**2.1. Сортировка включением**

Одним из наиболее простых и естественных методов внутренней сортировки является сортировка с простыми включениями. Идея алгоритма очень проста. Пусть имеется массив ключей a[1], a[2], ..., a[n]. Для каждого элемента массива, начиная со второго, производится сравнение с элементами с меньшим индексом (элемент a[i] последовательно сравнивается с элементами a[i-1], a[i-2] ...) и до тех пор, пока для очередного элемента a[j] выполняется соотношение a[j] > a[i], a[i] и a[j] меняются местами. Если удается встретить такой элемент a[j], что a[j] <= a[i], или если достигнута нижняя граница массива, производится переход к обработке элемента a[i+1] (пока не будет достигнута верхняя граница массива).

Легко видеть, что в лучшем случае (когда массив уже упорядочен) для выполнения алгоритма с массивом из n элементов потребуется n-1 сравнение и 0 пересылок. В худшем случае (когда массив упорядочен в обратном порядке) потребуется n?(n-1)/2 сравнений и столько же пересылок. Таким образом, можно оценивать сложность метода простых включений как O(n2).

Можно сократить число сравнений, применяемых в методе простых включений, если воспользоваться тем фактом, что при обработке элемента a[i] массива элементы a[1], a[2], ..., a[i-1] уже упорядочены, и воспользоваться для поиска элемента, с которым должна быть произведена перестановка, методом двоичного деления. В этом случае оценка числа требуемых сравнений становится O(n?log n). Заметим, что поскольку при выполнении перестановки требуется сдвижка на один элемент нескольких элементов, то оценка числа пересылок остается O(n2).

|  |  |
| --- | --- |
| Таблица 2.1 Пример сортировки методом простого включения | |
| **Начальное состояние массива** | 8 23 5 65 44 33 1 6 |
| **Шаг 1** | 8 23 5 65 44 33 1 6 |
| **Шаг 2** | 8 5 23 65 44 33 1 6  5 8 23 65 44 33 1 6 |
| **Шаг 3** | 5 8 23 65 44 33 1 6 |
| **Шаг 4** | 5 8 23 44 65 33 1 6 |
| **Шаг 5** | 5 8 23 44 33 65 1 6  5 8 23 33 44 65 1 6 |
| **Шаг 6** | 5 8 23 33 44 1 65 6  5 8 23 33 1 44 65 6  5 8 23 1 33 44 65 6  5 8 1 23 33 44 65 6  5 1 8 23 33 44 65 6  1 5 8 23 33 44 65 6 |
| **Шаг 7** | 1 5 8 23 33 44 6 65  1 5 8 23 33 6 44 65  1 5 8 23 6 33 44 65  1 5 8 6 23 33 44 65  1 5 6 8 23 33 44 65 |

Дальнейшим развитием метода сортировки с включениями является сортировка методом Шелла, называемая по-другому сортировкой включениями с уменьшающимся расстоянием. Мы не будем описывать алгоритм в общем виде, а ограничимся случаем, когда число элементов в сортируемом массиве является степенью числа 2. Для массива с 2n элементами алгоритм работает следующим образом. На первой фазе производится сортировка включением всех пар элементов массива, расстояние между которыми есть 2(n-1). На второй фазе производится сортировка включением элементов полученного массива, расстояние между которыми есть 2(n-2). И так далее, пока мы не дойдем до фазы с расстоянием между элементами, равным единице, и не выполним завершающую сортировку с включениями. Применение метода Шелла к массиву, используемому в наших примерах, показано в таблице 2.2.

*Таблица 2.2. Пример сортировки методом Шелл*

|  |  |
| --- | --- |
| **Начальное состояние массива** | 8 23 5 65 44 33 1 6 |
| **Фаза 1 (сортируются элементы, расстояние между которыми четыре)** | 8 23 5 65 44 33 1 6  8 23 5 65 44 33 1 6  8 23 1 65 44 33 5 6  8 23 1 6 44 33 5 65 |
| **Фаза 2 (сортируются элементы, расстояние между которыми два)** | 1 23 8 6 44 33 5 65  1 23 8 6 44 33 5 65  1 23 8 6 5 33 44 65  1 23 5 6 8 33 44 65  1 6 5 23 8 33 44 65  1 6 5 23 8 33 44 65  1 6 5 23 8 33 44 65 |
| **Фаза 3 (сортируются элементы, расстояние между которыми один)** | 1 6 5 23 8 33 44 65  1 5 6 23 8 33 44 65  1 5 6 23 8 33 44 65  1 5 6 8 23 33 44 65  1 5 6 8 23 33 44 65  1 5 6 8 23 33 44 65  1 5 6 8 23 33 44 65 |

В общем случае алгоритм Шелла естественно переформулируется для заданной последовательности из t расстояний между элементами h1, h2, ..., ht, для которых выполняются условия h1 = 1 и h(i+1) < hi. Дональд Кнут показал, что при правильно подобранных t и h сложность алгоритма Шелла является O(n(1.2)), что существенно меньше сложности простых алгоритмов сортировки.

