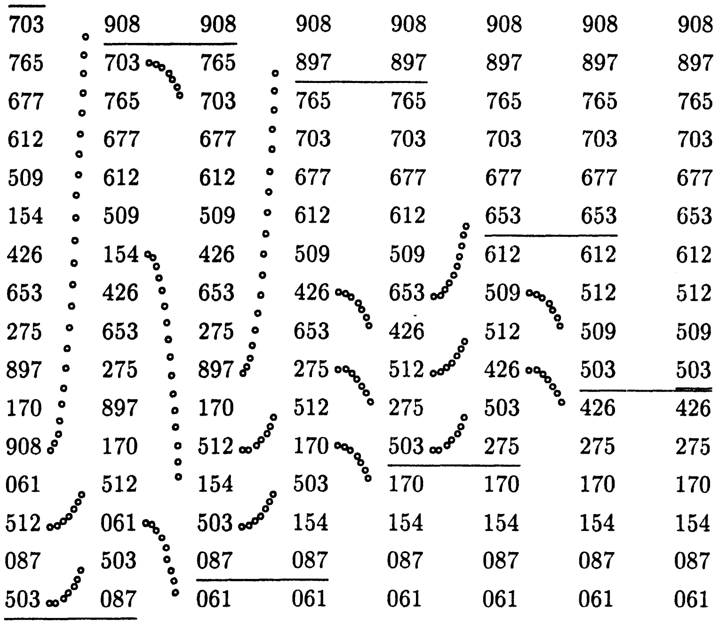


703	908	908	908	908	908	908	908
765	703	765	897	897	897	897	897
677	765	703	765	765	765	765	765
612	677	677	703	703	703	703	703
509	612	612	677	677	677	677	677
154	509	509	612	612	653	653	653
426	154	426	509	509	612	612	612
653	426	653	426	653	509	512	512
275	653	275	653	426	512	509	509
897	275	897	275	512	426	503	503
170	897	170	512	275	503	426	426
908	170	512	170	503	275	275	275
061	512	154	503	170	170	170	170
512	061	503	154	154	154	154	154
087	503	087	087	087	087	087	087
503	087	061	061	061	061	061	061

Рис. 16. Шейкер-сортировка.

Параллельная сортировка Бэтчера. Чтобы получить алгоритм обменной сортировки, время работы которого имеет порядок, меньший N^2, необходимо подобрать для сравнений пары *несоседних* ключей (K_i, K_j); иначе придется выполнить столько операций обмена записей, сколько инверсий имеется в исходной перестановке. Среднее число инверсий равно $\frac{1}{4}(N^2 - N)$. В 1964 году К. Э. Бэтчер [см. K. E. Batcher *Proc. AFIPS Spring Joint Computer Conference* **32** (1968), 307–314] открыл интересный способ программирования последовательности сравнений, предназначенной для поиска возможных обменов. Его метод далеко не очевиден. В самом деле, обосновать его справедливость весьма сложно, поскольку выполняется относительно мало сравнений. Рассмотрим два доказательства: одно в этом разделе, а другое — в разделе 5.3.4.

Схема сортировки Бэтчера несколько напоминает сортировку Шелла, но сравнения выполняются по-новому, а потому цепочки операций обмена записей не возникает. В качестве примера сравним табл. 1 и 5.2.1–3. Сортировка Бэтчера действует, как 8-, 4-, 2- и 1-сортировка, но сравнения не перекрываются. Поскольку в алгоритме Бэтчера, по существу, происходит слияние пар рассортированных подпоследовательностей, его можно назвать обменной сортировкой со слиянием.

Алгоритм M (*Обменная сортировка со слиянием*). Записи $R_1, \ldots, R_N$ перекомпоновываются в пределах того же пространства в памяти. После завершения сортировки их ключи будут упорядочены: $K_1 \leq \cdots \leq K_N$. Предполагается, что $N \geq 2$ (рис. 17).

M1. [Начальная установка p.] Установить $p \leftarrow 2^{t-1}$, где $t = \lceil \lg N \rceil$ — наименьшее целое число, такое, что $2^t \geq N$. (Шаги M2–M5 будут выполняться с $p = 2^{t-1}$, $2^{t-2}, \ldots, 1$.)

