В данном разделе рассматривается альтернативный подход к разработке алгоритмов, известный как метод разбиения ("разделяй и властвуй"). Разработаем с помощью этого подхода алгоритм сортировки, время работы которого в наихудшем случае намного меньше времени работы алгоритма, работающего по методу включений. Одно из преимуществ алгоритмов, разработанных методом разбиения, заключается в том, что время их работы часто легко определяется с помощью технологий, описанных в главе 4.

## 2.3.1 Метод декомпозиции

Многие полезные алгоритмы имеют ***рекурсивную*** структуру: для решения данной задачи они рекурсивно вызывают сами себя один или несколько раз, чтобы решить вспомогательную задачу, имеющую непосредственное отношение к поставленной задаче. Такие алгоритмы зачастую разрабатываются с помощью метода ***декомпозиции***, или ***разбиения***: сложная задача разбивается на несколько более простых, которые подобны исходной задаче, но имеют меньший объем; далее эти вспомогательные задачи решаются рекурсивным методом, после чего полученные решения комбинируются с целью получить решение исходной задачи.

Парадигма, лежащая в основе метода декомпозиции "разделяй и властвуй", на каждом уровне рекурсии включает в себя три этапа.

**Разделение** задачи на несколько подзадач.

**Покорение** — рекурсивное решение этих подзадач. Когда объем подзадачи достаточно мал, выделенные подзадачи решаются непосредственно.

**Комбинирование** решения исходной задачи из решений вспомогательных задач.

Алгоритм ***сортировки слиянием*** (merge sort) в большой степени соответствует парадигме метода разбиения. На интуитивном уровне его работу можно описать таким образом.

**Разделение:** сортируемая последовательность, состоящая из $n$ элементов, разбивается на две меньшие последовательности, каждая из которых содержит $n/2$ элементов.

**Покорение:** сортировка обеих вспомогательных последовательностей методом слияния.

**Комбинирование:** слияние двух отсортированных последовательностей для получения окончательного результата.

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 слияний, — это объединение двух отсортированных последовательностей в ходе комбинирования (последний этап). Это делается с помощью вспомогательной процедуры MERGE$(A, p, q, r)$, где $A$ — массив, а $p$, $q$ и $r$ — индексы, нумерующие элементы массива, 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 $A[p..r]$.

Для выполнения процедуры MERGE требуется время $\Theta(n)$, где $n = r - p + 1$ — количество подлежащих слиянию элементов. Процедура работает следующим образом. Возвращаясь к наглядному примеру сортировки карт, предположим, что на столе лежат две стопки карт, обращенных лицевой стороной вниз. Карты в каждой стопке отсортированы, причем наверху находится карта наименьшего достоинства. Эти две стопки нужно объединить в одну выходную, в которой карты будут рассортированы и также будут обращены рубашкой вверх. Основной шаг состоит в том, чтобы из двух младших карт выбрать самую младшую, извлечь ее из соответствующей стопки (при этом в данной стопке верхней окажется новая карта) и поместить в выходную стопку. Этот шаг повторяется до тех пор, пока в одной из входных стопок не кончатся карты, после чего оставшиеся в другой стопке карты нужно поместить в выходную стопку. С вычислительной точки зрения выполнение каждого основного шага занимает одинаковые промежутки времени, так как все сводится к сравнению достоинства двух верхних карт. Поскольку необходимо выполнить, по крайней мере, $n$ основных шагов, время работы процедуры слияния равно $\Theta(n)$.

