Быстрая сортировка

Слёлин А.В.

18 мая, 2024 г.

Описание алгортима

Пусть дан массив $A[0..n-1]$, такой что $n = \text{length}[A]$, причём индексация начинается с нуля для облегчения понимания кода.

Рассмотрим случай когда нам нужно отсортировать массив A в неубывающую последовательность; случай, когда нужно отсортировать в невозрастающую последовательность аналогичен.

Быстрая сортировка, подобно сортировке слияниями основана на парадигме «разделяй и властвуй».

Разделение

Массив $A[p..r]$ разбивается на два подмассива $A[p..q-1]$ и $A[q+1..r]$. Каждый элемент подмассива $A[p..q-1]$ не превышает элемент $A[q]$, а каждый элемент подмассива $A[q+1..r]$ не меньше элемента $A[q]$. Индекс q вычисляется в ходе вычисления процедуры разбиения.

Покорение

Подмассивы $A[p..q-1]$ и $A[q+1..r]$ сортируются путём рекурсивного вызова процедуры быстрой сортировки.

Комбинирование

Поскольку подмассивы сортируются путём рекурсивного вызова на месте, для их объединения не нужны никакие действия: весь массив $A[p..r]$ оказывается отсортирован.

Пример

Пусть массив $A = [8, 3, 12, 16, 10, 4, 1, 9, 7, 14]$. count - означает номер рекурентного вызова функции быстрой сортировки QUICKSORT, input_array - входной массив в QUICKSORT(выделен красным цветом), а output_array - выходной, q - опорный элемент (выделен жёлтым цветом), L - подмассив $A[p..q-1]$ (выделен синим цветом), R - подмассив $A[q+1..r]$ (выделен оранжевым цветом). В исходном массиве A показываются изменения зелёным цветом.

Общий картина вызовов:

1) $in = [8, 3, 12, 16, 10, 4, 1, 9, 7, 14]$

$out = [8, 3, 12, 10, 4, 1, 9, 7, \mathbf{14}, 16]$

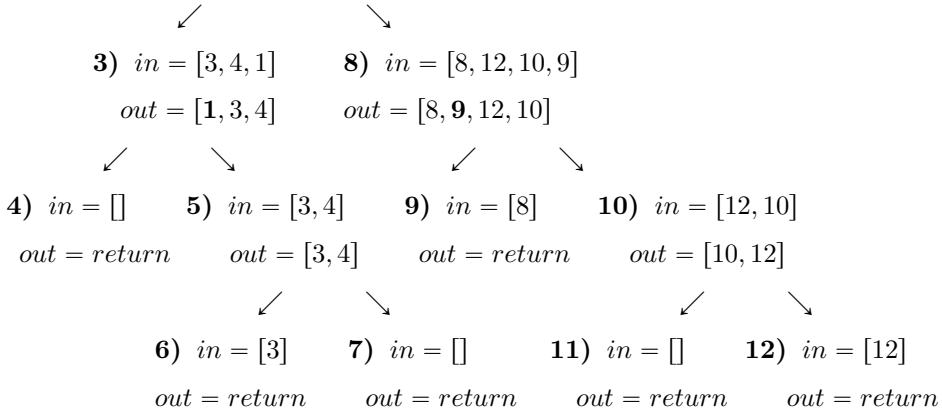
↙ ↘

2) $in = [8, 3, 12, 10, 4, 1, 9, 7]$

$out = [3, 4, 1, \mathbf{7}, 8, 12, 10, 9]$

13) $in = [16]$

$out = return$



count = 1:

$$\text{input\_array} = [8, 3, 12, 16, 10, 4, 1, 9, 7, 14];$$

$$\text{output\_array} = [8, 3, 12, 10, 4, 1, 9, 7, 14, 16];$$

$$A = [8, 3, 12, 10, 4, 1, 9, 7, 14, 16];$$

count = 2:

$$\text{input\_array} = [8, 3, 12, 10, 4, 1, 9, 7];$$

$$\text{output\_array} = [3, 4, 1, 7, 8, 12, 10, 9];$$

$$A = [3, 4, 1, 7, 8, 12, 10, 9, 14, 16];$$

count = 3:

$$\text{input\_array} = [3, 4, 1];$$

$$\text{output\_array} = [1, 3, 4];$$

$$A = [1, 3, 4, 7, 8, 12, 10, 9, 14, 16];$$

count = 4:

$$\text{input\_array} = [];$$

return;

count = 5:

$$\text{input\_array} = [3, 4];$$

$$\text{output\_array} = [3, 4];$$

$$A = [1, 3, 4, 7, 8, 12, 9, 14, 16];$$

count = 6:

$$\text{input\_array} = [3];$$

return;

count = 7:

$$\text{input\_array} = [];$$

return;

count = 8:

$$\text{input\_array} = [8, 12, 10, 9];$$

$$\text{output\_array} = [8,9,12, 10];$$

$$A = [1, 3, 4, 7, 8, 9, 12, 10,14, 16];$$

count = 9:

$$\text{input\_array} = [8];$$

return;

count = 10:

$$\text{input\_array} = [12, 10];$$

$$\text{output\_array} = [10,12];$$

$$A = [1, 3, 4, 7, 8, 9, 10, 12,14, 16];$$

count = 11:

$$\text{input\_array} = [];$$

return;

count = 12:

$$\text{input\_array} = [12];$$

return;

count = 13:

$$\text{input\_array} = [16];$$

return.

