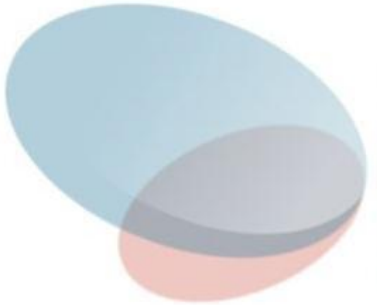


Алгоритмы и структуры данных
Лекция 1
Алгоритмы сортировки

Сортировка – это процесс упорядочения некоторого множества элементов, на котором определено отношение порядка >, <, ≥, ≤, =. Когда говорят о сортировке, подразумевают упорядочение множества элементов по возрастанию или убыванию.

Традиционно различают

внутреннюю сортировку, в которой предполагается, что

- данные находятся в оперативной памяти, и
- важно оптимизировать число действий программы (для методов, основанных на сравнении, число сравнений, обменов элементов и пр.), и

внешнюю, в которой

- данные хранятся на внешнем устройстве и,
- прежде всего, требуется снизить число обращений к этому устройству.

Оценка сложности алгоритма сортировки

- **Алгоритмы сортировки ведут себя по-разному в различных обстоятельствах.**

  Например, пузырьковая сортировка опережает быструю сортировку по скорости работы, если сортируемые элементы уже были почти упорядочены, но работает медленнее, если элементы были расположены хаотично.

- **Алгоритмы сортировки оцениваются по скорости выполнения и эффективности использования памяти.**

Время — основной параметр, характеризующий быстродействие алгоритма. Называется также ***вычислительной сложностью***. Для упорядочения важны ***худшее***, ***среднее*** и ***лучшее*** поведение алгоритма в терминах мощности входного множества A. Если на вход алгоритму подаётся множество A, то обозначим n = | A |. Для типичного алгоритма хорошее поведение — это *O(n log n),* и плохое поведение — это *O(n^2)*. Идеальное поведение для упорядочения — *O(n)*.

Алгоритмы сортировки, использующие только абстрактную операцию *сравнения ключей*, всегда нуждаются по меньшей мере в *O(n log n)* сравнениях.

Память – второй параметр, характеризующий эффективность алгоритма**. Ряд алгоритмов требует выделения *дополнительной* памяти под временное хранение данных**. Как правило, эти алгоритмы требуют *O(log n)* памяти. Алгоритмы сортировки, не потребляющие дополнительной памяти, относят к **сортировкам на месте**.

Классификация алгоритмов сортировки

- ***Устойчивость*** (stability) — устойчивая сортировка **не меняет взаимного расположения равных элементов.**
- ***Естественность поведения*** — эффективность метода **при обработке уже упорядоченных**, или частично упорядоченных данных. **Алгоритм** ведёт себя естественно, если учитывает эту характеристику входной последовательности и **работает быстрее**.
- ***Использование операции сравнения***. Алгоритмы, использующие для сортировки **сравнение элементов между собой**, называются **основанными на сравнениях**. **Минимальная** трудоемкость худшего случая для этих алгоритмов составляет *O(n log n)*, но они отличаются гибкостью применения. Для специальных случаев (типов данных) существуют более эффективные алгоритмы.

Общие принципы преобразования данных при использовании алгоритмов сортировки

- ***Таблицы указателей***

При сортировке элементов данных программа перестраивает их в некоторую структуру данных. Скорость этого процесса зависит от **типа** элементов. Перемещение целого числа на новое место в массиве может быть намного быстрее, чем перемещение определенной пользователем структуры данных. **Для повышения производительности при сортировке больших объектов**, например, записей, можно **помещать ключевые поля данных, используемые для сортировки, в таблицу индексов (указателей).**

Замечание

- При добавлении или удалении записи необходимо обновлять каждую таблицу индексов независимо.
- Таблицы индексов занимают дополнительную память. Если создать таблицу индексов для каждого из полей данных, объем занимаемой памяти более чем удвоится.

