УДК 004.424.5 : 519.683.6

С. Д. Винничук, д-р техн. наук
Ин-т проблем моделирования в энергетике
им. Г. Е. Пухова НАН Украины
(Украина, 03164, Киев, ул. Ген. Наумова, 15,
тел.(044)4249171, E-mail: vynnychuk@i.ua)

Естественная сортировка слиянием с минимизацией объема дополнительной памяти

Предложен алгоритм, требующий объема дополнительной памяти $O(\log n)$, с трудоемкостью в худшем случае $O(n\log^2 n)$. Предложены также алгоритмы устойчивой нерекурсивной сортировки слиянием, позволяющие учитывать естественную упорядоченность исходного массива данных длиной n при уменьшении объема дополнительной памяти до величин $n/2 + O(1)$, $n/4 + O(1)$, $n/8 + O(1)$ и трудоемкости в случае наихудшего расположения элементов порядка $O(n\log n)$.

Запропоновано алгоритм, що потребує обсягу додаткової пам'яті $O(\log n)$, з трудомісткістю для гіршого випадку $O(n\log^2 n)$. Запропоновано також стійкі нерекурсивні алгоритми сортування злиттям, які дозволяють враховувати природню впорядкованість початкового масиву даних довжиною n при зменшенні обсягу додаткової пам'яті до величин $n/2 + O(1)$, $n/4 + O(1)$, $n/8 + O(1)$ з трудомісткістю для гіршого випадку $O(n\log n)$.

К л ю ч е в ы е с л о в а: сортировка слиянием, алгоритм, оценка трудоемкости.

Постановка задачи. Алгоритм сортировки слиянием предложен Джоном фон Нейманом в 1945 г. [1]. Сложность исходного алгоритма оценивается величиной порядка $O(n\log n)$, где n — число элементов массива независимо от упорядоченности. Такой порядок сложности имеет и быстрая сортировка Хоара, но только для лучшего расположения элементов.

Обозначим алгоритм слияния $\boldsymbol{S0}$. Кратко его можно описать так. Имеются два отсортированных (упорядоченных) массива, которые для удобства изложения назовем фрагментами. Их объединение в один упорядоченный массив реализуется простым циклом, в котором меньший из двух текущих элементов упорядочиваемых фрагментов переносится в выходной массив, и так продолжается до исчерпания обоих входных массивов.

Одним из достоинств сортировки слиянием является возможность реализации алгоритма без использования рекурсий, его устойчивость (не меняется порядок следования элементов с одинаковыми значениями) и возможность сортировки массивов большой длины. Устойчивость сортировки слиянием обусловливает ее предпочтительность по сравнению с

методами быстрой сортировки и пирамидальной сортировки для тех приложений, в которых устойчивость важна. Кроме того, сортировка слиянием удобна для упорядочения связных списков, для которых применим метод последовательного доступа.

Различают несколько вариантов реализации сортировки слиянием: нисходящая и восходящая (указывают на способ формирования массивов элементов меньшей длины), естественная (указывает на использование информации об упорядоченности исходных данных), многофазная и ленточная (сортировки данных большой длины, превышающей объем оперативной памяти).

К основным недостаткам алгоритма сортировки слиянием ***S*0** относят требование выделения дополнительной памяти, пропорциональной длине массива (возможно однократное выделение памяти в начале работы алгоритма), а также тот факт, что даже для полностью отсортированного массива количество вычислений будет таким же, как для произвольного неупорядоченного массива.

Существуют способы устранения этих недостатков, в частности рекурсивные алгоритмы, использующие только $O(\log^2 n)$ стековой памяти (так называемое условие минимальной памяти). Но тогда время расчета оценивается величиной порядка $O(n\log^2 n)$, где упорядоченность исходного массива не учитывается. Такие алгоритмы описаны в [2]. В них использована идея Пратта специального разделения пары последовательно расположенных упорядоченных частей исходного массива на две пары таких частей меньшего размера (в дальнейшем способ такого деления будем называть делением по Пратту), где все элементы первой пары не превосходят элементы второй.

Суть идеи алгоритма Пратта состоит в следующем. Два фрагмента, U и V, разделяют на части: массив U на U_1 и U_2, массив V на V_1 и V_2. В результате все элементы массивов U_2 и V_2 больше, чем элементы массивов U_1 и V_1, или равны им. Тогда элементы массивов U_2 и V_1 можно переставить местами так, что образуются две пары фрагментов (U_1, V_1) и (U_2, V_2) меньшего размера, и сортировка пары фрагментов (U, V) сводится к последовательной сортировке пар фрагментов (U_1, V_1) и (U_2, V_2). Процедура такого разбиения может быть рекурсивно продолжена до достижения некоторой минимальной длины фрагментов, когда можно реализовать упорядочение элементов пары фрагментов (например, для составляющих частей фрагментов длиной 1). Сложность алгоритма слияния двух массивов общей длиной n равна $O(n\log n)$.

Существуют способы учета естественной упорядоченности [1] для ленточной сортировки. В то же время, в современных программных реали-

зациях сортировки слиянием данные естественной сортировки практически не используются. Одна из современных реализаций естественной сортировки слиянием приведена в [3], где показано, что при использовании данных естественной сортировки в случае поиска границ отсортированных массивов при их слиянии время работы алгоритма для случайных наборов чисел примерно совпадает с аналогичным временем работы исходного алгоритма сортировки слиянием. При этом отмечено, что алгоритм должен обеспечить более быструю сортировку для массивов, близких к упорядоченным.

Можно предположить, что основной проблемой, которая препятствует использованию в алгоритмах сортировки слиянием информации об упорядоченности данных, является необходимость выделения дополнительной памяти для запоминания границ отсортированных фрагментов данных. Эта проблема решена в предлагаемых алгоритмах устойчивой сортировки слиянием, учитывающих естественную упорядоченность исходного массива данных для случаев, когда исходный и дополнительный массивы размещены в оперативной памяти либо в файле подкачки.

Все алгоритмы сортировки слиянием, использующие естественную упорядоченность элементов массива (естественная сортировка слиянием), содержат начальный шаг, на котором происходит выделение упорядоченных частей исходного массива. Каждая из этих частей имеет неизвестную заранее длину. Практически любой из алгоритмов формирования таких частей массива (фрагментов) имеет сложность порядка $O(n)$, что не меняет оценки общей сложности метода сортировки.

Формирование упорядоченных фрагментов. В работе [1] описаны способы формирования упорядоченных фрагментов для вариантов ленточной сортировки слиянием с традиционными последовательностями возрастающих и убывающих значений чисел. Рассмотрим простейший из таких способов — выделение частей массива, в которых значение следующих элементов не меньше предыдущих, в котором (для обеспечения длины фрагмента не менее 2) при необходимости реализуется первичная сортировка двух элементов (их перестановка местами).

С учетом такого способа первичного формирования упорядоченных фрагментов заранее неизвестной длины опишем алгоритм слияния $\boldsymbol{A}$:

0. На начальном шаге выполняется формирование упорядоченных фрагментов нулевого уровня ($u = 0$) длиной не менее двух элементов с занесением информации о длине каждого из вновь образованных фрагментов в массиве фрагментов *MF* в порядке, обратном номеру фрагмента, начиная с последнего элемента массива *MF*.

1. Если на уровне u число фрагментов равно единице, то процедура сортировки завершена, иначе — переход к п. 2.

2. $u = u + 1$. Переход к п. 3, т.е. слиянию пар последовательных фрагментов и формированию фрагментов более высокого уровня.

