Algoritmos y estructuras de datos

Ordenamiento por partición (*Quicksort*)

Francisco Javier Zaragoza Martínez

Universidad Autónoma Metropolitana Unidad Azcapotzalco
Departamento de Sistemas

29 de noviembre de 2023

Citas

Alexander Pope

¡Qué delgadas particiones dividen la percepción del pensamiento!

William Shakespeare

Así crecimos juntos, como una cereza doble, aparentemente partida, pero aún una unión en partición.

Anónimo

Delgadas particiones dividen los límites donde residen el bien y el mal, que nada es perfecto aquí abajo, pero la dicha aún limita la aflicción.

Ordenamiento interno

Definición

Problema abstracto

Dado un arreglo A con n elementos, se desea reacomodar sus elementos de modo que $A_0 \leq A_1 \leq \cdots \leq A_{n-1}$.

Problema concreto

Dado un arreglo `int a[MAX]` con `int n` elementos en las posiciones `a[0], ..., a[n-1]` se desea reacomodar sus elementos de modo que `a[0] <= a[1] <= ... <= a[n-1]`.

Ordenamiento interno

Un problema, mil soluciones

El problema de ordenamiento interno es uno de los más estudiados en la computación y para el que se conocen muchos algoritmos. Algunos son:

- Ordenamiento por cuenta (*Counting sort*).
- Ordenamiento de burbuja (*Bubble sort*).
- Ordenamiento por inserción (*Insertion sort*).
- Ordenamiento por mezcla (*Merge sort*).
- Ordenamiento por selección (*Selection sort*).
- Ordenamiento por partición (*Quicksort*).
- Ordenamiento por árbol (*Tree sort*).
- Ordenamiento por montículo (*Heapsort*).

Selección directa

Idea: Selecciona el máximo elemento y ponlo en su lugar

Ejemplo

3 1 4 1 5 9 2 6 5 3 5 8 9 7 9 3

Selección directa

Idea: Selecciona el máximo elemento y ponlo en su lugar

Ejemplo

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	4	1	5	9	2	6	5	3	5	8	9	7	3	9

Selección directa

Idea: Selecciona el máximo elemento y ponlo en su lugar

Ejemplo

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	4	1	5	9	2	6	5	3	5	8	9	7	3	**9**
3	1	4	1	5	9	2	6	5	3	5	8	3	7	**9**	**9**

Selección directa

Idea: Selecciona el máximo elemento y ponlo en su lugar

Ejemplo

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	4	1	5	9	2	6	5	3	5	8	9	7	3	**9**
3	1	4	1	5	9	2	6	5	3	5	8	3	7	**9**	**9**
3	1	4	1	5	7	2	6	5	3	5	8	3	**9**	**9**	**9**

Selección directa

Idea: Selecciona el máximo elemento y ponlo en su lugar

Ejemplo

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	4	1	5	9	2	6	5	3	5	8	9	7	3	9
3	1	4	1	5	9	2	6	5	3	5	8	3	7	9	**9**
3	1	4	1	5	7	2	6	5	3	5	8	3	9	**9**	**9**
3	1	4	1	5	7	2	6	5	3	5	3	8	**9**	**9**	**9**

Selección directa

Idea: Selecciona el máximo elemento y ponlo en su lugar

Ejemplo

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	4	1	5	9	2	6	5	3	5	8	9	7	3	9
3	1	4	1	5	9	2	6	5	3	5	8	3	7	9	**9**
3	1	4	1	5	7	2	6	5	3	5	8	3	9	**9**	**9**
3	1	4	1	5	7	2	6	5	3	5	3	8	**9**	**9**	**9**
3	1	4	1	5	3	2	6	5	3	5	7	**8**	**9**	**9**	**9**

Selección directa

Idea: Selecciona el máximo elemento y ponlo en su lugar

Ejemplo

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	4	1	5	9	2	6	5	3	5	8	9	7	3	9
3	1	4	1	5	9	2	6	5	3	5	8	3	7	9	**9**
3	1	4	1	5	7	2	6	5	3	5	8	3	9	**9**	**9**
3	1	4	1	5	7	2	6	5	3	5	3	8	**9**	**9**	**9**
3	1	4	1	5	3	2	6	5	3	5	7	**8**	**9**	**9**	**9**
3	1	4	1	5	3	2	5	5	3	6	**7**	**8**	**9**	**9**	**9**

