Алгоритмы сортировки

Класс алгоритмов сортировки один из самых больших и наиболее часто используемых.

Д. Кнут

Искусство программирования для ЭВМ

Т. 3. Сортировка и поиск

ООО «И.Д. Вильямс» 2001

Общие положения

- Сортировка – это процесс целенаправленного перемещения элементов заданной конечной последовательности, результатом которой является последовательность, в которой элементы расположены в порядке возрастания (или убывания) их значений.
- Если объем данных такой большой, что перемещение самих записей является нецелесообразным из-за больших накладных расходов, то в этом случае может быть использована вспомогательная таблица указателей, так, что вместо перемещения самих данных перемещаются эти указатели. Это называется сортировкой таблицы адресов.

Терминология

- Сортировка называется *устойчивой*, если она удовлетворяет такому дополнительному условию, что записи с одинаковыми ключами остаются в прежнем порядке
- Сортировка классифицируется как внутренняя, если сортируемые записи находятся в оперативной памяти, и как внешняя, если некоторые из сортируемых записей находятся во вспомогательной памяти.

Цель изучения алгоритмов сортировки

- разобраться в принципах построения этих алгоритмов, чтобы не тратить время на изобретение велосипеда;
- не использовать заведомо неэффективные алгоритмы сортировки;
- определить характерные ситуации, в которых эффективны те или иные алгоритмы или их комбинации;
- определить принципы оценки эффективности алгоритмов сортировки.

Факторы, влияющие на производительность сортировки

(параметры сортировки)

- -число и размер сортируемых записей;
- -тип и размер ключа;
- -характер упорядоченности сортируемых данных (почти отсортированная последовательность, случайная последовательность, почти отсортированная наоборот);
- -повторяемость значений ключа;
- -используемый резерв памяти.

Критерии эффективности

- 1. Основным критерием является *время выполнения* (T), поскольку именно им характеризуется скорость сортировки.
- 2. Время выполнения напрямую зависит от количества сравнений ключей (C) и количества перестановок элементов(P).
- 3. Размер оперативной памяти компьютера ограничен, поэтому немаловажным критерием является *использование памяти* в процессе сортировки (M).
- 4. Стек программы в адресном пространстве ограничен, поэтому при анализе алгоритма нужно учесть критерий *использования рекурсии*(R).
- Большинство классических сортировок имеют временные затраты, которые варьируются от 0(nlogn) до $0(n^2)$.

методы определения временных затрат сортировки

- Имеются два способа определения **временных затрат сортировки**, причем ни один из них не дает результатов, которые применимы для всех случаев.
- Первый состоит в проведении анализа различных случаев (наилучшего, наихудшего, среднего). Результатом этого анализа часто является некоторая формула, задающая среднее время (или число операций) сортировки в зависимости от размера файла n.
- Второй метод - запуск программы на ЭВМ. При этом набирается статистика на различных файлах.

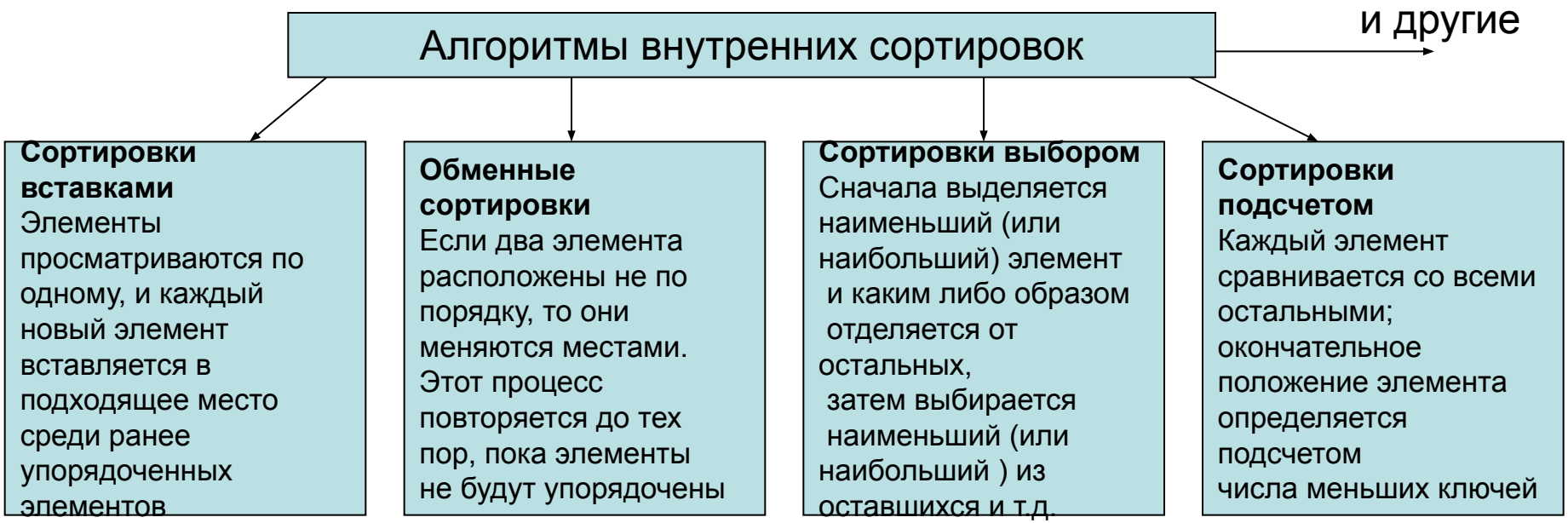
Рекомендации

- Какая либо сортировка не должна выбираться просто потому, что она имеет порядок 0(nlogn), т.к. при разных n она может вести себя по разному.
- В качестве критерия сравнения различных методов сортировки удобно использовать сравнение критических ситуаций (почти упорядоченная, случайная упорядоченность, почти упорядоченная наоборот).