3. Выполняется слияние всех последовательных пар фрагментов уровня $(u - 1)$ в цикле и формирование вместо них упорядоченных массивов (фрагментов более высокого уровня u) длиной, равной суммарной длине рассматриваемой пары фрагментов. Данные о длине фрагментов уровня u записываются в массив MF в обратном новому номеру фрагмента порядке, начиная с последнего элемента массива MF.

По завершении цикла — переход к п. 1.

Пример формирования фрагментов с помощью алгоритма ***A*** приведен в табл. 1.

Поскольку известный алгоритм слияния ***S*0** при слиянии пар фрагментов всегда требует объема памяти, равного их суммарной длине, для двух фрагментов, представляющих данные всего массива, требуется дополнительная память, равная длине массива. Следовательно, при прямом использовании алгоритма слияния ***S*0** невозможно добиться экономии дополнительной памяти. Рассмотрим модифицированный алгоритм слияния, позволяющий уменьшить требуемый объем дополнительной памяти при сохранении той же сложности.

Модифицированный алгоритм процедуры слияния двух упорядоченных фрагментов. В известных алгоритмах слияния используется три массива: два упорядоченных, которые объединяются в третьем. Если все три массива располагаются в оперативной памяти, то процедура слияния может быть выполнена с использованием одного дополнительного массива размером, равным меньшей из длин фрагментов, что реализуется в рассматриваемом далее алгоритме слияния ***S*1**. Для упрощения рассмотрим случай, когда два сортируемых массива являются упорядоченными фрагментами одного массива, где они размещены последовательно.

Пусть в исходном массиве M первый фрагмент начинается с ячейки с номером n_1, и имеет длину k_1, а второй фрагмент — с ячейки $n_1 + k_1$ и

Таблица 1

Номер элемента в массиве	0	1	2	3	4	5	6	7	8	9	10
Исходная последовательность	7	5	11	8	3	2	9	12	14	6	4
Преобразованная последовательность	5	7	8	11	2	3	9	12	14	4	6
Сформированные фрагменты	1		2		3					4	

Рис. 1. Модифицированный алгоритм слияния двух фрагментов одного массива с копированием в буфер: А — первого фрагмента; Б — второго фрагмента; цифры указывают порядок занесения данных

имеет длину k_2. Тогда модифицированный алгоритм слияния ***S*1** можно представить такой последовательностью шагов:

1. Копируем один из фрагментов в дополнительный массив *MF*.

2. Если в массив *MF* скопирован первый фрагмент, переходим к п. 3, если иначе — к п. 4.

3. Реализуем традиционный способ слияния k_1 элементов массива *MF* и k_2 элементов второго фрагмента массива *M* с размещением элементов в исходном массиве *M*, начиная с ячейки n_1.

4. Реализуем слияние k_2 элементов массива *MF* и k_1 элементов первого фрагмента массива *M* в порядке убывания значений элементов массива с размещением элементов в исходном массиве *M* в обратном порядке, начиная с ячейки $n_1 + k_1 + k_2 - 1$.

Пример модифицированной процедуры слияния с помощью алгоритма ***S*1** представлен на рис. 1, где использованы третий и четвертый фрагменты из табл. 1. Полужирным шрифтом выделены численные данные, переданные в буфер (массив *MF*) как копии соответствующих фрагментов. Варианты А и Б описывают два способа сортировки с использованием полей исходного массива:

1. В порядке возрастания элементов, начиная с местоположения первого элемента первого фрагмента, где в буфер (массив *MF*) скопирован первый фрагмент.

2. В порядке убывания элементов, начиная с местоположения последнего элемента второго фрагмента, где в буфер (массив *MF*) скопирован второй фрагмент.

Как видно из рис. 1, при упорядочении данные массива *M* не теряются, но возможны случаи, когда часть данных уже находится в нужном месте (вариант Б), т.е. процедура слияния может быть прекращена досрочно.

Оба способа сортировки всегда приводят к одинаковому результату, но с различным объемом данных, копируемых в буфер. Следовательно, при разработке алгоритма сортировки слиянием с уменьшенным объемом дополнительной памяти целесообразно учесть это обстоятельство и копировать в буфер тот из пары фрагментов массива, который имеет меньшую длину.

Особенности реализации и оценка сложности алгоритма *A* естественной сортировки слиянием. С помощью алгоритма ***A*** необходимо разместить информацию (массив *MF*) о фрагментах, а также выделить память для буфера, в который будет копироваться фрагмент меньшего размера. Такой объем памяти предполагается выделить в начале работы программы. Установим минимально необходимое значение числа ячеек памяти массива *MF*, если исходный массив *M* имеет длину *n*.

Утверждение 1. Минимально необходимое число ячеек памяти массива *MF* не может превысить числа $m = (n + 1)/2 + O(1)$, где *n* — длина массива *M*.

Доказательство. Максимальное число фрагментов, на которое может быть разбит исходный массив *M* на начальном шаге, составляет $(n + 1)/2$. Тогда каждый из фрагментов имеет длину 2 либо 1 (возможно, только в последнем фрагменте). В этом же массиве располагается фрагмент меньшего размера из двух последовательных упорядочиваемых. Рассмотрим случай, когда оба фрагмента имеют одинаковую длину *k*.

Если $k = 2$, то для двух фрагментов необходимо разместить данные об их длине и сам фрагмент длины 2. Следовательно, кроме данных о длине предельного числа фрагментов, необходимо предусмотреть две дополнительные ячейки в массиве *MF*, что вместе составит величину $(n + 1)/2 + 2$.

Пусть $k > 2$. Тогда для части массива *M* длиной $2k$ максимально возможное число фрагментов длины 2 равно *k*. Однако их фактическое число равно двум, поэтому максимально возможное число фрагментов уменьшится на $k - 2$. Следовательно, суммарный объем требуемой памяти для массива *MF* при размещении в нем данных о длинах всех фрагментов и данных об одном из фрагментов не превысит значения $(n + 1)/2 - (k - 2) + k = (n + 1)/2 + 2$. При этом предельное значение будет достигнуто только в случае, когда все другие фрагменты имеют длину 2 (последний может иметь длину 1).

Пусть теперь два объединяемых фрагмента имеют различные длины: k_1 и k_2. Тогда максимально возможное число фрагментов, соответствующее длине массива $k_1 + k_2$, будет не меньше, чем $(k_1 + k_2)/2 = k_0$. Но $k_0 \geq \min(k_1, k_2)$. Следовательно, общий объем требуемой для массива *MF* памяти по сравнению с объемом памяти для фрагментов равной длины может уменьшиться на величину $k_0 - \min(k_1, k_2)$. Поэтому и в данном случае предельное число ячеек в массиве *MF* не может превысить $(n + 1)/2 + 2$.

В случае, когда имеется более двух фрагментов длиной, превышающей 2, при фиксированных двух выбранных фрагментах суммарное число оставшихся фрагментов может только уменьшиться по сравнению с максимально возможным. Следовательно, требуемое число ячеек памяти не превысит $(n + 1)/2 + 2$. Утверждение доказано.

Согласно утверждению 1 объем требуемой для массива памяти может быть ограничен числом $(n + 1)/2 + O(1)$, а размерность массива *MF* может быть определена в самом начале процедуры сортировки и однократно выделена требуемая память размером $c = (n + 1)/2 + 4$.

