МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Нижегородский государственный технический университет им. Р.Е. Алексеева»

Кафедра "Информационные радиосистемы"

Методы сортировки

Методические указания к лабораторной работе № 2
по дисциплине «Информационные технологии» для студентов
направления подготовки бакалавра 210400 «Радиотехника»
дневной формы обучения

Нижний Новгород
2012

Составители Е.Н.Приблудова, С.Б.Сидоров

УДК 621.325.5-181.4

Методы сортировки: метод. указания к лаб. работе по дисциплине «Информационные технологии» для студентов направления подготовки бакалавра 210400 «Радиотехника» дневной формы обучения / НГТУ; Сост.: Е.Н.Приблудова, С.Б.Сидоров. Н.Новгород, 2012, 11 с.

Сформулирована задача сортировки и рассмотрены области ее применения. Изложены краткие сведения о методах сортировки. Сформулированы задания и порядок выполнения для лабораторной работы.

Подп. к печ. . Формат $60\times84\,{}^{1}\!/_{16}$. Бумага газетная.

Печать офсетная. Печ.л. . Уч.-изд.л. . Тираж . Заказ .

Нижегородский государственный технический университет им. Р.Е.Алексеева
Типография НГТУ. 603950, Н.Новгород, ул.Минина, 24.

1. Цель работы

Изучение различных методов сортировки и проведение их сравнительного анализа.

2. Краткие сведения

2.1 Значимость и суть задачи

Сортировка - это процесс упорядочения некоторого множества элементов, на котором определены отношения порядка >, <, >=, <= (по возрастанию или убыванию).

Алгоритмы сортировки имеют большое практическое применение. Их можно встретить почти везде, где речь идет об обработке и хранении больших объемов информации. Некоторые задачи обработки данных решаются проще, если данные упорядочены.

Сортировка применяется во всех без исключения областях программирования, например, базы данных или математические программы.

Оценка алгоритмов сортировки

При выборе алгоритмов сортировки необходимо поставить перед собой вопрос. Существует ли наилучший алгоритм? Имея приблизительные характеристики входных данных, можно подобрать метод, работающий оптимальным образом.

Рассмотрим ***параметры***, по которым будет производиться оценка алгоритмов.

1. *Число операций сортировки (сравнения и перемещения)* - параметры, характеризующие быстродействие алгоритма.
2. *Объем памяти* – параметр, характеризующий использование алгоритмом дополнительной памяти.

Практически время выполнения алгоритма зависит не только от объема множества данных (размера задачи), но и от их значений. Например, время работы некоторых алгоритмов сортировки значительно сокращается, если первоначально данные частично упорядочены, тогда как другие методы оказываются нечувствительными к этому свойству. Чтобы учитывать этот факт, полностью сохраняя при этом возможность анализировать алгоритмы независимо от данных, различают:

- *максимальную сложность*, или сложность наиболее неблагоприятного случая, когда метод работает дольше всего;
- *среднюю сложность* - сложность метода в среднем (обычном) случае;

– *минимальную сложность* - сложность в наиболее благоприятном случае, когда метод справляется быстрее всего.

Алгоритмы могут иметь разную трудоемкость (количество операций с точностью до постоянного множителя), например, $T(n^2)$, $T(n*log(n))$, $T(n)$. Минимальная сложность всякого алгоритма сортировки не может быть меньше $T(n)$. Максимальная сложность метода, оперирующего сравнениями, не может быть меньше $T(n*log(n))$.

Рассмотрим временные сложности в порядке возрастания.

1. ***Сложность T(n).***

Такая сложность получается в обычных итерационных процессах, когда на каждом шаге метода задача размерности *n* сводится к задаче размерности *n-1*. Примером такой сложности может служить любой цикл, в котором число итераций прямо зависит от *n*.

2. ***Сложность T(n*log(n)).***

Эта сложность получается, если на каждом шаге метода выполняется некоторое количество действий *C*n*, то есть зависящее от размера задачи прямо пропорционально, и задача размерности *n* сводится к задаче размерностью *n/m*, где *m* – целая положительная константа.