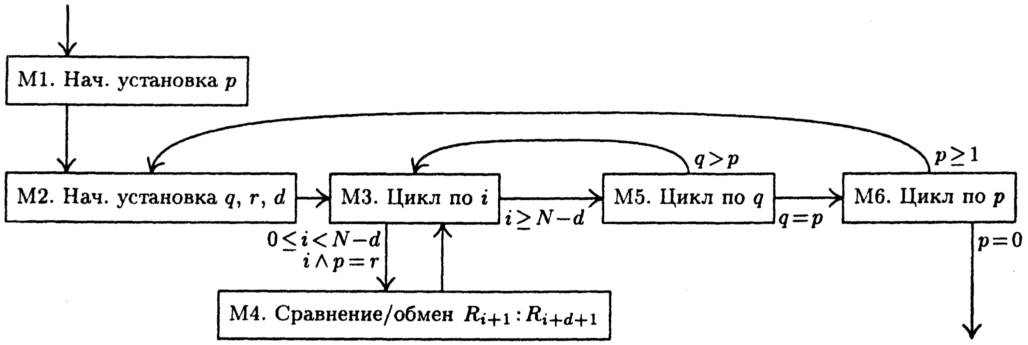


Рис. 17. Алгоритм M.

M2. [Начальная установка q, r, d.] Установить $q \leftarrow 2^{t-1}$, $r \leftarrow 0$, $d \leftarrow p$.

M3. [Цикл по i.] Для всех i, таких, что $0 \leq i < N - d$ и $i \wedge p = r$, выполнить шаг M4. Затем перейти к шагу M5. (Здесь через $i \wedge p$ обозначена операция "поразрядное логическое И" над представлениями целых чисел i и p; все биты результата равны 0, кроме тех битов, для которых в соответствующих разрядах i и p находятся 1. Так, $13 \wedge 21 = (1101)_2 \wedge (10101)_2 = (00101)_2 = 5$. К этому моменту d — нечетное кратное p (т. е. частное от деления d на p нечетно), а p — степень двойки, так что $i \wedge p \neq (i + d) \wedge p$. Отсюда следует, что шаг M4 можно выполнять при всех нужных значениях i в любом порядке или даже одновременно.)

M4. [Сравнение/обмен $R_{i+1}:R_{i+d+1}$.] Если $K_{i+1} > K_{i+d+1}$, поменять местами записи $R_{i+1} \leftrightarrow R_{i+d+1}$.

M5. [Цикл по q.] Если $q \neq p$, установить $d \leftarrow q - p$, $q \leftarrow q/2$, $r \leftarrow p$ и возвратиться к шагу M3.