Описанная идея реализована в представленном ниже псевдокоде, однако в нем также есть дополнительное ухищрение, благодаря которому в ходе каждого основного шага не приходится проверять, является ли каждая из двух стопок пустой. Идея заключается в том, чтобы поместить в самый низ обеих объединяемых колод так называемую ***сигнальную*** карту особого достоинства, что позволяет упростить код. Для обозначения сигнальной карты используется символ $\infty$. Не существует карт, достоинство которых больше достоинства сигнальной карты. Процесс продолжается до тех пор, пока проверяемые карты в обеих стопках не окажутся сигнальными. Как только это произойдет, это будет означать, что все несигнальные карты уже помещены в выходную стопку. Поскольку заранее известно, что в выходной стопке должно содержаться ровно $r - p + 1$ карта, выполнив такое количество основных шагов, можно остановиться:

MERGE$(A, p, q, r)$
$n_1 \leftarrow q - p + 1$
$n_2 \leftarrow r - q$
Создаем массивы $L[1..n_1 + 1]$ и $R[1..n_2 + 1]$

```
4  for i ← 1 to n1
5      do L[i] ← A[p + i − 1]
6  for j ← 1 to n2
7      do R[j] ← A[q + j]
8  L[n1 + 1] ← ∞
9  R[n2 + 1] ← ∞
10 i ← 1
11 j ← 1
12 for k ← p to r
13     do if L[i] ⩽ R[j]
14           then A[k] ← L[i]
15                i ← i + 1
16           else A[k] ← R[j]
17                j ← j + 1
```

Подробно опишем работу процедуры MERGE. В строке 1 вычисляется длина $n_1$ подмассива $A[p..q]$, а в строке 2 — длина $n_2$ подмассива $A[q+1..r]$. Далее в строке 3 создаются массивы $L$ ("левый" — "left") и $R$ ("правый" — "right"), длины которых равны $n_1 + 1$ и $n_2 + 1$ соответственно. В цикле **for** в строках 4 и 5 подмассив $A[p..q]$ копируется в массив $L[1..n_1]$, а в цикле **for** в строках 6 и 7 подмассив $A[q+1..r]$ копируется в массив $R[1..n_2]$. В строках 8 и 9 последним элементам массивов $L$ и $R$ присваиваются сигнальные значения.

Как показано на рис. 2.3, в результате копирования и добавления сигнальных карт получаем массив $L$ с последовательностью чисел $\langle 2, 4, 5, 7, \infty \rangle$ и массив $R$ с последовательностью чисел $\langle 1, 2, 3, 6, \infty \rangle$. Светло-серые ячейки массива $A$ содержат конечные значения, а светло-серые ячейки массивов $L$ и $R$ — значения, которые еще только должны быть скопированы обратно в массив $A$. В светло-серых ячейках находятся исходные значения из подмассива $A[9..16]$ вместе с двумя сигнальными картами. В темно-серых ячейках массива $A$ содержатся значения, которые будут заменены другими, а в темно-серых ячейках массивов $L$ и $R$ — значения, уже скопированные обратно в массив $A$. В частях рисунка *а–з* показано состояние массивов $A$, $L$ и $R$, а также соответствующие индексы $k$, $i$ и $j$ перед каждой итерацией цикла в строках 12–17. В части *и* показано состояние массивов и индексов по завершении работы алгоритма. На данном этапе подмассив $A[9..16]$ отсортирован, а два сигнальных значения в массивах $L$ и $R$ — единственные элементы, оставшиеся в этих массивах и не скопированные в массив $A$. В строках 10–17, проиллюстрированных на рис. 2.3, выполняется $r - p + 1$ основных шагов, в ходе каждого из которых производятся манипуляции с инвариантом цикла, описанным ниже.

