МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Брянский государственный технический университет

Утверждаю
Ректор университета
_____О.Н. Федонин
«_____»_____2013 г.

ЯЗЫКИ ПРОГРАММИРОВАНИЯ

МЕТОДЫ СОРТИРОВКИ ДАННЫХ

Методические указания
к выполнению лабораторной работы
для студентов очной формы обучения
специальностей 090303 – «Информационная безопасность автоматизированных систем»,
090900 – «Информационная безопасность»

Брянск 2013

УДК 004.43

Языки программирования. Методы сортировки данных: методические указания к выполнению лабораторной работы для студентов очной формы обучения специальностей 090303 – «Информационная безопасность автоматизированных систем», 090900 – «Информационная безопасность». – Брянск: БГТУ, 2013. – 19 с.

Разработали:
Ю.А. Леонов, к.т.н., доц.,
Е.А. Леонов, к.т.н., доц.

Научный редактор: Ю.М. Казаков
Редактор издательства: Л.И. Афонина
Компьютерный набор: Ю.А. Леонов

Рекомендовано кафедрой «Компьютерные технологии и системы» БГТУ (протокол № 2 от 19.09.2013)

Темплан 2013 г., п.

Подписано в печать Формат 60x84 1/16. Бумага офсетная.
Офсетная печать.
Усл. печ. л. 1,16 Уч. – изд. л. 1,16 Тираж 20 экз. Заказ Бесплатно

Издательство брянского государственного технического университета,
241035, Брянск, бульвар 50-летия Октября, 7, БГТУ. 58-82-49
Лаборатория оперативной полиграфии БГТУ, ул. Харьковская, 9

1. ЦЕЛЬ РАБОТЫ

Целью работы является изучение методов сортировки и приобретение практических навыков в программировании данных методов.

Продолжительность работы – 1ч.30мин.

2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

2.1. Сортировка данных

Алгоритмы сортировки информации образуют основу для огромного большинства прикладных программ. Сортировка информации является одной из стандартных функций, возникающих в процессе решения задач. *Сортировка данных* – это процесс изменения порядка расположения элементов в неупорядоченных структурах данных таким образом, чтобы обеспечить возрастание или убывание числового значения элемента данных или определенного числового параметра, связанного с каждым элементом данных (ключа), при переходе от предыдущего элемента к последующему. То есть для любой пары чисел должны быть определены отношения "больше" или "меньше".

Для переменных символьного типа понятия "возрастание" и "убывание" относятся к значениям машинного кода, используемого для представления символов в памяти компьютера. Так как все буквенные символы располагаются в таблице кодов по алфавиту, то сортировка слов текста всегда приводит к их упорядочению в алфавитной последовательности.

Сортировка данных используется для эффективного решения задач в программировании.

При решении задачи сортировки обычно выдвигается требование минимального использования дополнительной памяти, из которого вытекает недопустимость применения дополнительных массивов.

Для оценки быстродействия алгоритмов различных методов сортировки, как правило, используют два показателя:

- количество присваиваний;
- количество сравнений.

Все методы сортировки можно разделить на две большие группы:

- прямые методы сортировки;
- улучшенные методы сортировки.

Прямые методы сортировки по принципу, лежащему в основе метода, в свою очередь разделяются на три подгруппы:

1) сортировка обменом («пузырьковая» сортировка);
2) сортировка выбором (выделением);
3) сортировка вставкой (включением).

Улучшенные методы сортировки основываются на тех же принципах, что и прямые, но используют некоторые оригинальные идеи для ускорения процесса сортировки. Прямые методы на практике используются довольно редко, так как имеют низкое быстродействие относительно улучшенных методов. Однако они хорошо показывают суть основанных на них улучшенных методов. Кроме того, в некоторых случаях (как правило, при небольшой длине массива и/или особом исходном расположении элементов массива) некоторые из прямых методов могут даже превзойти улучшенные методы.

2.1.1. Сортировка обменом (bubble sort, «пузырьком»)

Принцип метода

Слева направо поочередно сравниваются два соседних элемента, и если их взаиморасположение не соответствует заданному условию упорядоченности, то они меняются местами. Далее берутся два следующих соседних элемента и так далее до конца массива.

После одного такого прохода на последней *n*-й позиции массива в случае сортировки по возрастанию будет стоять максимальный элемент («всплыл» первый «пузырек»). Поскольку максимальный элемент уже стоит на своей последней позиции, то второй проход обменов выполняется до *n*-1 элемента. И так далее. Всего требуется *n*-1 проход.