Рассмотрим правило размещения данных в массиве *MF*. Разместим данные фрагмента меньшей длины $k_{\min}$ в ячейках от 1 до $k_{\min}$. В ячейку 0 может быть занесено значение, меньшее любого из чисел фрагментов, если длина второго фрагмента меньше длины первого. Соответственно в ячейку $k_{\min} + 1$ может быть занесено значение, большее любого из чисел фрагментов, если длина второго фрагмента больше длины первого. Данные о длинах фрагментов следует заносить, начиная с ячейки $c - 1$, в обратном порядке по отношению к номерам фрагментов.

Алгоритм ***A***, в котором процедура слияния реализована в соответствии с алгоритмом ***S*1**, полностью описывает весь процесс сортировки произвольного массива данных, размещаемого в оперативной памяти. Оценим трудоемкость совместного алгоритма, который обозначим ***A*** + ***S*1**.

Согласно алгоритму ***A*** на нулевом уровне формирование упорядоченных фрагментов с числом элементов не менее двух реализуется одним циклом длины n. Поэтому его трудоемкость пропорциональна длине массива, т.е. является величиной порядка $O(n)$.

На каждом новом уровне $u > 0$ происходит слияние всех последующих пар фрагментов. При обработке двух фрагментов число операций присваивания оценивается величиной, пропорциональной суммарному числу элементов обоих фрагментов. Но последующие пары фрагментов охватывают либо весь массив (при четном числе фрагментов), либо все его элементы, кроме элементов последнего фрагмента. Поэтому число операций по упорядочению элементов на любом из уровней $u > 0$ также является величиной порядка $O(n)$.

Определим максимально возможное число уровней. На начальном шаге алгоритма ***A*** может быть сформировано число фрагментов, не превышающее $(n + 1)/2$. На каждом из последующих уровней $u > 0$ пара фрагментов преобразуется в один фрагмент. При этом суммарное число фрагментов уменьшается приблизительно в два раза. Это продолжается до тех пор, пока не останется один фрагмент, являющийся результатом упорядочения исходного массива. Поэтому максимально возможное число уровней $u_{\max}$ не превысит $\log_2 n + 1$.

Следовательно, общая трудоемкость алгоритма ***A*+*S*1** для наиболее сложного случая оценивается величиной $O(n \log n)$. При этом учтена естественная упорядоченность исходного массива, требуемая дополнительная память, равная $n/2 + O(1)$, выделяется один раз в начале работы и не используется стековая память.

Дальнейшее уменьшение объема дополнительной памяти. Пусть на начальном шаге, на котором происходит выделение упорядоченных фрагментов, длина их будет не меньше четырех. Это всегда можно обеспечить, например, объединением двух последовательных фрагментов, содержащих по два или три элемента. Тогда общее число фрагментов на начальном этапе уменьшится вдвое. Но уменьшится ли вдвое необходимая дополнительная память? Естественно, нет, если использовать алгоритм ***S*1** сортировки пар фрагментов.

Для произвольной пары фрагментов суммарной длины $n0$ можно однократно использовать алгоритм их деления по Пратту так, что число элементов в новых парах фрагментов будет равным или меньшим числа $(n0 + 1)/2$. Но тогда для каждой новой пары фрагментов длина меньшего из них не может превысить числа $(n0 + 1)/4$. Поскольку в алгоритме слияния ***S*1** требуемый объем дополнительной памяти определяется длиной меньшего из фрагментов, максимально возможный объем требуемой памяти в случае $n0 = n$ является величиной порядка $n/4 + O(1)$, что в два раза меньше величины, полученной с помощью алгоритма ***A*+*S*1**. При этом незначительно увеличится программный код и возрастет число операций (на реализацию разового деления по Пратту и формирование массивов половинной длины — величина порядка $O(n)$). Следовательно, общая трудоемкость процедуры слияния всех пар фрагментов на каждом уровне останется $O(n)$, а общая трудоемкость алгоритма сортировки будет $O(n \log n)$.

Если на начальном шаге сортировки добиваться минимальной длины фрагментов не менее восьми и дважды использовать деление по Пратту двух сортируемых фрагментов, то требуемая дополнительная память уменьшится до величины $n/8 + O(1)$ при дальнейшем увеличении объема вычислений и, возможно, длины программного кода. Однако и в этом

случае общая трудоемкость процедуры слияния всех пар фрагментов на каждом уровне останется $O(n)$, а общая трудоемкость алгоритма сортировки — $O(n\log n)$.

Такое совершенствование алгоритма ***A*** можно продолжать и далее. Покажем, что существует устойчивый нерекурсивный алгоритм сортировки слиянием, учитывающий естественную упорядоченность исходного массива данных, в котором обеспечивается величина дополнительной памяти $O(\log n)$ при общей сложности $O(n(\log n)^2)$.

Устойчивый нерекурсивный алгоритм естественной сортировки слиянием с объемом дополнительной памяти *O* (log*n*). Предлагаемый алгоритм ***A*1** основан на идее одновременного определения упорядоченных фрагментов и попарного слияния фрагментов одного и того же уровня. Два последовательных фрагмента одного уровня можно объединить и получить фрагмент более высокого уровня. То же самое можно сделать и для пар таких фрагментов более высокого уровня. Число уровней таких объединений составляет величину не более $O(\log n)$ (поскольку на каждом новом уровне число фрагментов уменьшается почти вдвое). При этом оказывается, что для реализации объединения пар фрагментов необходимо знать только границы их расположения в исходном массиве данных.

Пусть $X0$ — массив данных о номере элемента массива M, с которого начинается первый из пары фрагментов соответствующего уровня, или число – 1, если информация еще не определена; $X1$ — массив данных о длине (числе элементов массива M) для первого фрагмента соответствующего уровня, или признак того, что информация еще не определена; $X2$ — массив данных о длине (числе элементов массива M) для второго фрагмента соответствующего уровня, или признак того, что информация еще не определена.

Алгоритм ***A*1**.

1. *Формирование пары последовательных фрагментов нулевого уровня.* При последовательном просмотре элементов массива M формируем пару фрагментов уровня $u = 0$ с занесением информации в массивы $X0$, $X1$ и $X2$. Начальный номер ячейки для пар фрагментов — это первый номер ячейки первого фрагмента и он записывается в ячейке $X0[0]$. Если для первых двух элементов фрагмента их значения убывают, то элементы меняются местами. Если значение следующего элемента в массиве M больше, чем предыдущего, то длина фрагмента увеличивается на единицу, а иначе — определена длина фрагмента. При определении длины первого фрагмента ее значение записывается в ячейке $X1[0]$. При определении длины второго фрагмента ее значение записывается в ячейке $X2[0]$. Если достигнут конец массива M, то определена длина текущего фрагмента

(может равняться единице), и переход к п. 4. Если сформирована пара фрагментов и конец массива M не достигнут, то переход к п. 2.

2. *Слияние двух фрагментов одного уровня и*. Формируется длина объединенного фрагмента уровня $u + 1$. Переход к п. 3.

3. *Анализ данных о паре фрагментов уровня* $u + 1$. Если длина первого фрагмента не занесена ($X0[u + 1] < 0$), то $X0[u + 1] = X0[u]$; $X1[u + 1] = X1[u] + X2[u]$; $X1[u] = 0$; $X2[u] = 0$ и переход к п. 1. Если длина первого фрагмента занесена, то $X2[u + 1] = X1[u] + X2[u]$; $X1[u] = 0$; $X2[u] = = 0$; $u = u + 1$ и переход к п. 2.