Selección directa

Idea: Selecciona el máximo elemento y ponlo en su lugar

Ejemplo

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	4	1	5	9	2	6	5	3	5	8	9	7	3	9
3	1	4	1	5	9	2	6	5	3	5	8	3	7	9	9
3	1	4	1	5	7	2	6	5	3	5	8	3	9	9	9
3	1	4	1	5	7	2	6	5	3	5	3	8	9	9	9
3	1	4	1	5	3	2	6	5	3	5	7	8	9	9	9
3	1	4	1	5	3	2	5	5	3	6	7	8	9	9	9
3	1	4	1	5	3	2	5	3	5	6	7	8	9	9	9

Selección directa

Idea: Selecciona el máximo elemento y ponlo en su lugar

Ejemplo

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	4	1	5	9	2	6	5	3	5	8	9	7	3	9
3	1	4	1	5	9	2	6	5	3	5	8	3	7	9	9
3	1	4	1	5	7	2	6	5	3	5	8	3	9	9	9
3	1	4	1	5	7	2	6	5	3	5	3	8	9	9	9
3	1	4	1	5	3	2	6	5	3	5	7	8	9	9	9
3	1	4	1	5	3	2	5	5	3	6	7	8	9	9	9
3	1	4	1	5	3	2	5	3	5	6	7	8	9	9	9
3	1	4	1	5	3	2	3	5	5	6	7	8	9	9	9

Selección directa

Idea: Selecciona el máximo elemento y ponlo en su lugar

Ejemplo

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	4	1	5	9	2	6	5	3	5	8	9	7	3	9
3	1	4	1	5	9	2	6	5	3	5	8	3	7	9	9
3	1	4	1	5	7	2	6	5	3	5	8	3	9	9	9
3	1	4	1	5	7	2	6	5	3	5	3	8	9	9	9
3	1	4	1	5	3	2	6	5	3	5	7	8	9	9	9
3	1	4	1	5	3	2	5	5	3	6	7	8	9	9	9
3	1	4	1	5	3	2	5	3	5	6	7	8	9	9	9
3	1	4	1	5	3	2	3	5	5	6	7	8	9	9	9
3	1	4	1	3	3	2	5	5	5	6	7	8	9	9	9

Selección directa

Idea: Selecciona el máximo elemento y ponlo en su lugar

Ejemplo

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	4	1	5	9	2	6	5	3	5	8	9	7	3	9
3	1	4	1	5	9	2	6	5	3	5	8	3	7	9	9
3	1	4	1	5	7	2	6	5	3	5	8	3	9	9	9
3	1	4	1	5	7	2	6	5	3	5	3	8	9	9	9
3	1	4	1	5	3	2	6	5	3	5	7	8	9	9	9
3	1	4	1	5	3	2	5	5	3	6	7	8	9	9	9
3	1	4	1	5	3	2	5	3	5	6	7	8	9	9	9
3	1	4	1	5	3	2	3	5	5	6	7	8	9	9	9
3	1	4	1	3	3	2	5	5	5	6	7	8	9	9	9
3	1	2	1	3	3	4	5	5	5	6	7	8	9	9	9

Selección directa

Implementación de la selección del máximo

Tenemos un arreglo desordenado `a[0]`, `a[1]`, ..., `a[j]` en el que queremos encontrar el máximo `a[i]` e intercambiarlo por `a[j]`.

```
void maximo(int j, int a[]) {
  int i = j;                          // maximo en i = j
  for (int k = j-1; k >= 0; k--)
    if (a[k] > a[i])                  // nuevo maximo
      i = k;
  intercambio(a, i, j);
}
```

Selección directa

Implementación iterativa

Seleccionamos el máximo a[n-1], ..., a[0] y lo ponemos en ese orden en su lugar.

```
void seleccion(int n, int a[]) {
  for (int j = n-1; j > 0; j--)
    maximo(j, a);
}
```

Selección directa

Implementación iterativa

Seleccionamos el máximo `a[n-1]`, ..., `a[0]` y lo ponemos en ese orden en su lugar.

```
void seleccion(int n, int a[]) {
  for (int j = n-1; j > 0; j--)
    maximo(j, a);
}
```

Cantidad de comparaciones

La llamada a `maximo(j, a)` hace exactamente `j` comparaciones. En total se hacen $1 + 2 + \cdots + (n-1) = \frac{1}{2}n(n-1) \approx \frac{1}{2}n^2$ comparaciones.

