МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский
Нижегородский государственный университет им. Н.И. Лобачевского»
(ННГУ)
Институт информационных технологий, математики и механики

ЛАБОРАТОРНАЯ РАБОТА
«Изучение методов сортировки числовых массивов»

Выполнил: студент группы 381603-4

Леонтьев Вадим Евгеньевич

Подпись _____

Руководитель:

Барышева Ирина Викторовна

Подпись _____

Нижний Новгород
2016

Оглавление

Введение.

Говорят, что 90% времени работы программы приходится на так называемые узкие места, занимающие в общем объеме программы не более 10% команд. Поиск таких мест и улучшение их временных характеристик - один из основных методов совершенствования программ.

К числу таких узких мест относятся фрагменты программ, занимающиеся упорядочением достаточно объемной информации. Поэтому сортировкам уделяется большое внимание.

Сортировка - достаточно хороший пример задачи, которую можно решать с помощью различных алгоритмов. В данной работе я рассмотрю различные методы сортировок. У каждой из них есть свои достоинства и недостатки, а выбор алгоритма зависит из конкретной постановки задачи.

Алгоритмы сортировки имеют большое практическое применение. Их можно встретить почти везде, где речь идет об обработке и хранении больших объемов информации. Очевидно, что с "отсортированными" данными работать легче и быстрее, чем с произвольно расположенными. Когда элементы "отсортированы", проще найти и быстрее, например, найти слово в словаре на 700 страниц.

Наш первоначальный интерес к сортировке основывается на том, что при построении алгоритмов мы сталкиваемся со многими, весьма фундаментальными приемами. В частности, сортировка – это идеальный объект для демонстрации огромного разнообразия алгоритмов, все они изобретены для одной и той же задачи.

Важность темы "сортировки" определяется и особой ролью таблиц. Все применения ЭВМ основаны на их способности к быстрой и точной обработке больших объемов информации, а это возможно только когда информация однородна и отсортирована. Таким образом, таблицы как основное средство представления однородной информации неизбежно используются во всех реальных компьютерных программах. На табличном принципе основана и архитектура современных ЭВМ: память машины можно рассматривать как большой массив байтов, адреса которых располагаются по возрастанию.

Сортировку массивов следует понимать как процесс перегруппировки элементов массива с целью облегчения их упорядоченности, для облегчения последующего поиска элементов в массиве. Основное условие: выбранный метод сортировки массивов должен экономно использовать доступную память. Поэтому это еще и идеальный объект, демонстрирующий необходимость анализа производительности алгоритмов. К тому же на примерах сортировок можно показать, как путем усложнения алгоритма можно добиться значительного выигрыша в эффективности.

Хорошие алгоритмы сортировки массивов требуют порядка n*log2n (к ним относится такие сортировки, как: сортировка вставками, слиянием), а для простых методов, их называют прямыми, требуются порядка n2 (к ним относится такие сортировки, как: пузырьковые сортировки, и немногим быстрее сортировка Шелла и поиском минимального элемента) сравнений элементов. Прямые методы особенно удобны для объяснения характерных черт основных принципов большинства сортировок. Программы этих методов легко понимать, и они коротки. Усложненные методы требуют небольшого числа операций, но эти операции обычно сами более сложны, и поэтому для достаточно малых N прямые методы оказываются быстрее, хотя при больших N их использовать, конечно, не следует.

Рассмотрю описание нескольких методов упорядочения числовых массивов и тексты реализующих их процедур.

1. Постановка задачи

На данный момент не существует алгоритма сортировки, который является лучше других, иначе бы ими перестали пользоваться. Поэтому целью моей работы является проведение эксперимента, по итогам которого можно будет установить преимущества и недостатки различных методов сортировок на различных данных.

Эксперимент будет проведён на различных, по способу формирования, данных, таких как:

1. Константный (постоянный) массив;
2. Привычным считаются сортировки на случайном (рандомном) массиве;
3. Известно, что некоторые сортировки разного вида пузырька плохи на случайном массиве, поэтому использую массив из файла.

По каждому из способов формирования массива, проверю работу следующих сортировок:

1. Сортировка «пузырьком»;
2. Сортировка «улучшенным пузырьком»;
3. Сортировка Шелла;
4. Сортировка поиском минимального элемента;
5. Сортировка вставками;
6. Сортировка слиянием.