4. *Анализ числа сформированных фрагментов уровня и*. Если при достижении конца массива M в цикле по формированию пар фрагментов сформированы два фрагмента, то переход к п. 5, а иначе — к п. 6.

5. *Слияние двух фрагментов одного уровня и*. Формируется длина объединенного фрагмента уровня $u + 1$. Переход к п. 6.

6. *Анализ данных о фрагментах уровня* $u + 1$. Если $X0[u + +1] < 0$ (признак того, что информация не определена), то процесс упорядочения массива M завершен, а иначе — переход к п. 7.

7. *Анализ данных о первом из фрагментов уровня* $u + 1$. Если длина первого фрагмента не занесена, то $X1[u + 1] = X1[u] + X2[u]$, $u = u + 1$ и переход к п. 6. Если длина первого фрагмента занесена, то $X2[u + +1] = X1[u] + X2[u]$, $u = u + 1$ и переход к п. 5.

Работа предлагаемого алгоритма ***A*1** представлена в табл. 2 по шагам, где на одном шаге либо формируется один фрагмент, либо реализуется одно слияние. Для произвольного массива M, $M[11]=\{5,6,4,7,3,9,8,2,10,1, 11\}$, число уровней не превышает $\log_2 n + 1$ и для выбранного примера число уровней равно целой части от $(\log_2 11 + 1)$ и равно четырем. Поэтому длина каждого массива $X0$, $X1$ и $X2$ равна четырем.

Для отображения начального состояния переменных все их значения представлены на шаге 0. В данном алгоритме требуется отдельная обработка последнего фрагмента, что отражено в табл. 2. Здесь происходит передача данных от уровня к уровню до тех пор, пока на одном из них не окажутся заполненны числами больше нуля длины двух фрагментов.

Особенности предлагаемого алгоритма следующие.

1. Строго последовательная обработка входных данных, что является преимуществом алгоритма обычной сортировки слиянием при обработке свя́зных списков, для которых применим метод последовательного доступа.

2. Объем дополнительной памяти, необходимый для слияния сформированных пар фрагментов, составляет величину $O(\log n)$.

3. Общее число шагов, обеспечивающее выделение фрагментов, определяется по их слиянию и в зависимости от числа выделяемых фрагментов

Таблица 2

Массив	Шаг алгоритма									Последний фрагмент		
	0	1	2	3	4	5	6	7	8			
M	5	5	5	4	5	5		2	5	2	2	1
	6	6	6	5	6	6		3	6	3	3	2
	4	4	4	6	4	4		4	4	4	4	3
	7	7	**7**	7	7	7		5	7	5	5	4
	3	3	3		3	3	2	6	3	6	6	5
	8	8	8		8	8	3	7	8	7	7	6
	9	9	9		9	9	8	8	9	8	8	7
	2	2	2		2	2	9	9	2	9	9	8
	10	10	10		10	10	10	10	10	10	10	9
	1	1	1		1	1			1	1	1	10
	11	11	11		11	11			11	11	11	11
$X0_0$	– 1	0	0		4	4			9			
$X1_0$	– 1	2	2		3	3			2			
$X2_0$	– 1		2			2						
$X0_1$	– 1			0	0	0	0			9		
$X1_1$	– 1			4	4	4	4			2		
$X2_1$	– 1						5					
$X0_2$	– 1							0	0	0	0	
$X1_2$	– 1							9	9	9	9	
$X2_2$	– 1										2	
$X0_3$	– 1											0
$X1_3$	– 1											11
$X2_3$	– 1											

Примечания: 1) в первом столбце нижний индекс указывает уровень, соответствующий номеру ячейки; 2) значения — 1 отображаются в ячейках только нулевого шага, а на всех других шагах, включая обработку последнего фрагмента, вместо — 1 пустая ячейка; 3) на третьем шаге представлен результат сортировки пары фрагментов, для которых данные сформированы на предыдущем шаге; 4) на втором шаге в массивах $X0$, $X1$, $X2$ происходит слияние фрагментов с выделенными длинами; 5) в последнем столбце представлен результат последнего слияния и окончательные данные.

s, которое не должно превышать $(n+1)/2$. Если $s = 1$, то массив упорядочен и работа алгоритма завершена; если $s > 1$, то суммарное число шагов N_s, на каждом из которых либо формируется фрагмент, либо происходит слияние двух фрагментов одного уровня, может быть оценено сверху по формуле

$$N_s \le s + \sum_{k=1}^{r} (s-1)/2^k + r < 2s + \log_2 s \le n + 1 + \log_2 n, \qquad (1)$$

где r равно целой части $\log_2 s$, а последняя составляющая суммы, равная r, соответствует шагам обработки последнего фрагмента.

Действительно, в (1) составляющая суммы для $k = 1$ соответствует числу пар фрагментов нулевого уровня за исключением последней пары (или последнего фрагмента), обрабатываемых отдельно как исключение. Поэтому число пар без обработки исключений равно результату деления без остатка $(s - 1)/2$, число пар фрагментов второго уровня ($k = 2$) будет результатом деления без остатка $(s - 1)/4$ и так далее, что обусловливает первое из неравенств формулы (1). При этом строгое неравенство

$$N_s < s + \sum_{k=1}^{r} (s-1)/2^k$$

возможно только в случае, когда число фрагментов является степенью числа 2. Второе неравенство,

$$s + \sum_{k=1}^{r} (s-1)/2^k + r < 2s + \log_2 s,$$

при r, равном целой части $\log_2 s$, следует из соотношения для геометрической прогрессии

$$s + \sum_{k=1}^{r} (s-1)/2^k \le s + \sum_{k=1}^{r} s/2^k \le \frac{s - s/2^r}{1-1/2} < \frac{s}{1-1/2} = 2s.$$

В результате расчета по формуле (1) для данных табл. 2 при $s = 5$, $r = 2$ получим $N_s = s + (s-1)/2 + (s-1)/4 + r = 5 + 2 + 1 + 2 = 8 + 2 = 10 < 2 \cdot 5 + 2 = 12 < 11 + 1 + 3 = 15$.

Формула (1) позволяет определить сложность алгоритма первичного формирования фрагментов, которая составляет $O(n)$.

Перейдем к рассмотрению алгоритма нерекурсивного устойчивого слияния пар фрагментов.

Простейшие случаи слияния двух фрагментов. Пусть для двух упорядоченных фрагментов известны номер ячейки z, с которой начинается первый фрагмент, и длины двух фрагментов: $n1$ и $n2$. Пусть также имеется массив дополнительной памяти некоторой фиксированной длины от единицы до величины $O(\log n)$. Покажем, что существует нерекурсивный алгоритм слияния с учетом упорядоченности, для которого достаточный объем дополнительной памяти составляет $O(\log n)$. Пусть длина такого массива равна L и это массив $MF[L]$. Рассмотрим три исключительных случая:

$$\min(n1, n2) <= L, \tag{2}$$

$$M[z + n1 - 1] \le M[z + n1], \tag{3}$$

$$M[z + n1 + n2 - 1] > M[z]. \tag{4}$$

Условие (2) соответствует случаю, когда можно воспользоваться дополнительным массивом $MF[L]$ и реализовать сортировку слиянием с использованием алгоритма ***S*1** при трудоемкости $O(n1 + n2)$. Поскольку при выполнении условия (3) часть массива M, соответствующая двум выделенным фрагментам, уже отсортирована, процедура сортировки двух фрагментов закончена. В этом случае не требуется дополнительной памяти, а время выполнения процедуры сортировки составляет величину $O(1)$.

