ОЛИМПИАДА ШКОЛЬНИКОВ «ШАГ В БУДУЩЕЕ»

НАУЧНО-ОБРАЗОВАТЕЛЬНОЕ СОРЕВНОВАНИЕ «ШАГ В БУДУЩЕЕ, МОСКВА»

10824

регистрационный номер

_____Информатика и системы управления (ИУ)_____
название факультета

__Программное обеспечение ЭВМ и информационные технологии (ИУ-7)__
название кафедры

Программа исследования эффективности алгоритмов сортировки данных

название работы

Автор: ***_Борисов Дмитрий Евгеньевич_____***
фамилия, имя, отчество
_ГБОУ Школа №67, 11а_____
наименование учебного заведения, класс

Научный руководитель: _____
фамилия, имя, отчество
_____
место работы
_____
звание, должность

_____
подпись научного руководителя

Москва - 2019

Программа исследования эффективности алгоритмов сортировки данных

Аннотация

Целью данной работы является исследование эффективности алгоритмов сортировки данных в сравнении друг с другом и в зависимости от входного набора данных. Для исследования выбраны наиболее популярные алгоритмы сортировки данных, как простые, так и достаточно сложные, но весьма эффективные. В рамках работы разобраны алгоритмы реализации различных методов сортировки, приведены математические оценки их эффективности, и даны фрагменты кода для реализации разобранных алгоритмов на языке Паскаль. Для наглядной демонстрации работы различных алгоритмов создана программа в среде системы разработки программного обеспечения Delphi. С помощью этой программы можно сравнить скорость работы различных алгоритмов сортировки на одинаковых массивах данных для среднего, худшего и лучшего случая. Для большей наглядности результаты сравнения могут быть представлены в виде гистограммы. Также в программе есть пошаговый демонстратор работы выбранного алгоритма сортировки, который с помощью простой анимации показывает процесс сортировки тестового массива. Для всех использованных алгоритмов в программе дано краткое описание и математический анализ эффективности. Данная программа может служить наглядным пособием при изучении алгоритмов сортировки, а также позволяет оценить временные затраты на сортировку массивов при числе элементов до 200 тысяч.

Оглавление

Введение

Практически любая достаточно сложная программа (будь то операционная система, система управления базами данных, система автоматизированного проектирования и т.п.) обязательно включает в себя механизмы сортировки данных. Объёмы данных в настоящее время крайне велики, и их обработка может занимать весьма существенное время, несмотря даже на прогресс в росте производительности современных компьютеров. Не является исключением в процессе обработки и сортировка. Поэтому крайне важным является вопрос выбора оптимальных методов сортировки с целью повышения производительности работы (и, соответственно, сокращения времени обработки) современных информационных систем. Алгоритмов сортировки существует достаточно много, ряд из них является оптимальными для определённых наборов данных, другие могут иметь некоторые преимущества в различных ситуациях. В виду этого возникает необходимость сравнительного анализа алгоритмов.

Описание исследованных алгоритмов сортировки

Под сортировкой понимают процесс перестановки элементов какого-либо массива данных в определённом порядке. Сортировка применяется практически в любой программе, начиная от простейших программ для личного пользования, заканчивая мощнейшими современными информационными системами.

Алгоритм сортировки – это последовательность шагов для упорядочения элементов в списке. В случае, когда элемент списка имеет несколько полей, поле, по которому необходимо произвести упорядочивание, называется ключом сортировки. В качестве ключа сортировки зачастую выступает число, остальные поля хранят какие-либо данные, не влияющие на работу алгоритма.

Алгоритмов сортировки существует довольно много, некоторые из которых имеют преимущества перед другими, зачастую в определённых условиях. Сравнение алгоритмов сортировки позволяет выявить их сильные и слабые стороны, а также показать, что усложнением алгоритмов можно добиться значительного повышения производительности по сравнению с простыми методами.