Рекомендации по выбору метода сортировки

- В большинстве случаев время, необходимое для некоторой сортировки, зависит от первоначальной последовательности данных.
- Если имеется некоторая информация о первоначальной последовательности данных, то можно принять более обоснованное решение о том, какой метод сортировки использовать. Если нет такой информации, можно выбрать некоторую сортировку основываясь на самом худшем или на среднем варианте.
- Также выбор сортировки зависит от того, как часто изменяется последовательность данных.
- Желательно иметь дело с сортировками имеющими сложность O(nlogn)

Классификация методов внутренних сортировок



СОРТИРОВКИ ВСТАВКАМИ

- Простые вставки
- Бинарные вставки
- Двухпутевые вставки
- Вставки в список
- Вставка в дерево
- Сортировка Шелла
- Сортировка с вычислением адреса
- и т. д.

Простые вставки

- Сортировка в этом случае осуществляется путем вставки очередного элемента после меньшего.
- Рассмотрим на примере
- 25 57 48 37 12 92 86 33
- 25 57 48 37 12 92 86 33
- 25 48 57 37 12 92 86 33
- 25 37 48 57 12 92 86 33
- 12 25 37 48 57 92 86 33
- 12 25 37 48 57 92 86 33
- 12 25 37 48 57 86 92 33
- 12 25 33 37 48 57 86 92

Анализ

- Если последовательность отсортирована, то на каждом просмотре делается одно сравнение и сложность 0(n).
- Если последовательность отсортирована наоборот, то сложность $0(n^2)$.
- Чем ближе файл к отсортированному виду, тем более эффективной становится сортировка простыми вставками.
- Метод простых вставок в среднем требует $n^2/4$ сравнений.

Бинарные вставки

- Работу метода проиллюстрируем на примере. Если вставляется 64 элемент, то сначала он сравнивается с 32 элементом, затем если он меньше, то сравнивается с 16 элементом, а если больше с 48 элементом и т.д. Место для 64 элемента будет найдено за 6 сравнений.
- Однако бинарные вставки решают задачу только наполовину: после нахождения места вставки нужно передвинуть примерно n/2 ранее отсортированных записей, чтобы освободить место вставляемой записи.

Двухпутевые вставки

- Проблему сдвига можно решить методом двухпутевых вставок. Суть его в следующем: первый элемент вставляется в середину последовательности, а место для последующих элементов освобождается при помощи сдвигов влево или вправо.
- Например, имеем последовательность:
- 25 57 48 37 12 92 86 33
- 25
- 25 57
- 25 48 57
- 25 37 48 57
- 12 25 37 48 57
- 12 25 37 48 57 92
- 12 25 37 48 57 86 92
- 12 25 33 37 48 57 86 92

Вставки в список

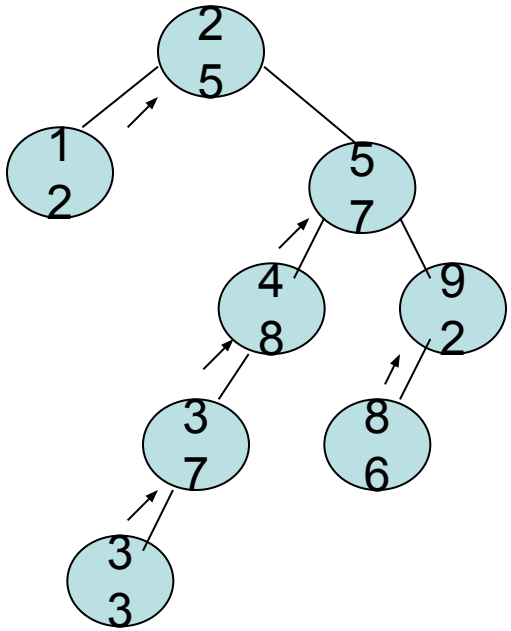
- Анализ структур данных, позволяющий избежать ненужных операций, часто дает существенный эффект.
- Рассматриваемую последовательность можно представить как линейный список.
- Для того, чтобы вставить k-й элемент, организуется прохождение данного связанного списка до тех пор, пока не будут найдены нужная позиция для x(k) или конец списка. В этот момент x(k) может быть вставлен в список при помощи настройки указателей, без сдвига элементов в последовательности.
- Это уменьшает время необходимое для вставки, но не время необходимое для поиска. Требование на пространство также возрастает из-за дополнительного массива p.

Вставка в дерево

- Простейшая сортировка с использованием бинарных деревьев состоит в просмотре каждого элемента входной последовательности и помещение элемента в соответствующую позицию в некотором бинарном дереве.
- Для того чтобы найти соответствующую позицию некоторого элемента, в каждом узле происходит выбор левой или правой ветви в зависимости от того, меньше вставляемый элемент элемента в этом узле, либо больше или равен ему.
- Когда каждый элемент находится на своем месте в дереве, отсортированная последовательность может быть получена при помощи просмотра этого дерева в симметричном порядке.

пример

- 25 57 48 37 12 92 86 33



Можно показать, что такая сортировка имеет сложность порядка 0(n log n) (отметим, что если дерево является прошитым в момент его создания, время прохождения его сильно уменьшается).

Сортировка Шелла

- Существенного улучшения можно достигнуть используя сортировку Шелла (или сортировку с убывающим шагом). Суть метода состоит в разделении последовательности на части и сортировка этих частей (обычно при помощи метода простых вставок). Количество частей k называется шагом.
- Затем выбирается меньшее значение шага k и процесс повторяется.
- Последнее значение шага k должно быть 1.