Критерии, по которым сравниваются сортировки:

1. Время сортировки - основной параметр, характеризующий быстродействие алгоритма;
2. Память – ряд алгоритмов требует выделения дополнительной памяти под временное хранение данных. При оценке используемой памяти не будет учитываться место, которое занимает исходный массив и независящие от входной последовательности затраты, например, на хранение кода программы;
3. Устойчивость - устойчивая сортировка не меняет взаимного расположения равных элементов. Такое свойство может быть очень полезным, если они состоят из нескольких полей, а сортировка происходит по одному из них;
4. Естественность поведения - эффективность метода при обработке уже отсортированных, или частично отсортированных данных. Алгоритм ведет себя естественно, если учитывает эту характеристику входной последовательности и работает лучше.

В качестве результата будет выводиться на экран:

1. Выбранный способ формирования данных;
2. Таблица типа: «Название сортировки» - «время её работы».

2. Структура проекта

Для решения поставленной задачи разработан проект (рисунок 1).

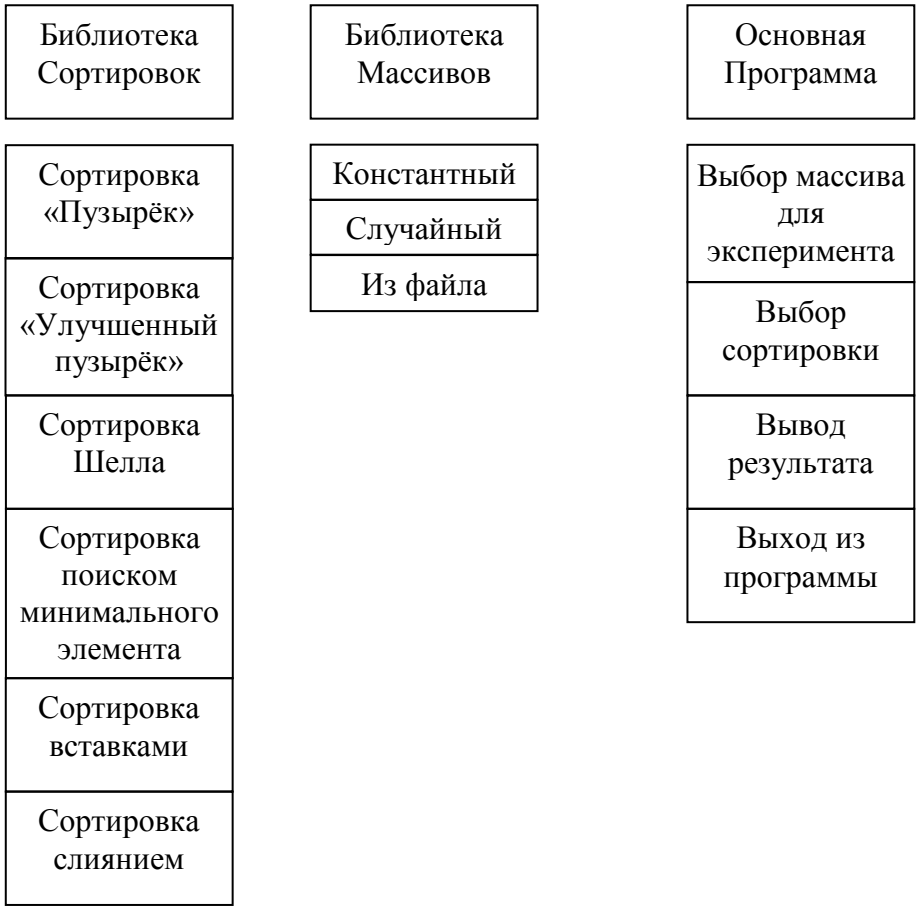

Рисунок 1. Схема проекта.

Проект содержит основную программу, запускающую диалоговое меню с пользователем в котором содержатся пункты: Выбор массива для эксперимента, Выбор сортировки, Вывод результата, Выход из программы. К основной программе подключены 3 библиотеки (Библиотека работы с графическими окнами, сортировок и массивов), которые обеспечивают функционирование основной программы.

Библиотека сортировок содержит процедуры шести сортировок (сортировка «пузырёк», сортировка «улучшенный пузырёк», сортировка Шелла, сортировка поиском минимального элемента, сортировка вставками и сортировка слиянием). Исходные данные у всех сортировок одинаковые – это массив, содержащий числа, количество этих чисел и максимально допустимый размер этого массива.

Библиотека массивов содержит подпрограммы формирования массива (константный – состоящий из семнадцати элементов, случайный – создаваемый рандомно из пользовательского числа количества элементов).

Далее описаны структуры библиотек (сортировок и массивов), процедур сортировок, и функций, задающие формирование массивов.

2.1. Библиотека сортировок

Библиотека содержит процедур сортировок. Реализации общей (“открытой”) части библиотеки сортировок на языке программирования Pascal приведена в приложении (Библиотека сортировок).