Общие принципы преобразования данных при использовании алгоритмов сортировки

- ***Объединение и сжатие ключей***

- В некоторых случаях можно **хранить ключи списка в комбинированной или сжатой форме.**

  Например, **можно объединять (combine) в программе два поля,** соответствующих имени и фамилии, в один ключ. Это позволит упростить и ускорить сравнение.

- **Иногда можно сжимать (compress) ключи.** Сжатые ключи занимают меньше места, уменьшая размер таблиц индексов. Это позволяет сортировать списки большего размера без перерасхода памяти, быстрее перемещать элементы в списке.

- **Один из методов сжатия строк — кодирование их целыми числами или данными другого числового формата**. Числовые данные занимают меньше места, чем строки и сравнение двух численных значений также происходит намного быстрее, чем сравнение двух строк.

Пример. Требуется **закодировать строки, состоящие из заглавных латинских букв**. Закодируем каждый символ числом по основанию 27 (26 латинских букв и еще одна цифра для обозначения конца слова).

Пусть сравниваются слова максимальной длины 3.

Код по основанию 27 для строки из трех символов дает формула

27^2 * (ord(первая буква) - ord(A) + 1) + 27 * (ord(вторая буква) - ord(A) + 1) + (ord(третья буква) - ord(A) + 1).

Если в строке меньше трех символов, вместо значения (ord(третья буква) - ord(A) + 1) подставляется 0. Например, строка FOX кодируется так:

27^2 * (ord(F) - ord(A) + 1) + 27 * (ord(O) - ord(A) + 1) + (ord(X) - ord(A) +1) = 4803

Строка NO кодируется следующим образом:

27^2 * (ord(N) - ord(A) + 1) + 27 * (ord(O) - ord(A) + 1) + (0) = 10611

Заметим, что 10611 больше 4803, поскольку NO > FOX.

ord(A) = 65 ; // ASCII код для символа "A".

1. Сортировка выбором

Сортировка выбором (selection sort) — **простой алгоритм** сортировки, является **неустойчивым.** На массиве из n элементов имеет **время выполнения в худшем, среднем и лучшем случае O(n^2),** (предполагается, что сравнения осуществляются за постоянное время).

Идея состоит в

- **поиске наименьшего (наибольшего) элемента** в списке, **который затем меняется местами с элементом в начале (в конце) списка**.
- Далее находится наименьший (наибольший) элемент из оставшихся, и меняется местами со вторым элементом (предпоследним).
- Процесс продолжается до тех пор, пока все элементы не займут свое конечное положение.

Усовершенствование алгоритма

Использовать **двунаправленный вариант сортировки методом выбора**, в котором на каждом проходе отыскиваются и устанавливаются на свои места и минимальное, и максимальное значения.

1. Сортировка выбором

Вычислительная сложность алгоритма

При поиске i-го наибольшего (наименьшего) элемента алгоритму приходится перебрать n-i элементов, которые еще не заняли свое конечное положение. Время выполнения алгоритма пропорционально (n - 1) + (n - 2) + … + 1=n*n/2, или порядка O(n^2).

Особенности алгоритма

- Сортировка выбором **неплохо работает со списками, элементы в которых расположены случайно или в прямом порядке**, но несколько **хуже**, если список изначально отсортирован **в обратном порядке.**
- **Алгоритм чрезвычайно прост**. Это не только облегчает его разработку и отладку,
- но и делает сортировку выбором **достаточно быстрой для задач небольшой размерности.**

1. Сортировка выбором

Пример упорядочения по возрастанию

СОРТИРОВКА ПОСРЕДСТВОМ ПРОСТОГО ВЫБОРА

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

2. Сортировка вставкой

Идея сортировки вставками:

очередная запись вставляется среди ранее отсортированных записей.

К i-у шагу (i=2,…,n) первые i-1 записей исходного массива отсортированы. Если запись k(i) < k(i-1) (в случае, когда массив сортируется по возрастанию), запись k(i) необходимо вставить в отсортированную часть массива, не нарушая упорядочивания.

