СОДЕРЖАНИЕ:

[Введение 2](#_Toc185344188)

[1. Задачи сортировки. 2](#_Toc185344189)

[1.1.Общие положения. 2](#_Toc185344190)

[1.2. Постановка задачи сортировки массивов. 4](#_Toc185344191)

[2. Методы сортировки массивов. 5](#_Toc185344192)

[2.1. Простые методы сортировки массивов. 5](#_Toc185344193)

[2.1.1. Сортировка с помощью прямого включения. 5](#_Toc185344194)

[2.1.2.Сортирвка с помощью прямого выбора. 8](#_Toc185344195)

[2.1.3. Сортировка с помощью прямого обмена 9](#_Toc185344196)

[2.2. Улучшенные методы сортировки массивов. 12](#_Toc185344197)

[2.2.1.Метод Шелла. 12](#_Toc185344198)

[2.2.2.Сортировка с помощью дерева 14](#_Toc185344199)

[2.2.3. Сортировка с помощью разделения 18](#_Toc185344200)

[Тесты 21](#_Toc185344201)

[Заключение 31](#_Toc185344202)

[Используемая литература 33](#_Toc185344203)

Введение**.**

Около трех с половиной десятилетий минуло с тех пор, как в педвузах введено в качестве учебной дисциплины программирование для ЭВМ. При колоссальной скорости изменений в самом предмете, всегда существенно превышавшей скорость центральных издательских механизмов, специально ориентированные на программы педвузов книги выходили не чаще, чем раз в десятилетие – едва ли не соразмерно скорости смены поколений ЭВМ. Сегодня полки книжных магазинов ломятся от изданий по информатике. Однако преподавателю (а более всего студенту) специальные учебные книги, содержание и направленность которых отвечают заданному учебному плану и программе все-таки очень нужны. Сейчас помимо программирования на некоторых специальностях в педвузах введены и другие более сложные спецкурсы, находящиеся на стыке прикладной (дискретной) математики и информатики.

В данной курсовой работе можно познакомится с массивами и узнать о простых и сложных методах их сортировки, а также о том, какие из них наиболее эффективны и в каких случаях.

### 1. Задачи сортировки.

### 1.1.Общие положения.

Основная задача – продемонстрировать различные методы сортировки и выделить наиболее эффективные из них. Сортировка – достаточно хороший пример задачи, которую можно решать с помощью многих различных алгоритмов. Каждый из них имеет и свои достоинства, и свои недостатки, и выбирать алгоритм нужно, исходя из конкретной постановки задачи.

В общем сортировку следует понимать как процесс перегруппировки заданного множества объектов в некотором определенном порядке. Цель сортировки – облегчить последующий поиск элементов в таком отсортированном множестве. Это почти универсальная, фундаментальная деятельность. Мы встречаемся с отсортированными объектами в телефонных книгах, в списках подоходных налогов, в оглавлениях книг, в библиотеках, в словарях, на складах – почти везде, где нужно искать хранимые объекты.

Таким образом, разговор о сортировке вполне уместен и важен, если речь идет об обработке данных. Первоначальный интерес к сортировке основывается на том, что при построении алгоритмов мы сталкиваемся со многими весьма фундаментальными приемами. Почти не существует методов, с которыми не приходится встречаться при обсуждении этой задачи. В частности, сортировка – это идеальный объект для демонстрации огромного разнообразия алгоритмов, все они изобретены для одной и той же задачи, многие в некотором смысле оптимальны, большинство имеет свои достоинства. Поэтому это еще и идеальный объект, демонстрирующий необходимость анализа производительности алгоритмов. К тому же на примерах сортировок можно показать, как путем усложнения алгоритма, хотя под рукой и есть уже очевидные методы, можно добиться значительного выигрыша в эффективности.

Выбор алгоритма зависит от структуры обрабатываемых данных – это почти закон, но в случае сортировки такая зависимость столь глубока, что соответствующие методы разбили на два класса – сортировку массивов и сортировку файлов (последовательностей). Иногда их называют внутренней и внешней сортировкой, поскольку массивы хранятся в быстрой, оперативной, внутренней памяти машины со случайным доступом, а файлы обычно размещаются в более медленной, но и более емкой внешней памяти, на устройствах, основанных на механических перемещениях (дисках или лентах).