Например, для массива «5 3 2 8 7 1» сортировка обменом по возрастанию будет проходить следующим образом:

- исходное положение: 5 3 2 8 7 1;
- 1 итерация: **3 5** 2 8 7 1;
- 2 итерация: 3 **2 5** 8 7 1;
- 3 итерация: 3 2 **5 8** 7 1;
- 4 итерация: 3 2 5 **7 8** 1;
- 5 итерация: 3 2 5 7 **1 8**;

- 6 итерация: **2 3** 5 7 1 8;
- 7 итерация: 2 **3 5** 7 1 8;
- 8 итерация: 2 3 **5 7** 1 8;
- 9 итерация: 2 3 5 **1 7** 8;
- 10 итерация: **2 3** 5 7 1 8;
- 11 итерация: 2 **3 5** 7 1 8;
- 12 итерация: 2 3 **5 7** 1 8;
- 13 итерация: 2 3 5 **1 7** 8;
- 14 итерация: **2 3** 5 1 7 8;
- 15 итерация: 2 **3 5** 1 7 8;
- 16 итерация: 2 3 **1 5** 7 8;
- 17 итерация: **2 3** 1 5 7 8;
- 18 итерация: 2 **1 3** 5 7 8;
- конечный результат: **1 2** 3 5 7 8.

Пример сортировки методом обмена

```
static void ChangeSort(int[] A)
{
    for (int i = 0; i < A.Length; i++)
        for (int j = 0; j < A.Length - 1 - i; j++)
            if (A[j] > A[j + 1])
            {
                int tmp = A[j];
                A[j] = A[j + 1];
                A[j + 1] = tmp;
            }
}
```

В среднем число операций сравнений равно n^2, где n – количество элементов в массиве.

2.1.2. Сортировка выбором (selection sort)

Принцип метода

При сортировке выбором находится наименьший элемент на интервале от первого до последнего элемента массива и делается его обмен с первым элементом массива. Затем находится минимальный элемент на интервале от второго до последнего и делается его обмен со вторым элементом и т.д. до обмена двух последних элементов.

Например, если сортировку выбором применить для массива «5 3 2 8 7 1», то будут получены следующие проходы:

- исходное положение: 5 3 2 8 7 1;
- 1 итерация: **1** 3 2 8 7 **5**;
- 2 итерация: 1 **2 3** 8 7 5;
- 3 итерация: 1 2 **3** 8 7 5;
- 4 итерация: 1 2 3 **5** 7 **8**;
- конечный результат: 1 2 3 5 7 8.

Пример сортировки выбором

```
static void ChoiceSort(int[] A)
{
    for (int i = 0; i < A.Length; i++)
    {
        int indMin = i;
        int min = A[indMin];
        for (int j = i + 1; j < A.Length; j++)
            if (A[j] < min)
            {
                indMin = j;
                min = A[j];
            }
        A[indMin] = A[i];
        A[i] = min;
    }
}
```

В среднем число операций сравнений равно n^2, где n – количество элементов в массиве.

2.1.3. Сортировка вставками (insertion sort)

Принцип метода

При сортировке вставками массив разделяется на две части: отсортированную и неотсортированную. В начале сортировки первый элемент массива (крайний слева) находится в отсортированной части, а остальные элементы в неотсортированной. Последовательно берется первый элемент из неотсортированной части и ищется позиция вставки его в отсортированной части. Для того чтобы вставить элемент необходимо сдвинуть группу элементов начиная от найденной позиции вставки до последнего элемента в

отсортированной части включительно на один вправо. Таким образом, позиция вставки освобождается для вставки взятого элемента из неотсортированной части. После проделанных операций отсортированная часть увеличилась на один элемент, а неотсортированная часть уменьшилась на один элемент. Описанные действия повторяются до тех пор, пока все элементы из неотсортированной части не перейдут в отсортированную часть массива.

Например, для массива «5 3 2 8 7 1» сортировка вставками будет проходить следующим образом:

- исходное положение: 5 | 3 2 8 7 1;
- 1 итерация: **3** 5 | 2 8 7 1;
- 2 итерация: **2** 3 5 | 8 7 1;
- 3 итерация: 2 3 5 **8** | 7 1;
- 4 итерация: 2 3 5 **7** 8 | 1;
- конечный результат: **1** 2 3 5 7 8 |.