1. Запись k(i) сохраняется во вспомогательной переменной temp.
2. Последовательно сравниваются k(j), j= i-1, …, 1, со значением temp, пока k(j) > temp, при этом записи k(j+1) присваивается значение k(j). В результате находится такое j, что либо k(j) <= temp, либо j=0 (значение temp меньше любого k(j)).
3. Записи k(j+1) присваивается значение temp.

i-й шаг алгоритма завершен.

2. Сортировка вставкой

Метод выбора очередного элемента из исходного массива произволен; может использоваться практически любой алгоритм выбора. **Обычно** (и с целью получения устойчивого алгоритма сортировки), **элементы вставляются по порядку их появления во входном массиве**. С целью сохранения **естественности** алгоритма поиск места для вставки в упорядоченной части массива осуществляется от конца этой части к началу.

Алгоритм может сортировать список по мере его получения.

На каждом шаге алгоритма

- **выбираем** один из элементов входных данных и
- **вставляем** его на нужную позицию в уже отсортированном списке,
- до тех пор, пока набор входных данных не будет исчерпан.

2. Сортировка вставкой

Сортировка вставкой (insertion sort) — алгоритм со **сложностью порядка O(n^2).**

Особенности алгоритма

- **прост** в реализации;
- **эффективен на небольших наборах данных**, на наборах данных до десятков элементов может оказаться лучшим;
- **эффективен на наборах данных, которые уже частично отсортированы**;
- устойчивый алгоритм сортировки (**не меняет порядок элементов, которые уже отсортированы**);
- не требует временной памяти, даже под стек;
- **может сортировать список по мере его получения.**

2. Сортировка вставкой

Пример

ПРИМЕР СОРТИРОВКИ МЕТОДОМ ПРОСТЫХ ВСТАВОК

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

2. Сортировка вставкой

Полное число шагов, которые потребуется выполнить, составляет 1 + 2 + 3 + … + (n - 1), то есть O(n^2). Фактически, этот алгоритм не слишком быстр даже в сравнении с другими алгоритмами порядка O(n^2), такими как сортировка выбором.

Достаточно много времени тратится на поиск правильного положения для нового элемента.

Усовершенствование алгоритма

Применение эффективного алгоритма поиска (в уже отсортированной части массива!) ускоряет выполнение алгоритма сортировки вставкой.

Пример. Отсортировать массив 6, 1, 8, 2, 5, 3 методом вставки.

Вставка в связных списках

Можно использовать вариант сортировки вставкой для упорядочения элементов не в массиве, а в связном списке. Алгоритм ищет требуемое положение элемента в возрастающем связном списке, и затем помещает туда новый элемент, используя операции работы со связными списками.

Наилучший случай для этого алгоритма достигается, когда исходный список первоначально отсортирован в обратном порядке. В этом случае каждый последующий элемент меньше, чем предыдущий, поэтому алгоритм помещает его в начало отсортированного списка, при условии, что *список просматривается с начала*. При этом требуется выполнить только одну операцию сравнения элементов, и **в наилучшем случае время выполнения алгоритма будет порядка O(n).**

В **наихудшем и среднем случаях** вычислительная сложность алгоритма **порядка O(n^2).**

Преимущество использования связных списков для вставки в том, что при этом *перемещаются только указатели, а не сами записи данных*. **Передача указателей может быть быстрее, чем копирование записей целиком, если элементы представляют собой большие структуры данных.**

3. Сортировка пузырьком

Сортировка простыми обменами, сортировка пузырьком (***bubble sort***) — **простой** алгоритм сортировки. Для понимания и реализации этот алгоритм — простейший, но **эффективен он лишь для небольших массивов**. Сложность алгоритма: $O(n^2)$.