Выполнение условия (4) соответствует случаю, когда после сортировки происходит перестановка всех элементов второго фрагмента в начало первого фрагмента, что характерно для алгоритмов сортировки делением по Пратту при циклической перестановке элементов фрагментов U_2 и V_1. Известный алгоритм циклической перестановки представляется в данном случае как вариант реализации с дополнительным анализом выполнения условий (2) — (4).

Если длины фрагментов равны ($n1 = n2$), то выполняется простейшая разовая перестановка соответствующих пар элементов, для которой требуется одна ячейка дополнительной памяти, а трудоемкость оценивается величиной $O(n1)$. Но, как правило, длины $n1$ и $n2$ не равны. Покажем, что и в этом случае алгоритм сортировки требует $O(1)$ дополнительной памяти, а трудоемкость его оценивается величиной $O(n1 + n2)$.

Алгоритм сортировки ***S*2**.

1. Проверить соотношения $n1 \le L$ или $n2 \le L$. Если хотя бы одно из них выполнено, то реализовать описанный выше модифицированный алгоритм сортировки и закончить процедуру сортировки, иначе перейти к п. 2.

2. Проверить соотношение между $n1$ и $n2$. Если $n1 = n2$, перейти к п. 3, если $n1 > n2$, перейти к п. 4, если $n1 < n2$, перейти к п. 5.

3. В цикле по j от 0 до $n1 - 1$ переставить местами элементы $M\,[z + j]$ и $M\,[z + n1 + j]$ и установить значение $n1 = n2 = 0$.

4. Реализовать циклическую перестановку элементов второго фрагмента в начало первого, которая может быть представлена на языке *C* следующим двойным циклом:

```
for(j1=z+n1+n2–1; j1>=z+n1; j1--)
{
  a=m[j1];
  for(j2=j1; j2>=z+n2; j2=j1–n2) m[j2]=m[j2–n2];
  m[j2]=a;
}
```

а также установить новые границы фрагментов, подлежащих сортировке:

$a = n2 \,\%\, n1$; $n2 = n1 - a$; $n1 = a$; $z = z + n1$; $n1 = n1 \,\%\, n2$; $n2 = n2 - n1$.

5. Реализовать циклическую перестановку элементов первого фрагмента в конец второго, которая может быть представлена на языке *C* следующим двойным циклом:

```
for(j1=z; j1<z+n1; j1++)
{
    a=m[j1];
    for(j2=j1; j2<z+n2; j2=j1+n1) m[j2]=m[j2+n2];
    m[j2]=a;
}
```

а также установить новые границы фрагментов, подлежащих сортировке:

$z = z + n1$; $n1 = n1 \,\%\, n2$; $n2 = n2 - n1$; .

6. Если $n1 = 0$ или $n2 = 0$, то завершить процедуру сортировки, иначе перейти к п. 1.

Согласно алгоритму слияния ***S*2** требуемая дополнительная память ограничивается одной ячейкой. Оценим его трудоемкость для наиболее сложного случая, когда процедура слияния завершается в п. 1. Тогда во всех случаях $n1 \neq n2$ и одна итерация может содержать шаги 1 – 2 – 4 – 6 – 1 либо 1 – 2 – 5 – 6 – 1. При этом суммарная трудоемкость выполнения операций на шагах 1, 2 и 6 составляет O (1). Следовательно, оценка трудоемкости одной итерации алгоритма от шага 1 до шага 1 практически совпадает с оценкой трудоемкости шага 4 или 5. Поскольку эти шаги симметричны (с переменой местами $n1$ и $n2$), достаточно оценить трудоемкость одного из них, например шага 4.

Оценим число операций присвоения для шага 4, которые определяют общую сложность одной итерации алгоритма. Для этого рассмотрим работу алгоритма на примере массива *M*, состоящего из двух фрагментов:

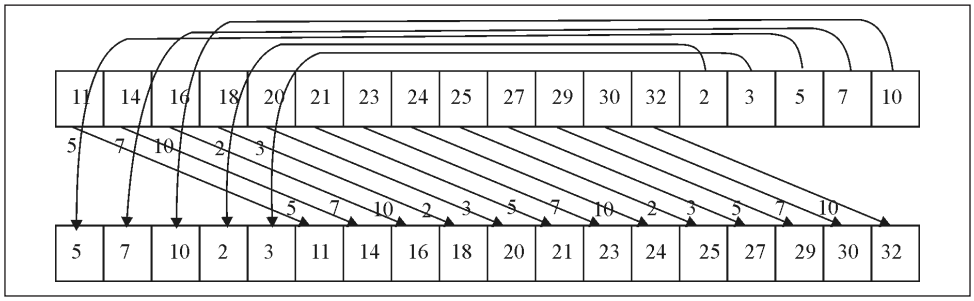

Рис. 2. Схема циклической перестановки элементов

M [18] = {11, 14, 16, 18, 20, 21, 23, 24, 25, 27, 29, 30, 32, 2, 3, 5, 7,10}, где z = = 0; $n1 = 13$; $n2 = 5$.

На рис. 2 представлена работа внутреннего цикла, осуществляющего циклическую перестановку элементов, и результаты перестановки всех элементов. Работе внутреннего цикла соответствует значение элемента в конце стрелок, для которого осуществляется циклическая перестановка. Так, на место элемента 10 становится элемент 32, на место элемента 32 — элемент 24, на место элемента 24 — элемент 16, а на место 16 — исходный элемент 10. В конце стрелок установлено значение 10.

Аналогично для элемента 3: на его место становится элемент 27, на место элемента 27 — элемент 20, на место элемента 20 — исходный элемент со значением 3. В конце стрелок установлено значение 3.

Следует заметить, что для приведенного примера в одних случаях на внутреннем цикле реализовывалось три перестановки, а в других — две. Это связано с жесткими границами фрагментов, за которые нельзя выходить. При этом число перестановок, равное трем, реализовано для числа элементов, равного остатку от деления 18 на 5 (значение 18 % 5). Число элементов, для которых выполнено всего две перестановки, определяется как разность между числом 5 (меньшая из длин фрагментов) и числом 18 % 5.

Как видим, однократное выполнение шага 4 (шага 5) приводит к тому, что упорядоченной оказывается часть массива, соответствующая большей из длин фрагментов. При этом оставшаяся неупорядоченная часть состоит также из двух фрагментов, для которых в случае $n1 > n2$ первый уменьшенный фрагмент имеет длину ($n1$ % $n2$) — остаток от деления $n1$ на $n2$, а второй — ($n2 - n1$ % $n2$); в случае $n1 < n2$ первый уменьшенный фрагмент имеет длину ($n2$ % $n1$), а второй — ($n1 - n2$ % $n1$). При этом в случае $n1 > n2$ начало первого из фрагментов совпадает с предыдущим его значением, а при $n1 < n2$ начало первого из фрагментов сдвигается вправо (увеличивается) на величину $n2$.

Анализ шагов 4 и 5 алгоритма ***S*2** свидетельствует о том, что при их однократном выполнении число операций присвоения равно $n1 + n2 + n0$, где $n0 = \min(n1, n2)$. Однако этим процесс сортировки не заканчивается. Он продолжается при все более уменьшающихся длинах фрагментов.

