Сортировка слиянием

Слёлин А.В.

17 мая, 2024 г.

Описание алгоритма

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.

Алгоритм сотрировки слиянием основан на парадигме «разделяй и властвуй». Его работу можно описать следующим образом:

Разделение
Сортируемый массив, состоящий из n элементов разбивается на два подмассива, каждый из которых содержит $\frac{n}{2}$ элементов.

Покорение
Далее каждый массив мы сортируем методом слияния.

Комбинирование
Слияние двух отсортированных массива для получения окончательного результата.

Данный алгоритм хорошо представляется в виде рекурсии. Рекурсия достигает своего нижнего предела, когда длина сортируемой последовательности становится равной 1. В этом случае вся работа уже сделана, поскольку любую такую последовательность можно считать упорядоченной.

Основная операция, которая производится в процессе сортировки по методу слияния. Это делается с помощью вспомогательной процедуры $\text{MERGE}(A, p, q, r)$, где A - массив, а p, q и r - индексы, нумерующие элементы подмассива, такие, что $p \leqslant q < r$. В этой процедуре предполагается, что элементы подмассивов $A[p..q]$ и $A[q+1..r]$ упорядочены. Она сливает эти два подмассива в один отсортированный, элементы которого заменяют текущие элементы подмассива $A[p..r]$.

В процедуре $\text{MERGE}(A, p, q, r)$ мы воспользуемся дополнительной памятью, так что новый подмассив $L = A[p..q]$, а $R = A[q+1..r]$, причём добавим в конец каждого массива inf, означающее очень большое число, значение которого станет ясно чуть позже, то есть $L[q+1] = R[r+1] = \text{inf}$. Пройдёмся в цикле по k по массиву A от p до q и каждый раз будем обновлять значение $A[k]$.

Пусть указатель i смотрит на индекс самого маленького элемента L (в начале это $L[p]$), а j - на индекс самого маленького элемента R (в начале это $R[q+1]$). В $A[k]$ подставляем наименьший из $A[i]$ и $A[j]$ и при этом сдвигаем указатель наименьшего элемента на единицу. Таким образом, если $L[p] < R[q+1]$, тогда $A[p] = L[p]$ и теперь указатель i показывает на $L[p+1]$, так как наименьший $L[p]$ уже использован, а сам массив L отсортирован.

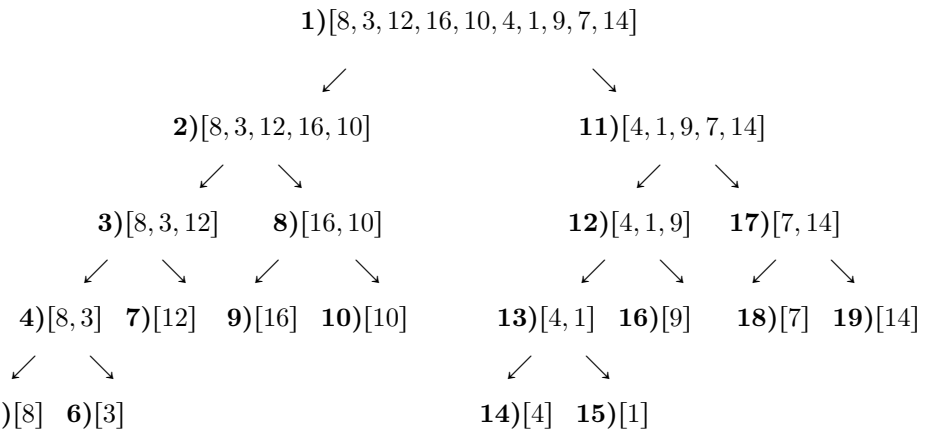
Теперь поймём почему $L[q+1] = R[r+1] = \text{inf}$. Рассмотрим такую ситуацию: все элементы L меньше $R[q+1]$, то есть нам надо просто «склеить» два массива.