Прежде чем идти дальше, введем некоторые понятия и обозначения. Если у нас есть элементы а, a, …… , а то сортировка есть перестановка этих элементов в массив аk, ak, …… ,ak где ak <= ak <= ….. <= ak.

Считаем, что тип элемента определен как INTEGER .

Const n=???; //здесь указывается нужная длина массива

Var A: array[1..n] of integer;

Выбор INTEGER до некоторой степени произволен. Можно было взять и

другой тип, на котором определяется общее отношение порядка.

Метод сортировки называют устойчивым, если в процессе сортировки относительное расположение элементов с равными ключами не изменяется. Устойчивость сортировки часто бывает желательной, если речь идет об элементах, уже упорядоченных (отсортированных) по некотором вторичным ключам ( т.е. свойствам ), не влияющим на основной ключ.

### 1.2. Постановка задачи сортировки массивов.

Основное условие: выбранный метод сортировки массивов должен экономно использовать доступную память. Это предполагает, что перестановки, приводящие элементы в порядок, должны выполняться на том же месте т. е. методы, в которых элементы из массива а передаются в результирующий массив b, представляют существенно меньший интерес. Мы будем сначала классифицировать методы по их экономичности, т. е. по времени их работы. Хорошей мерой эффективности может быть C – число необходимых сравнений ключей и M – число пересылок (перестановок) элементов. Эти числа суть функции от n – числа сортируемых элементов. Хотя хорошие алгоритмы сортировки требуют порядка n\*log n сравнений, мы сначала разберем несколько простых и очевидных методов, их называют прямыми, где требуется порядка n2 сравнений ключей. Начинать разбор с прямых методов, не трогая быстрых алгоритмов, нас заставляют такие причины:

1. Прямые методы особенно удобны для объяснения характерных черт основных принципов большинства сортировок.
2. Программы этих методов легко понимать, и они коротки.
3. Усложненные методы требуют большого числа операций, и поэтому для достаточно малых n прямые методы оказываются быстрее, хотя при больших n их использовать, конечно, не следует.

Методы сортировки “ на том же месте “ можно разбить в соответствии с определяющими их принципами на три основные категории:

* + Сортировки с помощью включения (by insertion).
  + Сортировки с помощью выделения (by selection).
  + Сортировка с помощью обменов (by exchange).

Теперь мы исследуем эти принципы и сравним их. Все программы оперируют массивом а.

Const n=<длина массива>

a: array[1..n] of integer;

### 2. Методы сортировки массивов.

### 2.1. Простые методы сортировки массивов.

### 2.1.1. Сортировка с помощью прямого включения.

Такой метод широко используется при игре в карты. Элементы мысленно делятся на уже “готовую” последовательность а, … , а и исходную последовательность. При каждом шаге, начиная с I = 2 и увеличивая i каждый раз на единицу, из исходной последовательности извлекается i- й элементы и перекладывается в готовую последовательность, при этом он вставляется на нужное место.

ПРОГРАММА 2.1. ССОРТИРОВКА С ПОМОЩЬЮ ПРЯМОГО ВКЛЮЧЕНИЯ.

PROGRAM SI;

VAR

I,J,N,X:INTEGER;

A:ARRAY[0..50] OF INTEGER;

BEGIN

WRITELN(‘Введите длину массива’);

READ(N);

WRITELN(‘Введите массив’);

FOR I:=1 TO N DO READ(A[I]);

FOR I:=2 TO N DO

BEGIN

X:=A[I];

A[0]:=X;

J:=I;

WHILE X<A[J-1] DO

BEGIN

A[J]:=A[J-1];

DEC(J)

END;

A[J]:=X

END;

WRITELN('Результат:');

FOR I:=1 TO N DO WRITE(A[I],' ')

END.

Такой типичный случай повторяющегося процесса с двумя условиями

окончания позволяет нам воспользоваться хорошо известным приемом

“барьера” (sentinel).

Анализ метода прямого включения. Число сравнений ключей (Ci) при i- м просеивании самое большее равно i-1, самое меньшее – 1; если предположить, что все перестановки из n ключей равновероятны, то среднее число сравнений – I/2. Число же пересылок Mi равно Ci + 2 (включая барьер). Поэтому общее число сравнений и число пересылок таковы:

Cmin = n-1 (2.1.) Mmin = 3\*(n-1) (2.4.)

Cave = (n2+n-2)/4 (2.2.) Mave = (n2+9\*n-10)/4 (2.5.)