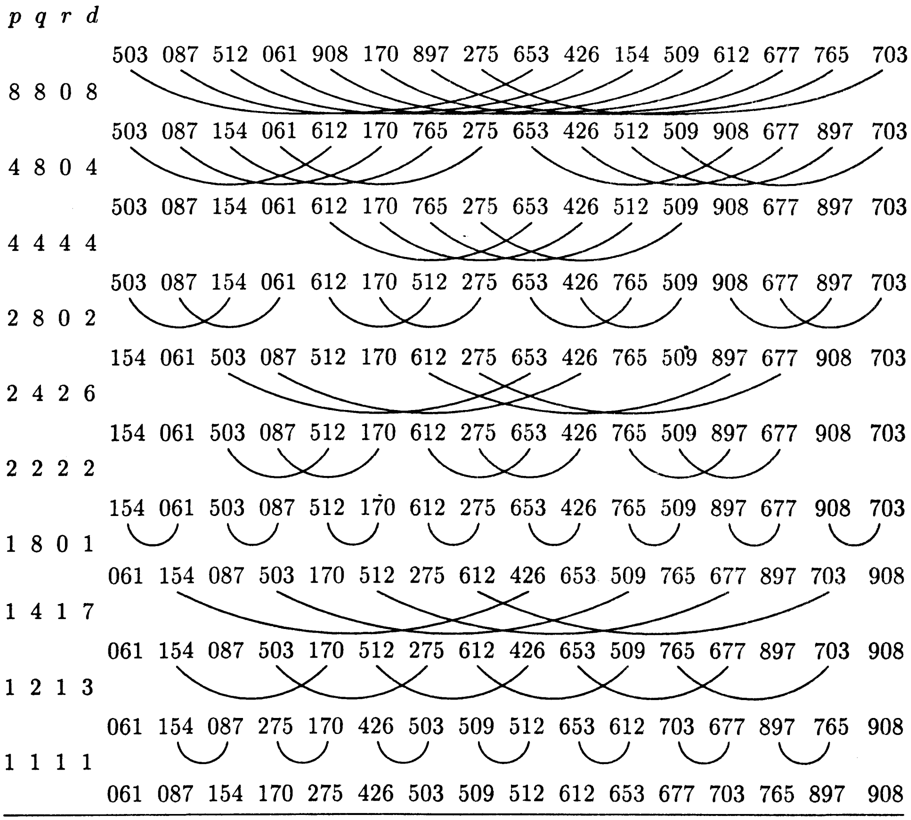
M6. [Цикл по p.] (К этому моменту перестановка $K_1\, K_2 \ldots K_N$ будет p-упорядочена.) Установить $p \leftarrow \lfloor p/2 \rfloor$. Если $p > 0$, возвратиться к шагу M2. ▮

В табл. 1 этот метод проиллюстрирован при $N = 16$. Обратите внимание на то, что, по существу, алгоритм сортирует N элементов путем независимой сортировки подмассивов $R_1, R_3, R_5, \ldots$ и $R_2, R_4, R_6, \ldots$, после чего выполняются шаги M2–M5 при $p = 1$ для слияния двух рассортированных последовательностей.

Чтобы доказать, что магическая последовательность сравнений и/или обменов, описанная в алгоритме M, действительно позволяет рассортировать любую последовательность $R_1\, R_2 \ldots R_N$, необходимо показать только, что в результате выполнения шагов M2–M5 при $p = 1$ будет слита любая 2-упорядоченная последовательность $R_1\, R_2 \ldots R_N$. С этой целью можно воспользоваться методом решеточных диаграмм из раздела 5.2.1 (см. рис. 11 на с. 106); каждая 2-упорядоченная перестановка множества $\{1, 2, \ldots, N\}$ соответствует на решетке единственному пути из вершины (0,0) к $(\lceil N/2 \rceil, \lfloor N/2 \rfloor)$. На рис. 18, (a) показан пример при $N = 16$, соответствующий перестановке 1 3 2 4 10 5 11 6 13 7 14 8 15 9 16 12. При $p = 1$, $q = 2^{t-1}$, $r = 0$, $d = 1$ на шаге M3 выполняется сравнение (и, возможно, обмен) записей $R_1 : R_2$, $R_3 : R_4$ и т. д. Этой операции соответствует простое преобразование пути на решетке:

Таблица 1

ОБМЕННАЯ СОРТИРОВКА СО СЛИЯНИЕМ (МЕТОД БЭТЧЕРА)



p q r d																
	503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
8 8 0 8																
	503	087	154	061	612	170	765	275	653	426	512	509	908	677	897	703
4 8 0 4																
	503	087	154	061	612	170	765	275	653	426	512	509	908	677	897	703
4 4 4 4																
	503	087	154	061	612	170	512	275	653	426	765	509	908	677	897	703
2 8 0 2																
	154	061	503	087	512	170	612	275	653	426	765	509	897	677	908	703
2 4 2 6																
	154	061	503	087	512	170	612	275	653	426	765	509	897	677	908	703
2 2 2 2																
	154	061	503	087	512	170	612	275	653	426	765	509	897	677	908	703
1 8 0 1																
	061	154	087	503	170	512	275	612	426	653	509	765	677	897	703	908
1 4 1 7																
	061	154	087	503	170	512	275	612	426	653	509	765	677	897	703	908
1 2 1 3																
	061	154	087	275	170	426	503	509	512	653	612	703	677	897	765	908
1 1 1 1																
	061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908