Важным свойством алгоритма является его сфера применения. Как правило выделяют два основных класса:

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

Внешняя сортировка используется при работе с запоминающими устройствами большого объёма, но с доступом не произвольным, а последовательным. Такая сортировка применяется для упорядочения файлов. Её особенностью является то, что в каждый конкретный момент времени мы

работаем только с одним элементом, а затраты на доступ к другим (по сравнению с оперативной памятью) неоправданно велики. Это накладывает некоторые дополнительные ограничения на алгоритм и приводит к специальным методам упорядочения, обычно использующим дополнительное дисковое пространство. К тому же, доступ к данным на дисках производится намного медленнее, чем операции с оперативной памятью.

Специфика доступа к данным в дисковых накопителях проявляется в том, что операции чтения/записи происходят последовательным образом: в каждый момент времени можно считать или записать только элемент, следующий за текущим. Второй особенностью является то, что, как правило, объём данных не позволяет их разместить целиком в оперативной памяти.

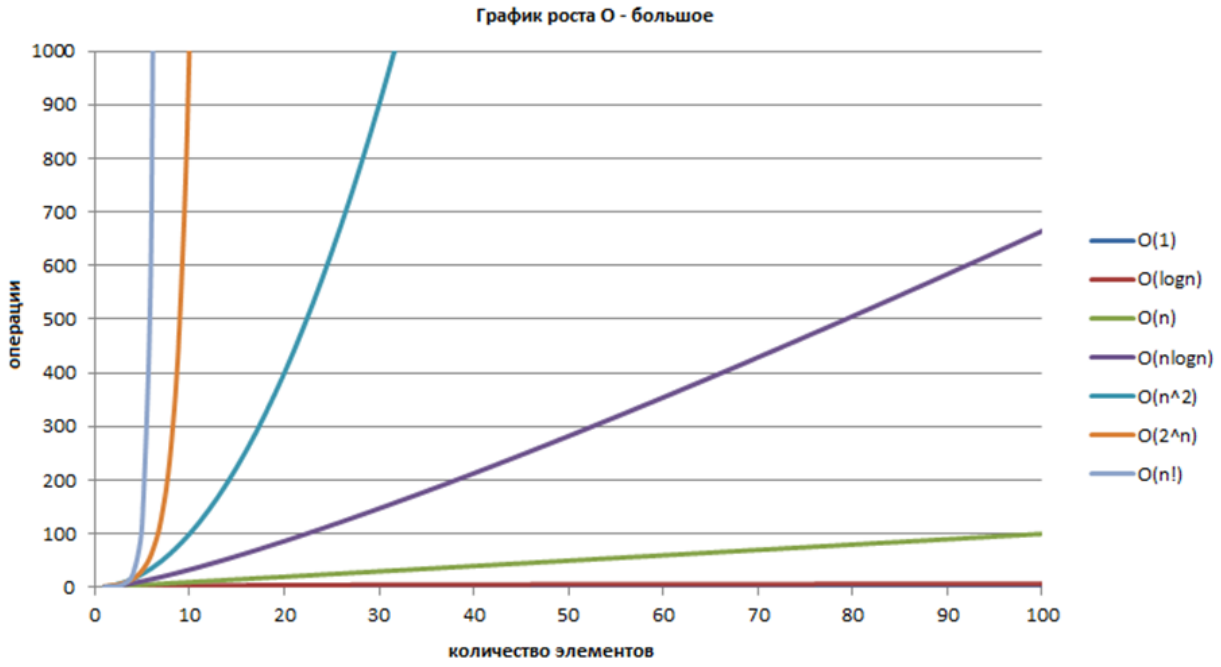
Оценка эффективности алгоритмов сортировки

Не существует единственного, наиболее эффективного в любой ситуации, алгоритма сортировки. Поэтому важным является определение критериев оценки эффективности алгоритмов.

В оценках будет использоваться математическое обозначение «O» большое, которое применяется для сравнения асимптотического (в данном случае – стремящегося к бесконечности) поведения функций. Данная нотация используется в различных разделах математики: в математическом анализе, теории чисел и комбинаторике, а также при оценке сложности алгоритмов. Например, фраза «сложность алгоритма O(n!)» означает, что при больших n время работы алгоритма (или общее количество операций) не превышает C×n!, где C – некая положительная константа (обычно в качестве параметра n берут объем входной информации алгоритма).