Пример сортировки вставками

```
static void InsertSort(int[] A)
{
    for (int i = 1; i < A.Length; i++)
    {
        int elem = A[i];
        int pos = 0;
        while (elem > A[pos]) pos++;
        for (int j = i - 1; j >= pos; j--) A[j + 1] = A[j];
        A[pos] = elem;
    }
}
```

Среднее, а также худшее число сравнений и перестановок оцениваются как n^2. Хорошим показателем сортировки является весьма естественное поведение: почти отсортированный массив будет отсортирован очень быстро.

2.1.4. Быстрая сортировка (quick sort)

Принцип метода

В основе быстрой сортировки лежит принцип разбиения. Сначала выбирается опорный элемент *x*, относительно которого

переупорядочиваются остальные элементы массива. Все элементы, меньшие *x*, переставляются перед *x*, а больше или равные *x* – после. В итоге массив оказывается, разбит на две части. Далее к первой и второй частям рекурсивно применяется описанный алгоритм до тех пор, пока в каждой части не останется по одному элементу. Желательно выбрать элемент *x* таким, чтобы количество элементов в первой и во второй части было примерно одинаковым.

Например, для массива «5 3 2 8 7 1» в качестве опорного элемента выбираем средний элемент массива с индексом (*low* + *high*) / 2 = (0 + 5) / 2 = 2. В результате сортировки для первого рекурсивного вызова будут получены следующие проходы:

- начальное положение: 5 3 2 8 7 1;
- 1 итерация: **1** 3 2 8 7 **5**;
- 2 итерация: 1 **2** **3** 8 7 5;
- новые рекурсивные вызовы для двух частей: 1 2; 3 8 7 5;
- конечный результат: 1 2 3 5 7 8.

Пример быстрой сортировки

```
static void QuickSort(int[] A, int low, int high)
{
    int i = low;
    int j = high;
    int x = A[(low + high) / 2];
    do {
        while (A[i] < x) i++;
        while (A[j] > x) j--;
        if (i <= j)
        {
            int temp = A[i];
            A[i] = A[j];
            A[j] = temp;
            i++; j--;
        }
    } while (i < j);
    if (low < j) QuickSort(A, low, j);
    if (i < high) QuickSort(A, i, high);
}
```

В приведенном алгоритме *low* и *high* – левая и правая границы сортируемой части массива, (*low* + *high*) / 2 – индекс среднего элемента.

Среднее число сравнений и перестановок оцениваются как $n \cdot \log n$. В худшем случае число сравнений и перестановок равняется n^2.

2.1.5. Сортировка слиянием (merge sort)

Принцип метода

Массив разделяется на две части, каждая из этих частей разделяется также на две части и так далее, до тех пор, пока в каждой из частей массива не будет по одному элементу. При разбиении используется рекурсивный алгоритм.

Далее после того как рекурсия достигнет максимальной глубины происходит слияние каждой из двух частей в порядке сортировки. Сначала объединяются самые мелкие части массива, состоящие из одного элемента в двойки, затем двойки на более высоком уровне рекурсии объединяются в четвертки и т.д. до тех пор, пока алгоритм не выйдет на начальный вызов рекурсивной функции, когда слияние произойдет всего массива.

Например, для массива «5 3 2 8 7 1 6 4» на начальном уровне массив разобьется на две части: «5, 3, 2, 8», «7, 1, 6, 4» и так далее пока на нижних уровнях рекурсии массив не будет представлять собой совокупность частей по одному элементу: «5», «3», «2», «8», «7», «1», «6», «4». Затем соседние элементы объединяются в двойки согласно условию сортировки в нашем случае по возрастанию: «3, 5», «2, 8», «1, 7», «4, 6». Далее в двойки объединяются в четверки: «2, 3, 5, 8», «1, 4, 6, 7». И в начальном рекурсивном вызове массив будет отсортирован: «1, 2, 3, 4, 5, 6, 7, 8».