**2.2. Обменная сортировка**

Простая обменная сортировка (в просторечии называемая "методом пузырька") для массива a[1], a[2], ..., a[n] работает следующим образом. Начиная с конца массива сравниваются два соседних элемента (a[n] и a[n-1]). Если выполняется условие a[n-1] > a[n], то значения элементов меняются местами. Процесс продолжается для a[n-1] и a[n-2] и т.д., пока не будет произведено сравнение a[2] и a[1]. Понятно, что после этого на месте a[1] окажется элемент массива с наименьшим значением. На втором шаге процесс повторяется, но последними сравниваются a[3] и a[2]. И так далее. На последнем шаге будут сравниваться только текущие значения a[n] и a[n-1]. Понятна аналогия с пузырьком, поскольку наименьшие элементы (самые "легкие") постепенно "всплывают" к верхней границе массива. Пример сортировки методом пузырька показан в таблице 2.3.

*Таблица 2.3. Пример сортировки методом пузырька*

|  |  |
| --- | --- |
| **Начальное состояние массива** | 8 23 5 65 44 33 1 6 |
| **Шаг 1** | 8 23 5 65 44 33 1 6  8 23 5 65 44 1 33 6  8 23 5 65 1 44 33 6  8 23 5 1 65 44 33 6  8 23 1 5 65 44 33 6  8 1 23 5 65 44 33 6  1 8 23 5 65 44 33 6 |
| **Шаг 2** | 1 8 23 5 65 44 6 33  1 8 23 5 65 6 44 33  1 8 23 5 6 65 44 33  1 8 23 5 6 65 44 33  1 8 5 23 6 65 44 33  1 5 8 23 6 65 44 33 |
| **Шаг 3** | 1 5 8 23 6 65 33 44  1 5 8 23 6 33 65 44  1 5 8 23 6 33 65 44  1 5 8 6 23 33 65 44  1 5 6 8 23 33 65 44 |
| **Шаг 4** | 1 5 6 8 23 33 44 65  1 5 6 8 23 33 44 65  1 5 6 8 23 33 44 65  1 5 6 8 23 33 44 65 |
| **Шаг 5** | 1 5 6 8 23 33 44 65  1 5 6 8 23 33 44 65  1 5 6 8 23 33 44 65 |
| **Шаг 6** | 1 5 6 8 23 33 44 65  1 5 6 8 23 33 44 65 |
| **Шаг 7** | 1 5 6 8 23 33 44 65 |

Для метода простой обменной сортировки требуется число сравнений nx(n-1)/2, минимальное число пересылок 0, а среднее и максимальное число пересылок - O(n2).

Метод пузырька допускает три простых усовершенствования. Во-первых, как показывает таблица 2.3, на четырех последних шагах расположение значений элементов не менялось (массив оказался уже упорядоченным). Поэтому, если на некотором шаге не было произведено ни одного обмена, то выполнение алгоритма можно прекращать. Во-вторых, можно запоминать наименьшее значение индекса массива, для которого на текущем шаге выполнялись перестановки. Очевидно, что верхняя часть массива до элемента с этим индексом уже отсортирована, и на следующем шаге можно прекращать сравнения значений соседних элементов при достижении такого значения индекса. В-третьих, метод пузырька работает неравноправно для "легких" и "тяжелых" значений. Легкое значение попадает на нужное место за один шаг, а тяжелое на каждом шаге опускается по направлению к нужному месту на одну позицию.

На этих наблюдениях основан метод шейкерной сортировки (ShakerSort). При его применении на каждом следующем шаге меняется направление последовательного просмотра. В результате на одном шаге "всплывает" очередной наиболее легкий элемент, а на другом "тонет" очередной самый тяжелый. Пример шейкерной сортировки приведен в таблице 2.4.

*Таблица 2.4. Пример шейкерной сортировки*

|  |  |
| --- | --- |
| **Начальное состояние массива** | 8 23 5 65 44 33 1 6 |
| **Шаг 1** | 8 23 5 65 44 33 1 6  8 23 5 65 44 1 33 6  8 23 5 65 1 44 33 6  8 23 5 1 65 44 33 6  8 23 1 5 65 44 33 6  8 1 23 5 65 44 33 6  1 8 23 5 65 44 33 6 |
| **Шаг 2** | 1 8 23 5 65 44 33 6  1 8 5 23 65 44 33 6  1 8 5 23 65 44 33 6  1 8 5 23 44 65 33 6  1 8 5 23 44 33 65 6  1 8 5 23 44 33 6 65 |
| **Шаг 3** | 1 8 5 23 44 6 33 65  1 8 5 23 6 44 33 65  1 8 5 6 23 44 33 65  1 8 5 6 23 44 33 65  1 5 8 6 23 44 33 65 |
| **Шаг 4** | 1 5 6 8 23 44 33 65  1 5 6 8 23 44 33 65  1 5 6 8 23 44 33 65  1 5 6 8 23 33 44 65 |
| **Шаг 5** | 1 5 6 8 23 33 44 65  1 5 6 8 23 33 44 65  1 5 6 8 23 33 44 65 |

Шейкерная сортировка позволяет сократить число сравнений (по оценке Кнута средним числом сравнений является (n2 - n?(const + ln n)), хотя порядком оценки по-прежнему остается n2. Число же пересылок, вообще говоря, не меняется. Шейкерную сортировку рекомендуется использовать в тех случаях, когда известно, что массив "почти упорядочен".

**2.3. Сортировка выбором**

При сортировке массива a[1], a[2], ..., a[n] методом простого выбора среди всех элементов находится элемент с наименьшим значением a[i], и a[1] и a[i] обмениваются значениями. Затем этот процесс повторяется для получаемых подмассивов a[2], a[3], ..., a[n], ... a[j], a[j+1], ..., a[n] до тех пор, пока мы не дойдем до подмассива a[n], содержащего к этому моменту наибольшее значение. Работа алгоритма иллюстрируется примером в таблице 2.5.