3. ***Сложность $T(n^2)$.***

Такая сложность получается, когда мы имеем два вложенных цикла, число итераций, которых зависит от *n* прямо пропорционально. Или другими словами, на каждом шаге метода задача размерности *n* сводится к задаче размерности *n-1*, и при этом выполняется некоторое количество действий *C*n*, то есть зависящее от *n* прямо пропорционально.

Примеры алгоритмов с квадратичной сложностью будут рассмотрены ниже. Это все простые, но малоэффективные при больших *n* методы сортировок (в том числе получившая широкую известность пузырьковая сортировка).

Еще одним важным свойством алгоритма является его ***сфера применения***. Здесь основных позиций две:

– внутренняя сортировка;
– внешняя сортировка.

Внутренняя сортировка оперирует с массивами, помещающимися в оперативной памяти с произвольным доступом к любой ячейке. Данные обычно сортируются на том же месте, без дополнительных затрат.

Внешняя сортировка оперирует с запоминающими устройствами большого объема.

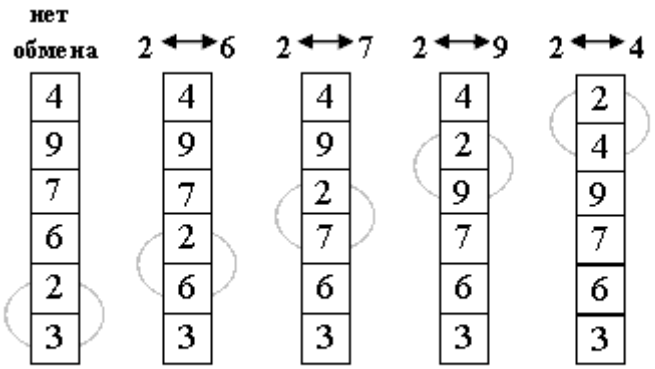
Проблемы оптимизации процесса сортировки различаются в обоих случаях: во внутренней сортировке стремятся сократить ***число сравнений*** и других внутренних операций, во внешней сортировке тоже, кроме числа сравнений, решающим фактором является ***количество операций по вводу и выводу*** дан-

ных, т.к. доступ к данным на носителе производится намного медленнее, чем операции с оперативной памятью.

Сортировка «пузырьком»

Расположим массив сверху вниз, от нулевого элемента - к последнему.

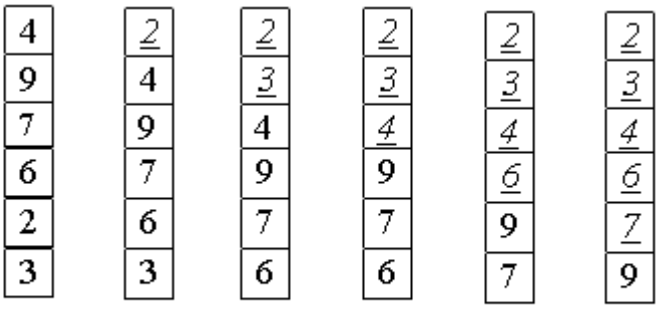
Идея метода: шаг сортировки состоит в проходе снизу вверх по массиву. По пути просматриваются пары соседних элементов. Если элементы некоторой пары находятся в неправильном порядке (верхний элемент пары больше нижнего), то меняем их местами.



Нулевой проход, сравниваемые пары выделены

После нулевого прохода по массиву "вверху" оказывается минимальный (самый "легкий") элемент - отсюда аналогия с пузырьком. Следующий проход делается до второго сверху элемента, таким образом, второй по величине элемент поднимается на правильную позицию.

Делаем проходы по все уменьшающейся нижней части массива до тех пор, пока в ней не останется только один элемент. На этом сортировка заканчивается, т. к. последовательность упорядочена по возрастанию.



