Ordenación

Fernando Bobillo

Enunciado

- Ordenar una lista de números enteros de modo que cada nº sea menor o igual que su sucesor en la lista ordenada

Enunciado

- Entrada: lista $a_0, a_1, \ldots, a_{n-1}$
- Salida: reordenación $a_{\sigma(0)}, a_{\sigma(1)}, \ldots, a_{\sigma(n-1)}$ tal que

$$a_{\sigma(0)} \leq a_{\sigma(1)} \leq \ldots \leq a_{\sigma(n-1)}$$

donde σ es una permutación de los índices {0, 1, … , n – 1}

- Ejemplo:
  - Entrada: {8, 15, 42, 23, 4, 16}
  - Salida: {4, 8, 15, 16, 23, 42}

Selección

- El algoritmo de ordenamiento por selección (*select-sort*) se basa en la búsqueda de mínimo (si el orden es ascendente)

http://www.youtube.com/watch?v=Ns4TPTC8whw

Idea del select-sort

- Tenemos que ordenar una lista de n elementos
- Buscamos el elemento mínimo de todo la lista
  - Supongamos que es el elemento a[i]
- Colocamos ese elemento en la primera posición de la lista
  - Para ello, intercambiamos a[1] y a[i]
  - Ahora la primera posición ya está ordenada
- Buscamos el elemento mínimo de la sublista [2, n], sea a[j]
- Lo colocamos en la 2ª posición intercambiando a[2] y a[j]
- Repetimos el proceso para el resto de la lista

Pseudocódigo

```
seleccion(int[] a)

   para (izq = 0; hasta longitud de a - 1; de uno en uno)

        // Calcular posición del mínimo entre izq y el final
        posMin = izq;
        para (i = izq + 1; hasta longitud de a; de uno en uno)
                si a[i] < a[posMin]
                        posMin = i;

        // Intercambiar los elementos en izq y posMin
        int aux = a[izq];
        a[izq] = a[posMin];
        a[posMin] = aux;
```

Ejemplo

- 1. Vector inicial
  - [44 55 12 42 94 6 18 67]
- 2. Mínimo del subvector [0 , 7]: 6
  - [44 55 12 42 94 6 18 67]

  Intercambio de 6 y 44
  - [6 55 12 42 94 44 18 67]
- 3. Mínimo del subvector [1 , 7]: 12
  - [6 55 12 42 94 44 18 67]

  Intercambio de 55 y 12
  - [6 12 55 42 94 44 18 67]

Ejemplo

- 4. Mínimo del subvector [2 , 7]: 18
  - [6 12 55 42 94 44 18 67]

  Intercambio de 55 y 18
  - [6 12 18 42 94 44 55 67]
- 5. Mínimo del subvector [3 , 7]: 42
  - [6 12 18 42 94 44 55 67]

  Intercambio de 42 y 42
  - [6 12 18 42 94 44 55 67]

Ejemplo

- 6. Mínimo del subvector [4 , 7]: 44
  - [6 12 18 42 94 44 55 67]

  Intercambio de 94 y 44
  - [6 12 18 42 44 94 55 67]
- 7. Mínimo del subvector [5 , 7]: 55
  - [6 12 18 42 44 94 55 67]

  Intercambio de 94 y 55
  - [6 12 18 42 44 55 94 67]