Selección directa

Implementación recursiva

Se pone en su lugar el máximo de los n elementos y luego se ordenan recursivamente con selección directa los demás elementos. Nota que si $n = 1$ no se hace nada (caso base).

```
void seleccion(int n, int a[]) {
  if (n > 1) {
    maximo(n-1, a);
    seleccion(n-1, a);
  }
}
```

Selección directa

Implementación recursiva

Se pone en su lugar el máximo de los n elementos y luego se ordenan recursivamente con selección directa los demás elementos. Nota que si $n = 1$ no se hace nada (caso base).

```
void seleccion(int n, int a[]) {
  if (n > 1) {
    maximo(n-1, a);
    seleccion(n-1, a);
  }
}
```

Cantidad de comparaciones

Si `seleccion(n, a)` hace $T(n)$ comparaciones, entonces $T(n) = T(n-1) + (n-1)$ si $n > 1$ y $T(1) = 0$. En total son $(n-1) + \cdots + 2 + 1 = \frac{1}{2}n(n-1)$ comparaciones.

Selección directa

Ejercicios

1. Reescribe `seleccion` para que ordene decrecientemente.
2. Reescribe `seleccion` para que inicie con el elemento más pequeño.
3. Reescribe `seleccion` como dos ciclos anidados.
4. ¿Cuántas veces se lee del arreglo `a`?
5. ¿Cuántas veces se escribe en el arreglo `a`?

Quicksort

Idea: Pon un elemento en su lugar, separa los demás en menores y mayores

Ejemplo

3 1 4 1 5 9 2 6 5 3 5 8 9 7 9 3

Quicksort

Idea: Pon un elemento en su lugar, separa los demás en menores y mayores

Ejemplo

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	1	2	3	**3**	4	6	5	5	5	8	9	7	9	9

Quicksort

Idea: Pon un elemento en su lugar, separa los demás en menores y mayores

Ejemplo

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	1	2	3	**3**	4	6	5	5	5	8	9	7	9	9
	¿		?		**3**										

Quicksort

Idea: Pon un elemento en su lugar, separa los demás en menores y mayores

Ejemplo

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	1	2	3	**3**	4	6	5	5	5	8	9	7	9	9
	¿		?		**3**										
1	1	2	3	3	**3**										

Quicksort

Idea: Pon un elemento en su lugar, separa los demás en menores y mayores

Ejemplo

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	1	2	3	**3**	4	6	5	5	5	8	9	7	9	9
	¿		?		**3**										
1	1	2	3	3	**3**										
					3	4	6	5	5	5	8	9	7	9	9

Quicksort

Idea: Pon un elemento en su lugar, separa los demás en menores y mayores

Ejemplo

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	1	2	3	**3**	4	6	5	5	5	8	9	7	9	9
	¿		?		**3**										
1	1	2	3	3	**3**										
					3	4	6	5	5	5	8	9	7	9	9
					3				¿		?				

Quicksort

Idea: Pon un elemento en su lugar, separa los demás en menores y mayores

Ejemplo

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	1	2	3	**3**	4	6	5	5	5	8	9	7	9	9
	¿		?		**3**										
1	1	2	3	3	**3**										
					3	4	6	5	5	5	8	9	7	9	9
					3				¿		?				
					3	4	5	5	5	6	7	8	9	9	9