3.1.1. Сортировка «Пузырёк»

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются ‘N – 1’ раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован.

При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции, как пузырёк в воде, отсюда и название алгоритма). Эффективен лишь для небольших массивов.

Описание алгоритма:

1. Определим число пар соседних элементов, которые участвуют в сравнении L:= n-1;
2. До тех пор пока все не упорядоченно:
    2.1. Для каждой пары организуем сравнение двух соседних элементов, если соседи стоят неправильно:
        2.1.1. Меняем соседей местами;
        2.1.2. Фиксируем факт перестановки;
    2.2. Уменьшаем L на 1, так как ещё один элемент встал на своё место.

Описание алгоритма сортировки в ‘unit’ на языке программирования Pascal приведено в приложении (Сортировка «Пузырёк»).

3.1.2. Сортировка «Улучшенный пузырёк»

Представляет собой улучшенный метод «пузырька».

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 лёгкий пузырек снизу поднимается 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»

1. Положим номер элемента, который сравниваем с соседом R=n-1;
2. Положим номер элемента, с которого начинаем сравнение L=n-1;
3. До тех пор, пока все не встанут на свои места:
    3.1. Фиксируем признак наличия перестановки F:=1;
    3.2. Для каждого элемента, начиная с номера L до номера R, начинаем сравнение с его соседом. Если соседи стоят не правильно, тогда:

3.2.1 Меняем соседей местами;

3.2.2.Фиксируем признак наличия перестановки F:=1;

3.3. Уменьшаем R на 1, так как большой встал на свое место. Если перестановки были:

3.3.1. F:=0;

3.3.2. Для каждого элемента массива, начиная с right до L+1, организуем сравнение с его левым соседом. Если соседи стоят не правильно:

3.3.2.1. Меняем соседей местами;

3.3.2.2. Фиксируем факт перестановки F:=1;

3.3.3. Увеличиваем L на 1, так как ещё один меньший встал на свое место.

Описание алгоритма сортировки в ‘unit’ на языке программирования Pascal приведено в приложении (Сортировка «Улучшенный пузырёк»).

3.1.3. Сортировка Шелла

Идея метода Шелла состоит в сравнении элементов, стоящих не только рядом, но и на определённом расстоянии друг от друга. При сортировке Шелла сначала сравниваются и сортируются между собой значения, стоящие один от другого на некотором расстоянии d. После этого процедура повторяется для некоторых меньших значений d, а завершается сортировка Шелла упорядочиванием элементов при $d = 1$. Эффективность сортировки Шелла в определённых случаях обеспечивается тем, что элементы «быстрее» встают на свои места. Описание алгоритма:

1. Положим шаг между сравниваемыми элементами равным m/2. Положим число L:=L-1, до тех пор, пока все элементы не встали на свои места и при этом k:=1:

1.1. Установим признак упорядоченности F:=0;

1.2. Для каждой пары организуем сравнение a[i] с a[i+k], при условии, что a[i+k]=b, если сравниваемые элементы стоят не правильно, тогда поменяем a[i] с a[i+k] местами. Установим признак упорядоченности F:=1;

2. Если k:=1, тогда уменьшим L на 1;

3. Изменим k по формуле k=(k+1) div 3.

Описание алгоритма сортировки в ‘unit’ на языке программирования Pascal приведено в приложении (Сортировка Шелла).

3.1.4. Сортировка поиском минимального элемента

Сортировка поиском минимального элемента может быть как устойчивый, так и неустойчивый. На массиве из n элементов имеет время T=(n-1), предполагая, что сравнения делаются за постоянное время.

Описание алгоритма:

1. Определим номер элемента, начиная с которого ведём поиск минимального

2. Для каждого к от 1 до N-1:

2.1. Найдём минимальный и его место

2.2. Поменяем местами минимальный элемент с a[k]

Описание алгоритма сортировки в ‘unit’ на языке программирования Pascal приведено в приложении (Сортировка поиском минимального элемента).

3.1.5. Сортировка вставками

Сортировка выбором может быть как устойчивый, так и неустойчивый. На массиве из n элементов имеет время выполнения в худшем, среднем и лучшем случае $T=(n^2)$, предполагая, что сравнения делаются за постоянное время.