Для оценки эффективности используют следующие параметры:

Время – основной параметр, характеризующий быстродействие алгоритма. Называется также вычислительной сложностью. Для оценки важно *худшее, среднее и лучшее* время работы алгоритма для массива размера (n). Временная сложность алгоритма обычно выражается с использованием приведённой выше нотации «O» большое. Для типичного алгоритма хороший результат – это O(n log n), а плохой – это $O(n^2)$. Идеальное поведение для алгоритма – O(n). Алгоритмы сортировки, использующие только операцию сравнения значений всегда нуждаются по меньшей мере в O(n log n) сравнениях. Наглядно разницу можно увидеть на приведённом ниже рисунке:



Память – ряд алгоритмов требует выделения дополнительной памяти для временного хранения данных. Для оценки эффективности использования памяти не учитывается место, которое занимает исходный массив и дополнительные затраты, например, на хранение кода программы;

Устойчивость – 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 обрабатываемого массива данных и работает эффективнее.

Список алгоритмов сортировки

В ниже приведённых таблицах: n – это количество записей, которые необходимо упорядочить:

Алгоритмы устойчивой сортировки

Название	**Сложность**	**Примечание**
Пузырьковая сортировка	$O(n^2)$	для каждой пары индексов производится обмен, если элементы расположены не по порядку
Сортировка перемешиванием	$O(n^2)$	
Сортировка вставками	$O(n^2)$	Для каждого элемента определяется его позиция в упорядоченном списке и выполняется вставка

Алгоритмы неустойчивой сортировки

Название	**Сложность (худшая)**	**Сложность (средняя)**	**Примечание**
Сортировка выбором	$O(n^2)$	-	поиск наименьшего или наибольшего элемента и помещение его в начало или конец упорядоченного списка
Сортировка Шелла	$O(n \log n)$	-	попытка улучшить сортировку вставками
Пирамидальная сортировка	$O(n \log n)$	$O(n \log n)$	Список преобразуется в "кучу". Наибольший элемент из кучи добавляется в отсортированный список.

Быстрая сортировка	$O(n^2)$	$O(n \log n)$	Считается самым быстрым из известных для упорядочения больших случайных списков. Исходный набор данных разбивается на две половины так, что любой элемент первой половины упорядочен относительно любого элемента второй половины; затем алгоритм применяется рекурсивно к каждой половине.

Описание алгоритмов сортировки

Сортировка выбором

Самый простой алгоритм сортировки. Его суть заключается в том, что перебором находится наименьший (или наибольший, в зависимости от необходимого порядка) элемент, который меняется местами с элементом, стоящим на первом месте. Далее находится наименьший (наибольший) среди оставшихся элементов и меняется местами с элементом, стоящим на втором месте. Цикл заканчивается, когда будут выбраны все элементы. На языке программирования Паскаль реализация алгоритма вставки выглядит следующим образом:

```
for i := 1 to  N-1  do
begin
    nMin = i;
    for j := i +1 to N do
        if A[j] < A[nMin] then nMin := j;
        if nMin <> i then
        begin
            c := A[i];
            A[i] := A[nMin];
            A[nMin] := c;
        end;
end;
```

Этот алгоритм отличается небольшим числом перестановок (n-1) (операций перемещения данных в три раза больше - 3×(n-1)), но очень значительным числом сравнений - n×(n-1)/2.

Пузырьковая сортировка

Пузырьковая сортировка основана на методе перестановок. Идея этого метода отражена в его названии. Самые легкие элементы массива «всплывают» наверх, самые «тяжелые» - тонут. В процессе работы очередной элемент сравнивается с соседним и при необходимости выполняется их перестановка. Таким образом, мы вытолкнем наверх самый «легкий» элемент всего массива. Элементы массива при этом уподобляются всплывающим пузырькам в сосуде с водой. Алгоритм реализуется следующей последовательностью команд:

```
for i := 1 to N-1 do
begin
    for j := 1 to N-1 do
        if A[j] > A[j+1] then
        begin
            c := A[j];
            A[j] := A[j+1];
            A[j+1] := c;
        end;
end;
```

Число сравнений всегда n×(n-1)/2, число перестановок в худшем случае такое же (3×n× (n-1)/ 2 перемещений данных). Алгоритм достаточно прост, но крайне неэффективен.