*Таблица 2.5. Пример сортировки простым выбором*

|  |  |
| --- | --- |
| **Начальное состояние массива** | 8 23 5 65 44 33 1 6 |
| **Шаг 1** | 1 23 5 65 44 33 8 6 |
| **Шаг 2** | 1 5 23 65 44 33 8 6 |
| **Шаг 3** | 1 5 6 65 44 33 8 23 |
| **Шаг 4** | 1 5 6 8 44 33 65 23 |
| **Шаг 5** | 1 5 6 8 33 44 65 23 |
| **Шаг 6** | 1 5 6 8 23 44 65 33 |
| **Шаг 7** | 1 5 6 8 23 33 65 44 |
| **Шаг 8** | 1 5 6 8 23 33 44 65 |

Для метода сортировки простым выбором требуемое число сравнений - nx(n-1)/2. Порядок требуемого числа пересылок (включая те, которые требуются для выбора минимального элемента) в худшем случае составляет O(n2). Однако порядок среднего числа пересылок есть O(n?ln n), что в ряде случаев делает этот метод предпочтительным.

**2.4. Сортировка разделением (Quicksort)**

Метод сортировки разделением был предложен Чарльзом Хоаром (он любит называть себя Тони) в 1962 г. Этот метод является развитием метода простого обмена и настолько эффективен, что его стали называть "методом быстрой сортировки - Quicksort".

Основная идея алгоритма состоит в том, что случайным образом выбирается некоторый элемент массива x, после чего массив просматривается слева, пока не встретится элемент a[i] такой, что a[i] > x, а затем массив просматривается справа, пока не встретится элемент a[j] такой, что a[j] < x. Эти два элемента меняются местами, и процесс просмотра, сравнения и обмена продолжается, пока мы не дойдем до элемента x. В результате массив окажется разбитым на две части - левую, в которой значения ключей будут меньше x, и правую со значениями ключей, большими x. Далее процесс рекурсивно продолжается для левой и правой частей массива до тех пор, пока каждая часть не будет содержать в точности один элемент. Понятно, что как обычно, рекурсию можно заменить итерациями, если запоминать соответствующие индексы массива. Проследим этот процесс на примере нашего стандартного массива (таблица 2.6).

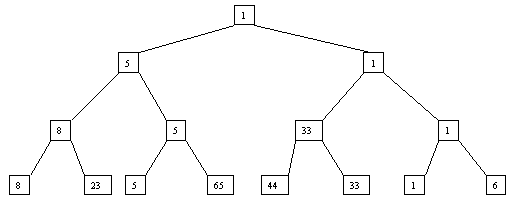
*Таблица 2.6. Пример быстрой сортировки*

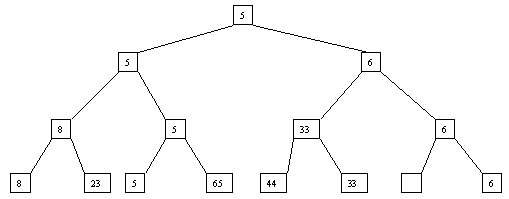
|  |  |
| --- | --- |
| **Начальное состояние массива** | 8 23 5 65 |44| 33 1 6 |
| **Шаг 1 (в качестве x выбирается a[5])** | |--------|  8 23 5 6 44 33 1 65  |---|  8 23 5 6 1 33 44 65 |
| **Шаг 2 (в подмассиве a[1], a[5] в качестве x выбирается a[3])** | 8 23 |5| 6 1 33 44 65  |--------|  1 23 5 6 8 33 44 65  |--|  1 5 23 6 8 33 44 65 |
| **Шаг 3 (в подмассиве a[3], a[5] в качестве x выбирается a[4])** | 1 5 23 |6| 8 33 44 65  |----|  1 5 8 6 23 33 44 65 |
| **Шаг 4 (в подмассиве a[3], a[4] выбирается a[4])** | 1 5 8 |6| 23 33 44 65  |--|  1 5 6 8 23 33 44 65 |

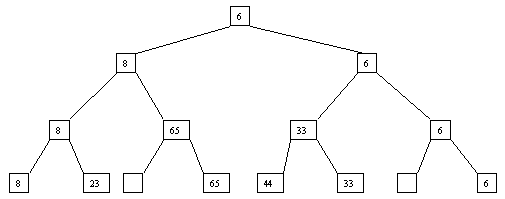
Алгоритм недаром называется быстрой сортировкой, поскольку для него оценкой числа сравнений и обменов является O(n?log n). На самом деле, в большинстве утилит, выполняющих сортировку массивов, используется именно этот алгоритм.