Шаги алгоритма:

1. находим номер минимального значения в текущем списке
2. производим обмен этого значения со значением первой неотсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции)
3. теперь сортируем хвост списка, исключив из рассмотрения уже отсортированные элементы

Описание алгоритма:

1. Положим d=1;
2. До тех пор, пока d<n: Положим
   - 2.1. k:=1;
   - 2.2. До тех пор, пока k<n:
     - 2.2.1. Элементы массива a начиная с k-того, d штук, если они есть, заносим в массив first.
     - 2.2.2. Элементы массива, начиная с k+d, d штук, если они есть, заносим в массив second.
   - 2.3. Вернем элементы массива first и second в массив a, начиная с k-того элемента так, чтобы они стояли по порядку.
3. Увеличим k, k=2+2*d
4. Увеличим d, d=d*2

Описание алгоритма сортировки в ‘unit’ на языке программирования Pascal приведено в приложении (Сортировка вставками).

3.1.6. Сортировка слиянием

Сортировка слиянием – алгоритм сортировки, который упорядочивает списки в определённом порядке. Эта сортировка — хороший пример использования принципа «разделяй и властвуй». Сначала задача разбивается на несколько подзадач меньшего размера. Затем эти задачи решаются с помощью рекурсивного вызова или непосредственно, если их размер достаточно мал. Наконец, их решения комбинируются, и получается решение исходной задачи.

Для решения задачи сортировки эти три этапа выглядят так:

1. Сортируемый массив разбивается на две части примерно одинакового размера;
2. Каждая из получившихся частей сортируется отдельно, например — тем же самым алгоритмом;
3. Два упорядочённых массива половинного размера соединяются в один.

Описание алгоритма:

1. Определим длину упорядоченных частей элементов массива a, h:=1;
2. До тех пор пока k<=n;
   - 2.1. Занесем элементы массива a, начиная с номера k, h штук (если они есть), в массив first и определим nf;
   - 2.2. Занесём элементы массива a, начиная с номера k+h, h штук(если они есть), в массив second и определим ns;
   - 2.3. Вернем элементы first, second в массив a с k-того элемента по алгоритму записанному в процедуре слияния.
3. Увеличим h вдвое.

2.2. Библиотека массивов

Библиотека массивов содержит процедуры образование массивов. Описание реализации библиотек на языке программирования Pascal приведена в приложении (Библиотека массивов).

2.3. Основная программа

Основная программа содержит порядок вызова меню в сочетании с подключенными библиотеками. Описание реализации программы на языке программирования Pascal приведена в приложении (Основная программа).

3. Инструкция для пользователя

После запуска программы следует выбрать пункт, из главного меню (рис. 2), «Выбор массива».

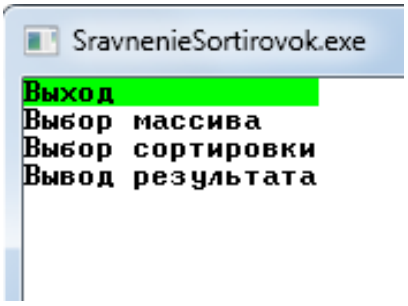

Рисунок 2. Главное меню.

Затем выбрать способ формирования массива (рис. 3).

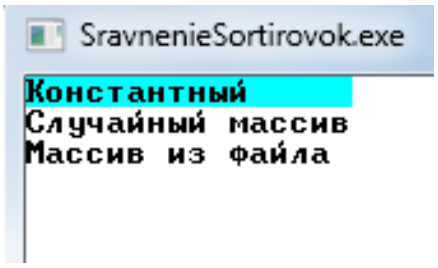

Рисунок 3. Выбор формирования массива

Потом в главном меню выбрать пункт «Выбор сортировки» (рис. 2)
Из предложенного списка (рис. 4) выбрать сортировку

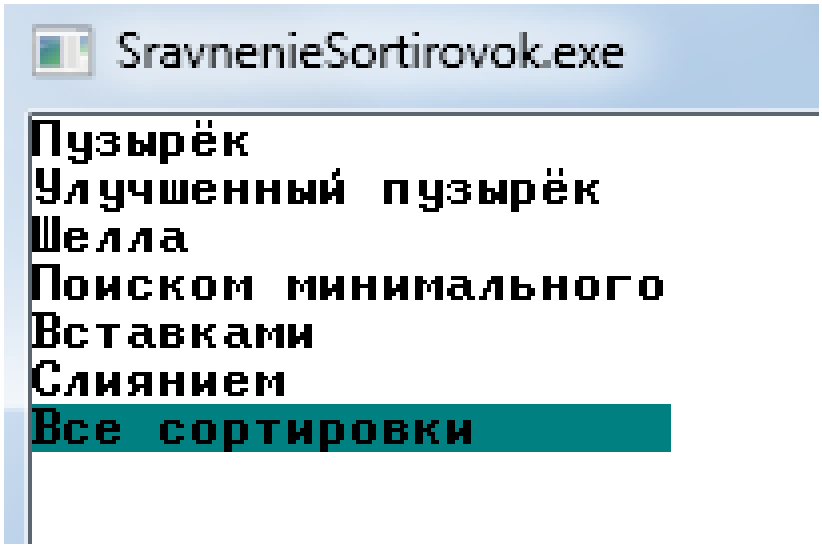

Рисунок 4. Меню сортировок.