пример

- Например, начальная последовательность имеет вид:
- 25 57 48 37 12 92 86 33, последовательность шагов 5 3 1.
- Первая итерация (k=5)
- (x(1)=25, x(6)=92)
- (x(2)=57, x(7)=86)
- (x(8)=33, x(3)=48)
- (x(4)=37)
- (x(5)=12)
- 25 57 33 37 12 92 86 48
- Вторая итерация (k=3)
- (x(1)=25, x(4)=37, x(7)=86)
- (x(5)=12, x(8)=48, x(2)=57)
- (x(3)=33, x(6)=92)
- 25 12 33 37 48 92 86 57
- Третья итерация (k=1)
- 12 25 33 37 48 57 86 92

Анализ

- Уже отмечалось, что сортировка простыми вставками очень эффективна для последовательности, которая почти упорядочена.
- Важно также понимать, что когда размер последовательности n мал, то сортировка порядка $0(n^2)$ часто более эффективна, чем сортировка порядка $0(n \log n)$, т.к. последняя требует как правило дополнительных действий.
- Анализ эффективности сортировки Шелла математически достаточно сложен. Можно показать, что порядок сложности сортировки Шелла может быть аппроксимирован величиной $0(n(\log n)^2)$, если используется подходящая последовательность шагов (т. е. количества шагов на разных итерациях должны быть взаимно простыми числами).

Сортировка с вычислением адреса

- Для сортировки здесь используется функция хеширования.
- Заданная последовательность разбивается на части в соответствии с функцией хеширования. В каждую часть включаются элементы, которые меньше или равны элементам в другой части. При этом элементы помещаются в каждую часть упорядоченно с помощью одного из методов сортировки (например, метода вставок).
- После того, как все элементы последовательности будут помещены в части, эти части будут соединены, чтобы получить отсортированный результат.

пример

- Рассмотрим последовательность
- 25 57 48 37 12 92 86 33
- Создадим десять частей для каждой из 10 возможных цифр элементов последовательности. Каждая часть строится как отсортированный связанный список. Первоначально каждая часть пуста. Используем функцию хеширования
- H = x div 10.
- Тогда части выглядят следующим образом:
- 0
- 1 12
- 2 25
- 3 33 37
- 4 48
- 5 57
- 6
- 7
- 8 86
- 9 92

Анализ

- Кнут показал, что среднем случае сложность метода равно 0(n).
- Этот результат справедлив только тогда, когда вероятность хеширования любого ключа в любое число от 1 до m равна 1/m.
- Наихудший случай, когда все ключи отображаются в одну часть. При этом оценка метода ухудшается до $0(n^2)$.

Сравнение алгоритмов *сортировок вставками*

	Среднее время	Требования к памяти	Примечания
Простые вставки	$O(n^2)$	нет	Хорош для списков частично упорядоченных и массивов n<20
Бинарные вставки	$O(n^2)$	нет	Легко определяется место вставки, но сама вставка требует много времени. Не рекомендуется при n>128
Двухпутевые вставки	$O(n^2)$	нет	Сохраняет достоинства бинарных вставок, частично решает проблему сдвига элементов. Интересен в учебных целях
Вставки в список	$O(n^2)$	требуется доп память для указателей	Легко осуществляется вставка, но число сравнений велико. Интересен в в учебных целях
Сортировка Шелла	$O(n^{1,25})$	нет	Хорош для общего образования и для n<50
Сортировка с вычислением адреса	$O(n)$	требуется доп память	Очень эффективна, когда ключи хешируются равномерно

ОБМЕННЫЕ СОРТИРОВКИ

- Сортировка методом пузырька
- Быстрая сортировка (Обменная сортировка с разделением)
- Параллельная сортировка Бэтчера (Обменная сортировка со слиянием. Алгоритм Бэтчера)
- Обменная поразрядная сортировка
- цифровая сортировка («карманная сортировка»)

Сортировка методом пузырька

- Сортировка методом пузырька проста для усвоения и программирования, но наверное самая неэффективная.
- Идея этого метода состоит в том, чтобы просмотреть файл последовательно несколько раз, а на каждом просмотре сравнить предыдущий и последующий элемент и в случае необходимости поменять местами (транспозиция).

Пример

- Рассмотрим следующий файл:
- 25 57 48 37 12 92 86 33
- итер.1) 25 48 37 12 57 86 33 92
- итер.2) 25 37 12 48 57 33 86 92
- итер.3) 25 12 37 48 33 57 86 92
- итер.4) 12 25 37 33 48 57 86 92
- итер.5) 12 25 33 37 48 57 86 92
- итер.6) 12 25 33 37 48 57 86 92
- итер.7) 12 25 33 37 48 57 86 92

улучшения метода

- Поскольку все элементы в позициях, больших или равных n-i+1, уже находятся после i- ой итерации, на нужных местах, то нет необходимости рассматривать их на последующих итерациях.
- Файл может быть отсортирован раньше чем за n-1 просмотр (см. пример). Для исключения ненужных просмотров целесообразно сохранять информацию о транспозиции элементов на каждом просмотре. Отсутствие транспозиции свидетельствует о том, что файл отсортирован.
- Возможны и другие усовершенствования этого метода (например чередование просмотров в прямом и обратном направлении – *шейкер сортировка*).
- Этот метод нежелательно использовать для таблиц большого размера (более 15).

Анализ

- Заметим, что для файла из n элементов требуется не более n-1 итераций.
- Оценка эффективности сортировки методом пузырька проста. Имеется (n-1) просмотров и (n-1) сравнений на каждом просмотре. Общее число сравнений $(n-1)*(n-1)=n^2-2n+1$, что составляет $0(n^2)$. Число транспозиций меньше числа сравнений.
- Возможные улучшения метода приводят к тому, что среднее число итераций меньше, но сложность остаётся $0(kn^2)$, и сомножитель k меньше.