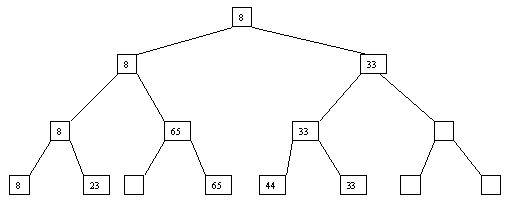
**2.5. Сортировка с помощью дерева (Heapsort)**

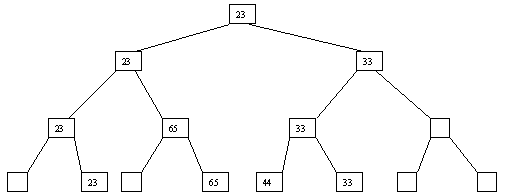
Начнем с простого метода сортировки с помощью дерева, при использовании которого явно строится двоичное дерево сравнения ключей. Построение дерева начинается с листьев, которые содержат все элементы массива. Из каждой соседней пары выбирается наименьший элемент, и эти элементы образуют следующий (ближе к корню уровень дерева). Из каждой соседней пары выбирается наименьший элемент и т.д., пока не будет построен корень, содержащий наименьший элемент массива. Двоичное дерево сравнения для массива, используемого в наших примерах, показано на рисунке 2.1. Итак, мы уже имеем наименьшее значение элементов массива. Для того, чтобы получить следующий по величине элемент, спустимся от корня по пути, ведущему к листу с наименьшим значением. В этой листовой вершине проставляется фиктивный ключ с "бесконечно большим" значением, а во все промежуточные узлы, занимавшиеся наименьшим элементом, заносится наименьшее значение из узлов - непосредственных потомков (рис. 2.2). Процесс продолжается до тех пор, пока все узлы дерева не будут заполнены фиктивными ключами (рисунки 2.3 - 2.8).

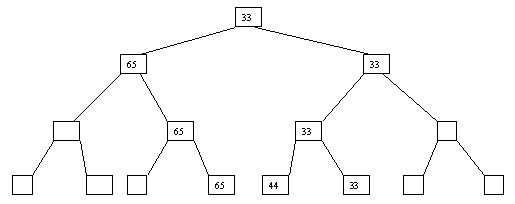
  
*Рис. 2.1.*

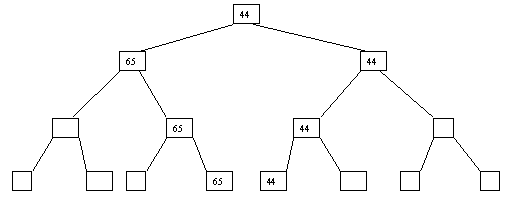
  
*Рис. 2.2. Второй шаг*

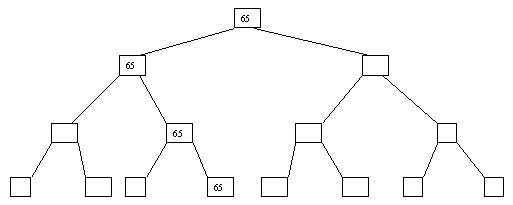
  
*Рис. 2.3. Третий шаг*

  
*Рис. 2.4. четвертый шаг*

  
*Рис. 2.5. Пятый шаг*

  
*Рис. 2.6. Шестой шаг*

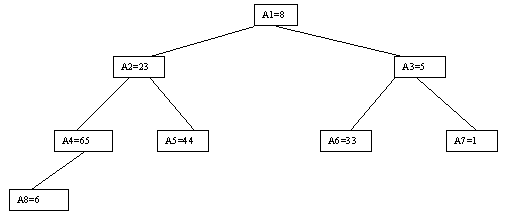
  
*Рис. 2.7. Седьмой шаг*

  
*Рис. 2.8. Восьмой шаг*

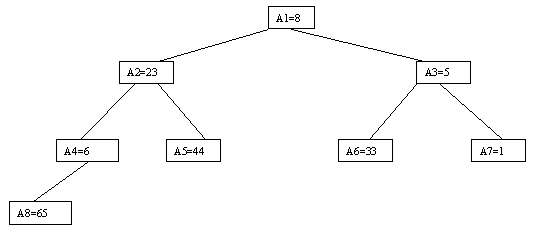
На каждом из n шагов, требуемых для сортировки массива, нужно log n (двоичный) сравнений. Следовательно, всего потребуется n?log n сравнений, но для представления дерева понадобится 2n - 1 дополнительных единиц памяти.

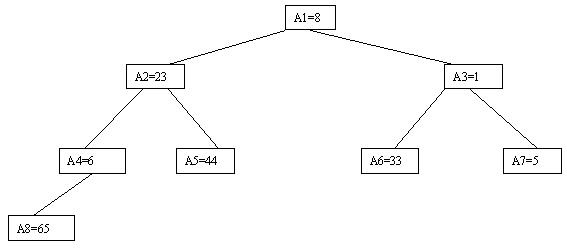
Имеется более совершенный алгоритм, который принято называть пирамидальной сортировкой (Heapsort). Его идея состоит в том, что вместо полного дерева сравнения исходный массив a[1], a[2], ..., a[n] преобразуется в пирамиду, обладающую тем свойством, что для каждого a[i] выполняются условия a[i] <= a[2i] и a[i] <= a[2i+1]. Затем пирамида используется для сортировки.

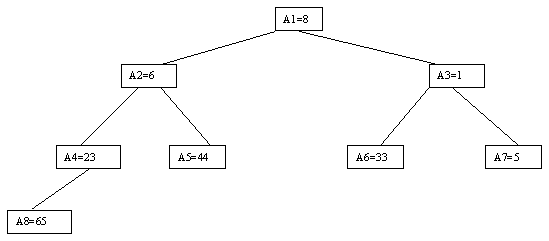
Наиболее наглядно метод построения пирамиды выглядит при древовидном представлении массива, показанном на рисунке 2.9. Массив представляется в виде двоичного дерева, корень которого соответствует элементу массива a[1]. На втором ярусе находятся элементы a[2] и a[3]. На третьем - a[4], a[5], a[6], a[7] и т.д. Как видно, для массива с нечетным количеством элементов соответствующее дерево будет сбалансированным, а для массива с четным количеством элементов n элемент a[n] будет единственным (самым левым) листом "почти" сбалансированного дерева.