В этом случае указатель i будет увеличиваться, пока не дойдёт до $L[q]$, а потом снова увеличиться на единицу. И тут мы выйдем за пределы массива; можно и это обработать, но j не будет дальше двигаться, так как $L[q] < R[q+1]$. То есть конечный элемент inf должен говорить, что один из указателей кончился, теперь идём по другому указателю, так как условие $R[j] < \inf$ всегда верно.

Пример

Пусть массив $A = [8, 3, 12, 16, 10, 4, 1, 9, 7, 14]$. count - означает номер рекурентного вызова функции сортировки слиянием MERGESORT, input_array - входной массив в MERGESORT (выделен красным цветом), L - подмассив $A[p..q]$ (выделен синим цветом), R - подмассив $A[q+1..r]$ (выделен оранжевым цветом). Если около count стоит слово MERGE, значит мы исользуем функцию MERGE для соединения двух входных подмассивов L (выделен синим цветом) и R (выделен оранжевым цветом); после этого в исходном массиве A показываются изменения в связи с выполнением функции MERGE (выделено зелёным цветом).

Общий картина вызовов:



count = 1:

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

$$L = [8, 3, 12, 16, 10]; \quad R = [4, 1, 9, 7, 14];$$

count = 2:

$$\text{input\_array} = [8, 3, 12, 16, 10];$$

$$L = [8, 3, 12]; \quad R = [16, 10];$$

count = 3:

$$\text{input\_array} = [8, 3, 12];$$

$$L = [8, 3]; \quad R = [12];$$

count = 4:

$$\text{input\_array} = [8, 3];$$

$$L = [8]; \quad R = [3];$$

count = 5:

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

return;

count = 6:

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

return;

count = 4 (MERGE):

$$L = [8]; \quad R = [3];$$

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

count = 7:

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

return;

count = 3 (MERGE):

$$L = [3, 8]; \quad R = [12];$$

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

count = 8:

$$\text{input\_array} = [16, 10];$$

$$L = [16]; \quad R = [10];$$

count = 9:

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

return;

count = 10:

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

return;

count = 8 (MERGE):

$$L = [16]; \quad R = [10];$$

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

count = 2 (MERGE):

$$L = [3, 8, 12]; \quad R = [10, 16];$$

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

count = 11:

$$\text{input\_array} = [4, 1, 9, 7, 14];$$

$$L = [4, 1, 9]; \quad R = [7, 14];$$

count = 12:

$$\text{input\_array} = [4, 1, 9];$$

$$L = [4, 1]; \quad R = [9];$$

count = 13:

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

$$L = [4]; \quad R = [1];$$

count = 14:

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

return;

count = 15:

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

return;

count = 13 (MERGE):

$$L = [4]; \quad R = [1];$$

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

count = 16:

$$\text{input\_array} = [9];$$

return;

count = 12 (MERGE):

$$L = [1, 4]; \quad R = [9];$$

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

count = 17:

$$\text{input\_array} = [7, 14];$$

$$L = [7]; \quad R = [14];$$

count = 18:

$$\text{input\_array} = [7];$$

return;

count = 19:

$$\text{input\_array} = [14];$$

return;

count = 17 (MERGE):

$$L = [7]; \quad R = [14];$$

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

count = 11 (MERGE):

$$L = [1, 4, 9]; \quad R = [7, 14];$$

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

count = 1 (MERGE):

$$L = [3, 8, 12, 10, 16]; \quad R = [1, 4, 7, 9, 14];$$

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

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

Код сортировки

Листинг 1: mergeSort

```
1       public static void mergeSort(int[] A, int p, int r, boolean
            reverse) {
2           if (p < r) {
3               int q = (p + r) / 2;
4               mergeSort(A, p, q, reverse);
5               mergeSort(A, q + 1, r, reverse);
6               merge(A, p, q, r, reverse);
7           }
8       }
```

Запускаем реккурсию, которая сортирует левую часть подмассива, затем правую, а после вызывает метод слияния этих последовательностей.

