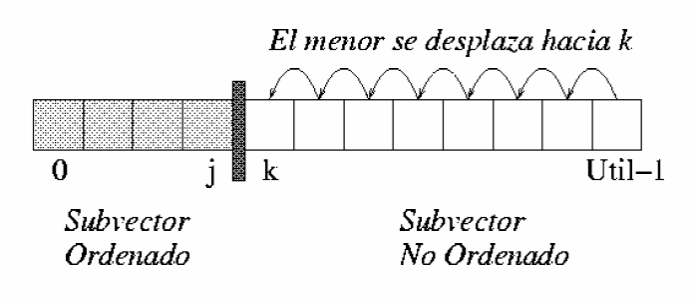



```
static void ordenarInsercion (double v[])
{
  double tmp;
  int i, j;
  int N = v.length;

  for (i=1; i<N; i++) {

      tmp = v[i];

      for (j=i; (j>0) && (tmp<v[j-1]); j--)
          v[j] = v[j-1];

      v[j] = tmp;
  }
}
```



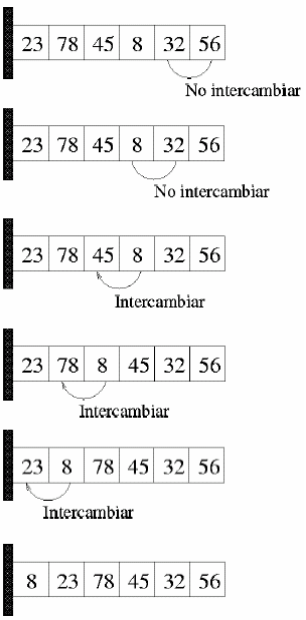
En cada iteración, se inserta un elemento del subvector no ordenado en la posición correcta dentro del subvector ordenado.

Ordenación por intercambio directo (método de la burbuja)

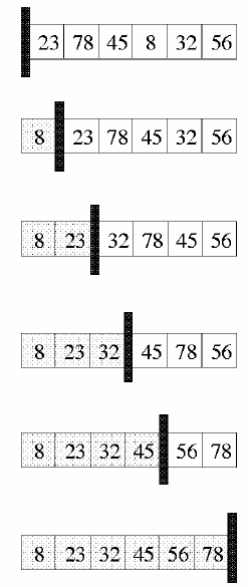



```
static void ordenarBurbuja (double v[])
{
  double tmp;
  int i, j;
  int N = v.length;

  for (i=1; i<N; i++)
      for (j=N-1; j>=i; j--)
          if (v[j] < v[j-1]) {
             tmp = v[j];
             v[j] = v[j-1];
             v[j-1] = tmp;
          }
}
```

En cada iteración



Estado del vector tras cada iteración:



Ordenación rápida (QuickSort)

1. Se toma un elemento arbitrario del vector, al que denominaremos pivote (p).

2. Se divide el vector de tal forma que todos los elementos a la izquierda del pivote sean menores que él, mientras que los que quedan a la derecha son mayores que él.

3. Ordenamos, por separado, las dos zonas delimitadas por el pivote.

```
static void quicksort
            (double v[], int izda, int dcha)
{
  int pivote;  // Posición del pivote

  if (izda<dcha) {

     pivote = partir (v, izda, dcha);

     quicksort (v, izda, pivote-1);

     quicksort (v, pivote+1, dcha);
  }
}
```

Uso:

```
    quicksort (vector, 0, vector.length-1);
```

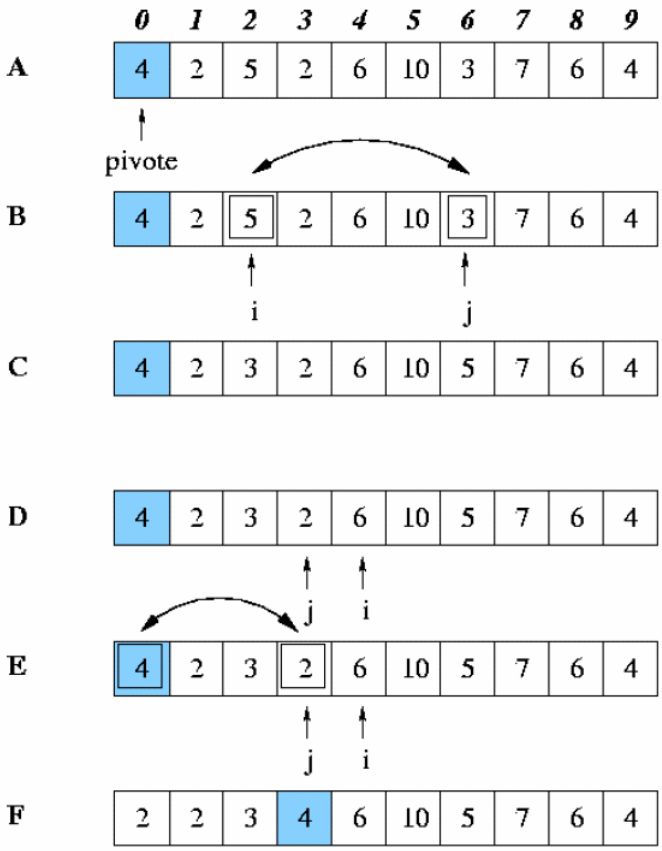
Obtención del pivote

Mientras queden elementos mal colocados respecto al pivote:

a. Se recorre el vector, de izquierda a derecha, hasta encontrar un elemento situado en una posición i tal que v[i] > p.

b. Se recorre el vector, de derecha a izquierda, hasta encontrar otro elemento situado en una posición j tal que v[j] < p.

c. Se intercambian los elementos situados en las casillas i y j (de modo que, ahora, v[i] < p < v[j]).



```
  /**
   * División del vector en dos partes
   * @see quicksort
   *
   * @param primero Índice del primer elemento
   * @param ultimo Índice del último elemento
   * @return Posición del pivote
   *
   * @pre (primero>=0)
   *      && (primero<=ultimo)
   *      && (ultimo<v.length)
   */

private static int partir
        (double v[], int primero, int ultimo)
{
  double pivote = v[primero]; // Valor del pivote
  double temporal;            // Variable auxiliar
  int izda = primero+1;
  int dcha = ultimo;

  do {  // Pivotear…

    while ((izda<=dcha) && (v[izda]<=pivote))
          izda++;

    while ((izda<=dcha) && (v[dcha]>pivote))
          dcha--;

    if (izda < dcha) {
       temporal = v[izda];
       v[izda] = v[dcha];
       v[dcha] = temporal;
       dcha--;
       izda++;
    }

  } while (izda <= dcha);

  // Colocar el pivote en su sitio
  temporal = v[primero];
  v[primero] = v[dcha];
  v[dcha] = temporal;

  return dcha; // Posición del pivote
}
```