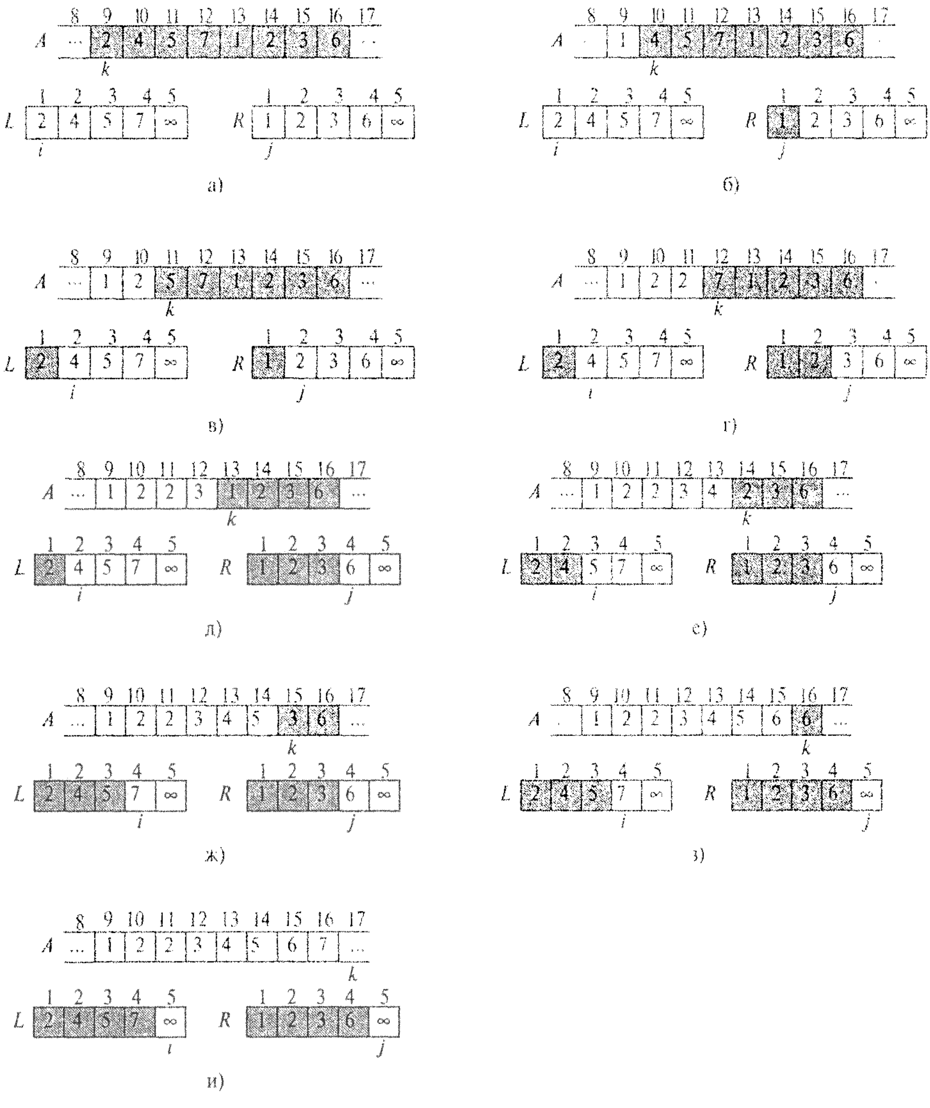

**Рис. 2.3.** Операции, выполняемые в строках 10–17 процедуры MERGE$(A, 9, 12, 16)$, когда в подмассиве $A[9..16]$ содержится последовательность $\langle 2, 4, 5, 7, 1, 2, 3, 6 \rangle$

Инвариант

Перед каждой итерацией цикла **for** в строках 12–17, подмассив $A[p..k-1]$ содержит $k-p$ наименьших элементов массивов $L[1..n_1+1]$ и $R[1..n_2+1]$ в отсортированном порядке. Кроме того, элементы $L[i]$ и $R[i]$ являются наименьшими элементами массивов $L$ и $R$, которые еще не скопированы в массив $A$.

Необходимо показать, что этот инвариант цикла соблюдается перед первой итерацией рассматриваемого цикла **for**, что каждая итерация цикла не нарушает его, и что с его помощью удается продемонстрировать корректность алгоритма, когда цикл заканчивает свою работу.

**Инициализация.** Перед первой итерацией цикла $k = p$, поэтому подмассив $A[p..k-1]$ пуст. Он содержит $k - p = 0$ наименьших элементов массивов $L$ и $R$. Поскольку $i = j = 1$, элементы $L[i]$ и $R[j]$ — наименьшие элементы массивов $L$ и $R$, не скопированные обратно в массив $A$.