Quicksort

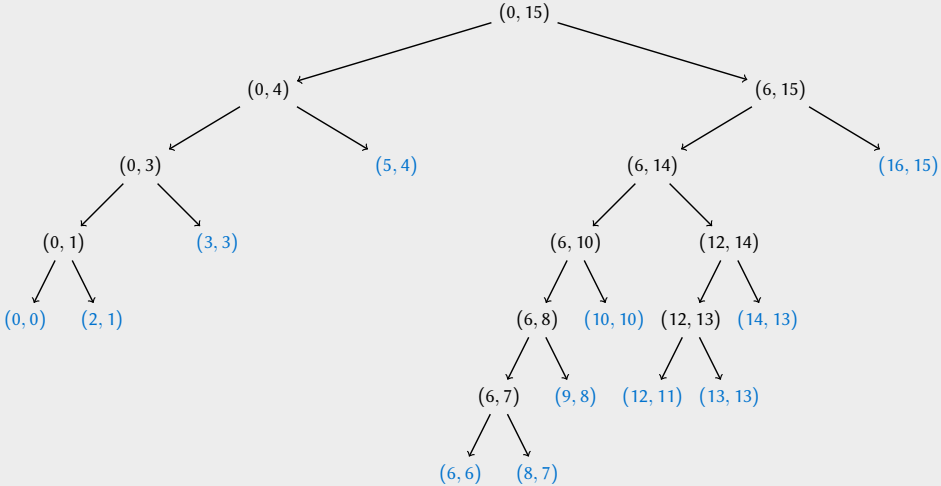
Idea: Pon un elemento en su lugar, separa los demás en menores y mayores

Ejemplo

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	1	2	3	**3**	4	6	5	5	5	8	9	7	9	9
	¿		?		**3**										
1	1	2	3	3	**3**										
					3	4	6	5	5	5	8	9	7	9	9
					3				¿		?				
					3	4	5	5	5	6	7	8	9	9	9
1	1	2	3	3	**3**	4	5	5	5	6	7	8	9	9	9

Quicksort

Árbol de recursión



Quicksort

Implementación de Lomuto para colocar el pivote en su lugar

Tenemos un arreglo desordenado a[i], ..., a[j] en el que queremos buscar el lugar p del elemento a[j] (el pivote). Además queremos reacomodar de modo que el pivote quede en a[p], los menores en a[i], ..., a[p-1] y los mayores en a[p+1], ..., a[j].

```
int pivote(int i, int j, int a[]) {
  int p = i-1;
  for (int k = i; k <= j; k++)
    if (a[k] <= a[j])          // no es mayor que el pivote
      intercambio(a, ++p, k); // hay uno mas que no es mayor
  return p;                    // pivote a[j] queda en a[p]
}
```

Quicksort

Implementación de Lomuto para colocar el pivote en su lugar

3 1 4 1 5 9 2 6 5 3 5 8 9 7 9 3

Quicksort

Implementación de Lomuto para colocar el pivote en su lugar

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	**1**	4	1	5	9	2	6	5	3	5	8	9	7	9	3

Quicksort

Implementación de Lomuto para colocar el pivote en su lugar

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	**1**	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	**4**	1	5	9	2	6	5	3	5	8	9	7	9	3

Quicksort

Implementación de Lomuto para colocar el pivote en su lugar

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	**1**	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	**4**	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	4	**1**	5	9	2	6	5	3	5	8	9	7	9	3

Quicksort

Implementación de Lomuto para colocar el pivote en su lugar

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	**1**	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	**4**	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	4	**1**	5	9	2	6	5	3	5	8	9	7	9	3
3	1	1	4	**5**	9	2	6	5	3	5	8	9	7	9	3

Quicksort

Implementación de Lomuto para colocar el pivote en su lugar

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	**1**	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	**4**	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	4	**1**	5	9	2	6	5	3	5	8	9	7	9	3
3	1	1	4	**5**	9	2	6	5	3	5	8	9	7	9	3
3	1	1	4	5	**9**	2	6	5	3	5	8	9	7	9	3

Quicksort

Implementación de Lomuto para colocar el pivote en su lugar

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	**1**	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	**4**	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	4	**1**	5	9	2	6	5	3	5	8	9	7	9	3
3	1	1	4	**5**	9	2	6	5	3	5	8	9	7	9	3
3	1	1	4	5	**9**	2	6	5	3	5	8	9	7	9	3
3	1	1	4	5	9	**2**	6	5	3	5	8	9	7	9	3