В главном меню выбрать пункт «Вывод результата» и выбрать нужную сортировку (рис. 4)

4. Контрольный пример

Выбран: Случайный массив

Название сортировка	Время работы
«Пузырёк»	312
«Улучшенный пузырёк»	243
Шелла	125
Вставками	227
Слиянием	6

Выбран: Массив из файла

Название сортировка	Время работы
«Пузырёк»	1119
«Улучшенный пузырёк»	1083
Шелла	418
Вставками	2121
Слиянием	9

Заключение

В ходе лабораторной работы был проведен обзор существующих алгоритмов сортировки, в том числе оценка их эффективности.

Был сделан вывод, что методы сложных сортировок (сортировки использующие копирование массива), более эффективны в целом, чем методы простых сортировок.

В ходе сравнения было выявлено, что сортировка методом вставок эффективнее сортировки методом пузырька, благодаря меньшему числу сравнений ключей и меньшему количеству пересылок.

С помощью разработанной программы был проведен экспериментальный анализ сортировок.

Проведённые тесты показывают, что на случайном массиве сортировки работают быстрее, нежели на массиве из файла. Самой быстрой, а следовательно и лучшей, оказалась сортировка слияния, она показала лучшее время на обоих массивах. Самой медленной сортировкой на случайном массиве оказалась сортировка пузырьком, на массиве из файла – поиском минимального.

Список литературы

1. Кетков А., Кетков Ю. Практика программирования: Бейсик, Си, Паскаль. Самоучитель.–СПб.: БХВ-Петербург, 2001. -480 с.
2. Вирт Н. Алгоритмы и структуры данных. – М.: Мир. 2011. -360.
3. Кнут Д.Э. Искусство программирования. Т. 3. Сортировка и поиск.–М.:Вильямс, 2004. -824.
4. Фаронов В.В. Турбо Паскаль 7.0. Начальный курс. Учебное пособие. -М.: «ОМДГрупп», 2003. -616 с.
5. Кетков Ю. Л., Кетков А. Ю. Практика программирования: Бейсик, Си, Паскаль. Самоучитель. — СПб.: БХВ-Петербург, 2001. — 480 с.

Приложение

Библиотека сортировок

```
unit MyLibrarySort;

interface

const
   NMAX = 10000001;
   NSOR = 6;

type
   TepMasIn = array[1..Nmax] of integer;
   TepHalfArrInt = array[1..Nmax div 2] of integer;
   TepTimeSort = array[1..NSOR] of real;

var
   F, S, Rez, ArBab, ArBestBab, ArShella, ArMin, ArPaste, ArSlit: TepMasIn;
   TimeWorkSort: TepTimeSort; // Массив, сщдержащий время работы сортировок
   N, Nf, Ns, i, k: integer;

procedure SortBable(var ArBab: TepMasIn; N: integer);
procedure SortBestBable(var ArBestBab: TepMasIn; N: integer);
procedure SortSHella(var ArShella: TepMasIn; N: integer);
procedure SortMinim(var ArMin: TepMasIn; N: integer);
procedure SortPaste(var ArPaste: TepMasIn; N, k: integer);
procedure SortSlet(var ArSlit: TepMasIn; N: integer);
procedure Sliyanie(F: TepMasIn; S: TepHalfArrInt; var Rez: TepMasIn;
Nf, Ns, nk: integer);
```

Сортировка «Пузырёк»

```
procedure SortBable(A: TepArrInt; var ArBab: TepArrInt; N: integer);
var
   i, L, F, b: integer;
begin
   Milliseconds;
   begin
      L := N - 1;
      repeat;
         F := 0;
         for i := 1 to L do
            if ArBab[i] > ArBab[i + 1] then
            begin
               b := ArBab[i];
               ArBab[i] := ArBab[i + 1];
               ArBab[i + 1] := b;
               F := 1;
            end;
         L := L - 1;
      until(F = 0);
   end;
   TimeWorkSort[1] := MillisecondsDelta * 0.001;
   for i := 1 to N do
      B[i, 1] := A[i];
end;
```

