Quicksort

Quicksort es un algoritmo de clasificación rápido, que se utiliza no solo con fines educativos, sino que se aplica ampliamente en la práctica. En promedio, tiene complejidad O (n log n), lo que hace que quicksort sea adecuado para clasificar grandes volúmenes de datos. La idea del algoritmo es bastante simple y una vez que te das cuenta, puedes escribir quicksort tan rápido como sort de burbuja .

Algoritmo

La estrategia de dividir y vencer se usa en quicksort. Debajo del paso de recursión se describe:

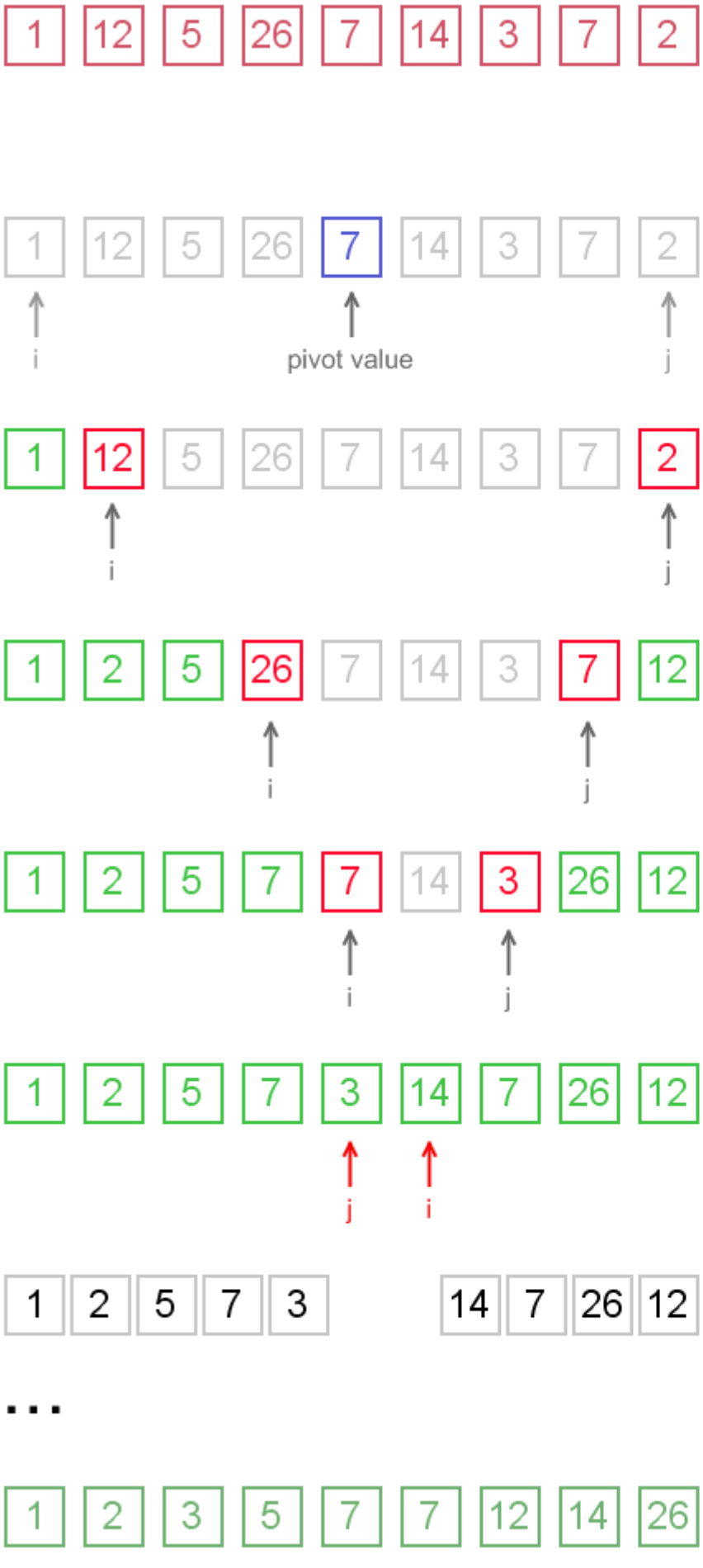
1. **Elija un valor de pivote.** Tomamos el valor del elemento medio como valor de pivote, pero puede ser cualquier valor, que está en el rango de valores ordenados, incluso si no está presente en la matriz.
2. **Dividir.** Reorganice los elementos de tal manera que todos los elementos que sean inferiores al pivote vayan a la parte izquierda de la matriz y todos los elementos superiores al pivote, vayan a la parte derecha de la matriz. Los valores iguales al pivote pueden permanecer en cualquier parte de la matriz. Tenga en cuenta que esa matriz se puede dividir en partes no iguales.
3. **Clasifica ambas partes.** Aplique el algoritmo quicksort recursivamente a las partes izquierda y derecha.