Быстрая сортировка
(Обменная сортировка с разделением)

- Эффективна для больших файлов (таблиц). Цель каждого шага метода – помещение очередного элемента на его конечную позицию внутри последовательности.
- В результате все предшествующие элементы будут иметь меньшее значение, а последующие – большее. При этом последовательность делится на две части.
- Аналогичный процесс может быть применим рекурсивно к каждой из этих частей до тех пор пока все элементы не станут на свои места

пример

- 25 57 48 37 12 92 86 33
- Используются два индекса i и j с начальными значениями 1 и n соответственно.
- Сравниваем элементы a_i и a_j . Если перестановка не требуется, то j уменьшается на единицу и процесс повторяется. Если $a_i \geq a_j$, они меняются местами, i увеличивается на единицу и процесс повторяется, пока не возникнет другая перестановка. В этом случае j будет уменьшена на единицу и т.д.
- Последовательность перестановок для помещения элемента 25 в его конечную позицию имеет вид:
- 25 57 48 37 12 92 86 33
- 25 57 48 37 12 92 86 33
- 25 57 48 37 12 92 86 33
- 25 57 48 37 12 92 86 33
- 12 57 48 37 25 92 86 33
- 12 25 48 37 57 92 86 33
- 12 25 48 37 57 92 86 33

Анализ

- В результате элементы разбиты на две части, к каждой из которых рекурсивно может применена эта процедура, пока файл не окажется полностью отсортирован.
- Для хранения границ подфайлов используется стек.
- При $n \to \infty$ оценка сложности асимптотически приближается к $O(n\log_2 n)$.

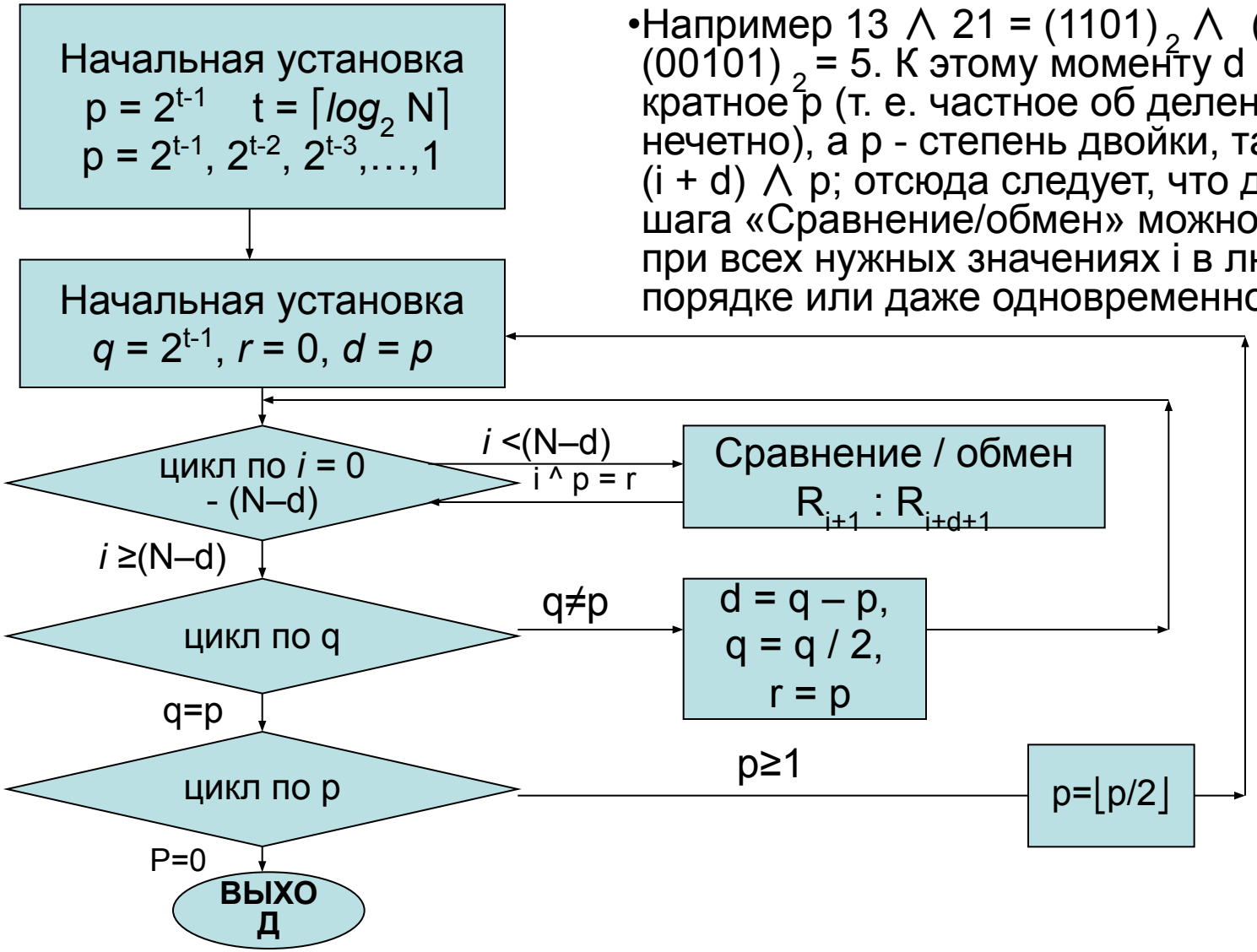
Параллельная сортировка Бэтчера

(Обменная сортировка со слиянием. Алгоритм Бэтчера)

- Суть – происходит слияние пар отсортированных подпоследовательностей. Проблема – как формировать эти пары.
- Метод далеко не очевиден. Два доказательства можно найти у Кнута т.3.
- Если мы хотим получить алгоритм обменной сортировки, время работы которого имеет порядок, меньший N^2, то необходимо подбирать для сравнений некоторые пары *несоседних* ключей ($К_i$, $К^j$); иначе придется выполнить столько обменов, сколько инверсий имеется в исходной перестановке, а среднее число инверсий равно $(N^2 - N)/4$
- Схема сортировки Бэтчера несколько напоминает сортировку Шелла, но сравнения выполняются по-новому, так что распространение обменов не обязательно.

