Algoritmos 2020/21

Grado en Ingeniería Informática

Práctica 4

Límite para la entrega: lunes 7 de diciembre, a las 23:59

Ordenación rápida

Algoritmo de ordenación rápida (*quicksort*) con selección aleatoria del pivote y un umbral para detectar vectores pequeños:

```
procedimiento OrdenarAux (V[izq..der])
  si izq+UMBRAL <= der entonces
    x := {nº aleatorio en el rango [izq..der]};
    pivote := V[x];
    intercambiar (V[izq], V[x]);
    i := izq + 1;
    j := der;
    mientras i <= j hacer
      mientras i <= der y V[i] < pivote hacer
        i := i + 1
      fin mientras;
      mientras V[j] > pivote hacer
        j := j - 1
      fin mientras;
      si i <= j entonces
        intercambiar (V[i], V[j]);
        i := i + 1;
        j := j - 1
      fin si
    fin mientras;
    intercambiar (V[izq], V[j]);
    OrdenarAux (V[izq..j-1]);
    OrdenarAux (V[j+1..der])
    fin si
fin procedimiento

procedimiento Ordenación Rápida (V[1..n])
  OrdenarAux(V[1..n]);
  si (UMBRAL > 1) entonces
    Ordenación por Inserción (V[1..n])
  fin si
fin procedimiento
```

Se pide:

1. Implemente el algoritmo de ordenación rápida.

```
void ord_rapida(int v [], int n) {
  rapida_aux(v, 0, n-1);
  if (UMBRAL > 1)
    ord_ins(v, n);
}
```

2. Valide el correcto funcionamiento de la implementación (con umbral $= 1$).

3. Ejecute el algoritmo con vectores de distinto tamaño y en distintas situaciones iniciales (vector ordenado ascendente o descendentemente, o desordenado), y con distintos valores de umbral: 1 (no se realiza la llamada a la ordenación por inserción), 10 y 100.

4. Compare entre si los tiempos obtenidos para cada umbral usado. En función de la situación inicial del vector, ¿con qué umbral se obtienen los mejores tiempos? ¿por qué?

5. Calcule empíricamente la complejidad del algoritmo para cada una de las diferentes situaciones iniciales del vector y cada uno de los umbrales (i.e., 9 tablas).