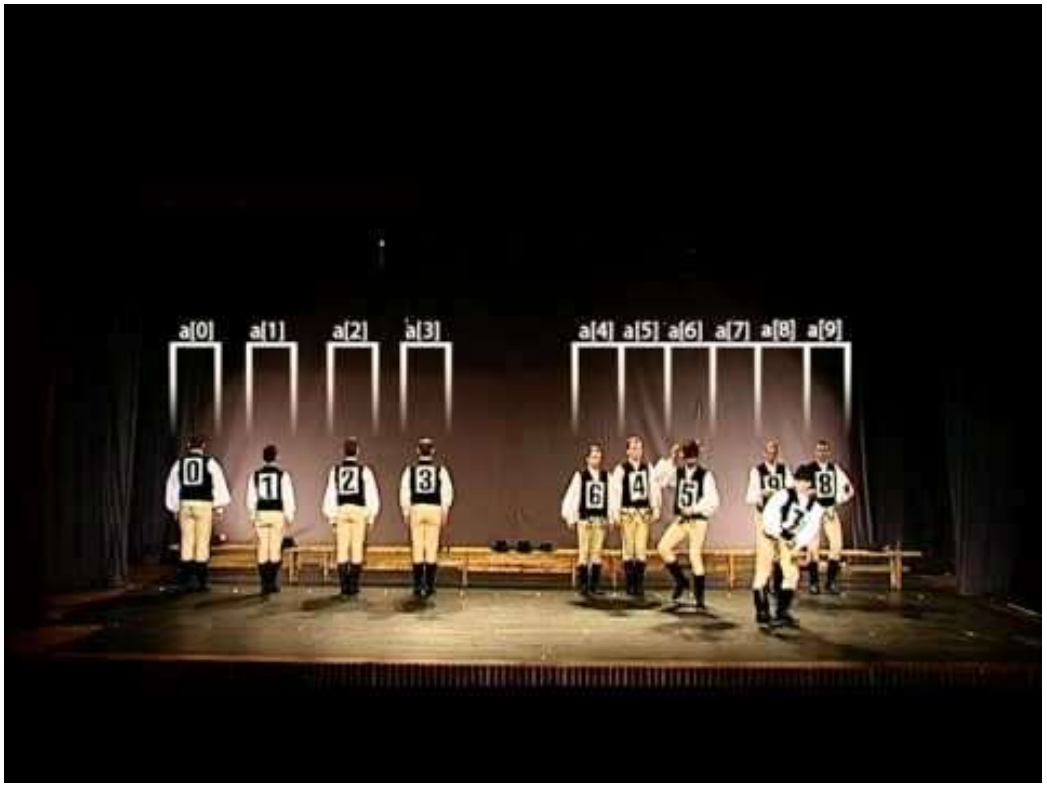
Ejemplo

- 8. Mínimo del subvector [6 , 7]: 67
  - [6 12 18 42 44 55 94 67]

  Intercambio de 94 y 67
  - [6 12 18 42 44 55 67 94]
- 9. Vector ordenado
  - [6 12 18 42 44 55 67 94]

Quicksort

- El ordenamiento rápido (*quicksort*) se basa en la técnica de divide y vencerás y en promedio es más rápido que otros



http://www.youtube.com/watch?v=ywWBy6J5gz8

Idea del quicksort

- Se elige un elemento de la lista de elementos llamado pivote
- Recolocar los demás elementos, de modo que a un lado del pivote queden los menores que él y a otro lado los mayores
  - En este momento, el pivote ocupa exactamente el lugar que le corresponde en la lista ordenada
- La lista queda separada en dos sublistas
  - Una formada por los elementos a la izquierda del pivote
  - Otra por los elementos a su derecha
- Se repite este proceso de forma recursiva para cada sublista mientras éstas contengan más de un elemento
- Una vez terminado, todos los elementos estarán ordenados

Consideraciones sobre el algoritmo

```
public static void quicksort(int[] a, int izq, int der) {
  int i = izq;
  int j = der;
  ...
}
```

- La lista de enteros se va a almacenar en un vector `a`
- El algoritmo es recursivo y en cada paso requiere conocer donde empieza y termina la sublista con la que trabajar
  - Se utilizarán dos variables enteras llamadas `izq` y `der`
- Búsqueda en la sublista de la izquierda, con la variable `i`
- Búsqueda en la sublista de la derecha, con la variable `j`
- Al final del proceso el vector `a` queda ordenado