Для пузырьковой сортировки существует модифицированный алгоритм, который называют «пузырёк с флагом». Суть этого алгоритма состоит в том, что если при выполнении прохода методом пузырька не было ни одного обмена элементов массива, то это означает, что массив уже отсортирован и остальные проходы не нужны. Реализация этого алгоритма выглядит следующим образом:

```
repeat
    flag := False;
    for j:=N-1 downto 1 do
```

```
            if A[j] > A[j+1] then
            begin
                c := A[j];
                A[j] := A[j+1];
                A[j+1] := c;
                flag := True;
            end;
until not flag;
```

Алгоритм может быть еще несколько улучшен если добавить счетчик проходов. После одного прохода один элемент будет стоять на своем месте, а значит с ним сравнивать не имеет смыслы. Реализация улучшенного метода сортировки пузырьком с флагом:

```
i := 0;
repeat
    i := i + 1;
    flag := False;
    for j := N-1 downto 1 do
        if A[j] > A[j+1] then
        begin
            c := A[j];
            A[j] := A[j+1];
            A[j+1] := c;
            Flag := True;
        end;
  until not flag;
```

Сортировка перемешиванием

Разновидность пузырьковой сортировки. Встречается также название «шейкерная сортировка». Отличается тем, что просмотры элементов выполняются один за другим в противоположных направлениях, при этом

большие элементы стремятся к концу массива, а маленькие — к началу. Лучший случай для этой сортировки — отсортированный массив (O(n)), худший — отсортированный в обратном порядке ($O(n^2)$). Пример реализации:

```
d := 1; i := 0;
for k := N-1 downto 1 do
    begin
        I := i+d;
        for j := 1 to k do
        begin
            if (A[i] - A[i+d]) * d > 0 then
            begin
                x := A[i];
                A[i] := A[i+d];
                A[i+d] := x;
            end;
            i:=i+d;
        end;
        d:=-d;
    end;
```

Сортировка вставками

Сортируемый массив просматривается в порядке возрастания номеров и каждый элемент вставляется в уже просмотренную часть массива так, чтобы сохранить порядок. На каждом шаге сортировки часть массива уже упорядочена, поэтому для поиска места вставки можно использовать метод половинного деления. Число сравнений значительно меньше, чем при сортировке выбором (<n log(n)), число перемещений данных в худшем случае n×(n+3)/2. Реализация алгоритма сортировки вставками на Паскале:

```
for i:= 2 to N do
```

```
if  A[i-1] > A[i]  then
begin
        x := A[i];
        j := i-1;
        while  (j > 0) and (A[j] > x)  do
        begin
                A[j+1] := A[j];
                J := j-1;
        end;
        A[j+1] := x;
end;
```

Сортировка Шелла

Этот метод был предложен Дональдом Шеллом в 1959 г. Основная идея этого алгоритма заключается в том, чтобы в начале устранить массовый беспорядок в массиве, сравнивая далеко стоящие друг от друга элементы. В процессе работы алгоритма интервал между сравниваемыми элементами постепенно уменьшается до единицы. Это означает, что на поздних стадиях сортировка сводится просто к перестановкам соседних элементов (если, конечно, такие перестановки являются необходимыми).

Сортировка методом Шелла базируется на известном алгоритме простых вставок. Смысл ее состоит в раздельной сортировке методом простых вставок нескольких частей, на которые разбивается исходный массив. Ее необычность состоит в том, что она рассматривает весь список как совокупность перемешанных подсписков.

При сортировке Шелла сначала выполняется сортировка элементов, отстоящих 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 (то есть, обычной

сортировкой вставками). Эффективность сортировки Шелла в определённых случаях обеспечивается тем, что элементы «быстрее» встают на свои места.