Сложность алгоритма: Среднее число сравнений и перемещений имеют квадратичный порядок роста: $T(n^2)$, отсюда можно заключить, что алгоритм пузырька малоэффективен при больших n.

Тем не менее, у него есть громадный плюс: он прост и его можно улучшать.

Во-первых, рассмотрим ситуацию, когда на каком-либо из проходов не произошло ни одного обмена. Что это значит?

Это значит, что все пары расположены в правильном порядке, так что массив уже отсортирован. И продолжать процесс не имеет смысла (особенно, если массив был отсортирован с самого начала).

Итак, первое улучшение алгоритма заключается в запоминании, производился ли на данном проходе какой-либо обмен. Если нет - алгоритм заканчивает работу.

Процесс улучшения можно продолжить, если запоминать не только сам факт обмена, но и индекс последнего обмена *k*. Действительно: все пары соседних элементов с индексами, меньшими *k*, уже расположены в нужном порядке. Дальнейшие проходы можно заканчивать на индексе *k*, вместо того чтобы двигаться до установленной заранее верхней границы *i*.

Качественно другое улучшение алгоритма можно получить из следующего наблюдения. Хотя легкий пузырек снизу поднимется наверх за один проход, тяжелые пузырьки опускаются с минимальной скоростью: один шаг за итерацию. Так что массив 2 3 4 5 6 1 будет отсортирован за 1 проход, а сортировка последовательности 6 1 2 3 4 5 потребует 5 проходов.

Чтобы избежать подобного эффекта, можно менять направление следующих один за другим проходов. Получившийся алгоритм иногда называют ***"шейкер-сортировкой"*.**

Насколько описанные изменения повлияли на эффективность метода? Среднее количество сравнений уменьшилось, но остается $\boldsymbol{T(n^2)}$**.** Число перемещений уменьшилось лишь на незначительную величину. Максимальная сложность остается квадратичной**.**

Дополнительная память, очевидно, не требуется.

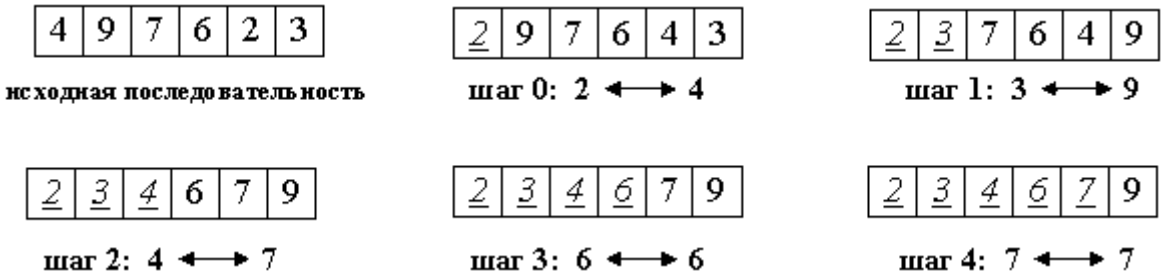
На практике метод пузырька, даже с улучшениями, для больших ***n*** работает слишком медленно.

Сортировка выбором

Идея метода состоит в том, чтобы создавать отсортированную последовательность путем присоединения к ней одного элемента за другим в правильном порядке.

Будем строить готовую последовательность, начиная с левого конца массива. Алгоритм состоит из *n* последовательных шагов, начиная от нулевого и заканчивая (*n*-1)-ым.

На *i*-м шаге выбираем наименьший из элементов *a*[*i*] ... *a*[*n*] и меняем его местами с *a*[*i*]. Последовательность шагов при *n*=6 изображена на рисунке ниже.



Вне зависимости от номера текущего шага i, последовательность $a[0]...a[i]$ (выделена курсивом) является упорядоченной. Таким образом, на (n-2)-ом шаге вся последовательность, кроме $a[n-1]$ оказывается отсортированной, а $a[n-1]$ стоит на последнем месте по праву: все меньшие элементы уже ушли влево.

Сложность алгоритма: $\boldsymbol{T(n^2)}$.