Quicksort

Implementación de Lomuto para colocar el pivote en su lugar

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	**1**	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	**4**	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	4	**1**	5	9	2	6	5	3	5	8	9	7	9	3
3	1	1	4	**5**	9	2	6	5	3	5	8	9	7	9	3
3	1	1	4	5	**9**	2	6	5	3	5	8	9	7	9	3
3	1	1	4	5	9	**2**	6	5	3	5	8	9	7	9	3
3	1	1	2	5	9	4	**6**	5	3	5	8	9	7	9	3

Quicksort

Implementación de Lomuto para colocar el pivote en su lugar

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	**1**	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	**4**	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	4	**1**	5	9	2	6	5	3	5	8	9	7	9	3
3	1	1	4	**5**	9	2	6	5	3	5	8	9	7	9	3
3	1	1	4	5	**9**	2	6	5	3	5	8	9	7	9	3
3	1	1	4	5	9	**2**	6	5	3	5	8	9	7	9	3
3	1	1	2	5	9	4	**6**	5	3	5	8	9	7	9	3
3	1	1	2	5	9	4	6	**5**	3	5	8	9	7	9	3

Quicksort

Implementación de Lomuto para colocar el pivote en su lugar

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	**1**	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	**4**	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	4	**1**	5	9	2	6	5	3	5	8	9	7	9	3
3	1	1	4	**5**	9	2	6	5	3	5	8	9	7	9	3
3	1	1	4	5	**9**	2	6	5	3	5	8	9	7	9	3
3	1	1	4	5	9	**2**	6	5	3	5	8	9	7	9	3
3	1	1	2	5	9	4	**6**	5	3	5	8	9	7	9	3
3	1	1	2	5	9	4	6	**5**	3	5	8	9	7	9	3
3	1	1	2	5	9	4	6	5	**3**	5	8	9	7	9	3

Quicksort

Implementación de Lomuto para colocar el pivote en su lugar

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	**1**	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	**4**	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	4	**1**	5	9	2	6	5	3	5	8	9	7	9	3
3	1	1	4	**5**	9	2	6	5	3	5	8	9	7	9	3
3	1	1	4	5	**9**	2	6	5	3	5	8	9	7	9	3
3	1	1	4	5	9	**2**	6	5	3	5	8	9	7	9	3
3	1	1	2	5	9	4	**6**	5	3	5	8	9	7	9	3
3	1	1	2	5	9	4	6	**5**	3	5	8	9	7	9	3
3	1	1	2	5	9	4	6	5	**3**	5	8	9	7	9	3
3	1	1	2	3	9	4	6	5	5	**5**	8	9	7	9	3

Quicksort

Implementación de Lomuto para colocar el pivote en su lugar

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	**1**	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	**4**	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	4	**1**	5	9	2	6	5	3	5	8	9	7	9	3
3	1	1	4	**5**	9	2	6	5	3	5	8	9	7	9	3
3	1	1	4	5	**9**	2	6	5	3	5	8	9	7	9	3
3	1	1	4	5	9	**2**	6	5	3	5	8	9	7	9	3
3	1	1	2	5	9	4	**6**	5	3	5	8	9	7	9	3
3	1	1	2	5	9	4	6	**5**	3	5	8	9	7	9	3
3	1	1	2	5	9	4	6	5	**3**	5	8	9	7	9	3
3	1	1	2	3	9	4	6	5	5	**5**	8	9	7	9	3
3	1	1	2	3	9	4	6	5	5	5	**8**	9	7	9	3

Quicksort

Implementación de Lomuto para colocar el pivote en su lugar

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	**1**	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	**4**	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	4	**1**	5	9	2	6	5	3	5	8	9	7	9	3
3	1	1	4	**5**	9	2	6	5	3	5	8	9	7	9	3
3	1	1	4	5	**9**	2	6	5	3	5	8	9	7	9	3
3	1	1	4	5	9	**2**	6	5	3	5	8	9	7	9	3
3	1	1	2	5	9	4	**6**	5	3	5	8	9	7	9	3
3	1	1	2	5	9	4	6	**5**	3	5	8	9	7	9	3
3	1	1	2	5	9	4	6	5	**3**	5	8	9	7	9	3
3	1	1	2	3	9	4	6	5	5	**5**	8	9	7	9	3
3	1	1	2	3	9	4	6	5	5	5	**8**	9	7	9	3
3	1	1	2	3	9	4	6	5	5	5	8	**9**	7	9	3