Cmax = (n2+n-4)/4 (2.3.) Mmax = (n2+3\*n-4)/2 (2.6.)

Алгоритм с прямыми включениями можно легко улучшить, если обратить внимание на то, что готовая последовательность, в которую надо вставить новый элемент, сама уже упорядочена. Естественно остановиться на двоичном поиске, при котором делается попытка сравнения с серединой готовой последовательности, а затем процесс деления пополам идет до тех пор, пока не будет найдена точка включения. Такой модифицированный алгоритм сортировки называется методом с двоичным включением ( binary insertion).

ПРОГРАММА 2.2. СОРТИРОВКА МЕТОДОМ ДЕЛЕНИЯ ПОПОЛАМ.

PROGRAM BI;

VAR

I,J,M,L,R,X,N:INTEGER;

A:ARRAY[0..50] OF INTEGER;

BEGIN

WRITELN('Введи длину массива');

READ(N);

WRITELN('Введи массив');

FOR I:=1 TO N DO READ(A[I]);

FOR I:=2 TO N DO

BEGIN

X:=A[I];L:=1;R:=I;

WHILE L<R DO

BEGIN

M:=(L+R) DIV 2;

IF A[M]<=X THEN L:=M+1 ELSE R:=M

END;

FOR J:=I DOWNTO R+1 DO A[J]:=A[J-1];

A[R]:=X

END;

WRITELN('Результат:');

FOR I:=1 TO N DO WRITE(A[I],' ')

END.

Анализ двоичного включения. Место включения обнаружено, если L=R. Таким образом, в конце поиска интервал должен быть единичной длины; значит, деление его пополам происходит I\*log I раз. Таким образом:

C = Si: 1<=i<=n: [log I ] (2.7.)

Аппроксимируем эту сумму интегралом

Integral (1:n) log x dx = n\*(log n – C) + C (2.8.)

Где C = log e = 1/ln 2 = 1.4426… . (2.9.)

### 2.1.2.Сортирвка с помощью прямого выбора.

Этот прием основан на следующих принципах:

1. Выбирается элемент с наименьшим ключом
2. Он меняется местами с первым элементом а1.
3. Затем этот процесс повторяется с оставшимися n-1 элементами, n-2 элементами и т.д. до тех пор, пока не останется один, самый большой элемент.

ПРОГРАММА 2.3. СОРТИРВКА С ПОМОЩЬЮ ПРЯМОГО ВЫБОРА.

PROGRAM SS;

VAR

I,J,R,X,N:INTEGER;

A:ARRAY[0..50] OF INTEGER;

BEGIN

WRITELN('Введи длину массива');

READ(N);

WRITELN('Введи массив');

FOR I:=1 TO N DO READ(A[I]);

FOR I:=1 TO N-1 DO

BEGIN

R:=I;

X:=A[I];

FOR J:=I+1 TO N DO IF A[J]<X THEN

BEGIN

R:=J;

X:=A[R]

END;

A[R]:=A[I];

A[I]:=X

END;

WRITELN('Результат:');

FOR I:=1 TO N DO WRITE(A[I],' ')

END.

Анализ прямого выбора. Число сравнений ключей (С), очевидно не зависит от начального порядка ключей. Для С мы имеем C = (n2 – n)/2 (2.10.).

Число перестановок минимально: Mmin = 3\*(n-1) (2.11.).

В случае изначально упорядоченных ключей и максимально Mmax = n2/4 + 3\*(n-1) (2.12.)

Определим Мavg . Для достаточно больших n мы можем игнорировать дробные составляющие и поэтому аппроксимировать среднее число присваиваний на i-м просмотре выражением Fi = ln i + g + 1 (2.13.), где g = 0.577216… - константа Эйлера.

Среднее число пересылок Mavg в сортировке с выбором есть сумма Fi с i от 1 до n

Mavg = n\*(g + 1) + (Si: 1<=i<=n: ln i) (2.14.)

Вновь аппроксимируя эту сумму дискретных членов интегралом

Integral (1:n) ln x dx = x\*(ln x – 1) = n\*ln (n) – n + 1 (2.15.)

Получаем приблизительное значение

Mavg = n\*(ln (n) + g) . (2.16.)

2.1.3. Сортировка с помощью прямого обмена**.**

Как и в методе прямого выбора, мы повторяем проходы по массиву, сдвигая каждый раз наименьший элемент оставшейся последовательности к левому концу массива. Если мы будем рассматривать как вертикальные, а не горизонтальные построения, то элементы можно интерпретировать как пузырьки в чане с водой, причем вес каждого соответствует его ключу.