Для нахождения наименьшего элемента алгоритм совершает n-1 сравнений. С учетом того, что количество рассматриваемых на очередном шаге элементов уменьшается на единицу, общее количество операций: $\boldsymbol{T(n^2)}$.

Алгоритм не использует дополнительной памяти.

Особенностью данного метода является то, что он требует всего лишь n перемещений элементов (n - размер массива) и в то же время n^2 операций сравнения. Отсюда можно сделать вывод, что этот метод можно применять в случае, когда затраты на перемещение элементов существенно выше, чем затраты на их сравнение.

Сортировка вставками

Сортировка простыми вставками в чем-то похожа на вышеизложенные методы.

Аналогичным образом делаются проходы по части массива, и аналогичным же образом в его начале "вырастает" отсортированная последовательность.

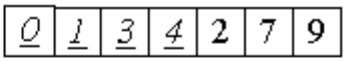
Однако в сортировке пузырьком или выбором можно было четко заявить, что на i-м шаге элементы $a[0]...a[i]$ стоят на правильных местах и никуда более не переместятся. Здесь же подобное утверждение будет более слабым: последовательность $a[0]...a[i]$ упорядочена. При этом по ходу алгоритма в нее могут вставляться все новые элементы.

Будем разбирать алгоритм, рассматривая его действия на i-м шаге. Последовательность к этому моменту разделена на две части: готовую $a[0]...a[i]$ и неупорядоченную $a[i+1]...a[n]$.

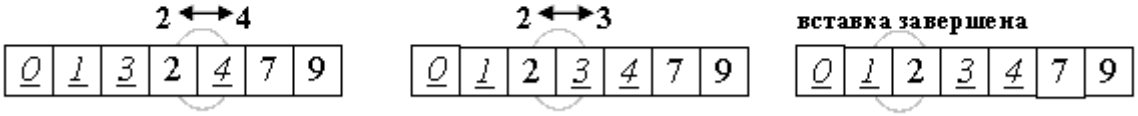
На следующем, (i+1)-ом каждом шаге алгоритма берем $a[i+1]$ и вставляем на нужное место в готовую часть массива.

Поиск подходящего места для очередного элемента входной последовательности осуществляется путем последовательных сравнений с элементом, стоящим перед ним.

В зависимости от результата сравнения элемент либо остается на текущем месте (вставка завершена), либо они меняются местами и процесс повторяется.



Последовательность на текущий момент. Часть a[0]..a[2] уже упорядочена.



Вставка числа 2 в отсортированную подпоследовательность. Сравниваемые пары выделены.

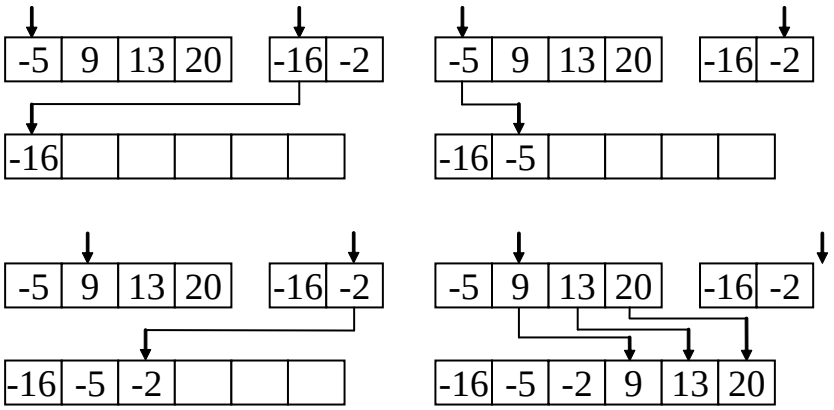
Таким образом, в процессе вставки мы "просеиваем" элемент *x* к началу массива. Останавливаемся в следующих случаях:

- найден элемент, меньший *x*;
- достигнуто начало последовательности.