В результате однократной реализации шага 4 или 5 алгоритма длина оставшейся части массива, подлежащей сортировке, уменьшается до меньшей из длин фрагментов, т.е. более чем в два раза. Поэтому общее число операций присвоения может быть определено по формуле

$$3/2(n1+n2)(1+1/2+1/2^2+1/2^3+...)=3(n1+n2), \qquad (5)$$

т.е. составит величину $O(n1 + n2)$. Следует заметить, что некоторые вычисления, связанные с формированием данных для новой итерации, можно опустить, поскольку число итераций имеет порядок $O(\log_2(n1+n2))$. Поэтому общая оценка сложности алгоритма остается $O(n1+n2)$. Из (5) видно, что случай с обратной упорядоченностью исходного массива не является наиболее сложным для сортировки.

Теперь рассмотрим два фрагмента, для которых не выполнено ни одно из условий (2) — (4) и для сортировки пар фрагментов требуется их разбиение по Пратту. Нерекурсивный алгоритм такого однократного устойчивого деления учитывает естественную упорядоченность исходных фрагментов.

Устойчивый шаг деления пар фрагментов по Пратту с учетом естественной упорядоченности. Согласно идее Пратта о делении пар фрагментов длины $n1$ и $n2$ на две пары фрагментов меньшей длины процесс деления должен продолжаться до тех пор, пока не окажется возможной сортировка пары фрагментов, например, для длин фрагментов равных единице. При этом с добавлением нового уровня разбиений всех сформированных пар фрагментов число операций увеличивается на величину, пропорциональную суммарной длине $(n1 + n2)$. Поэтому целесообразно использовать такой алгоритм деления, при котором число уровней будет минимальным.

Очевидным является тот факт, что минимальное число уровней деления на фрагменты достигается в случае, когда на каждом новом уровне суммарная длина пары фрагментов уменьшается вдвое (медианное деление). Поэтому будем использовать именно такой способ деления. Однако этот способ не всегда является наиболее эффективным, поскольку существует ряд исключений, для которых деление по Пратту более эффективно, чем медианное.

В табл. 3 приведены два примера такого деления при числе ячеек дополнительной памяти $L = 1$. При формировании эффективного деления по Пратту из фрагмента меньшей длины выбирали элемент на расстоянии

L от границы фрагмента и определяли элемент большего из фрагментов так, чтобы вновь образованная пара фрагментов допускала сортировку без последующего дополнительного деления. В примере 1 во втором фрагменте выбран элемент 14 и в первом фрагменте найден первый элемент, больший, чем 14 (элемент 18). В примере 2 во втором фрагменте выбран элемент 7 и в первом фрагменте найден первый элемент, меньший, чем 7 (элемент 6). В обоих случаях получено такое деление на фрагменты, при котором для реализации сортировки не потребовалось дальнейшее разбиение на фрагменты меньшего размера. Поэтому в предлагаемом алгоритме ***D*1** реализуем деление по Пратту.

Пусть для определенности два фрагмента, расположенных в массиве M последовательно, начиная с ячейки z, имеют длины $n1$ и $n2$ и $n0 = (n1 + n2 + 1)/2$. Тогда для случая $n1 > n2$ алгоритм деления по Пратту следующий.

А л г о р и т м ***D*1**.

1. Если $M[z+n1+L-1] \geq M[z+n0-L-1]$, перейти к п. 2, иначе — к п. 5.

2. Если $M[z+n1+L-1] \geq M[z+n1-1]$, то сортировке подлежат только первый фрагмент и первые L элементов второго. Число элементов в первой паре фрагментов будет равно $n1$ и L, а во второй паре — 0 и $n2 - L$. Дальнейшего разбиения каждой пары полученных фрагментов не потребуется, поскольку выполняется условие (2).

3. Если $M[z+n1+L-1] < M[z+n1-1]$, то среди элементов первого фрагмента в диапазоне ячеек от $z+n0-L$ до $z+n1-1$ необходимо найти такую

Таблица 3

Пример	Первый фрагмент		Второй фрагмент
1	Исходные данные	1 2 3 4 5 6 7 8 10 18 19	14 15 16
	Медианное деление	1 2 3 4 5 6 7 8	**10 18 19** 14 15 16
	Эффективное деление по Пратту	1 2 3 4 6 7 8 10 18 **14**	**19** 15 16
2	Исходные данные	1 6 8 10 14 15 16 18 19	2 3 4 7
	Медианное деление	1 6 8 **10** 14 **2 3**	**15 16 18 19** 4 7
	Эффективное деление по Пратту	1 **2 3 4**	**6 8 2 14 15 16 18 19** 7

Примечание. Полужирным шрифтом указаны переставленные элементы.

ячейку $z1$, для которой $M[z+n1+L-1]< M[z1]$, но $M[z+n1+L-1]\geq M[z1-1]$, что реализуется бинарным поиском. Тогда число элементов в первой паре фрагментов будет равно $(z1-z)$ и L, а во второй паре — $[n1-(z1-z)]$ и $n2-L$.

4. Перейти к п. 10.

5. Если $M[z-1+n1+n2-L]< M[z-1+(n1-n0)+L]$, перейти к п. 6, иначе — к п. 8.

6. Если $M[z-1+n1+n2-L]< M[z]$, то сортировке подлежат только первый фрагмент и последние L элементов второго. Число элементов в первой паре фрагментов будет равно 0 и $n2-L$, а во второй паре — $n1$ и L. Дальнейшее разбиение каждой пары полученных фрагментов не потребуется, поскольку выполняется условие (2).

7. Если $M[z-1+n1+n2-L]\geq M[z]$, то среди элементов первого фрагмента в диапазоне ячеек от z до $z-1+n1-n0+L$ необходимо найти такую ячейку $z2$, в которой $M[z-1+n1+n2-L]\geq M[z2-1]$, но $M[z-1+n1+n2-L]< M[z2]$, что реализуется бинарным поиском. Тогда число элементов в первой паре фрагментов равно $(z2-z)$ и $n2-L$, а во второй паре — $[n1-(z2-z)]$ и L.

8. Перейти к п. 10.

9. В диапазоне ячеек второго фрагмента массива M от $L+1$ до $n2-L-1$, считая от начала фрагмента, с использованием алгоритма деления отрезка пополам найти ячейку $z3>L$ такую, в которой выполнены условия $M[z+n0-z3]> M[z-1+n1+z3]$, но $M[z-1+n0-z3]\leq M[z-1+n1+z3]$. Тогда число элементов для первой пары фрагментов равно $(n0-z3)$ и $z3$, а для второй пары — $n1-n0+z3$ и $(n2-z3)$.

10. Разбиение пары фрагментов на две пары меньшего размера завершено.

Варианты формирования пар фрагментов меньшего размера по алгоритму ***D*1** деления по Пратту представлены на рис. 3, где число элементов исходной пары фрагментов отображено длиной линии n. При $n1 \leq n2$ алгоритм ***D*1** деления по Пратту отображается симметрично.