Quicksort

Implementación de Lomuto para colocar el pivote en su lugar

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	**1**	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	**4**	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	4	**1**	5	9	2	6	5	3	5	8	9	7	9	3
3	1	1	4	**5**	9	2	6	5	3	5	8	9	7	9	3
3	1	1	4	5	**9**	2	6	5	3	5	8	9	7	9	3
3	1	1	4	5	9	**2**	6	5	3	5	8	9	7	9	3
3	1	1	2	5	9	4	**6**	5	3	5	8	9	7	9	3
3	1	1	2	5	9	4	6	**5**	3	5	8	9	7	9	3
3	1	1	2	5	9	4	6	5	**3**	5	8	9	7	9	3
3	1	1	2	3	9	4	6	5	5	**5**	8	9	7	9	3
3	1	1	2	3	9	4	6	5	5	5	**8**	9	7	9	3
3	1	1	2	3	9	4	6	5	5	5	8	**9**	7	9	3
3	1	1	2	3	9	4	6	5	5	5	8	9	**7**	9	3

Quicksort

Implementación de Lomuto para colocar el pivote en su lugar

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	**1**	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	**4**	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	4	**1**	5	9	2	6	5	3	5	8	9	7	9	3
3	1	1	4	**5**	9	2	6	5	3	5	8	9	7	9	3
3	1	1	4	5	**9**	2	6	5	3	5	8	9	7	9	3
3	1	1	4	5	9	**2**	6	5	3	5	8	9	7	9	3
3	1	1	2	5	9	4	**6**	5	3	5	8	9	7	9	3
3	1	1	2	5	9	4	6	**5**	3	5	8	9	7	9	3
3	1	1	2	5	9	4	6	5	**3**	5	8	9	7	9	3
3	1	1	2	3	9	4	6	5	5	**5**	8	9	7	9	3
3	1	1	2	3	9	4	6	5	5	5	**8**	9	7	9	3
3	1	1	2	3	9	4	6	5	5	5	8	**9**	7	9	3
3	1	1	2	3	9	4	6	5	5	5	8	9	**7**	9	3
3	1	1	2	3	9	4	6	5	5	5	8	9	7	**9**	3

Quicksort

Implementación de Lomuto para colocar el pivote en su lugar

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	**1**	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	**4**	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	4	**1**	5	9	2	6	5	3	5	8	9	7	9	3
3	1	1	4	**5**	9	2	6	5	3	5	8	9	7	9	3
3	1	1	4	5	**9**	2	6	5	3	5	8	9	7	9	3
3	1	1	4	5	9	**2**	6	5	3	5	8	9	7	9	3
3	1	1	2	5	9	4	**6**	5	3	5	8	9	7	9	3
3	1	1	2	5	9	4	6	**5**	3	5	8	9	7	9	3
3	1	1	2	5	9	4	6	5	**3**	5	8	9	7	9	3
3	1	1	2	3	9	4	6	5	5	**5**	8	9	7	9	3
3	1	1	2	3	9	4	6	5	5	5	**8**	9	7	9	3
3	1	1	2	3	9	4	6	5	5	5	8	**9**	7	9	3
3	1	1	2	3	9	4	6	5	5	5	8	9	**7**	9	3
3	1	1	2	3	9	4	6	5	5	5	8	9	7	**9**	3
3	1	1	2	3	9	4	6	5	5	5	8	9	7	9	**3**

Quicksort

Implementación de Lomuto para colocar el pivote en su lugar

3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	**1**	4	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	**4**	1	5	9	2	6	5	3	5	8	9	7	9	3
3	1	4	**1**	5	9	2	6	5	3	5	8	9	7	9	3
3	1	1	4	**5**	9	2	6	5	3	5	8	9	7	9	3
3	1	1	4	5	**9**	2	6	5	3	5	8	9	7	9	3
3	1	1	4	5	9	**2**	6	5	3	5	8	9	7	9	3
3	1	1	2	5	9	4	**6**	5	3	5	8	9	7	9	3
3	1	1	2	5	9	4	6	**5**	3	5	8	9	7	9	3
3	1	1	2	5	9	4	6	5	**3**	5	8	9	7	9	3
3	1	1	2	3	9	4	6	5	5	**5**	8	9	7	9	3
3	1	1	2	3	9	4	6	5	5	5	**8**	9	7	9	3
3	1	1	2	3	9	4	6	5	5	5	8	**9**	7	9	3
3	1	1	2	3	9	4	6	5	5	5	8	9	**7**	9	3
3	1	1	2	3	9	4	6	5	5	5	8	9	7	**9**	3
3	1	1	2	3	9	4	6	5	5	5	8	9	7	9	**3**
3	1	1	2	3	3	4	6	5	5	5	8	9	7	9	9