"перегиб" относительно диагонали при необходимости, чтобы путь нигде не проходил выше диагонали (рис. 18, (b) и доказательство в упр. 10). На следующей итерации шага M3 имеем $p = r = 1$ и $d = 2^{t-1} - 1, 2^{t-2} - 1, \ldots, 1$. В результате произойдет сравнение и/или обмен записей $R_2 : R_{2+d}$, $R_4 : R_{4+d}$ и т. д. Опять же, имеется простая геометрическая интерпретация: путь "перегибается" относительно прямой, расположенной на $\frac{1}{2}(d + 1)$ единиц ниже диагонали (рис. 18, (c) и (d)). В конце концов, получаем путь, изображенный на рис. 18, (e), который соответствует полностью рассортированной перестановке. На этом "геометрическое доказательство" справедливости алгоритма Бэтчера завершается (данный алгоритм можно было бы назвать сортировкой посредством перегибов!).

`MIX`-программа для алгоритма M приведена в упр. 12. К сожалению, количество вспомогательных операций, необходимых для управления последовательностью сравнений, весьма велико, так что программа менее эффективна, чем другие рассмотренные выше методы. Однако она обладает одним важным компенсирующим качеством: все сравнения и/или обмены, определяемые данной итерацией шага M3, можно выполнять *одновременно* на компьютерах или в сетях, которые реализуют параллельные вычисления. С такими параллельными операциями сортировка осу-

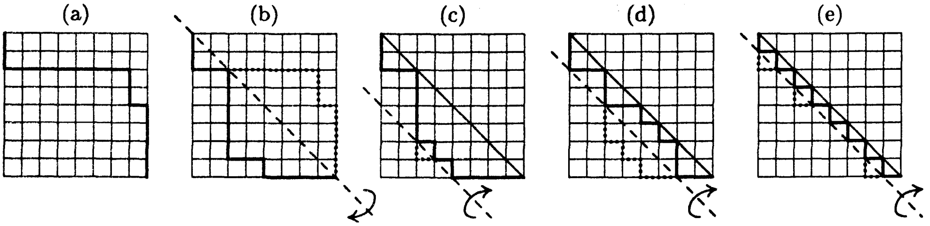


Рис. 18. Геометрическая интерпретация метода Бэтчера, $N = 16$.

ществляется за $\frac{1}{2}\lceil\lg N\rceil(\lceil\lg N\rceil + 1)$ шагов, и это один из самых быстрых общих методов среди всех известных. Например, *1 024 элемента можно рассортировать методом Бэтчера всего за 55 параллельных шагов.* Его ближайшим соперником является метод Пратта (см. упр. 5.2.1–30), который затрачивает 40 или 73 шага в зависимости от того, как считать: если допускать перекрытие сравнений до тех пор, пока не потребуется выполнять перекрывающиеся обмены, то для сортировки 1 024 элементов методом Пратта требуется всего 40 циклов сравнения и/или обмена. Дальнейшие пояснения приводятся в разделе 5.3.4.

Быстрая сортировка. В методе Бэтчера последовательность сравнений предопределена: каждый раз сравниваются одни и те же пары ключей независимо от информации о сортируемой последовательности, которую могут предоставить уже выполненные операции сравнения. Это утверждение в большой мере справедливо и применительно к методу пузырька, хотя алгоритм B использует в ограниченной степени полученные сведения, чтобы сократить объем работы в крайней справа части последовательности. Обратимся теперь к совсем иной стратегии, при которой для определения, какие ключи сравнивать следующими, используется результат каждого сравнения. Такая стратегия не годится для параллельных вычислений, но она может оказаться плодотворной для обычных компьютеров с последовательным выполнением операций.