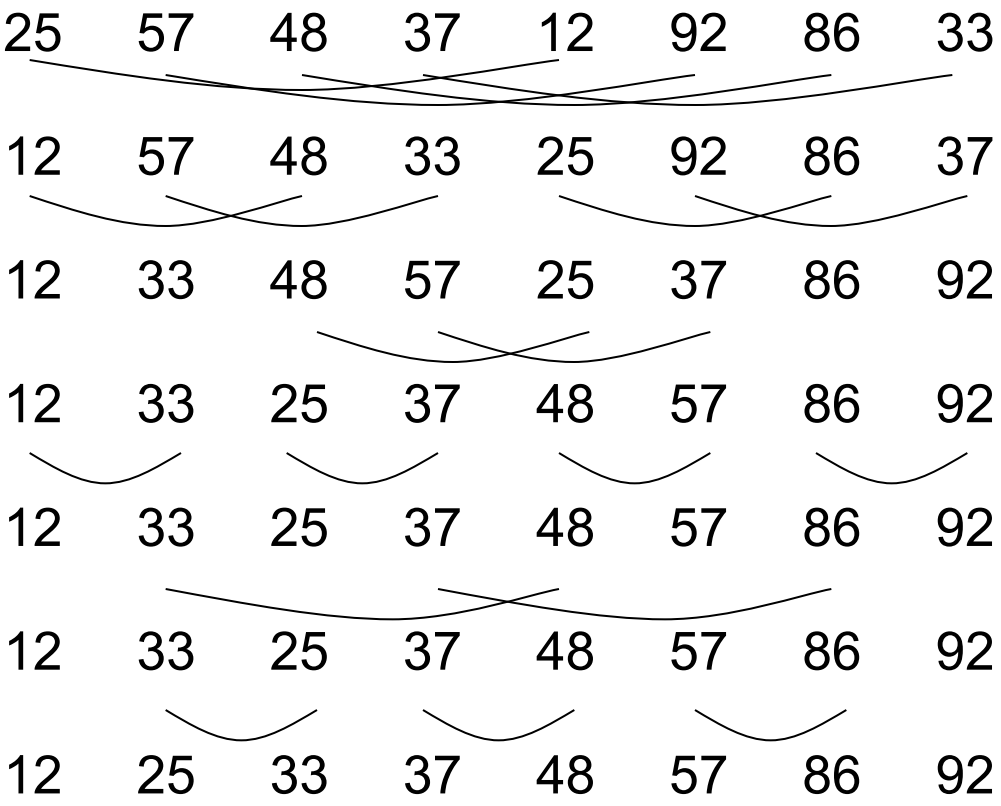
Схема алгоритма

- (^-поразрядное логическое «и»)
- Например 13 ⋀ 21 = $(1101)_2$ ⋀ $(10101)_2$ = $(00101)_2$ = 5. К этому моменту d - нечетное кратное p (т. е. частное об деления d на p нечетно), а p - степень двойки, так что i ⋀ p ≠ (i + d) ⋀ p; отсюда следует, что действия шага «Сравнение/обмен» можно выполнять при всех нужных значениях i в любом порядке или даже одновременно.



Пример

- Рассмотрим работу этого алгоритма на последовательности
- p q r d
- 25 57 48 37 12 92 86 33
- 4 4 0 4
- 12 57 48 33 25 92 86 37
- 2 4 0 2
- 12 33 48 57 25 37 86 92
- 2 2 2 2
- 12 33 25 37 48 57 86 92
- 1 4 0 1
- 12 33 25 37 48 57 86 92
- 1 2 1 3
- 12 33 25 37 48 57 86 92
- 1 1 1 1
- 12 25 33 37 48 57 86 92



Анализ

- Алгоритм достаточно сложный. Однако он обладает компенсирующим качеством. Все обмены и сравнения можно вести на устройствах допускающих параллельные вычисления.
- В этом случае это один из самых быстрых методов и выполняется за $1/2\lceil \log_2 N \rceil$ ($\lceil \log_2 N \rceil+1$) шагов.
- Например, 1024 *элемента можно отсортировать методом Бэтчера всего за 55 параллельных шагов*. Его ближайшим соперником является метод Пратта
- если мы готовы допустить перекрытие сравнений до тех пор, пока не потребуется выполнять перекрывающиеся обмены, то для сортировки 1024 элементов методом Пратта требуется всего 40 циклов сравнения/обмена.

Обменная поразрядная сортировка

- ***Отличительная черта этого метода – двоичное представление ключей и сравниваются отдельные биты ключей.***
- В общих чертах этот алгоритм можно описать следующим образом:
- Последовательность сортируется по старшему биту так, чтобы все ключи, начинающиеся с 0, оказались перед всеми ключами, начинающимися с 1. Для этого надо найти самый левый ключ $К_i$, начинающийся с 1, и самый правый ключ $К_j$, начинающийся с 0. меняются местами. Процесс повторяется, пока не станет $i = j$.
- Пусть F_0 - множество элементов, начинающихся с 0, а F_1 – все остальные. Применим к F_0 поразрядную сортировку (начав со второго бита слева) до тех пор, пока множество F_0 не будет полностью отсортировано, затем проделаем тоже с F_1

Анализ

- Число стадий и число проверок несколько больше чем в быстрой сортировке, но при больших n в действительности меньше, в предположении о равномерном расположении ключей.
- Как и в быстрой сортировке, для хранения «информации о границах» подфайлов, ожидающих сортировки, можно воспользоваться стеком.
- Обменная поразрядная сортировка в стандартном варианте не очень эффективна, если имеются одинаковые ключи.

«цифровая сортировка»

(«карманная сортировка»)

- Рассмотрим обобщение обменной поразрядной сортировки на случай счисления с основанием большим 2.
- Например необходимо отсортировать колоду из 52 игральных карт в соответствии Т< 2< 3< …..< 10< В< Д< К и по масти пика<трефа<бубна<черва Это частный случай лексикографического упорядочения на множестве.
- Естественно отсортировать карты сначала по масти в четыре стопки, а затем отсортировать карты в каждой стопке.
- Но существует более быстрый способ! Сначала разложить карты в 13 стопок по их достоинству (тузу к тузам и т.д.). Затеи сложив их вместе разложить на четыре стопки по мастям. И наконец в соответствии со статусом мастей сложить эти четыре стопки.
- Доказательство такой сортировки приводится у Кнута.