Алгоритм S3 сортировки двух произвольных фрагментов. При оценке сложности алгоритмов сортировки слиянием в [1] показано, что наиболее сложным является случай сортировки двух фрагментов длины k вида $(a_1, a_2, ..., a_k, b_1, b_2, ..., b_k)$, где

$$b_1 < a_1 < b_2 < a_2 < ... < b_k < a_k. \qquad (6)$$

Пусть $Y0$ — массив данных о номере элемента массива M, с которого начинается первый из пары фрагментов соответствующего уровня, подлежащих слиянию, или – 1, если информация еще не определена; $Y1$ — массив данных о длине (числе элементов массива M) для первого фрагмента соответствующего уровня, или – 1, если информация еще не опреде-

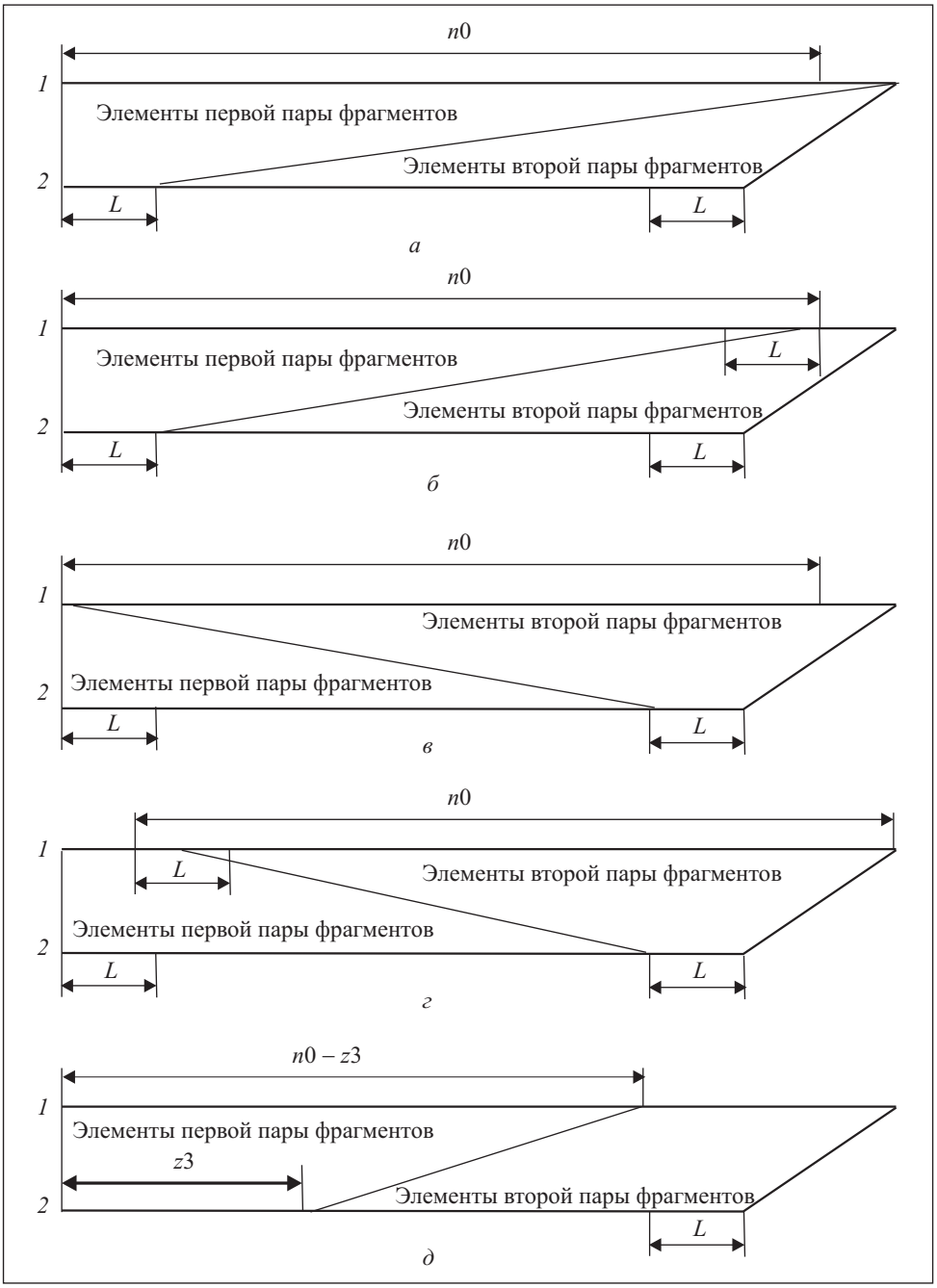

Рис. 3. Варианты деления пары фрагментов по алгоритму ***D*1** при $n1 > n2$: *а* — шаг 2; *б* — шаг 3; *в* — шаг 6; *г* — шаг 7; *д* — шаг 9; *1* и *2* — первый и второй фрагменты исходной пары

лена; $Y2$ — массив данных о длине (числе элементов массива M) для второго фрагмента соответствующего уровня, или – 1, если информация еще не определена; U — порядковый номер текущего уровня.

Алгоритм слияния ***S*3**.

0. Входные данные для работы алгоритма: номер $Y0[0]$ элемента массива M, с которого начинается первый из пары исходных фрагментов, подлежащих слиянию; $Y1[0]$ — длина первого фрагмента; $Y2[0]$ — длина второго фрагмента; номер уровня $U = 0$.

1. Для фрагментов на текущем уровне U проверяем выполнение условий (2)—(4). Если выполнено одно из них, реализуем слияние фрагментов и переходим к п. 2, иначе — к п. 3.

2. Увеличиваем номер элемента массива M, с которого начинается первый из следующей пары фрагментов на величину $Y1[U] + Y2[U]$. Определяем уровень, на который следует перейти. Если завершено упорядочивание первой пары медианного деления фрагментов более высокого уровня, то текущий уровень не меняется, если иначе, то номер уровня уменьшается до тех пор, пока не обнаружится уровень, на котором реализовано упорядочение элементов только для первой пары медианного деления. Если упорядочена вторая пара медианного деления для нулевого уровня, то — завершение процесса сортировки.

3. Ищем медианы двух фрагментов, разбиваем пару фрагментов уровня U на две пары фрагментов уровня $U + 1$ и для первой такой пары формируем данные в массивах $Y0[U + 1]$, $Y1[U + 1]$ и $Y2[U + 1]$. Увеличиваем уровень на 1. Возвращаемся к п. 1.

Следует заметить, что на шаге 2 номер пары медианного деления определяется на основании истинности условия

$$Y0[U-1]+Y1[U+1]+Y2[U+1]=Y0[U]+Y1[U]+Y2[U]. \qquad (7)$$

Если условие (7) выполнено, то обработана вторая пара, если иначе, то первая.

Алгоритм многократно повторяется. Максимальное число повторений для $n1 = n2 = 2^k$ при $L = 1$ в случае, если ни разу не выполнялось условие (3) или (4), равно $n1 - 1$. Число уровней никогда не превышает целой части $(\log_2 n + 1)$.

Рассмотрим работу описанного алгоритма на примере массива $M\ [15] = \{2, 6, 9, 11, 14, 16, 1, 3, 4, 7, 8, 10, 12, 15, 20\}$, где первый фрагмент имеет длину 6, а второй — 9.