Основная идея, на которой базируется этот метод, состоит в том, чтобы взять одну из записей, скажем R_1, и передвинуть ее на то место, которое она должна занять после сортировки, скажем в позицию s. В поиске этой окончательной позиции придется несколько перекомпоновать и остальные записи таким образом, чтобы слева от позиции s не оказалось ни одной записи с бóльшим ключом, а справа — с меньшим. В результате последовательность окажется разбитой таким образом, что исходная задача сортировки всего массива записей будет сведена к задачам независимой сортировки пары подмассивов $R_1 \ldots R_{s-1}$ и $R_{s+1} \ldots R_N$ меньшей длины. Далее, тот же метод применяется и к полученным подмассивам до тех пор, пока не будет завершена сортировка всей последовательности. Существует несколько способов разбиения всей последовательности на подмассивы. Рассмотрим следующую процедуру, предложенную Седгевиком, которая, на наш взгляд, является лучшей из имеющихся на сегодняшний день, в основном, вследствие своей "прозрачности" и простоты анализа алгоритма. Итак, пусть имеются указатели i и j, причем вначале $i = 2$ и $j = N$. Предположив, что запись R_i должна после разделения принадлежать левому подмассиву (это легко определить, сравнив K_i с K_1), увеличим i на 1 и продолжим далее до тех пор, пока не обнаружим такую запись R_i, которая принадлежит правому подмассиву. Аналогично будем уменьшать j на 1 до тех пор, пока не обнаружим такую запись R_j, которая принадлежит левому подмассиву. Если $i < j$, поменяем местами R_i с R_j; затем повторим аналогичную процедуру со следующей записью, “сжигая свечку с обоих концов”, пока не станет $i \geq j$. Завершается процесс разделения массива обменом записей R_j с R_1. Посмотрим, например, что произойдет с нашей последовательностью из шестнадцати чисел.

		i ↓														j ↓
Исходный массив:	[503	**087**	512	061	908	170	897	275	653	426	154	509	612	677	765	**703**]
1-й обмен:	503	087	**512**	061	908	170	897	275	653	426	**154**	509	612	677	765	703
2-й обмен:	503	087	154	061	**908**	170	897	275	653	**426**	512	509	612	677	765	703
3-й обмен:	503	087	154	061	426	170	**897**	**275**	653	908	512	509	612	677	765	703
Переключение указателей:	503	087	154	061	426	170	**275**	**897**	653	908	512	509	612	677	765	703
Разделенный массив:	[275	087	154	061	426	170]	**503**	[**897**	653	908	512	509	612	677	765	703]
							↑ j	↑ i								

(Для удобства анализа положения i и j ключи K_i и K_j выделены в таблице полужирным шрифтом.)

В табл. 2 показано, как выбранная нами для примеров последовательность полностью сортируется при помощи этого метода за 11 итераций. В скобки заключены подмассивы, которые еще предстоит рассортировать; в программе эти подмассивы можно представлять двумя переменными двоичными (l, r) (границы рассматриваемого в данный момент массива) и стеком дополнительных пар (l_k, r_k). Каждый раз, когда массив разбивается, помещаем в стек *больший* из подмассивов и начинаем обрабатывать оставшийся; и так до тех пор, пока не будут получены тривиально короткие массивы. Как показано в упр. 20, такая процедура гарантирует, что в стеке никогда не будет находиться более $\lg N$ элементов.

Только что описанную процедуру сортировки можно назвать *обменной сортировкой с разделением*. Ее идея принадлежит Ч. Э. Р. Хоару, интереснейшая статья которого [C. A. R. Hoare, *Comp. J.* **5** (1962), 10–15] — одно из наиболее исчерпывающих из когда-либо опубликованных сообщений об этом методе. Хоар окрестил свой метод “quicksort” (“быстрая сортировка”), и это название вполне соответствует действительности, так как внутренний цикл вычислений при реализации на любом из современных компьютеров оказывается очень быстрым. При всех сравнениях, выполняемых на текущей итерации, используется один и тот же ключ, так что соответствующее значение можно держать в регистре. Обмен тактами между сравнениями выполняется только в отношении единственного индекса. Более того, количество перезаписей данных довольно мало́ — обработка табл. 2, например, требует всего лишь 17 операций перезаписи данных.

Вспомогательные операции (требуемые для управления стеком и переменными i, j) не сложны, но все же из-за них процедура быстрой сортировки посредством разделений пригодна, в основном, при больших значениях N. Поэтому в следующем алгоритме используется несколько измененная стратегия обработки коротких подмассивов.