Суть алгоритма

- поразрядную сортировку можно выполнить следующим образом :
  - сначала провести распределяющую сортировку по младшим цифрам ключей (в системе счисления с основанием M), переместив записи из области ввода во вспомогательную область,
  - затем произвести ещё одну распределяющую сортировку по следующей цифре, переместив записи обратно в исходную область
  - и т.д., до тех пор пока после завершения просмотра (сортировка по старшей цифре) все ключи не окажутся расположенными в нужном порядке.

пример

25 57 48 37 12 92 86 33

Удобно начать с младшей значащей цифры, т.к. числа могут иметь разное количество цифр.

- Очереди, организованные по МЗЦ.
- 0
- 1
- 2 12 92
- 3 33
- 4
- 5 25
- 6 86
- 7 57 37
- 8 48
- 9
- После первого просмотра
- 12 92 33 25 86 57 37 48

- Очереди, организованные по старшей значимой цифре
- 0
- 1 12
- 2 25
- 3 33 37
- 4 48
- 5 57
- 6
- 7
- 8 86
- 9 92
- Отсортированный файл
- 12 25 33 37 48 57 86 92

Анализ

- Временные затраты на метод поразрядной сортировки зависят от числа цифр (m) в ключе элемента и числа элементов в последовательности (n).
- Порядок сложности данной сортировки составляет O (m * n)

Сравнение алгоритмов обменных сортировок

	Среднее время	Требования к памяти	Примечания
метод пузырька	$O(n^2)$	нет	Лёгок в реализации, самый не эффективный
Быстрая сортировка	O(nlogn)	logn	Быстрейший метод для больших неупорядоченных массивов n>50, Худший случай маловероятен
Параллельная сортировка Бэтчера	$1/2\lceil \log_2 n \rceil (\lceil \log_2 n \rceil + 1)$	нет	Один из самых быстрых известных методов
Обменная поразрядная сортировка	O(nlogn)	нет	Эффективна в случае равномерного распределения ключей, не очень эффективна, если имеются одинаковые ключи
цифровая сортировка	O (m * n)	Требуется доп. память	Самая эффективная при условии n не слишком мало, а m –длина ключа, не слишком велика

Сортировки посредством выбора

- Сортировка посредством простого выбора
- Квадратичный выбор
- Выбор из дерева
- Сортировка методом турнира с выбыванием
- Пирамидальная сортировка (Сортировка с помощью кучи)
- Наибольший из включенных первым исключается
  - *Характерная черта – все сортировки требуют наличия всех элементов до начала сортировки*

Сортировка посредством простого выбора

- По сравнению с основными элементами таких алгоритмов здесь добавлены два улучшения:
  - выбранный элемент можно записывать в соответствующую позицию, а элемент который её занимает, переносится на место выбранного;
  - удобно выбирать наибольший элемент.

пример

- 25 57 48 37 12 92 86 33
- 25 57 48 37 12 33 86 92
- 25 57 48 37 12 33 86 92
- 25 33 48 37 12 57 86 92
- 25 33 12 37 48 57 86 92
- 25 33 12 37 48 57 86 92
- 25 12 33 37 48 57 86 92
- 12 25 33 37 48 57 86 92
- Анализ показывает, что алгоритм чуть медленнее простых вставок, но предпочтительней метода пузырька (меньше число обменов).

Квадратичный выбор

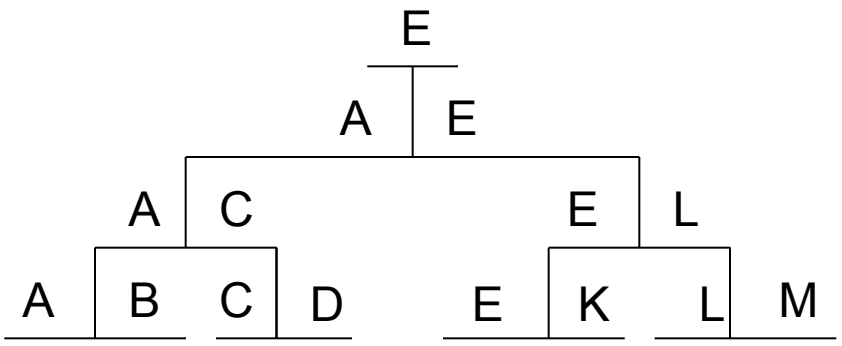
- Возможно улучшение алгоритма простого выбора за счёт использования извлечённой раньше информации при последующих выборах.
- Один из способов сэкономить время на последующих выборах – разбить последовательность на группы. В нашем случае на 2 группы и в каждой группе определяем наибольший элемент.
- 25 57 48 37 | 12 92 86 33

57 92	92
57 86	86 92
57 33	57 86 92
48 33	48 57 86 92
37 33	37 48 57 86 92
25 33	33 37 48 57 86 92
25 12	25 33 37 48 57 86 92
12	12 25 33 37 48 57 86 92

Оценка сложности этого метода $O(n\sqrt{n})$, что существенно лучше $O(n^2)$.
Эту идею можно обобщить:
Если разбить последовательность на три части, то получится в результате метод кубического выбора, если разбить на четыре части - выбор четвертой степени и т.д. В пределе выбор n степени основан на структуре бинарного дерева

Сортировка методом турнира с выбыванием

- Рассмотрим эту сортировку на примере турнира команд A,B,C,D,E,K,L,M.
- Команда E на первом месте за семь матчей (сравнений)
- Второе место можно определить, если команда A сыграет с командой K, а победитель с командой L , т.е. два дополнительных матча (сравнения). Аналогично определяем другие места.