**Сохранение.** Чтобы убедиться, что инвариант цикла сохраняется после каждой итерации, сначала предположим, что $L[i] \leqslant R[j]$. Тогда $L[i]$ — наименьший элемент, не скопированный в массив $A$. Поскольку в подмассиве $A[p..k-1]$ содержится $k-p$ наименьших элементов, после выполнения строки 14, в которой значение элемента $L[i]$ присваивается элементу $A[k]$, в подмассиве $A[p..k]$ будет содержаться $k - p + 1$ наименьший элемент. В результате увеличения параметра $k$ цикла **for** и значения переменной $i$ (строка 15), инвариант цикла восстанавливается перед следующей итерацией. Если же выполняется неравенство $L[i] > R[j]$, то в строках 16 и 17 выполняются соответствующие действия, в ходе которых также сохраняется инвариант цикла.

**Завершение.** Алгоритм завершается, когда $k = r + 1$. В соответствии с инвариантом цикла, подмассив $A[p..k-1]$ (т.е. подмассив $A[p..r]$) содержит $k - p = r - p + 1$ наименьших элементов массивов $L[1..n_1+1]$ и $R[1..n_2+1]$ в отсортированном порядке. Суммарное количество элементов в массивах $L$ и $R$ равно $n_1 + n_2 + 2 = r - p + 3$. Все они, кроме двух самых больших, скопированы обратно в массив $A$, а два оставшихся элемента являются сигнальными.

Чтобы показать, что время работы процедуры MERGE равно $\Theta(n)$, где $n = r - p + 1$, заметим, что каждая из строк 1–3 и 8–11 выполняется в течение фиксированного времени; длительность циклов **for** в строках 4–7 равна $\Theta(n_1 + n_2) = \Theta(n)$,[7] а в цикле **for** в строках 12–17 выполняется $n$ итераций, на каждую из которых затрачивается фиксированное время.

Теперь процедуру MERGE можно использовать в качестве подпрограммы в алгоритме сортировки слиянием. Процедура MERGE_SORT$(A, p, r)$ выполняет сортировку элементов в подмассиве $A[p..r]$. Если справедливо неравенство $p \geqslant r$, то в этом подмассиве содержится не более одного элемента, и, таким образом, он отсортирован. В противном случае производится разбиение, в ходе которого

[7] В главе 3 будет показано, как формально интерпретируются уравнения с $\Theta$-обозначениями.

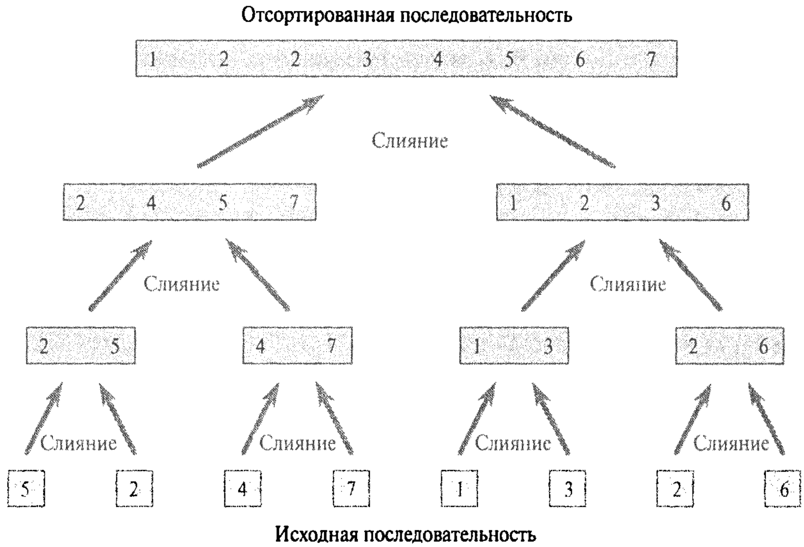


**Рис. 2.4.** Процесс сортировки массива $A = \langle 5, 2, 4, 7, 1, 3, 2, 6 \rangle$ методом слияния. Длины подлежащих объединению отсортированных подпоследовательностей возрастают по мере работы алгоритма