Сортировка «Улучшенный пузырёк»

```
procedure SortBestBable(A: TepArrInt; var ArBestBab: TepArrInt; N: integer);
var
   Beg, Fin, F, i, b: integer;
begin
   Milliseconds;
   Beg := 1;
   fin := N - 1;
   repeat;
      F := 0;
      for i := Beg to Fin do
         if ArBestBab[i] > ArBestBab[i + 1] then
         begin
            b := ArBestBab[i];
            ArBestBab[i] := ArBestBab[i + 1];
            ArBestBab[i + 1] := b;
            F := 1;
         end;
      Fin := Fin - 1;
      if (F > 0) then
      begin
         F := 0;
         for i := Fin downto Beg do
            if ArBestBab[i] > ArBestBab[i + 1] then
            begin
               b := ArBestBab[i];
               ArBestBab[i] := ArBestBab[i + 1];
               ArBestBab[i + 1] := b;
               F := 1;
            end;
      end;
   until(F = 0);
   TimeWorkSort[2] := MillisecondsDelta * 0.001;
   for i := 1 to N do
      B[i, 2] := A[i];
end;
```

Сортировка Шелла

```
procedure SortSHella(A: TepArrInt; var ArShella: TepArrInt; N: integer);
var
   H, F, i, b: integer;
begin
   Milliseconds;
   H := N;
   repeat;
      H := (H div 3) + 1;
      F := 0;
      for i := 1 to N - H do
         if ArShella[i] > ArShella[i + H] then
         begin
            b := ArShella[i];
            ArShella[i] := ArShella[i + 1];
            ArShella[i + 1] := b;
            F := 1;
         end;
   until(F = 0) and (H = 1);   TimeWorkSort[3] := MillisecondsDelta * 0.001;
   for i := 1 to N do
      B[i, 3] := A[i];
end;
```

Сортировка поиском минимального элемента

```
procedure SortMinim(A: TepArrInt; var ArMin: TepArrInt; N: integer);
var
   L, k, min, i: integer;
begin
   Milliseconds;
   for k := 1 to N - 1 do
   begin
      min := ArMin[k];
      L := k;
      for i := k to N do
         if min > ArMin[i] then
         begin
            min := ArMin[i];
            L := i;
         end;
      ArMin[L] := ArMin[k];
      ArMin[k] := min;
   end;
   TimeWorkSort[4] := MillisecondsDelta * 0.001;
   for i := 1 to N do
      B[i, 4] := A[i];
      end;
```

Сортировка вставками

```
procedure SortPaste(A: TepArrInt; var ArPaste: TepArrInt; N, k: integer);
var
   i, b, j: integer;
begin
   Milliseconds;   for i := k + 1 to N do
   begin
      b := ArPaste[i];
      j := i - 1;
      while(j > 0) and (b < ArPaste[j]) do
      begin
         ArPaste[j + 1] := ArPaste[j];
         j := j - 1;
      end;
      ArPaste[j + 1] := b;
   end;
   TimeWorkSort[5] := MillisecondsDelta * 0.001;
   for i := 1 to N do
      B[i, 5] := A[i];
      end;
```

Сортировка слиянием

```
procedure Sliyanie(F: TepMasIn; S: TepHalfArrInt; var Rez: TepMasIn; Nf, Ns, nk:
integer);
var
   Caunt, i, j: integer;
begin
   F[Nf + 1] := Abs(F[Nf]);
   if Ns > 0 then
      F[Nf + 1] := F[Nf + 1] + Abs(S[Ns]);
   S[Ns + 1] := F[Nf + 1];
   Caunt := 0;
   i := 1;
   j := 1;
```

```
   while(Caunt < Nf + Ns) do
   begin
      if F[i] < S[j] then
      begin
         Rez[nk + Caunt] := F[i];
         Caunt := Caunt + 1;
         i := i + 1;
      end
      else
      begin
         Rez[nk + Caunt] := S[j];
         Caunt := Caunt + 1;
         j := j + 1;
      end;
   end;
end;

procedure SortSlet(var ArSlit: TepMasIn; N: integer);
var
   First: TepMasIn;
   Second: TepHalfArrInt;
   H, Nf, Ns, k, i: integer;
begin
   Milliseconds;
   H := 1;
   while(H < N) do
   begin
      k := 1;
      while(k <= N) do
      begin
         Nf := 0;
         i := 0;
         while(Nf < H) and (k + i <= N) do
         begin
            Nf := Nf + 1;
            First[Nf] := ArSlit[k + i];
            i := i + 1;
         end;
         Ns := 0;
         while(Ns < H) and (k + i <= N) do
         begin
            Ns := Ns + 1;
            Second[Ns] := ArSlit[k + i];
            i := i + 1;
         end;

         Sliyanie(First, Second, ArSlit, Nf, Ns, k);
         k := k + 2 * H;
      end;
      H := 2 * H;
   end;
   TimeWorkSort[6] := MillisecondsDelta * 0.001;
end;
```