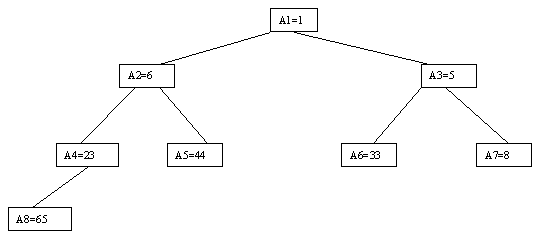
  
*Рис. 2.9.*

Очевидно, что при построении пирамиды нас будут интересовать элементы a[n/2], a[n/2-1], ..., a[1] для массивов с четным числом элементов и элементы a[(n-1)/2], a[(n-1)/2-1], ..., a[1] для массивов с нечетным числом элементов (поскольку только для таких элементов существенны ограничения пирамиды). Пусть i - наибольший индекс из числа индексов элементов, для которых существенны ограничения пирамиды. Тогда берется элемент a[i] в построенном дереве и для него выполняется процедура просеивания, состоящая в том, что выбирается ветвь дерева, соответствующая min(a[2?i], a[2?i+1]), и значение a[i] меняется местами со значением соответствующего элемента. Если этот элемент не является листом дерева, для него выполняется аналогичная процедура и т.д. Такие действия выполняются последовательно для a[i], a[i-1], ..., a[1]. Легко видеть, что в результате мы получим древовидное представление пирамиды для исходного массива (последовательность шагов для используемого в наших примерах массива показана на рисунках 2.10-2.13).

  
*Рис. 2.10.*

  
*Рис. 2.11.*

  
*Рис. 2.12.*

  
*Рис. 2.13.*

В 1964 г. Флойд предложил метод построения пирамиды без явного построения дерева (хотя метод основан на тех же идеях). Построение пирамиды методом Флойда для нашего стандартного массива показано в таблице 2.7.

*Таблица 2.7 Пример построения пирамиды*

|  |  |
| --- | --- |
| **Начальное состояние массива** | 8 23 5 |65| 44 33 1 6 |
| **Шаг 1** | 8 23 |5| 6 44 33 1 65 |
| **Шаг 2** | 8 |23| 1 6 44 33 5 65 |
| **Шаг 3** | |8| 6 1 23 44 33 5 65 |
| **Шаг 4** | 1 6 8 23 44 33 5 65  1 6 5 23 44 33 8 65 |

В таблице 2.8 показано, как производится сортировка с использованием построенной пирамиды. Суть алгоритма заключается в следующем. Пусть i - наибольший индекс массива, для которого существенны условия пирамиды. Тогда начиная с a[1] до a[i] выполняются следующие действия. На каждом шаге выбирается последний элемент пирамиды (в нашем случае первым будет выбран элемент a[8]). Его значение меняется со значением a[1], после чего для a[1] выполняется просеивание. При этом на каждом шаге число элементов в пирамиде уменьшается на 1 (после первого шага в качестве элементов пирамиды рассматриваются a[1], a[2], ..., a[n-1]; после второго - a[1], a[2], ..., a[n-2] и т.д., пока в пирамиде не останется один элемент). Легко видеть (это иллюстрируется в таблице 2.8), что в результате мы получим массив, упорядоченный в порядке убывания. Можно модифицировать метод построения пирамиды и сортировки, чтобы получить упорядочение в порядке возрастания, если изменить условие пирамиды на a[i] >= a[2?i] и a[1] >= a[2?i+1] для всех осмысленных значений индекса i.

*Таблица 2.8 Сортировка с помощью пирамиды*

|  |  |
| --- | --- |
| **Исходная пирамида** | 1 6 5 23 44 33 8 65 |
| **Шаг 1** | 65 6 5 23 44 33 8 1  5 6 65 23 44 33 8 1  5 6 8 23 44 33 65 1 |
| **Шаг 2** | 65 6 8 23 44 33 5 1  6 65 8 23 44 33 5 1  6 23 8 65 44 33 5 1 |
| **Шаг 3** | 33 23 8 65 44 6 5 1  8 23 33 65 44 6 5 1 |
| **Шаг 4** | 44 23 33 65 8 6 5 1  23 44 33 65 8 6 5 1 |
| **Шаг 5** | 65 44 33 23 8 6 5 1  33 44 65 23 8 6 5 1 |
| **Шаг 6** | 65 44 33 23 8 6 5 1  44 65 33 23 8 6 5 1 |
| **Шаг 7** | 65 44 33 23 8 6 5 1 |

Процедура сортировки с использованием пирамиды требует выполнения порядка nxlog n шагов (логарифм - двоичный) в худшем случае, что делает ее особо привлекательной для сортировки больших массивов.

**2.6. Сортировка со слиянием**

Сортировки со слиянием, как правило, применяются в тех случаях, когда требуется отсортировать последовательный файл, не помещающийся целиком в основной памяти. Методам внешней сортировки посвящается следующая часть книги, в которой основное внимание будет уделяться методам минимизации числа обменов с внешней памятью. Однако существуют и эффективные методы внутренней сортировки, основанные на разбиениях и слияниях.