ПРОГРАММА 2.4. ПУЗЫРЬКОВАЯ СОРТИРОВКА.

PROGRAM BS;

VAR

I,J,X,N:INTEGER;

A:ARRAY[0..50] OF INTEGER;

BEGIN

WRITELN('Введи длину массива');

READ(N);

WRITELN('Введи массив');

FOR I:=1 TO N DO READ(A[I]);

FOR I:=2 TO N DO

FOR J:=N DOWNTO I DO

IF A[J-1]>A[J] THEN

BEGIN

X:=A[J-1];

A[J-1]:=A[J];

A[J]:=X

END;

WRITELN('Результат:');

FOR I:=1 TO N DO

WRITE(A[I],' ')

END.

Улучшения этого алгоритма напрашиваются сами собой:

1. Запоминать, были или не были перестановки в процессе

некоторого прохода.

1. Запоминать не только сам факт, что обмен имел место, но и

положение (индекс) последнего обмена.

1. Чередовать направление последовательных просмотров.

Получающийся при этом алгоритм мы назовем “шейкерной” сортировкой (ShakerSoft).

Анализ пузырьковой и шейкерной сортировок. Число сравнений в строго обменном алгоритме C = (n2 – n)/2, (2.17.), а минимальное, среднее и максимальное число перемещений элементов (присваиваний) равно соответственно M = 0, Mavg = 3\*(n2 – n)/2, Mmax = 3\*(n2 – n)/4 (2.18.)

Анализ же улучшенных методов, особенно шейкерной сортировки довольно сложен. Минимальное число сравнений Cmin = n – 1. Кнут считает, что улучшенной пузырьковой сортировки среднее число проходов пропорционально n – k1n1/2, а среднее число сравнений пропорционально &#189;(n2 – n(k2 +ln n)).

ПРОГРАММА 2.5. ШЕЙКЕРНАЯ СОРТИРОВКА.

PROGRAM SS;

VAR

J,L,K,R,X,N,I:INTEGER;

A:ARRAY[0..50] OF INTEGER;

BEGIN

WRITELN('Введи длину массива’);

READ(N);

WRITELN('Введи массив');

FOR I:=1 TO N DO

READ(A[I]);

L:=2;

R:=N;

K:=N;

REPEAT

FOR J:=R DOWNTO L DO

IF A[J-1]>A[J] THEN

BEGIN

X:=A[J-1];

A[J-1]:=A[J];

A[J]:=X;

K:=J

END;

L:=K+1;

FOR J:=L TO R DO

IF A[J-1]>A[J] THEN

BEGIN

X:=A[J-1];

A[J-1]:=A[J];

A[J]:=X;

K:=J

END;

R:=K-1

UNTIL L>R;

WRITELN('Результат:');

FOR I:=1 TO N DO

WRITE(A[I],' ')

END.

Фактически в пузырьковой сортировке нет ничего ценного, кроме ее привлекательного названия. Шейкерная же сортировка широко используется в тех случаях, когда известно, что элементы почти упорядочены – на практике это бывает весьма редко.

### 2.2. Улучшенные методы сортировки массивов.

### 2.2.1.Метод Шелла.

В 1959 году Д. Шеллом было предложено усовершенствование сортировки с помощью прямого включения. Сначала отдельно сортируются и группируются элементы, отстоящие друг от друга на расстояние 4. Такой процесс называется четверной сортировкой. В нашем примере восемь элементов и каждая группа состоит из двух элементов. После первого прохода элементы перегруппировываются – теперь каждый элемент группы отстоит от другого на две позиции – и вновь сортируются. Это называется двойной сортировкой. На третьем подходе идет обычная или одинарная сортировка. На каждом этапе либо сортируется относительно мало элементов, либо элементы уже довольно хорошо упорядочены и требуют сравнительно немного перестановок.

ПРОГРАММА 2.6. СОРТИРОВКА ШЕЛЛА..