Algoritmo de partición en detalle

Hay dos índices **i** y **j** y en el comienzo de la partición de algoritmo **i** apunta al primer elemento de la matriz y **j**puntos a la última. Entonces se mueve de algoritmo **i** hacia adelante, hasta que se encuentra un elemento con valor mayor o igual que el pivote. El índice **j** se mueve hacia atrás, hasta que se encuentra un elemento con un valor menor o igual al pivote. Si **i ≤ j,** entonces se intercambian y yo paso a la siguiente posición (**i + 1**), j pasos a la anterior **(j - 1)** . El algoritmo se detiene, cuando **soy** mayor que **j** .

Después de la partición, todos los valores anteriores **al** elemento **i-ésimo** son menores o iguales que el pivote y todos los valores posteriores **al** elemento **j-ésimo** son mayores o iguales al pivote.

Ejemplo. Ordenar {1, 12, 5, 26, 7, 14, 3, 7, 2} usando quicksort.


1 12 5 26 7 14 3 7 2
unsorted
1 12 5 26 7 14 3 7 2
pivot value = 7
i
pivot value
j
1 12 5 26 7 14 3 7 2
12 ≥ 7 ≥ 2, swap 12 an
i
j
1 2 5 26 7 14 3 7 12
26 ≥ 7 ≥ 7, swap 26 an
i
j
1 2 5 7 7 14 3 26 12
7 ≥ 7 ≥ 3, swap 7 and
i
j
1 2 5 7 3 14 7 26 12
i > j, stop partition
j
i
1 2 5 7 3
14 7 26 12
run quick sort recursiv
...
1 2 3 5 7 7 12 14 26
sorted

Tenga en cuenta que mostramos aquí solo el primer paso de recursión, para no dar un ejemplo demasiado largo. Pero, de hecho, {1, 2, 5, 7, 3} y {14, 7, 26, 12} se ordenan recursivamente.

¿Por qué funciona?

En el algoritmo de paso de partición, la matriz se divide en dos partes y cada elemento **a** de la parte izquierda es menor o igual que cada elemento **b** de la parte derecha. También **a** y **b** satisfacen **un ≤ pivote ≤ b** desigualdad. Una vez completadas las llamadas de recursión, ambas partes se ordenan y, teniendo en cuenta los argumentos indicados anteriormente, se ordena todo el conjunto.

Análisis de complejidad

En el quicksort promedio tiene complejidad O (n log n), pero una fuerte prueba de este hecho no es trivial y no se presenta aquí. Aún así, puedes encontrar la prueba en [1] . En el peor de los casos, quicksort ejecuta el tiempo O (n^2), pero en la mayoría de los datos "prácticos" funciona bien y supera a otros algoritmos de clasificación O (n log n).

Fragmentos de código

El algoritmo de partición es importante per se, por lo tanto, se puede llevar a cabo como una función separada. El código para C ++ contiene una función sólida para quicksort, pero el código de Java contiene dos funciones separadas para la partición y ordena, en consecuencia.

Java

```
int partición ( int arr [], int izquierda, int derecha)
{
      int i = izquierda, j = derecha;
      int tmp;
      int pivot = arr [(izquierda + derecha) / 2];

      while (i <= j) {
            while (arr [i] <pivot)
                  i ++;
            while (arr [j]> pivote)
                  j--;
            if (i <= j) {
                  tmp = arr [i];
                  arr [i] = arr [j];
                  arr [j] = tmp;
                  i ++;
                  j--;
```

```
            }
      };

      devuelve i;
}

void quickSort ( int arr [], int izquierda, int derecha) {
      int index = partition (arr, left, right);
      if (izquierda <índice - 1)
            quickSort (arr, izquierda, índice - 1);
      if (índice <derecha)
            quickSort (arr, index, right);
}
```

C ++

```
void quickSort ( int arr [], int izquierda , int derecha ) {
      int i = izquierda , j = derecha ;
      int tmp ;
      int pivot = arr [( izquierda + derecha ) / 2];

      / * partición * /
      while ( i <= j ) {
            while ( arr [ i ] < pivot )
                  i ++;
            while ( arr [ j ]> pivote )
                  j -;
            if ( i <= j ) {
                  tmp = arr [ i ];
                  arr [ i ] = arr [ j ];
                  arr [ j ] = tmp ;
                  i ++;
                  j -;
            }
      };

      / * recursividad * /
      si ( izquierda < j )
            quickSort ( arr , izquierda , j );
      si ( yo < derecha )
            quickSort ( arr , i , right );
}
```