Один из популярных алгоритмов внутренней сортировки со слияниями основан на следующих идеях (для простоты будем считать, что число элементов в массиве, как и в нашем примере, является степенью числа 2). Сначала поясним, что такое слияние. Пусть имеются два отсортированных в порядке возрастания массива p[1], p[2], ..., p[n] и q[1], q[2], ..., q[n] и имеется пустой массив r[1], r[2], ..., r[2?n], который мы хотим заполнить значениями массивов p и q в порядке возрастания. Для слияния выполняются следующие действия: сравниваются p[1] и q[1], и меньшее из значений записывается в r[1]. Предположим, что это значение p[1]. Тогда p[2] сравнивается с q[1] и меньшее из значений заносится в r[2]. Предположим, что это значение q[1]. Тогда на следующем шаге сравниваются значения p[2] и q[2] и т.д., пока мы не достигнем границ одного из массивов. Тогда остаток другого массива просто дописывается в "хвост" массива r.

Пример слияния двух массивов показан на рисунке 2.14.

  
*Рис. 2.14.*

Для сортировки со слиянием массива a[1], a[2], ..., a[n] заводится парный массив b[1], b[2], ..., b[n]. На первом шаге производится слияние a[1] и a[n] с размещением результата в b[1], b[2], слияние a[2] и a[n-1] с размещением результата в b[3], b[4], ..., слияние a[n/2] и a[n/2+1] с помещением результата в b[n-1], b[n]. На втором шаге производится слияние пар b[1], b[2] и b[n-1], b[n] с помещением результата в a[1], a[2], a[3], a[4], слияние пар b[3], b[4] и b[n-3], b[n-2] с помещением результата в a[5], a[6], a[7], a[8], ..., слияние пар b[n/2-1], b[n/2] и b[n/2+1], b[n/2+2] с помещением результата в a[n-3], a[n-2], a[n-1], a[n]. И т.д. На последнем шаге, например (в зависимости от значения n), производится слияние последовательностей элементов массива длиной n/2 a[1], a[2], ..., a[n/2] и a[n/2+1], a[n/2+2], ..., a[n] с помещением результата в b[1], b[2], ..., b[n].

Для случая массива, используемого в наших примерах, последовательность шагов показана в таблице 2.9.

*Таблица 2.9. Пример сортировки со слиянием*

|  |  |
| --- | --- |
| **Начальное состояние массива** | 8 23 5 65 44 33 1 6 |
| **Шаг 1** | 6 8 1 23 5 33 44 65 |
| **Шаг 2** | 6 8 44 65 1 5 23 33 |
| **Шаг 3** | 1 5 6 8 23 33 44 65 |

При применении сортировки со слиянием число сравнений ключей и число пересылок оценивается как O(n?log n). Но следует учитывать, что для выполнения алгоритма для сортировки массива размера n требуется 2?n элементов памяти.

**2.7. Сравнение методов внутренней сортировки**

Для рассмотренных в начале этой части простых методов сортировки существуют точные формулы, вычисление которых дает минимальное, максимальное и среднее число сравнений ключей (C) и пересылок элементов массива (M). Таблица 2.10 содержит данные, приводимые в книге Никласа Вирта.

*Таблица 2.10. Характеристики простых методов сортировки*

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Min** | **Avg** | **Max** |
| **Прямое включение** | C = n-1  M = 2x(n-1) | (n2 + n - 2)/4  (n2 - 9n - 10)/4 | (n2 -n)/2 - 1  (n2 -3n - 4)/2 |
| **Прямой выбор** | C = (n2 - n)/2  M = 3x(n-1) | (n2 - n)/2  nx(ln n + 0.57) | (n2 - n)/2  n2/4 + 3x(n-1) |
| **Прямой обмен** | C = (n2 - n)/2  M = 0 | (n2 - n)/2  (n2 - n)x0.75 | (n2 - n)/2  (n2 - n)x1.5 |

Для оценок сложности усовершенствованных методов сортировки точных формул нет. Известно лишь, что для сортировки методом Шелла порядок C и M есть O(n(1.2)), а для методов Quicksort, Heapsort и сортировки со слиянием - O(n?log n). Однако результаты экспериментов показывают, что Quicksort показывает результаты в 2-3 раза лучшие, чем Heapsort (в таблице 2.11 приводится выборка результатов из таблицы, опубликованной в книге Вирта; результаты получены при прогоне программ, написанных на языке Модула-2). Видимо, по этой причине именно Quicksort обычно используется в стандартных утилитах сортировки (в частности, в утилите sort, поставляемой с операционной системой UNIX).

*Таблица 2.11. Время работы программ сортировки*

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Упорядоченный массив** | **Случайный массив** | **В обратном порядке** |
|  | n = 256 | | |
| **Heapsort Quicksort Сортировка со слиянием** | 0.20  0.20  0.20 | 0.08  0.12  0.08 | 0.18  0.18  0.18 |
|  | n = 2048 | | |
| **Heapsort Quicksort Сортировка со слиянием** | 2.32  2.22  2.12 | 0.72  1.22  0.76 | 1.98  2.06  1.98 |

**3. Методы внешней сортировки**

Принято называть "внешней" сортировкой сортировку последовательных файлов, располагающихся во внешней памяти и слишком больших, чтобы можно было целиком переместить их в основную память и применить один из рассмотренных в предыдущем разделе методов внутренней сортировки. Наиболее часто внешняя сортировка применяется в системах управления базами данных при выполнении запросов, и от эффективности применяемых методов существенно зависит производительность СУБД.