PROGRAM SHELLS;

CONST T=4;

H: ARRAY[1..4] OF INTEGER = (15,7,3,1);

VAR

I,J,K,S,X,N,M:INTEGER;

A:ARRAY[-16..50] OF INTEGER;

BEGIN

WRITELN('Введи длину массива');

READ(N);

WRITELN('Введи массив');

FOR I:=1 TO N DO

READ(A[I]);

FOR M:=1 TO T DO

BEGIN

K:=H[M];

S:=-K;

FOR I:=K+1 TO N DO

BEGIN

X:=A[I];

J:=I-K;

IF S=0 THEN S:=-K;

INC(S);

A[S]:=X;

WHILE X<A[J] DO

BEGIN

A[J+K]:=A[J];

J:=J-K

END;

A[J+K]:=X;

END;

END;

WRITELN('Результат:');

FOR I:=1 TO N DO

WRITE(A[I],' ')

END.

Анализ сортировки Шелла. Нам не известно, какие расстояния дают наилучший результат. Но они не должны быть множителями один другого. Справедлива такая теорема: если k-отсортированную последовательность i-отсортировать, то она остается k-отсортированной. Кнут показывает, что имеет смысл использовать такую последовательность, в которой hk-1 = 3hk + 1, ht = 1 и t = [log2 n] – 1. (2.19.)

2.2.2.Сортировка с помощью дерева**.**

Метод сортировки с помощью прямого выбора основан на повторяющихся поисках наименьшего ключа среди n элементов, среди n-1 оставшихся элементов и т. д. Как же усовершенствовать упомянутый метод сортировки? Этого можно добиться, действуя согласно следующим этапам сортировки:

1. Оставлять после каждого прохода больше информации, чем просто идентификация единственного минимального элемента. Проделав n-1 сравнений, мы можем построить дерево выбора вроде представленного на рисунке 2.1.

06

12

06

44 12 18 06

44 55 12 42 94 18 06 67

РИСУНОК 2.1. ПОВТОРЯЮЩИЙСЯ ВЫБОР СРЕДИ ДВУХ КЛЮЧЕЙ.

2. Спуск вдоль пути, отмеченного наименьшим элементом, и исключение его из дерева путем замены либо на пустой элемент в самом низу, либо на элемент из соседний ветви в промежуточных вершинах (см. рисунки 2.2 и 2.3.).

Д. Уилльямсом был изобретен метод Heapsort, в котором было получено существенное улучшение традиционных сортировок с помощью деревьев. Пирамида определяется как последовательность ключей a[L], a[L+1], … , a[R], такая, что a[i] <= a[2i] и a[i] <= a[2i+1] для i=L…R/2.

Р. Флойдом был предложен некий “лаконичный” способ построения пирамиды на ”том же месте”. Здесь a[1] … a[n] – некий массив, причем a[m]…a[n] (m = [n DIV 2] + 1 ) уже образуют пирамиду, поскольку индексов i и j, удовлетворяющих соотношению j = 2i (или j = 2i + 1), просто не существует.

12

44 12 18

44 55 12 42 94 18 67

РИСУНОК 2.2.ВЫБОР НАИМЕНЬШЕГО КЛЮЧА.

06

12 18

44 12 18 67

44 55 12 42 94 18 67

РИСУНОК 2.3. ЗАПОЛНЕНИЕ ДЫРОК.

Эти элементы образуют как бы нижний слой соответствующего двоичного дерева (см. рисунок 2.4.), для них никакой упорядоченности не требуется. Теперь пирамида расширяется влево; каждый раз добавляется и сдвигами ставится в надлежащую позицию новый элемент. Получающаяся пирамида показана на рисунке 2.4.

a[1]

a[2] a[3]

a[4] a[5] a[6] a[7]

a[8] a[9] a[10] a[11] a[12] a[13] a[14] a[15]

РИСУНОК 2.4.МАССИВ, ПРЕДСТАВЛЕННЫЙ В ВИДЕ ДВОИЧНОГО ДЕРЕВА.

ПРОГРАММА 2.7. HEARSORT.

PROGRAM HS;

VAR

I,X,L,N,R:INTEGER;

A:ARRAY[0..50] OF INTEGER;

PROCEDURE SIFT(L,R: INTEGER);

VAR

I,J,X: INTEGER;

BEGIN

I:=L;

J:=2\*L;

X:=A[L];

IF (J<R)AND(A[J]<A[J+1]) THEN INC(J);

WHILE (J<=R)AND(X<A[J]) DO

BEGIN

A[I]:=A[J];

A[J]:=X;

I:=J;

J:=2\*J;

IF (J<R)AND(A[J]<A[J+1]) THEN INC(J);

END;

END;

BEGIN

WRITELN('Введи длину массива');

READ(N);

WRITELN('Введи массив');

FOR I:=1 TO N DO

READ(A[I]);

L:=(N DIV 2)+1;

R:=N;

WHILE L>1 DO

BEGIN

DEC(L);

SIFT(L,N)

END;

WHILE R>1 DO

BEGIN

X:=A[1];

A[1]:=A[R];

A[R]:=X;

DEC(R);

SIFT(1,R);

END;

WRITELN(‘Результат:');

FOR I:=1 TO N DO

WRITE(A[I],' ')

END.