Расстояния между сравниваемыми элементами могут изменяться по-разному. Обязательным является лишь то, что последний шаг должен равняться единице. Следует избегать последовательностей степени двойки, которые, как показывают сложные математические выкладки, снижают эффективность алгоритма сортировки. Эксперименты показывают, что величины шагов не должны быть кратны друг другу для получения лучших результатов. Дональд Кнут в своей книге «Искусство программирования» рекомендует такие последовательности шагов, записанные в обратном порядке:

- 1, 4, 13, 40, 121, …, где $h_{i-1}=3\times h_i+1$;
- 1, 3, 7, 15, 31, …, где $h_{i-1}=2\times h_i+1$;

Роберт Седжвик предложил специальные формулы для получения шага. Одна из таких формул дает следующую последовательность значений шага:

- 1, 5, 19, 41, 109, 209, 505 и т.д.

Пример реализации сортировки Шелла на Паскале:

```
incr := n div 2;
    while  incr > 0 do
    begin
        for  I := incr+1 to n do
        begin
            j := i-incr;
            while  j > 0 do
            if A[j] > A[j+incr] then
            begin
                c := A[j];
                A[j] := A[j+incr];
                A[j+incr] := A[j];
```

```
                j := j-incr
            end
            else  j := 0
        end;
        incr := incr div 2
    end;
```

Пирамидальная сортировка

Пирамидальная сортировка основывается на организации элементов в массиве по типу двоичного (бинарного) дерева. Двоичным деревом называют иерархическую структуру данных, в которой каждый элемент имеет не более двух потомков.

Пирамида представляет собой особый вид бинарного дерева, в котором значение каждого элемента больше, чем значение каждого из его потомков. Непосредственные потомки каждого узла не упорядочены, поэтому иногда левый потомок может оказаться больше правого, а иногда – наоборот. Пирамида представляет собой полное дерево, в котором заполнение нового уровня начинается только после того, как предыдущий уровень полностью заполнен, а все узлы на одном уровне заполняются последовательно.

Сортировка начинается с построения пирамиды, содержащей все элементы исходной последовательности. Ясно, что максимальный элемент окажется в вершине дерева, поскольку все её потомки обязательно должны быть меньше. Затем, вершина пирамиды записывается в конец последовательности, а пирамида с удаленным максимальным элементом переформировывается. В результате, на её вершине оказывается максимальный из оставшихся элементов, он записывается на соответствующее место, и процедура продолжается до тех пор, пока пирамида не опустеет.

Таким образом, основную нагрузку алгоритма несут две процедуры: начальное построение пирамиды и переформирование пирамиды на каждом шаге.

Первая процедура создания пирамиды называется просеиванием. Пример её реализации:

```
for i := (N div 2) downto 1 do
begin
    j := i;
    while j <= (N div 2) do
    begin
        k := 2 * j;
        if (k + 1 <= N) and (A[k] < A[k + 1]) then
        k := k + 1;
        if A[k] > A[j] then
        begin
            x := A[j];
            A[j] := A[k];
            A[k] := x;
            j := k
        end
        else
            break
    end
end;
```

Вторая процедура – это непосредственно сортировка полученной пирамиды:

```
for i := N downto 2 do
begin
    x := A[1];
    A[1] := A[i];
```

```
        A[i] := x;
        j := 1;
        while j <= ((i - 1) div 2) do
        begin
            k := 2 * j;
            if (k + 1 <= i - 1) and (A[k] < A[k + 1]) then
                k := k + 1;
            if A[k] > A[j] then
            begin
                x := A[j];
                A[j] := A[k];
                A[k] := x;
                j := k
            end
            else
                break
        end
end;
```

Быстрая сортировка

Этот метод был разработан в 1962 г. Чарльзом Хоаром, поэтому его иногда называют методом Хоара.

Суть метода заключается в том, чтобы найти такой элемент множества, подлежащего сортировке, который разобьет его на два подмножества: те элементы, что меньше делящего элемента, и те, что не меньше его.

Алгоритм основан на разделении сортируемого массива на части и перестановке элементов таким образом, чтобы элементы в левой части массива не превосходили элементов в правой. На каждом шаге выбирается «средний» элемент массива, в результате ряда перестановок он занимает свое законное место, затем каждая из частей упорядочивается.

Метод быстрой сортировки является одним из самых быстрых известных универсальных алгоритмов сортировки массивов, его вычислительная сложность в среднем не превышает O(n log n).