Consideraciones sobre el algoritmo

- Se elige como pivote el valor situado en el centro (si hay un nº impar de elementos) o el valor más cercano (si es par)
- Las búsquedas en la sublistas de la izquierda pretenden identificar, entre aquellos números que no han sido visitados, el siguiente valor que es mayor que el pivote
- Las búsquedas en la sublistas de la derecha pretenden encontrar en el siguiente valor que es menor que él
- Si ambos se encuentran, se intercambian
- El proceso se repite hasta que no haya más intercambios

Código 1

```
public static void quicksort(int[] a, int izq, int der) {
  int i = izq;
  int j = der;
  int pivote = a[(izq + der) / 2];
  do {
    while (a[i] < pivote)
      i++;
    while (a[j] > pivote)
      j--;
    if (i <= j) {
      if (i < j) {
        int aux = a[i];
        a[i] = a[j];
        a[j] = aux;
      }
      i++;
      j--;
    }
  } while (i <= j);
  if (izq < j)
    quicksort(a, izq, j);
  if (i < der)
    quicksort(a, i, der);
}
```

Código 1

```
public static void main(String[] args) {
    Random r = new Random();
    Scanner sc = new Scanner(System.in);
    System.out.print("Dimensión de la lista de números: ");
    int size = sc.nextInt();
    int [] a = new int[size];
    for (int i = 0; i < size; i++) {
         int x = r.nextInt();
         x = Math.abs(x) % 50;
         a[i] = x;
    }
    System.out.println("Vector inicial");
    for (int i = 0; i < size; i++)
         System.out.print(a[i] + "\t");
    quicksort(a, 0, size - 1);
    System.out.println("\nVector ordenado");
    for (int i = 0; i<size; i++)
         System.out.print(a[i] + "\t");
}
```

Ejemplo

- 1. Vector inicial
  - [44 55 12 42 94 6 18 67]
- 2. Ordenar subvector [0 , 7]

  Pivote: 42

  Intercambio: 44 y 18
  - [18 55 12 42 94 6 44 67]

  Intercambio: 55 y 6
  - [18 6 12 42 94 55 44 67]

  Llamadas recursivas: ordenar subvectores [0, 2] y [4, 7]

Ejemplo

- 3. Ordenar subvector [0 , 2]
  - [18 6 12 42 94 55 44 67]

  Pivote: 6

  Intercambio: 18 y 6
  - [6 18 12 42 94 55 44 67]

  Llamada recursiva: ordenar subvectores [1, 2]

- 4. Ordenar subvector [1 , 2]
  - [6 18 12 42 94 55 44 67]

  Pivote: 18

  Intercambio: 18 y 12
  - [6 12 18 42 94 55 44 67]

Ejemplo

- 5. Ordenar subvector [4 , 7]
  - [6 12 18 42 94 55 44 67]

  Pivote: 55

  Intercambio: 94 y 44
  - [6 12 18 42 44 55 94 67]

  Llamada recursiva: ordenar subvector [6, 7]

- 6. Ordenar subvector [6 , 7]
  - [6 12 18 42 44 55 94 67]