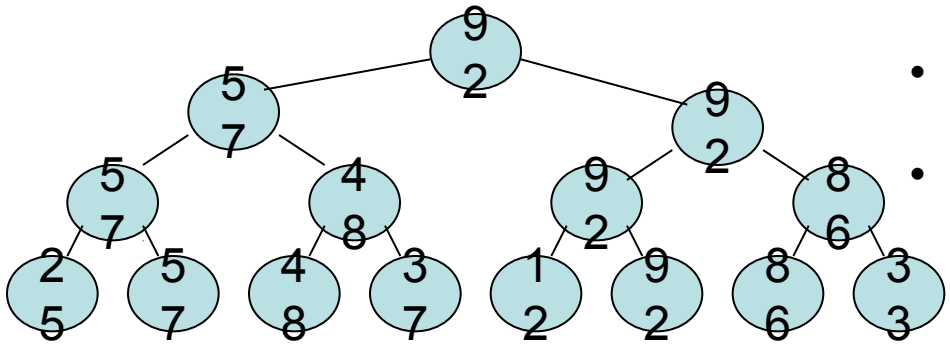


Этот подход используется в сортировке методом выбора из дерева

Выбор из дерева

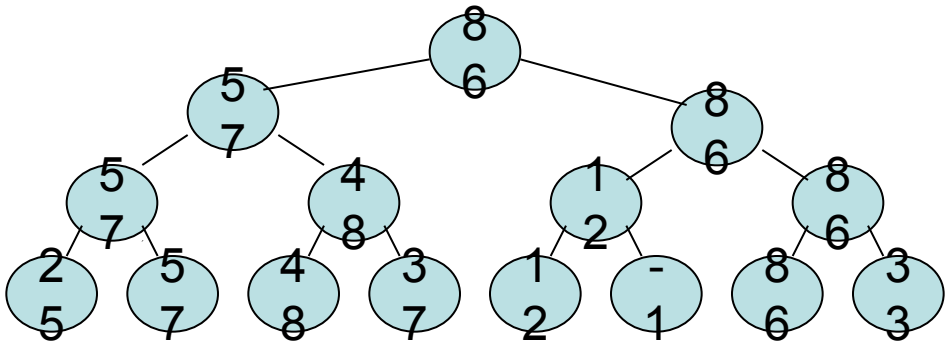
- Дерево строится следующим образом:
- каждому первоначальному элементу приписывается некоторый узел (лист) в некотором бинарном дереве;
- Затем выбираются два узла (листа). Наибольшее значение из них присваивается узлу – отцу. Этот процесс повторяется до тех пор пока не останется один лист или ни одного. Если остается один лист, то этот узел надо переместить вверх на следующий уровень.
- Затем процесс с заново созданными узлами надо повторить до тех пор, пока самое большое число не будет помещено в корень бинарного дерева.
- Узел являющийся листом со значением корня заменяется минимальным числом. Содержимое всех предков до корня этого листа повторно вычисляется.
- Этот процесс повторяется до тех пор, пока не будут выведены все листья первоначального дерева.

пример

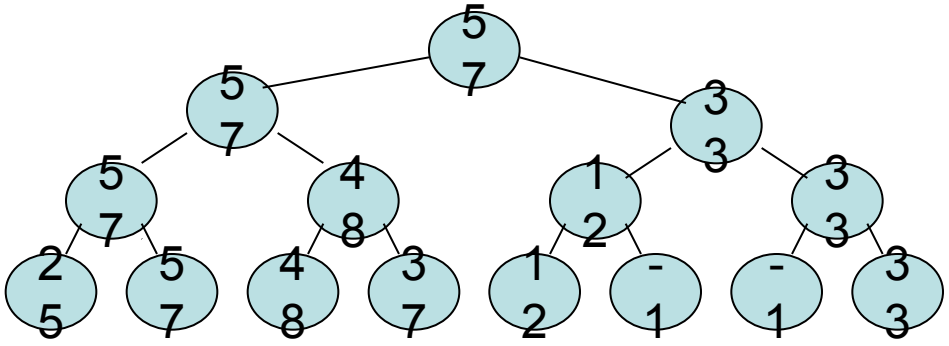


- Рассмотрим этот алгоритм на последовательности
- 25 57 48 37 12 92 86 33

выводим 92



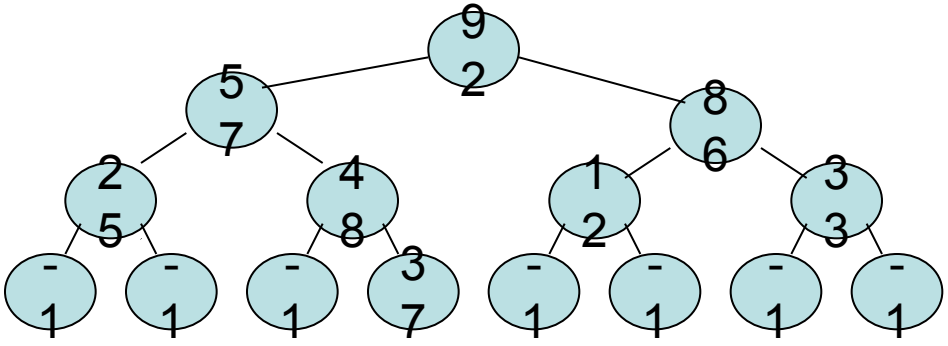
выводим 86



выводим 57 и т.д.