вычисляется индекс $q$, разделяющий массив $A[p..r]$ на два подмассива: $A[p..q]$ с $\lceil n/2 \rceil$ элементами и $A[q..r]$ с $\lfloor n/2 \rfloor$ элементами[8].

```
MERGE_SORT(A, p, r)
if p < r
  then q ← ⌊(p + r)/2⌋
       MERGE_SORT(A, p, q)
       MERGE_SORT(A, q + 1, r)
       MERGE(A, p, q, r)
```

Чтобы отсортировать последовательность $A = \langle A[1], A[2], \ldots, A[n] \rangle$, вызывается процедура MERGE_SORT($A, 1, \mathit{length}[A]$), где $\mathit{length}[A] = n$. На рис. 2.4 проиллюстрирована работа этой процедуры в восходящем направлении, если $n$ — это степень двойки. В ходе работы алгоритма происходит попарное объединение одноэлементных последовательностей в отсортированные последовательности длины 2, затем — попарное объединение двухэлементных последовательностей в отсортированные последовательности длины 4 и т.д., пока не будут

[8]Выражение $\lceil x \rceil$ обозначает наименьшее целое число, которое больше или равно $x$, а выражение $\lfloor x \rfloor$ — наибольшее целое число, которое меньше или равно $x$. Эти обозначения вводятся в главе 3. Чтобы убедиться в том, что в результате присваивания переменной $q$ значения $\lfloor (p + r)/2 \rfloor$ длины подмассивов $A[p..q]$ и $A[q + 1..r]$ получаются равными $\lceil n/2 \rceil$ и $\lfloor n/2 \rfloor$, достаточно проверить четыре возможных случая, в которых каждое из чисел $p$ и $r$ либо четное, либо нечетное.

получены две последовательности, состоящие из $n/2$ элементов, которые объединяются в конечную отсортированную последовательность длины $n$.

## 2.3.2 Анализ алгоритмов, основанных на принципе "разделяй и властвуй"

Если алгоритм рекурсивно обращается к самому себе, время его работы часто описывается с помощью ***рекуррентного уравнения***, или ***рекуррентного соотношения***, в котором полное время, требуемое для решения всей задачи с объемом ввода $n$, выражается через время решения вспомогательных подзадач. Затем данное рекуррентное уравнение решается с помощью определенных математических методов, и устанавливаются границы производительности алгоритма.

Получение рекуррентного соотношения для времени работы алгоритма, основанного на принципе "разделяй и властвуй", базируется на трех этапах, соответствующих парадигме этого принципа. Обозначим через $T(n)$ время решения задачи, размер которой равен $n$. Если размер задачи достаточно мал, скажем, $n \leqslant c$, где $c$ — некоторая заранее известная константа, то задача решается непосредственно в течение определенного фиксированного времени, которое мы обозначим через $\Theta(1)$. Предположим, что наша задача делится на $a$ подзадач, объем каждой из которых равен $1/b$ от объема исходной задачи. (В алгоритме сортировки методом слияния числа $a$ и $b$ были равны 2, однако нам предстоит ознакомиться со многими алгоритмами разбиения, в которых $a \neq b$.) Если разбиение задачи на вспомогательные подзадачи происходит в течение времени $D(n)$, а объединение решений подзадач в решение исходной задачи — в течение времени $C(n)$, то мы получим такое рекуррентное соотношение:

$$T(n) = \begin{cases} \Theta(1) & \text{при } n \leqslant c, \\ aT(n/b) + D(n) + C(n) & \text{в противном случае.} \end{cases}$$

В главе 4 будет показано, как решаются рекуррентные соотношения такого вида.

### Анализ алгоритма сортировки слиянием

Псевдокод MERGE_SORT корректно работает для произвольного (в том числе и нечетного) количества сортируемых элементов. Однако если количество элементов в исходной задаче равно степени двойки, то анализ рекуррентного уравнения упрощается. В этом случае на каждом шаге деления будут получены две подпоследовательности, размер которых точно равен $n/2$. В главе 4 будет показано, что это предположение не влияет на порядок роста, полученный в результате решения рекуррентного уравнения.

Чтобы получить рекуррентное уравнение для верхней оценки времени работы $T(n)$ алгоритма, выполняющего сортировку $n$ чисел методом слияния, будем