Алгоритм состоит из трёх шагов:

1. Выбор элемента из массива. Выбранный элемент назовём «опорным».
2. Разбиение: перераспределение элементов в массиве таким образом, чтобы элементы меньше «опорного» помещались перед ним, а больше или равные, после.
3. Рекурсивное применение первых двух шагов к двум подмассивам слева и справа от «опорного» элемента. Рекурсия не применяется к массиву, в котором только один элемент или отсутствуют элементы.

Реализация быстрой сортировки на Паскале:

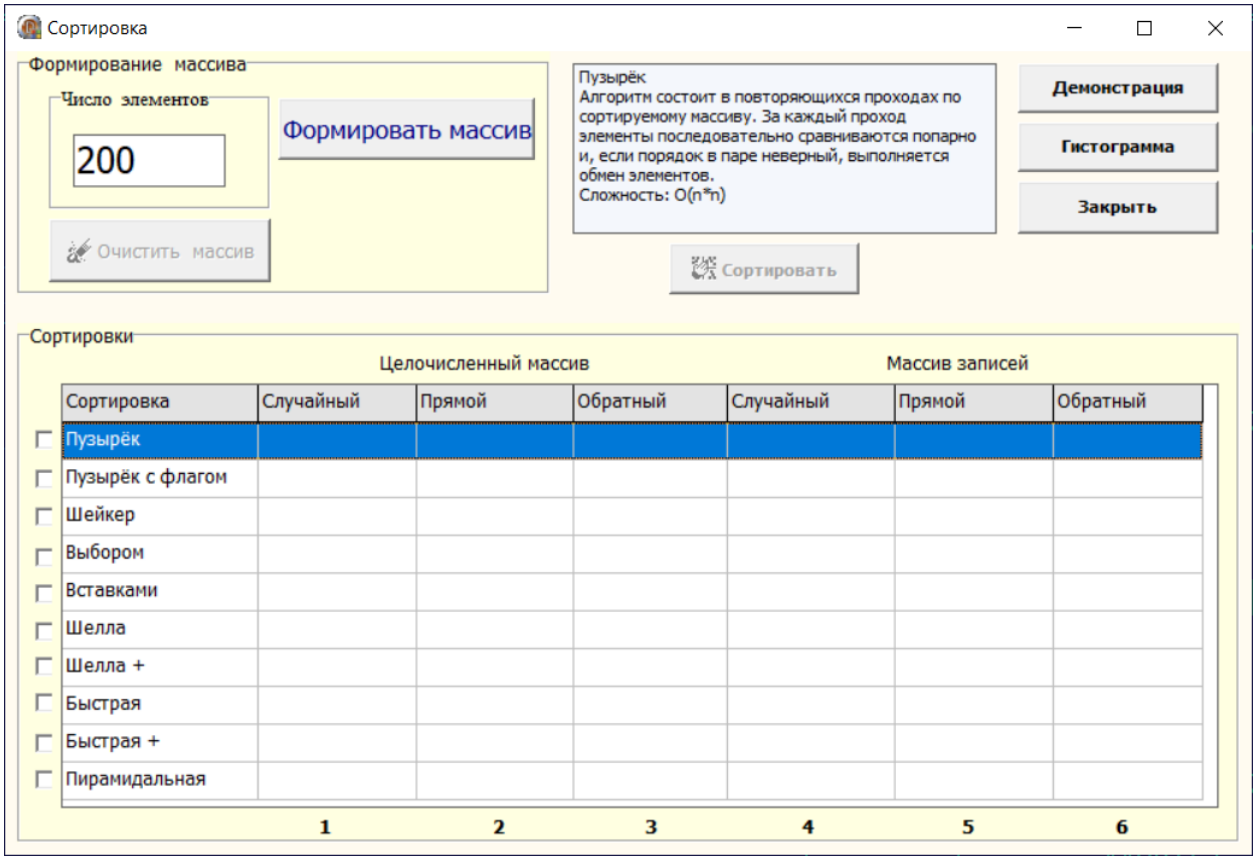
```
procedure QSort(first, last: integer);
var L, R, c, X: integer;
begin
    if first < last then
    begin
        X := A[(first + last) div 2];
        L := first;
        R := last;
        while L <= R do
        begin
            while A[L] < X do
                L := L + 1;
            while A[R] > X do
                R := R - 1;
            if L <= R then
            begin
                c := A[L];
```

```
                A[L] := A[R];
                A[R] := c;
                L := L + 1;
                R := R - 1;
            end;
        end;
        QSort(first, R);
        QSort(L, last);
    end;
end.
```