Анализ Heapsort. Heapsort очень эффективна для большого числа элементов n; чем больше n, тем лучше она работает.

В худшем случае нужно n/2 сдвигающих шагов, они сдвигают элементы на log(n/2), log(n/2 – 1), … ,log(n – 1) позиций (логарифм по [основанию 2] «обрубается» до следующего меньшего целого). Следовательно фаза сортировки будет n – 1 сдвигов с самое большое log(n-1), log(n-2), … , 1 перемещениями. Можно сделать вывод, что даже в самом плохом из возможных случаев Heapsort потребуется n\*log n шагов. Великолепная производительность в таких случаях – одно из привлекательных свойств Heapsort.

2.2.3. Сортировка с помощью разделения**.**

Этот улучшенный метод сортировки основан на обмене. Это самый лучший из всех известных на данный момент методов сортировки массивов. Его производительность столь впечатляюща, что изобретатель Ч. Хоар назвал этот метод быстрой сортировкой (Quicksort) . В Quicksort исходят из того соображения, что для достижения наилучшей эффективности сначала лучше производить перестановки на большие расстояния. Предположим, что у нас есть n элементов, расположенных по ключам в обратном порядке. Их можно отсортировать за n/2 обменов, сначала поменять местами самый левый с самым правым, а затем последовательно сдвигаться с двух сторон. Это возможно в том случае, когда мы знаем, что порядок действительно обратный. Однако полученный при этом алгоритм может оказаться и не удачным, что, например, происходит в случае n идентичных ключей: для разделения нужно n/2 обменов. Этих необязательных обменов можно избежать, если операторы просмотра заменить на такие:

WHILE a[i] <= x DO i := i + 1 END;

WHILE x <= a[i] DO j := j – 1 END

В этом случае x не работает как барьер для двух просмотров. В результате просмотры массива со всеми идентичными ключами приведут к переходу через границы массива.

Наша цель – не только провести разделение на части исходного массива элементов, но и отсортировать его. Будем применять процесс разделения к получившимся двум частям, до тех пор, пока каждая из частей не будет состоять из одного-единственного элемента. Эти действия описываются в программе 2.8.

ПРОГРАММА 2.8. QUICKSORT.

PROGRAM QS;

VAR

N,I:INTEGER;

A:ARRAY[0..50] OF INTEGER;

PROCEDURE SORT(L,R: INTEGER);

VAR

I,J,X,W: INTEGER;

BEGIN

I:=L;

J:=R;

X:=A[(L+R) DIV 2];

REPEAT

WHILE A[I]<X DO INC(I);

WHILE X<A[J] DO DEC(J);

IF I<=J THEN

BEGIN

W:=A[I];

A[I]:=A[J];

A[J]:=W;

INC(I);

DEC(J);

END;

UNTIL I>J;

IF L<J THEN SORT(L,J);

IF I<R THEN SORT(I,R);

END;

BEGIN

WRITELN('Введи длину массива');

READ(N);

WRITELN('Введи массив');

FOR I:=1 TO N DO READ(A[I]);

SORT(1, N);

WRITELN('Результат:');

FOR I:=1 TO N DO

WRITE(A[I],' ')

END.

Анализ Quicksort. Процесс разделения идет следующим образом: выбрав некоторое граничное значение x, мы затем проходим целиком по всему массиву. При этом выполняется точно n сравнений. Ожидаемое число обменов есть среднее этих ожидаемых значений для всех возможных границ x.

M = [Sx:1 <= x <= n:(x-1)\*(n-(x-1))/n]/n

= [Su:0 <= u <= n-1: u\*(n-u)]/n2

= n\*(n-1)/2n-(2n2-3n+1)/6n = (n-1/n)/6 (2.20.)