Усовершенствование

- Когда больший элемент перемещается вверх с одного разветвления на другое, его можно заменить наибольшим элементом, расположенных под ним. Выполнив эту операцию столько раз, сколько возможно, получим дерево



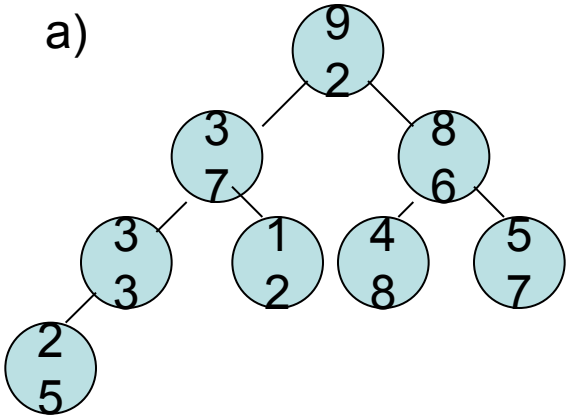
Для такого дерева можно продолжить сортировку не «восходящим» как ранее методом, а «нисходящим»: выводится элемент находящийся в корне, перемещается вверх наибольший из его потомков, перемещается вверх наибольший из потомков последнего и т.д.

Пирамидальная сортировка
(сортировка с помощью пирамиды)

- состоит из двух частей:
- Сначала вызывается процедура построения пирамиды (см. Бинарные кучи).
- Идея второй части проста:
  - максимальный элемент массива теперь находится в корне дерева (A[1]). Его следует поменять с A[n], уменьшить размер кучи на 1 и восстановить основное свойство в корневой вершине (поскольку поддеревья с корнями left(i) и right(i) не утратили основного свойства пирамиды, это можно сделать с помощью процедуры savek). После этого в корне будет находиться максимальный из оставшихся элементов.
  - Так делается до тех пор, пока в пирамиде не останется всего один элемент.

пример

а)



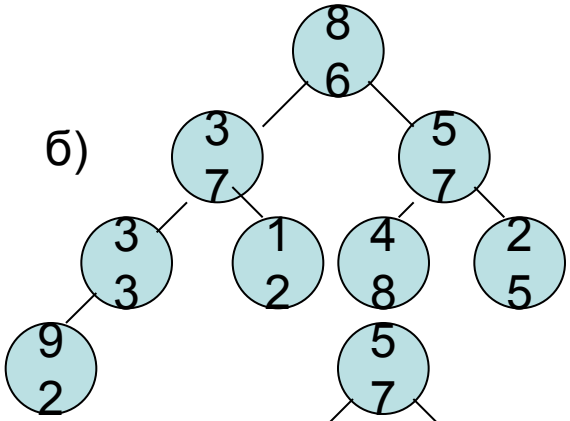
- Для последовательности

25 57 48 37 12 92 86 33

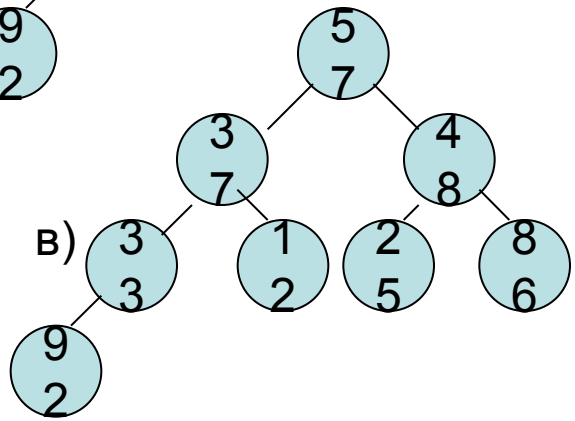
бинарная пирамида имеет вид как на рис Или в виде массива

92 37 86 33 12 48 57 25

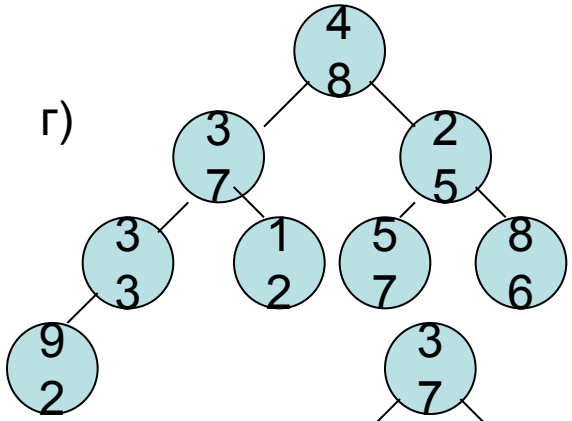
б)



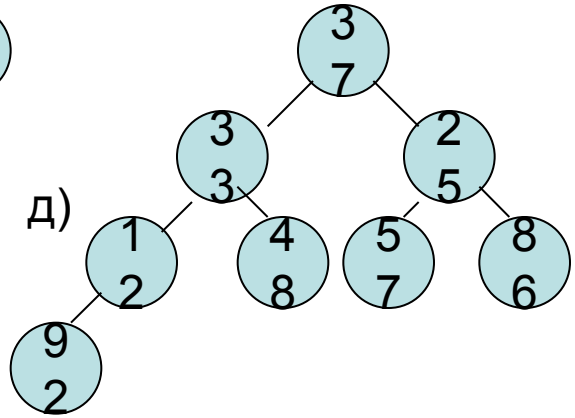
в)



г)



д)



и т. д.

Сравнение алгоритмов сортировок выбором

	Среднее время	Требования к памяти	Примечания
Простой выбор	$O(n^2)$	нет	алгоритм чуть медленнее простых вставок, но предпочтительней метода пузырька (меньше число обменов).
Квадратичный выбор	$0(n\sqrt{n})$	Требуется доп. память	В пределе (показатель корня $\rightarrow \infty$) получается выбор из дерева
Сортировка методом турнира с выбыванием	0(n log n)	Требуется доп. память	Интересна в учебных целях
Выбор из дерева	0(n log n)	нет	Эффективна для почти отсортированных и для почти отсортированных наоборот последовательностей
Пирамидальная сортировка	0(n log n)	нет	На практике пирамидальная сортировка не является самой быстрой - как правило, быстрая сортировка работает быстрее

Эффективность