Следует пояснить, почему речь идет именно о последовательных файлах, т.е. о файлах, которые можно читать запись за записью в последовательном режиме, а писать можно только после последней записи. Методы внешней сортировки появились, когда наиболее распространенными устройствами внешней памяти были магнитные ленты. Для лент чисто последовательный доступ был абсолютно естественным. Когда произошел переход к запоминающим устройствам с магнитными дисками, обеспечивающими "прямой" доступ к любому блоку информации, казалось, что чисто последовательные файлы потеряли свою актуальность. Однако это ощущение было ошибочным.

Все дело в том, что практически все используемые в настоящее время дисковые устройства снабжены подвижными магнитными головками. При выполнении обмена с дисковым накопителем выполняется подвод головок к нужному цилиндру, выбор нужной головки (дорожки), прокрутка дискового пакета до начала требуемого блока и, наконец, чтение или запись блока. Среди всех этих действий самое большое время занимает подвод головок. Именно это время определяет общее время выполнения операции. Единственным доступным приемом оптимизации доступа к магнитным дискам является как можно более "близкое" расположение на накопителе последовательно адресуемых блоков файла. Но и в этом случае движение головок будет минимизировано только в том случае, когда файл читается или пишется в чисто последовательном режиме. Именно с такими файлами при потребности сортировки работают современные СУБД.

Следует также заметить, что на самом деле скорость выполнения внешней сортировки зависит от размера буфера (или буферов) основной памяти, которая может быть использована для этих целей. Мы остановимся на этом в конце этой части книги. Сначала же мы рассмотрим основные методы внешней сортировки, работающие при минимальных расходах основной памяти.

**3.1. Прямое слияние**

Начнем с того, как можно использовать в качестве метода внешней сортировки алгоритм простого слияния, обсуждавшийся в конце предыдущей части. Предположим, что имеется последовательный файл A, состоящий из записей a1, a2, ..., an (снова для простоты предположим, что n представляет собой степень числа 2). Будем считать, что каждая запись состоит ровно из одного элемента, представляющего собой ключ сортировки. Для сортировки используются два вспомогательных файла B и C (размер каждого из них будет n/2).

Сортировка состоит из последовательности шагов, в каждом из которых выполняется распределение состояния файла A в файлы B и C, а затем слияние файлов B и C в файл A. (Заметим, что процедура слияния для файлов полностью иллюстрируется рисунком 2.14.) На первом шаге для распределения последовательно читается файл A, и записи a1, a3, ..., a(n-1) пишутся в файл B, а записи a2, a4, ..., an - в файл C (начальное распределение). Начальное слияние производится над парами (a1, a2), (a3, a4), ..., (a(n-1), an), и результат записывается в файл A. На втором шаге снова последовательно читается файл A, и в файл B записываются последовательные пары с нечетными номерами, а в файл C - с четными. При слиянии образуются и пишутся в файл A упорядоченные четверки записей. И так далее. Перед выполнением последнего шага файл A будет содержать две упорядоченные подпоследовательности размером n/2 каждая. При распределении первая из них попадет в файл B, а вторая - в файл C. После слияния файл A будет содержать полностью упорядоченную последовательность записей. В таблице 3.1 показан пример внешней сортировки простым слиянием.

*Таблица 3.1. Пример внешней сортировки прямым слиянием*

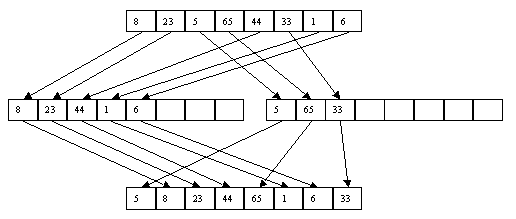
|  |  |
| --- | --- |
| **Начальное состояние файла A** | 8 23 5 65 44 33 1 6 |
| **Первый шаг Распределение Файл B Файл C Слияние: файл A** | 8 5 44 1  23 65 33 6  8 23 5 65 33 44 1 6 |
| **Второй шаг Распределение Файл B Файл C Слияние: файл A** | 8 23 33 44  5 65 1 6  5 8 23 65 1 6 33 44 |
| **Третий шаг Распределение Файл B Файл C Слияние: файл A** | 5 8 23 65  1 6 33 44  1 5 6 8 23 33 44 65 |

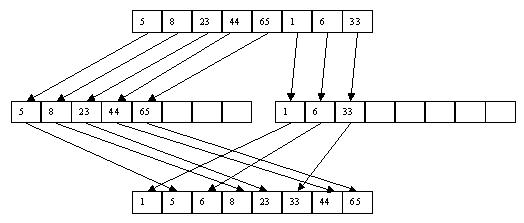
Заметим, что для выполнения внешней сортировки методом прямого слияния в основной памяти требуется расположить всего лишь две переменные - для размещения очередных записей из файлов B и C. Файлы A, B и C будут O(log n) раз прочитаны и столько же раз записаны.

**3.2. Естественное слияние**

При использовании метода прямого слияния не принимается во внимание то, что исходный файл может быть частично отсортированным, т.е. содержать упорядоченные подпоследовательности записей. Серией называется подпоследовательность записей ai, a(i+1), ..., aj такая, что ak <= a(k+1) для всех i <= k < j, ai < a(i-1) и aj > a(j+1). Метод естественного слияния основывается на распознавании серий при распределении и их использовании при последующем слиянии.