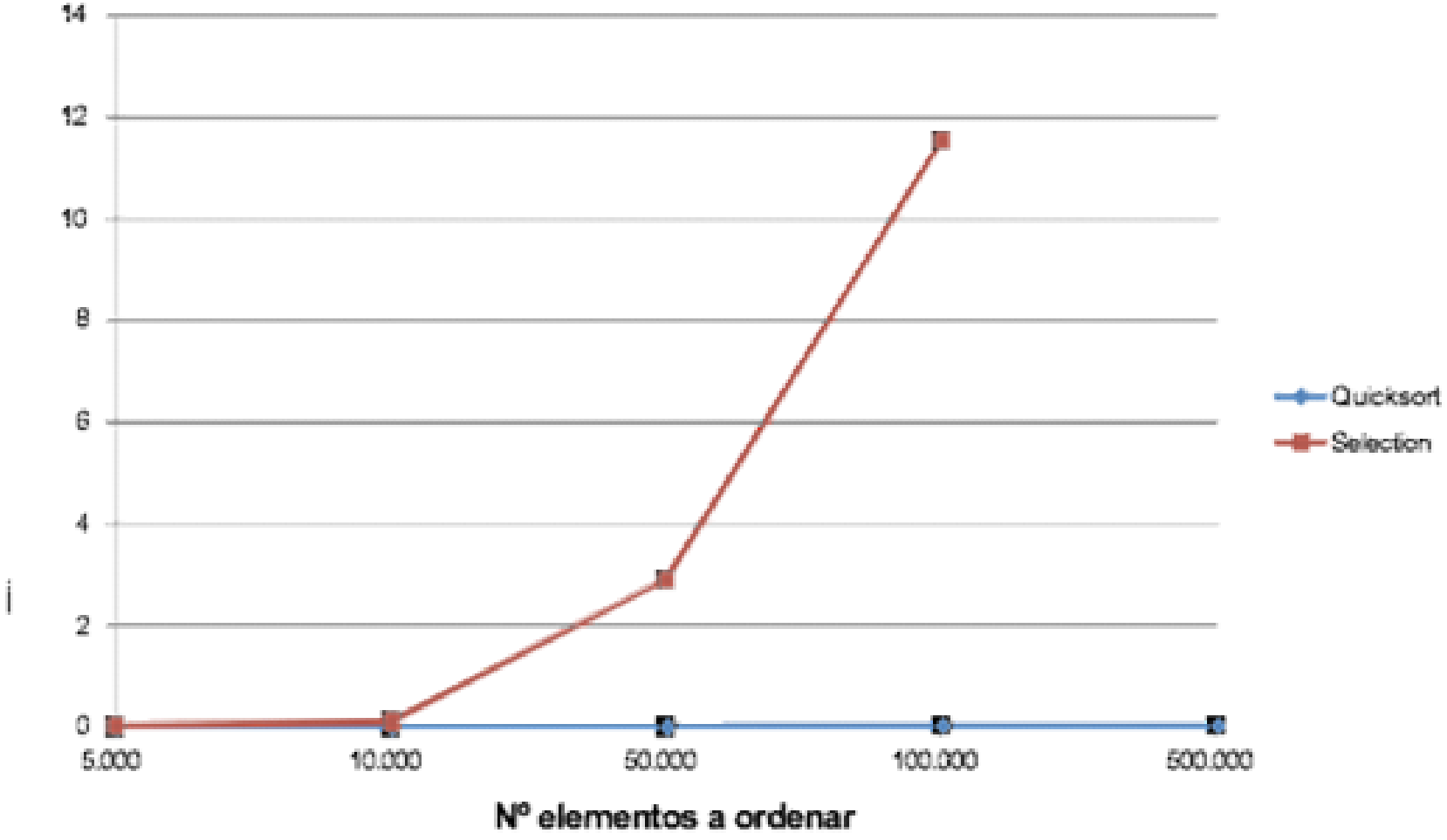
  Pivote: 94

  Intercambio: 94 y 67
  - [6 12 18 42 44 55 67 94]

Comparación de algoritmos

Name	Best	Average	Worst	Memory	Stable
Bubble sort	n	n^2	n^2	1	Yes
Selection sort	n^2	n^2	n^2	1	No
Insertion sort	n	n^2	n^2	1	Yes
Merge sort	$n \log n$	$n \log n$	$n \log n$	worst case is n	Yes
In-place merge sort	—	—	$n (\log n)^2$	1	Yes
Quicksort	$n \log n$	$n \log n$	n^2	$\log n$ on average, worst case is n	typical in-place sort is not stable;
Heapsort	$n \log n$	$n \log n$	$n \log n$	1	No

Comparación de algoritmos



Código completo en Java

¡TU TURNO!