В табл. 4 представлены шаги итераций. На одном шаге реализуется либо медианное деление, либо слияние двух фрагментов. Жирными ли-

Таблица 4

Массив	Шаг алгоритма													
M	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	2	2	2	2	1									
	6	6	1	1	2									
	9	9	3	3		3								
	11	1	4	4		4								
	14	3	6				6	6						
	16	4	9				7	7						
	1	7	7				9		8					
	3	8	8				8		9					
	4	11								11	11	10		
	7	14								14	10	11		
	8	16								10	14		12	
	10	10								12	12		14	
	12	12								16				15
	15	15								15				16
	20	20								20				20
$Y0_0$	0	0	0	0	0	0	0	0	0	0	0	0	0	15
$Y1_0$	6	8	8	8	8	8	8	8	8	8	8	8	8	0
$Y2_0$	9	7	7	7	7	7	7	7	7	7	7	7	7	0
$Y0_1$		0	0	0	0	0	0	0	8	8	8	8	8	15
$Y1_1$		3	4	4	4	4	4	4	0	4	4	4	4	0
$Y2_1$		5	4	4	4	4	4	4	0	3	3	3	3	0
$Y0_2$			0	0	0	4	4	4	8	8	8	8	12	15
$Y1_2$			2	2	2	0	2	2	0	2	2	2	1	0
$Y2_2$			2	2	2	0	2	2	0	2	2	2	2	0
$Y0_3$				0	2		4	6			8	10		
$Y1_3$				1	1		1	1			1	1		
$Y2_3$				1	1		1	1			1	1		

Примечание. В первом столбце нижний индекс указывает номер уровня.

ниями выделены формируемые фрагменты массива *M*. Заметим, что на шаге 14 не происходит деления по Пратту, поскольку минимальная длина фрагмента равна 1. При этом переход на более высокий уровень осуществляется на шагах 5 (2 + 1 + 1 = 0 + 2 + 2), 8 (6 + 1 + 1 = 4 + 2 + 2 = 0 + 4 + 4), 12 (8 + + 2 + 2 = 10 + 1 + 1) и 14 (12 + 1 + 2 = 8 + 4 + 3 = 0 + 8 + 7).

Данные табл. 4 подтверждают информацию о том, что в предлагаемом нерекурсивном алгоритме для организации процесса вычислений хватает объема дополнительной памяти в размере $6(\log_2(n1+n2)+1)$. Поэтому и для общего алгоритма справедливым остается вывод о том, что объединенный нерекурсивный алгоритм формирования пар фрагментов и их слияния требует объема дополнительной памяти порядка $O(\log_2 n)$ даже при условии, что и для массива $MF[L]$ выделяется память размером $O(\log_2 n)$.

Оценим общую трудоемкость объединенного алгоритма при $L = 1$. Как упомянуто выше, трудоемкость алгоритма в случае полностью упорядоченного массива длиной n оценивается величиной $O(n)$. Если рассматривается массив с обратной упорядоченностью, то трудоемкость алгоритма оценивается величиной $O(n\log n)$. Но в наиболее сложном случае для массива *M*, когда $n = 2^s$ и выполняется условие (6) для всех сортируемых фрагментов произвольной длины, трудоемкость оценивается величиной $O(n(\log_2 n)^2)$.

Действительно, в рассматриваемом случае для массива *M* при каждом медианном разбиении пар фрагментов переставляется местами половина элементов обоих фрагментов. Для всех пар фрагментов длины 2 требуется одно медианное разбиение и общая трудоемкость оценивается величиной $O(n)$. Для всех пар фрагментов длины 4 требуется два медианных разбиения и почти в два раза возрастает число операций при сортировке всех таких фрагментов $n0$ сравнению с фрагментами длины 2. Для пар фрагментов длины 8 требуется три медианных разбиения и примерно в три раза возрастает число операций при сортировке всех таких фрагментов по сравнению с фрагментами длины 2.

Для пар фрагментов нулевого уровня требуется $\log_2 n - 2$ медианных разбиений пар фрагментов длины $n/2$ и приблизительно в $\log_2 n - 2$ раза возрастает число операций при сортировке всех таких фрагментов по сравнению с фрагментами длины 2. Суммируя трудоемкость по всем уровням, получаем оценку общего алгоритма слияния всех фрагментов без использования дополнительной памяти сверх величины $O(\log_2 n)$, при $L = 1$ равную $O(n(\log_2 n)^2)$. Следует заметить, что на начальном этапе выделения упорядоченных фрагментов длины, не менее 2, и формирования пар фрагментов трудоемкость составляет $O(n)$. Поэтому трудоемкость общего алгоритма $\boldsymbol{A1} + \boldsymbol{S2} + \boldsymbol{S3}$ оценивается величиной $O(n(\log_2 n)^2)$.

Выводы

Предложенный модифицированный алгоритм слияния двух упорядоченных фрагментов позволяет ослабить требование к объему выделяемой дополнительной памяти до меньшей из длин упорядоченных частей элементов массива данных. Теоретически доказано, что использование алгоритма сортировки слиянием при сортировке массивов длины n, даже с учетом упорядоченности исходного массива, снижает требование к объему выделяемой дополнительной памяти до величины $n/2 + O(1)$. А при использовании одно- или двукратного деления по Пратту можно сократить требование к объему выделяемой дополнительной памяти до величины $n/4 + O(1)$, $n/8 + O(1)$ при трудоемкости в худшем случае порядка $O(n\log n)$.

Наибольшую экономию объема выделяемой дополнительной памяти обеспечивает последний из предложенных нерекурсивных алгоритмов устойчивой естественной сортировки слиянием. В нем выполняется один последовательный проход по массиву M, в ходе которого формируются пары упорядоченных фрагментов и реализуется их слияние. В этом алгоритме для формирования пар фрагментов требуется выделение объема дополнительной памяти $3\log_2 n + O(1)$. Сортировка двух фрагментов в худшем случае требует выделения такого же объема дополнительной памяти. Следовательно, общий объем дополнительной памяти составляет величину $O(\log_2 n)$, а сложность алгоритма для худшего случая оценивается величиной $O(n(\log_2 n)^2)$. Этот алгоритм более эффективен относительно требуемого объема выделяемой памяти по сравнению с известными алгоритмами слияния, основанными на рекурсивном применении идеи Пратта деления пар фрагментов на две пары меньшего размера, где при такой же трудоемкости минимально необходимый объем требуемой дополнительной памяти оценивается величиной $O(\log^2 n)$ стековой памяти.

An algorithm has been proposed, which requires additional memory of the order $O(\log n)$ with an estimate of the labour input in the worst case of order $O(n\log^2 n)$. Variants of resistant nonrecursive merge sorting algorithms have been proposed that take into account the natural ordering of the original data array of length n, with a decrease in the volume of additional memory to the values of $n/2 + O(1)$, $n/4 + O(1)$, $n/8 + O(1)$, and labour input in the worst case of elements location of the order $O(n\log n)$.

1. *Кнут Д. Э.* Искусство программирования. Т. 3. Сортировка и поиск. 2–е изд.: Пер с англ. — М. : Изд. дом «Вильямс», 2003. — 832 с.
2. *http:*//ru.wikipedia.org/wiki/%D0%A3%D1%81%D1%82%D0%BE%D0%B9%D1%87%D0%B8%D0%B2%D0%B0%D1%8F_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0

3. *Рабочий* алгоритм на C++ внешней сортировки «естественным слиянием» (Natural Merge Sort) с неубывающими и убывающими упорядоченными подпоследовательностями. — http://emery-emerald.narod.ru. — Январь 2010. — http://sql.ru/forum/actualthread.aspx?tid=727034.

Поступила 29.04.11

ВИННИЧУК Степан Дмитриевич, д-р техн. наук, вед. науч. сотр. Ин-та проблем моделирования в энергетике им. Г. Е. Пухова НАН Украины. В 1977 г. окончил Черновицкий госуниверситет. Область научных исследований — разработка методов, моделей и программных средств для анализа распределительных систем сжимаемой и несжимаемой жидкостей, авиационные системы кондиционирования воздуха; системная противоаварийная частотная автоматика электроэнергетических систем.