Алгоритм

Алгоритм состоит в повторяющихся проходах по сортируемому массиву. За каждый проход

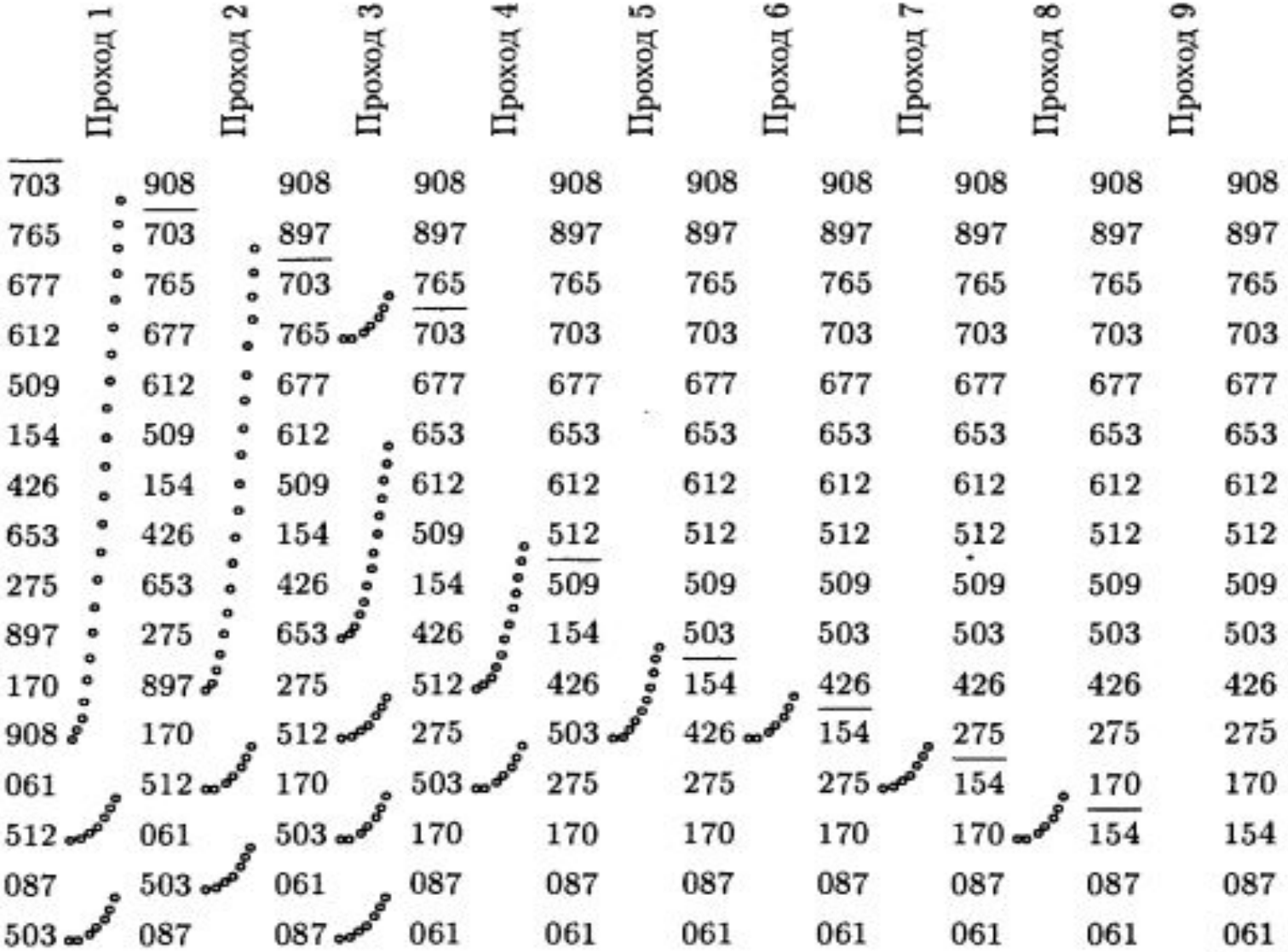
- **элементы последовательно сравниваются попарно** и, если порядок в паре неверный, **выполняется обмен** элементов.
- Проходы по массиву повторяются до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован.

При проходе алгоритма от начала к концу массива, больший элемент, стоящий не на своём месте, «всплывает» до нужной позиции как пузырёк в воде, отсюда и название алгоритма. В противоположность «методу погружения» (методу простых вставок), в котором элементы погружаются на соответствующий уровень.

3. Сортировка пузырьком

Пример. Сортировка пузырьком

Последовательность чисел представлена вертикально: первый элемент – внизу, последний – вверху.



Процесс сортировки методом пузырька.