Как и в случае прямого слияния, сортировка выполняется за несколько шагов, в каждом из которых сначала выполняется распределение файла A по файлам B и C, а потом слияние B и C в файл A. При распределении распознается первая серия записей и переписывается в файл B, вторая - в файл C и т.д. При слиянии первая серия записей файла B сливается с первой серией файла C, вторая серия B со второй серией C и т.д. Если просмотр одного файла заканчивается раньше, чем просмотр другого (по причине разного числа серий), то остаток недопросмотренного файла целиком копируется в конец файла A. Процесс завершается, когда в файле A остается только одна серия. Пример сортировки файла показан на рисунках 3.1 и 3.2.

  
*Рис. 3.1. Первый шаг*

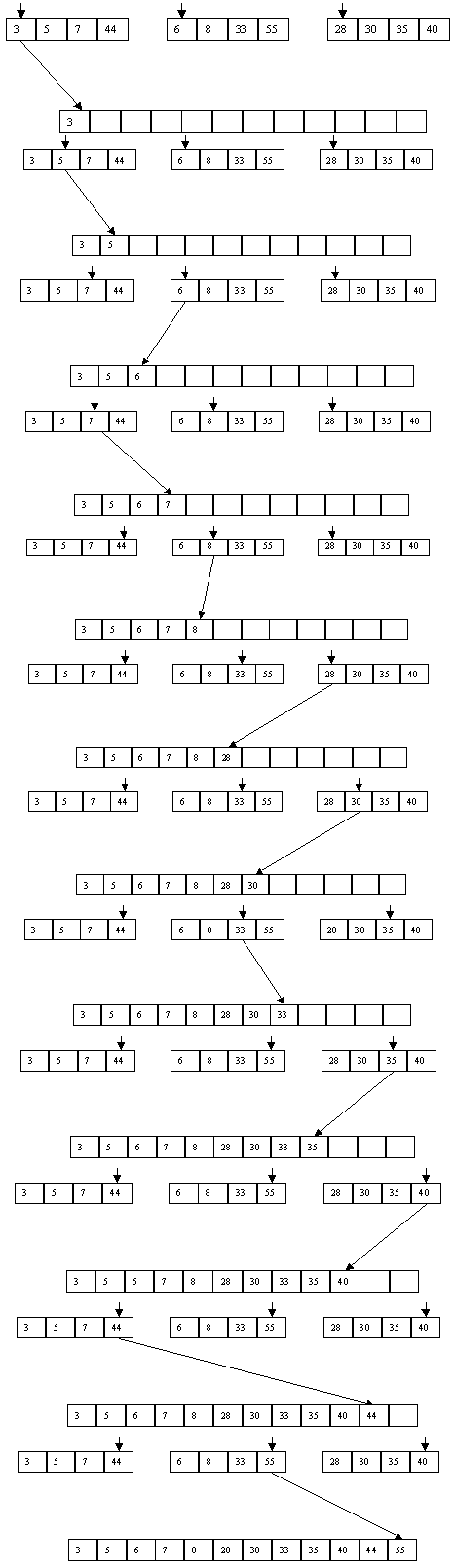
  
*Рис. 3.2. Второй шаг*

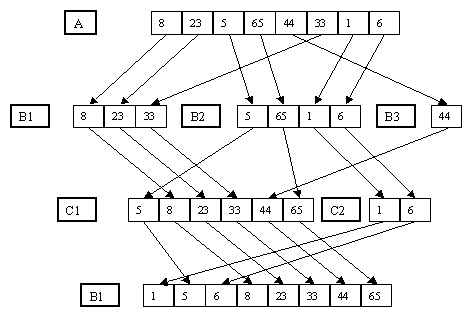
Очевидно, что число чтений/перезаписей файлов при использовании этого метода будет не хуже, чем при применении метода прямого слияния, а в среднем - лучше. С другой стороны, увеличивается число сравнений за счет тех, которые требуются для распознавания концов серий. Кроме того, поскольку длина серий может быть произвольной, то максимальный размер файлов B и C может быть близок к размеру файла A.

**3.3. Сбалансированное многопутевое слияние**

В основе метода внешней сортировки сбалансированным многопутевым слиянием является распределение серий исходного файла по m вспомогательным файлам B1, B2, ..., Bm и их слияние в m вспомогательных файлов C1, C2, ..., Cm. На следующем шаге производится слияние файлов C1, C2, ..., Cm в файлы B1, B2, ..., Bm и т.д., пока в B1 или C1 не образуется одна серия.

Многопутевое слияние является естественным развитием идеи обычного (двухпутевого) слияния, иллюстрируемого рисунком 2.14. Пример трехпутевого слияния показан на рисунке 3.3.

  
*Рис. 3.3.*

  
*Рис. 3.4.*

На рисунке 3.4 показан простой пример применения сортировки многопутевым слиянием. Он, конечно, слишком тривиален, чтобы продемонстрировать несколько шагов выполнения алгоритма, однако достаточен в качестве иллюстрации общей идеи метода. Заметим, что, как показывает этот пример, по мере увеличения длины серий вспомогательные файлы с большими номерами (начиная с номера n) перестают использоваться, поскольку им "не достается" ни одной серии. Преимуществом сортировки сбалансированным многопутевым слиянием является то, что число проходов алгоритма оценивается как O(log n) (n - число записей в исходном файле), где логарифм берется по основанию n. Порядок числа копирований записей - O(log n). Конечно, число сравнений не будет меньше, чем при применении метода простого слияния.