# ALGORITMOS Y ESTRUCTURAS DE DATOS

## Aproximación en C

Charles F. Bowman

OXFORD

104559

```
void clasi_rap( int datos[], int inf, int sup )
{
     int  i, j, tmp, elem_de_divi;

     if( sup > inf ){
          elem_de_divi = datos[sup];
          i = inf-1;
          j = sup;
          while( 1 ){
               while( datos[++i] < elem_de_divi )
                    ;
               while( datos[--j] > elem_de_divi )
                    ;
               if( i >= j )
                    break;

               tmp = datos[i];
               datos[i] = datos[j];
               datos[j] = tmp;
          }

          tmp = datos[i];
          datos[i] = datos[sup];
          datos[superior] = tmp;

          clasi_rap( datos, inf, i-1 );
          clasi_rap( datos, i+1, sup );
     }
}
```

**Cuadro 9.4**
Algoritmo de ordenamiento rápido (*quicksort*).

Ahora estamos listos para implementar el algoritmo fundamental. Como se muestra en el cuadro 9.4, la función `clasi_rap()` necesita tres argumentos. El primero apunta al arreglo de datos; los otros dos son, respectivamente, sus límites superior e inferior. (De inmediato veremos la necesidad de tener un índice que siga la pista del límite inferior.) La primera llamada establece estos valores en 0 y $n$ (el tamaño del arreglo), respectivamente. Observe que cuando

```
void
pasa_comb( int destino[], int fuente[], int tamano, int long )
{
        int inf = 0;

        while( inf < tamano - 2*long ){
                combina( destino, fuente, inf, inf+long-1, inf+2*long-1 );
                inf += 2 * long;
        }

        if( inf+long-1 < tamano ){
                combina( destino, fuente, inf, inf+long-1, tamano );
        } else {
                while( inf <= tamano ){
                        destino[inf] + fuente[inf];
                        inf++;
                }
        }
}

void clasi_combi( int datos[], int tamano )
{
        int             tmp[ 2048 ];            /* malloc */
        int long = 1;                           /* longitud del subarchivo */

        while( long < tamano ){
                pasa_comb( tmp, datos, tamano, long );
                long *= 2;
                pasa_comb( datos, tmp, tamano, long );
                long *= 2;
        }
}
```

**Cuadro 9.9**
Algoritmo de ordenamiento por combinación (*mergesort*).

# ALGORITMOS Y ESTRUCTURAS DE DATOS

## Aproximación en C

En esta obra se analizan las técnicas de implementación y se explican los conceptos de abstracción de datos y eficiencia de los algoritmos. Más importante aún, no presenta los algoritmos como una lista de compras; más bien, proporciona conocimientos prácticos sobre el proceso mismo del diseño de los algoritmos.

Apoyado en C, que es uno de los lenguajes de programación más utilizados, y sin perder de vista las situaciones que enfrentará el programador en su vida cotidiana, el libro guía al lector en cada uno de los pasos para [illegible] de algoritmos, con énfasis en las estructuras de datos [illegible] etc.) que los pondrán en marcha.

En este libro Charles F. Bowman explica por qué, al diseñar algoritmos, el programador debe:

- Analizar el problema antes que todo, procurando [illegible] información necesaria.
- Diseñar estructuras de datos adecuadas para el problema [illegible]
- Bosquejar el algoritmo a fin de analizar su complejidad.
- Instrumentarlo en el lenguaje de programación más conveniente y luego probarlo.
- Evaluar su desempeño y eficiencia para el logro de su cometido.

OXFORD
UNIVERSITY PRESS

UT UNIVERSIDAD TECNOLÓGICA DE SAN JUAN DEL RÍO