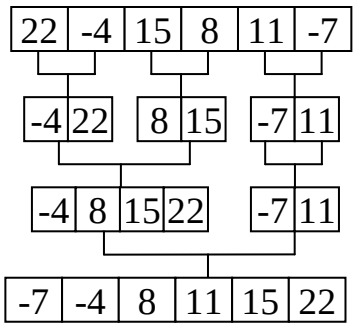
Среднее, а также худшее число сравнений и перемещений оцениваются как $\boldsymbol{T(n^2)}$, дополнительная память при этом не используется.

Сортировка слиянием

Сортировка слиянием является одним из самых быстрых способов сортировки. Она требует примерно $\boldsymbol{n*log(n)}$ сравнений и столько же перемещений элементов. Суть его заключается в следующем. Пусть имеются две упорядоченные последовательности по возрастанию. Эти последовательности можно специальным образом объединить, получив одну, состоящую из элементов обеих последовательностей и также упорядоченную по возрастанию. Способ объединения последовательностей называется слиянием. Для этого организуется последовательный перебор элементов из обеих последовательностей. На очередном шаге производится сравнение текущих элементов каждой из последовательностей. Если значение текущего элемента первой последовательности меньше значения текущего элемента второй, то во вспомогательный буфер заносится элемент из первой, и в первой последовательности переходим к следующему элементу. Аналогичные действия выполняем, если элемент из второй последовательности меньше элемента из первой. Слияние двух упорядоченных последовательностей по возрастанию изображено на рисунке ниже.



Исходную неупорядоченную последовательность, подлежащую сортировке, будем рассматривать как совокупность упорядоченных групп по одному элементу в каждой. Используя процедуру слияния соседних групп, получаем новые упорядоченные группы. Далее, полученные на предыдущем шаге группы укрупняются тем же способом и т.д. Понятно, что наступит такой момент, когда останется только одна упорядоченная группа. Это и означает, что наша исходная последовательность упорядочена. Ниже приведена схема алгоритма сортировки последовательности слиянием.



Результатом является упорядоченная последовательность, находящаяся в буфере. Малое количество операций сравнения и перемещения делает рассмотренный способ сортировки очень привлекательным для практического использования.

Однако, несмотря на хорошее общее быстродействие, у сортировки слиянием есть и минус: она требует дополнительной памяти такого же размера, что и исходная последовательность.

Сортировка слиянием является одним из наиболее эффективных методов, например, для файлов, когда есть лишь последовательный доступ к элементам.

3. Задания и порядок выполнения

Варианты заданий определяются преподавателем. В лабораторной работе необходимо выполнить ***внутреннюю*** сортировку данных, представленных в виде ***массивов***.

Перед выполнением задания:

- изучить рассмотренные методы сортировки и их сравнительный анализ;
- изучить постановку задачи;
- составить алгоритм для решения задачи с использованием следующих форм представления алгоритма: псевдокод и блок-схема.

4. Контрольные вопросы

1. Что такое сортировка?
2. Перечислите основные методы сортировки.
3. Приведите примеры практического использования сортировки.
4. По каким параметрам можно оценить алгоритмы сортировки?
5. Что такое трудоемкость алгоритма?
6. Чем отличаются внешняя и внутренняя сортировка?
7. Проведите сравнительный анализ эффективности алгоритмов сортировки.
8. Что объединяет методы сортировки «пузырьком», вставками и выбором?
9. В каком методе сортировки необходима дополнительная память?

5. Список рекомендуемой литературы

1. Павловская Т.А. C/C++. Программирование на языке высокого уровня / Т.А. Павловская.- Спб.: Питер, 2005.
2. Шилдт Г. Полный справочник по C: Пер.с англ. / Г. Шилдт. - 4-е изд. - М.: Изд.дом "Вильямс", 2008.
3. Кнут Д.Э. Искусство программирования: Пер.с англ.:В 3-х т. Т.3 : Сортировка и поиск / Д. Э. Кнут. - 2-е изд. - М.: Вильямс, 2003.