Библиотека массивов

```
unit MyLibraryToArr;

interface
uses MyLibrarySort;
const
   NMAX = 10000001;

var
   Work, A: TepMasIn;
   Nw, N: integer;
   NameFile: string = 'Benchmark.TXT';

procedure MasToMas(A: TepMasIn; N: integer; var Work: TepMasIn);

procedure MadeConstMas;
procedure MadeRandomMas(N: integer);
procedure MadeOfFileMas(NameFile: string);


implementation

procedure MadeConstMas;
begin
   N := 17;
   A[1] := 108;
   A[2] := 34;
   A[3] := 67;
   A[4] := 92;
   A[5] := 10;
   A[6] := 37;
   A[7] := 7;
   A[8] := 3;
   A[9] := 50;
   A[10] := 103;
   A[11] := 1;
   A[12] := 100;
   A[13] := 16;
   A[14] := 48;
   A[15] := 22;
   A[16] := 11;
   A[17] := 64;
end;

procedure MadeRandomMas(N: integer);
var
   B, i: integer;
begin
   Randomize;
   B := N;
   for i := 1 to B do
      A[i] := Random(10000001);
end;

procedure MadeOfFileMas(NameFile: string);
var
   F: TEXT;
begin
   AssignFile(F, NameFile);
   Reset(F);
   N := 0;
   while (not EOF(F)) do
   begin
```

```
      N := N + 1;
      Readln(F, A[N]);
   end;
   Close(F);
end;

procedure MasToMas(A: TepMasIn; N: integer; var Work: TepMasIn);
var
   Nw, i: integer;
begin
   Nw := N;
   for i := 1 to Nw do
      Work[i] := A[i];
end;
end.
```