Quicksort

Implementación recursiva

Esta función deberá llamarse inicialmente como `qsort(0, n-1, a)`.

```
void qsort(int i, int j, int a[]) {
  if (i < j) {                  // dos o mas elementos
    int p = pivote(i, j, a);    // el pivote va en a[p]
    qsort(i, p-1, a);           // ordena los menores
    qsort(p+1, j, a);           // ordena los mayores
  }
}
```

Comparaciones del pivote

Con un vector de tamaño n, `pivote` hace n comparaciones.

Quicksort

Cantidad de comparaciones

Comparaciones del pivote

Con un vector de tamaño n, `pivote` hace n comparaciones.

Comparaciones de Quicksort

Si $T(n)$ es el número de comparaciones que hace `qsort` con un vector de tamaño n, entonces $T(1) = 0$ y $T(n) = T(p-1) + T(n-p) + n$ si $n > 1$.

Quicksort

Cantidad de comparaciones

Comparaciones del pivote

Con un vector de tamaño n, `pivote` hace n comparaciones.

Comparaciones de Quicksort

Si $T(n)$ es el número de comparaciones que hace `qsort` con un vector de tamaño n, entonces $T(1) = 0$ y $T(n) = T(p-1) + T(n-p) + n$ si $n > 1$.

Estimación

Si $p \approx n/2$ entonces $T(n) \approx 2T(n/2) + n$, de donde $T(n) \approx n \log_2 n$.

Quicksort

Cantidad de comparaciones

Comparaciones del pivote

Con un vector de tamaño n, `pivote` hace n comparaciones.

Comparaciones de Quicksort

Si $T(n)$ es el número de comparaciones que hace `qsort` con un vector de tamaño n, entonces $T(1) = 0$ y $T(n) = T(p-1) + T(n-p) + n$ si $n > 1$.

Estimación

Si $p \approx n/2$ entonces $T(n) \approx 2T(n/2) + n$, de donde $T(n) \approx n \log_2 n$. El problema es que no sabemos cuanto vale p: si $p \approx 1$ o si $p \approx n$ consistentemente, entonces $T(n) \approx \frac{1}{2}n^2$.

Quicksort

Comparación con otros algoritmos de ordenamiento

Propiedad	Burbuja	Inserción	Selección	Mezcla	Quicksort
Memoria	n	n	n	$2n$	n
Peor tiempo	n^2	n^2	n^2	$n \log_2 n$	n^2
Mejor tiempo	n	n	n^2	$n \log_2 n$	$n \log_2 n$
Promedio	n^2	n^2	n^2	$n \log_2 n$	$n \log_2 n$

Tiempo de ejecución (1 ns por comparación)

n	Burbuja	Inserción	Selección	Mezcla	Quicksort
10^3	0.5 ms	0.5 ms	0.5 ms	0.01 ms	0.01 ms
10^4	50 ms	50 ms	50 ms	0.1 ms	0.1 ms
10^5	5 s	5 s	5 s	1 ms	1 ms
10^6	500 s	500 s	500 s	20 ms	20 ms

Quicksort

Ejercicios

1. Reescribe `pivote` usando `a[i]` como pivote.
2. Reescribe `pivote` usando un elemento aleatorio como pivote.
3. Reescribe `pivote` usando la mediana de `a[i]`, `a[(i+j)/2]` y `a[j]` como pivote.
4. Escribe una función `seleccion(int i, int j, int k, int a[])` que coloque en `a[k]` al elemento que quedaría en esta posición si ordenaras `a[i]..a[j]`. Pista: ¡No ordenes el arreglo! Aprovecha `pivote`.