Описание программы

Для разработки программы была использована среда разработки Delphi. Это современная система программирования, в основе которой лежит язык Паскаль и средства визуального программирования. Система Delphi относится к классу средств ускоренной разработки программного обеспечения за счёт использования визуального конструктора форм и библиотеки визуальных компонентов. Delphi представляет из себя очень мощный и гибкий язык, пригодный для создания весьма сложных проектов. Также Delphi обеспечивает высокую скорость разработки программ.

Разработанная программа позволяет провести сравнительный анализ экспериментально полученных данных с использование различных методов сортировок. Результаты отображаются в виде таблицы. Общий вид основного окна программы представлен на рисунке ниже:



В первом столбце этой таблицы указывается наименование метода сортировки. Некоторые из этих методов относятся к медленным: на получение результатов требуется существенное время. Однако достоинством подобных сортировок является простота их реализации. К ним относятся: пузырьковая сортировка, пузырьковая сортировка с флагом, шейкерная сортировка, сортировка простым выбором, сортировка вставкой. Остальные методы являются представителями более сложных алгоритмов, но зато обладают гораздо большей эффективностью. К этим методам относятся: сортировка Шелла, быстрая сортировка, пирамидальная сортировка. Сортировка Шелла представлена в двух вариантах, т.к. в ней важен выбор величины шагов.

Для каждого метода сортировки дается краткое содержание алгоритма и информация о его сложности.

Во втором столбце указывается время, потребовавшееся для сортировки целочисленных случайных величин.

Третий столбец – время сортировки уже отсортированного массива, а четвертый столбец – это массив с обратным порядком элементов.

Количество задаваемых элементов может колебаться от 200 до 200 000. Использование большого количества чисел для анализа вряд ли целесообразно, т.к. требует значительного времени для медленных алгоритмов.

Последующие столбцы – это массивы, в которых имеется сопутствующая информация. Увеличение размеров перемещаемых элементов демонстрирует влияние на скорость работы программы в сторону замедления. В данной программе сопутствующие данные представляют из себя переменные действительного типа.

В качестве времени окончания сортировки служит момент, когда разница двух моментов времени превысит данную величину, многократно пройдя при этом заданный алгоритм. Вычисляется количество повторений

сортировки, а затем окончательный результат получается путем деления времени на это количество.

С помощью режима «Демонстрация» можно наглядно, по шагам, понаблюдать за работой алгоритма. В анимированном режиме демонстрируется процесс перестановки элементов.

Также можно построить гистограмму на основе полученных в процессе сравнения алгоритмов данных.

Заключение

В представленной работе выполнен анализ алгоритмов сортировки одномерных массивов по количеству операций и временным затратам. Наглядное сравнение методов сортировки позволяет оценить их применимость в различных условиях и выбрать наиболее подходящий в конкретной ситуации. При обучении программированию большое значение имеет возможность графического представления алгоритма. Очень важно, чтобы обучаемый мог визуально увидеть каждый шаг выполнения программы. Для ряда алгоритмов это осуществляется в данном проекте.

Список использованной литературы

[1]. Вирт Н. Алгоритмы + структуры данных = программы: Пер. с англ. – М.: Мир, 1985. – 406 с., ил.

[2]. Кнут Дональд, Эрвин. Искусство программирования, том 3. Сортировка и поиск, 2-е изд. : Пер. с англ. – М. : ООО “И.Д. Вильямс”, 2007. – 832 с. : ил.

[3]. Стивенс, Род. Алгоритмы. Теория и практическое применение – М.: Издательство «Э», 2017. – 544 с.

[4]. Культин Н.Б. Delphi в задачах и примерах. – 3-е изд., перераб и доп. – СПб.: БХВ-Петербург, 2012. – 288 с.: ил.

[5]. Фленов М.Е. Библия Delphi. – 3-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2011. – 688 с.: ил.