Пример сортировки слиянием

```
static void MergeSort(int[] A, int low, int high)
{
    if (low < high)
    {
        int center = (low + high) / 2;
        MergeSort(A, low, center);
        MergeSort(A, center + 1, high);
        Merge(A, low, center, high);
```

```
        }
    }

    static void Merge(int[] A, int low, int center, int high)
    {
        int i = low, j = center + 1, tmpPos = 0;
        int[] tmp = new int[high - low + 1];

        while (i <= center && j <= high)
        {
            if (A[i] < A[j]) tmp[tmpPos++] = A[i++];
            else tmp[tmpPos++] = A[j++];
        }

        while (j <= high) tmp[tmpPos++] = A[j++];
        while (i <= center) tmp[tmpPos++] = A[i++];

        for (tmpPos = 0; tmpPos < tmp.Length; tmpPos++)
            A[low + tmpPos] = tmp[tmpPos];
    }
```

В данном примере рекурсивная функция *MergeSort* разбивает массив на части и вызывает для парных частей функцию *Merge* для объединения массивов в порядке сортировки.

Среднее число сравнений и перестановок оцениваются как $n \cdot \log n$, при этом отсутствует худший случай. Недостатком этого алгоритма является использование сравнительно большого количества дополнительной памяти.

3. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Для выполнения лабораторной работы необходимо первоначально ознакомиться с теоретической частью. В случае если материала представленного в теоретической части недостаточно для выполнения задания необходимо воспользоваться лекциями и книгами по данной тематике.

Затем требуется написать программу на языке C#, выполняющую задачу, представленную в списке заданий.

При выполнении задания необходимо предусмотреть следующие этапы работы программы:

1) выделение памяти под массив;

2) заполнение массива неупорядоченными данными;
3) вывод неупорядоченных данных массива на экран;
4) сортировка массива указанным методом;
5) вывод упорядоченных данных массива на экран.

4. СПИСОК ЗАДАНИЙ

№ варианта	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Задание	5A ↑	4B ↑	3C ↑	2D ↑	1E ↑	4A ↑	3B ↑	2C ↑	1D ↑	5E ↑	3A ↑
№ варианта	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**
Задание	2B ↑	1C ↑	5D ↑	4E ↑	2A ↓	1B ↓	5C ↓	4D ↓	3E ↓	1A ↓	5B ↓
№ варианта	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**			
Задание	3C ↓	3D ↓	2E ↓	3A ↓	4B ↓	5C ↓	4D ↓	5E ↓			

Примечание.

1. Методы сортировки:
1 – обменом;
2 – выбором;
3 – вставками;
4 – быстрая;
5 – слиянием.

2. Содержимое массива:
A – прописные латинские;
B – строчные латинские;
C – прописные русские;
D – строчные русские;
E – вещественные числа.

3. Направление сортировки:
↑ - по возрастанию;
↓ - по убыванию.

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что называют сортировкой данных?
2. Как оценивают эффективность алгоритмов сортировки?
3. Какие вы знаете прямые методы сортировки, в чем принцип каждого из этих методов?
4. Приведите примеры улучшенных методов сортировки.
5. В каких методах сортировки используются рекурсивные алгоритмы?
6. В чем разница метода быстрой сортировки от метода сортировки слиянием?

7. Расскажите принцип работы сортировок: обменом, выбором, вставкой.
8. Расскажите принцип работы сортировок: быстрая, слиянием.
9. Покажите графически (по отдельным итерациям) работу всех рассмотренных методов сортировки в данной лабораторной работе.

6. СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

Основная

1. Павловская Т.А. C#. Программирование на языке высокого уровня. – Изд.: Питер, 2009. – 432с.
2. Троелсен, Э. Язык программирования C# 2010 и платформа .NET 4. – Изд.: Вильямс, 2011. – 1392с.
3. Нейгел, К., C# 4.0 и платформа .NET 4 для профессионалов / Билл Ивьен, Джей Глинн, Карли Уотсон, Морган Скиннер. – Изд.: Питер, 2011. – 1440с.
4. Кнут, Д.Э. Искусство программирования. Сортировка и поиск. Том 3. – Изд.: Вильямс, 2012. – 824с.
5. Седжвик, Р. Алгоритмы на C++. – Изд.: Вильямс, 2011. – 1056с.

Дополнительная

6. Головешкин, В.А., Ульянов, М.В. Теория рекурсии для программистов. – Изд.: ФИЗМАТЛИТ, 2006. – 296с.
7. Марченков С.С. Рекурсивные функции. – Изд.: ФИЗМАТЛИТ, 2007. –64с.
8. Джесс Либерти. Программирование на C#. – Изд.: КноРус, 2003. – 688с.
9. Харви Дейтел. C# в подлиннике. Наиболее полное руководство. – Изд.: БХВ-Петербург, 2006. – 1056с.