Листинг 2: merge

```
1       public static void merge(int[] A, int p, int q, int r,
            boolean reverse) {
2           int len1 = q - p + 1, len2 = r - q;
3           int[] L = new int[len1 + 1], R = new int[len2 + 1];
4           for (int i = 0; i < len1; ++i)
5               L[i] = A[i + p];
6           for (int i = 0; i < len2; ++i)
7               R[i] = A[i + q + 1];
8           L[len1] = R[len2] = reverse ? Integer.MIN_VALUE  :
                Integer.MAX_VALUE;
9
10          int i = 0, j = 0;
11          for (int k = p; k <= r; ++k)
12              if (reverse ? L[i] > R[j] : L[i] < R[j]) A[k] = L[i
                    ++];
13              else A[k] = R[j++];
14      }
```

В методе слияния мы выделяем память под два подмассива L и R и копируем их из массива A. Затем соединяем их с помощью указателей i и j.

Исходный код

Представим весь код Main.java для тестирования алгоритма.

Листинг 3: Main

```
1   public class Main {
2       public static void merge(int[] A, int p, int q, int r,
            boolean reverse) {
3           int len1 = q - p + 1, len2 = r - q;
4           int[] L = new int[len1 + 1], R = new int[len2 + 1];
5           for (int i = 0; i < len1; ++i)
6               L[i] = A[i + p];
7           for (int i = 0; i < len2; ++i)
8               R[i] = A[i + q + 1];
9           L[len1] = R[len2] = reverse ? Integer.MIN_VALUE  :
                Integer.MAX_VALUE;
10
11          int i = 0, j = 0;
12          for (int k = p; k <= r; ++k)
13              if (reverse ? L[i] > R[j] : L[i] < R[j]) A[k] = L[i
                    ++];
```

```
14                  else A[k] = R[j++];
15          }
16
17          public static void mergeSort(int[] A, int p, int r, boolean
                reverse) {
18              if (p < r) {
19                  int q = (p + r) / 2;
20                  mergeSort(A, p, q, reverse);
21                  mergeSort(A, q + 1, r, reverse);
22                  merge(A, p, q, r, reverse);
23              }
24          }
25          public static void sort(int[] A, boolean reverse) {
26              mergeSort(A, 0, A.length - 1, reverse);
27          }
28
29          public static void print(int[] A) {
30              for (int i = 0; i < A.length; ++i)
31                  System.out.print(A[i] + " ");
32
33              System.out.println();
34          }
35
36          public static void main(String[] args) {
37              int[] A = {8, 3, 12, 16, 10, 4, 1, 9, 7, 14};
38
39              print(A);
40              sort(A, false);
41              print(A);
42          }
43      }
```

Сложность

Во-первых сразу заметим, что в методе слияния мы выделяем дополнительную память под два подмассива. Следовательно память зависит от длины n входного массива A. Пространственная сложность алгоритма сортировки слиянием равна $O(n)$.

Независимо от того какой массив нам подаётся на вход (отсортированный - лучший случай, неотсортированный - средний случай, отсортированный в обратную сторону - худший случай) MERGESORT разбивает массив на части, пока не получит массив длины 1. Значит во всех случаях MERGESORT работает за $O(\lg n)$. Каждый раз мы линейно проходим по части массива в любом случае (сортирован он или нет), а это значит, что во всех случаях MERGE работает за $O(n)$. Тогда если объединить полученные результаты, становится ясно, что

временная сложность сортировки слиянием в лучше, среднем и худшем случаях равна $\Omega(n \lg n),\ \Theta(n \lg n),\ O(n \lg n)$ соответственно.

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

Таблица 1: **Резюме**

Комментарии

Сортировку слиянием понять уже сложнее, чем пузырьковую сортировку или сортировку вставками, но она также является достаточно простой. Надо заметить, время работы данного алгоритма оптимально, но очень нежелательно выделять на реализацию алгоритма память, поэтому сортировку слиянием опять же лучше использовать только в учебных целях.