Основная программа

```
program SravnenieSortirovok;

uses MyLibrarySort, MyLibraryToArr, Crt;

const
   NMAX = 10000001;

type
   TepNamePunct = array[1..8] of string[20];

var
   NamePunctGL: TepNamePunct = ('Выход           ', 'Выбор массива   ', 'Выбор
сортировки', 'Вывод результата', '', '', '', '');
   NamePunctAr: TepNamePunct = ('Константный     ', 'Случайный массив', 'Массив из
файла ', '', '', '', '', '');
   NamePunctS: TepNamePunct = ('Пузырёк             ', 'Улучшенный пузырёк  ',
'Шелла               ', 'Поиском минимального',
                               'Вставками          ', 'Слиянием            ',
'Все сортировки      ', '');
   NamePunctRez: TepNamePunct = ('Пузырёк             ', 'Улучшенный пузырёк  ',
'Шелла               ', 'Поиском минимального',
                               'Вставками          ', 'Слиянием            ',
'Все сортировки      ', 'Назад               ');
   ColFon, SortK, ArrK, ResK, NpA, NpS, NpR, Np, k, i: integer;
   CBack: integer = 15;
   CText: integer = 0;
   Cod: char;


function FunMenu(var NamePunct: TepNamePunct; Np: integer; CBack: integer; CText:
integer; ColFon: integer): integer;
var
   Number, i: integer;
   Cod: char;
begin
   TextBackGround(CBack);
   TextColor(CText);
   Number := 1;
   repeat
      ClrScr;
      for i := 1 to Np do
         if i = Number then
         begin
            TextBackGround(ColFon);
```

```
            Writeln(NamePunct[i]);
            TextBackGround(CBack);
         end
         else
            Writeln(NamePunct[i]);
      Cod := ReadKey();
      if Cod = char(0) then
         Cod := ReadKey();
      if Cod = char(38) then
      begin
         Number := Number - 1;
         if Number = 0 then
            Number := Np;
      end;
      if Cod = char(40) then
      begin
         Number := Number + 1;
         if Number = Np + 1 then
            Number := 1;
      end;
   until (Cod = char(13)) or (Cod = char(39));
   FunMenu := Number;
end;


begin
   repeat
      Np := 4;
      ColFon := 10;
      ClrScr;
      k := FunMenu(NamePunctGL, Np, CBack, CText, ColFon);
      case k of
         2:
            begin
               ClrScr;
               ColFon := 11;
               NpA := 3;
               repeat
                  ClrScr;
                  ArrK := FunMenu(NamePunctAr, NpA, CBack, CText, ColFon);
                  case ArrK of
                     1:
                        begin
                           MadeConstMas;
                           MasToMas(A, N, Work);
                           break;
                        end;
                     2:
                        begin
                           Writeln('Введи количество элементов');
                           Readln(N);
                           MadeRandomMas(N);
                           MasToMas(A, N, Work);
                           break;
                        end;
                     3:
                        begin
                           MadeOfFileMas('Benchmark.TXT');
                           Writeln(N);
                           MasToMas(A, N, Work);
                           break;
                        end;
                  end;
               until (Cod = char(27));
```

```
   end;
3:
   begin
      ClrScr;
      ColFon := 3;
      NpS := 7;
      repeat
         ClrScr;
         SortK := FunMenu(NamePunctS, NpS, CBack, CText, ColFon);
         case SortK of
            1:
               begin
                  MasToMas(Work, N, A);
                  SortBable(A, N);
                  break;
               end;
            2:
               begin
                  MasToMas(Work, N, A);
                  SortBestBable(A, N);
                  break;
               end;
            3:
               begin
                  MasToMas(Work, N, A);
                  SortSHella(A, N);
                  break;
               end;
            4:
               begin
                  MasToMas(Work, N, A);
                  SortMinim(A, N);
                  break;
               end;
            5:
               begin
                  MasToMas(Work, N, A);
                  SortPaste(A, N, 1);
                  break;
               end;
            6:
               begin
                  MasToMas(Work, N, A);
                  SortSlet(A, N);
                  break;
               end;
            7:
               begin
                  MasToMas(Work, N, A);
                  SortBable(A, N);
                  MasToMas(Work, N, A);
                  SortBestBable(A, N);
                  MasToMas(Work, N, A);
                  SortSHella(A, N);
                  MasToMas(Work, N, A);
                  SortMinim(A, N);
                  MasToMas(Work, N, A);
                  SortPaste(A, N, 1);
                  MasToMas(Work, N, A);
                  SortSlet(A, N);
                  break;
               end;
         end;
      until (Cod = char(27));
```

```
   end;
4:
   begin
      ClrScr;
      ColFon := 6;
      NpR := 8;
      repeat
         ClrScr;
         ResK := FunMenu(NamePunctRez, NpR, CBack, CText, ColFon);
         case ResK of
            1:
               begin
                  Writeln;
                  Writeln('Выбран: ' + NamePunctAr[ArrK]);
                  Writeln;
                  Writeln('Сортировка', '               ',
                                                  'Время работы');
                  Writeln(NamePunctS[1], '        ', TimeWorkSort[1],
                                                          'sec');
                  Readln;
               end;
            2:
               begin
                  Writeln;
                  Writeln('Выбран: ' + NamePunctAr[ArrK]);
                  Writeln;
                  Writeln('Сортировка', '               '
                                                 'Время работы');
                  Writeln(NamePunctS[2], '        ', TimeWorkSort[2],
                                                          'sec');
                  Readln;
               end;
            3:
               begin
                  Writeln;
                  Writeln('Выбран: ' + NamePunctAr[ArrK]);
                  Writeln;
                  Writeln('Сортировка', '               ',
                                                'Время работы');
                  Writeln(NamePunctS[3], '        ', TimeWorkSort[3],
                                                          'sec');
                  Readln;
               end;
            4:
               begin
                  Writeln;
                  Writeln('Выбран: ' + NamePunctAr[ArrK]);
                  Writeln;
                  Writeln('Сортировка', '               ',
                                                  'Время работы');
                  Writeln(NamePunctS[4], '        ', TimeWorkSort[4],
                                                          'sec');
                  Readln;
               end;
            5:
               begin
                  Writeln;
                  Writeln('Выбран: ' + NamePunctAr[ArrK]);
                  Writeln;
                  Writeln('Сортировка', '               ',
                                                  'Время работы');
                  Writeln(NamePunctS[5], '        ', TimeWorkSort[5],
                                                          'sec');
                  Readln;
```

```
                        end;
                     6:
                        begin
                           Writeln;
                           Writeln('Выбран: ' + NamePunctAr[ArrK]);
                           Writeln;
                           Writeln('Сортировка', '               ',
                                                          'Время работы');
                           Writeln(NamePunctS[5], '        ', TimeWorkSort[6],
                                                                  'sec');
                           Readln;
                        end;
                     7:
                        begin
                           Writeln;
                           Writeln('Выбран: ' + NamePunctAr[ArrK]);
                           Writeln;
                           Writeln('Сортировка', '               ',
                                                           'Время работы');
                           for i := 1 to NSOR do
                              Writeln(NamePunctS[i], '        ', TimeWorkSort[i],
                                                                    'sec');
                           Readln;
                        end;
                  end;
               until (ResK = 8);
            end;
      end;
      if (Cod = char(27)) then Halt;
   until (k = 1);
end.
```