Массив A полностью отсортирован. Для получения невозрастающей последовательности следует просто поменять знак сравнения.

Код сортировки

Ключевой частью алгоритма быстрой сортировки является процедура PARTI TION, изменяющая порядок элементов подмассива $A[p..r]$ без привлечения дополнительной памяти.

Листинг 1: PARTITION

```
1       public static int PARTITION(int[] A, int p, int r, boolean
            reverse) {
2           int x = A[r];
3           int i = p - 1;
4           for (int j = p; j < r; ++j)
5               if (reverse ? A[j] >= x : A[j] <= x) {
6                   int tmp = A[++i];
7                   A[i] = A[j];
8                   A[j] = tmp;
9               }
10
11          int tmp = A[++i];
12          A[i] = A[r];
13          A[r] = tmp;
14
15          return i;
16      }
```

То есть в этой процедуре мы берём последний элемент и с помощью него делим алгоритм на две части: с одной стороны элементы, которые не больше, с другой - не меньше.

Для реализации этого алгоритма необязательно брать последний элемент как опорный, возможен также вариант добавить следующий рандомизированный код.

Листинг 2: RANDOMIZED_PARTITION

```
1       public static int RANDOMIZED_PARTITION(int[] A, int p, int r
            , boolean reverse) {
2           int i = ThreadLocalRandom.current().nextInt(p, r + 1);
3
4           int tmp = A[r];
5           A[r] = A[i];
6           A[i] = tmp;
7
8           return PARTITION(A, p, r, reverse);
9       }
```

И сам алгоритм быстрой сортировки.

Листинг 3: QUICKSORT

```
1       public static void QUICKSORT(int[] A, int p, int r, boolean
            reverse, boolean randomized) {
2           if (p < r) {
3               int q;
```

```
            if (randomized) q = RANDOMIZED_PARTITION(A, p, r,
                reverse);
            else q = PARTITION(A, p, r, reverse);
            QUICKSORT(A, p, q - 1, reverse, randomized);
            QUICKSORT(A, q + 1, r, reverse, randomized);
        }
    }
```

Чтобы выполнить соритровку всего массива A, вызов процедуры должен иметь вид QUICKSORT$(A, 0, A.length - 1, reverse, randomized)$.

Исходный файл

Представим весь код Main.java для тестирования алгоритма.

Листинг 4: Main

```
import java.util.concurrent.ThreadLocalRandom;

public class Main {

    public static int PARTITION(int[] A, int p, int r, boolean
        reverse) {
        int x = A[r];
        int i = p - 1;
        for (int j = p; j < r; ++j)
            if (reverse ? A[j] >= x : A[j] <= x) {
                int tmp = A[++i];
                A[i] = A[j];
                A[j] = tmp;
            }

        int tmp = A[++i];
        A[i] = A[r];
        A[r] = tmp;

        return i;
    }

    public static int RANDOMIZED_PARTITION(int[] A, int p, int r
        , boolean reverse) {
        int i = ThreadLocalRandom.current().nextInt(p, r + 1);

        int tmp = A[r];
        A[r] = A[i];
        A[i] = tmp;

```

```
29          return PARTITION(A, p, r, reverse);
30      }
31
32      public static void QUICKSORT(int[] A, int p, int r, boolean
            reverse, boolean randomized) {
33          if (p < r) {
34              int q;
35              if (randomized) q = RANDOMIZED_PARTITION(A, p, r,
                    reverse);
36              else q = PARTITION(A, p, r, reverse);
37              QUICKSORT(A, p, q - 1, reverse, randomized);
38              QUICKSORT(A, q + 1, r, reverse, randomized);
39          }
40      }
41
42      public static void SORT(int[] A, boolean reverse, boolean
            randomized) {
43          QUICKSORT(A, 0, A.length - 1, reverse, randomized);
44      }
45
46      public static void PRINT(int[] A) {
47          for (int i = 0; i < A.length; ++i)
48              System.out.print(A[i] + " ");
49
50          System.out.println();
51      }
52
53      public static void main(String[] args) {
54          int[] A = {8, 3, 12, 16, 10, 4, 1, 9, 7, 14};
55
56          PRINT(A);
57          SORT(A, false, false);
58          PRINT(A);
59      }
60  }
```

Сложность

Пространственная сложность алгоритма быстрой сортировки равна $O(\lg n)$ дополнительной памяти в виде стека.

Быстрая сортировка - это алгоритм сортировки, время работы которого для входного алгоритма из n чисел в наихудшем случае равно $O(n^2)$. Несмотря на такую медленную работу в наихудшем случае, этот алгоритм зачастую оказывается оптимальным благодаря тому, что в среднем время его работы намного лучше: $\Theta(n \lg n)$. Кроме того, постоянные множители не учтённые в выраже-

нии $\Theta(n \lg n)$, достаточно малы по величине. В лучшем случае всё аналогично среднему случаю.

Временная сложность			Пространственная сложность
Худший	Средний	Лучший	Худший
$O(n^2)$	$\Theta(n \lg n)$	$\Omega(n \lg n)$	$O(\lg n)$

Таблица 1: **Резюме**

Комментарии

Быстрая сортировка хороший алгоритм сортировки, который можно писать «из головы», если хорошо в нём разобраться. Он работает достаточно быстро и его можно использовать в проектах.