В том случае, если бы нам всегда удавалось выбирать в качестве границы медиану, то каждый процесс разделения расщеплял бы массив на две половинки, и для сортировки требовалось бы всего n\*log n подходов. В результате общее число сравнений было бы равно n\*log n, а общее число обменов – n\*log(n) /6. Но вероятность этого составляет только 1/n.

Главный из недостатков Quicksort – недостаточно высокая производительность при небольших n, впрочем этим грешат все усовершенствованные методы, одним из достоинств является то, что для обработки небольших частей в него можно легко включить какой-либо из прямых методов сортировки.

Тесты**.**

1. **Идея сортировки массива прямым включением заключается в том, что**
   1. *в сортируемой последовательности masi длиной n (i = 0..n - 1) последовательно выбираются элементы начиная со второго (i< 1) и ищутся их местоположения в уже отсортированной левой части последовательности.*
   2. в сортируемой последовательности masi длиной n (i=0..n-1) последовательно выбираются элементы начиная со первого (i< 1) и ищутся их местоположения в уже отсортированной левой части последовательности.
   3. в сортируемой последовательности masi длиной n (i=0..n-1) последовательно выбираются элементы начиная со второго (i< 1) и ищутся их местоположения в не отсортированной левой части последовательности.
   4. в сортируемой последовательности masi длиной n-1 (i=0..n-1) последовательно выбираются элементы начиная со второго (i< 1) и ищутся их местоположения в уже отсортированной левой части последовательности.
2. **При сортировке массива прямым включением поиск места включения очередного элемента х в левой части последовательности mas может закончиться двумя ситуациями:**
   1. найден элемент последовательности mas, для которого masi>x; достигнут левый конец отсортированной слева последовательности.
   2. *найден элемент последовательности mas, для которого masi<x; достигнут левый конец отсортированной слева последовательности.*
   3. найден элемент последовательности mas, для которого masi<x; достигнут правый конец отсортированной слева последовательности.
   4. найден элемент последовательности mas, для которого masi<x; достигнут левый конец не отсортированной слева последовательности.
3. **При сортировке массива прямым включением для отслеживания условия окончания просмотра влево отсортированной последовательности используется прием «барьера». Суть его в том, что**
   1. к исходной последовательности справа добавляется фиктивный элемент X. В начале каждого шага просмотра влево отсортированной части массива элемент X устанавливается в значение того элемента, для которого будет осуществляться поиск места вставки.
   2. к исходной последовательности слева добавляется фиктивный элемент X. В конце каждого шага просмотра влево отсортированной части массива элемент X устанавливается в значение того элемента, для которого будет осуществляться поиск места вставки.
   3. к исходной последовательности слева добавляется фиктивный элемент X. В начале каждого шага просмотра влево отсортированной части массива элемент X устанавливается в значение того элемента, для которого не будет осуществляться поиск места вставки.
   4. *к исходной последовательности слева добавляется фиктивный элемент X. В начале каждого шага просмотра влево отсортированной части массива элемент X устанавливается в значение того элемента, для которого будет осуществляться поиск места вставки.*
4. **Сортировка массива прямым включением требует в среднем**
   1. N^2/2 перемещений.
   2. *N^2/4 перемещений.*
   3. N^2 перемещений.
   4. N/4 перемещений.
5. **Выберите правильный вариант для вставки вместо знака «вопрос» во фрагмент кода сортировки массива прямым включением:**

For i:=2 to Сount do

Begin

Tmp:=Arr[i];

j:=i-1;

?

Begin

Arr[j+1]:=Arr[j];

j:=j-1;

End;

Arr[j+1]:=Tmp;

End;

* 1. While (j<0) and (Arr[j]<Tmp) do
  2. While (j>0) and (Arr[j]>Tmp) do
  3. *While (j>0) and (Arr[j]<Tmp) do*
  4. While (j=0) and (Arr[j]=Tmp) do

1. **Алгоритм сортировки массива бинарными включениями**
   1. вставляет *i-*й элемент в готовую последовательность, которая пока не отсортирована, для нахождения места для *i-*го элемента используется метод бинарного поиска элемента.
   2. *вставляет i-й элемент в готовую последовательность, которая уже отсортирована, для нахождения места для i-го элемента используется метод бинарного поиска элемента.*
   3. вставляет *i-*й элемент в готовую последовательность, которая уже отсортирована, для нахождения места для *i-*го элемента используется метод Шелла поиска элемента.
   4. вставляет *i-*й элемент в пока готовую последовательность, которая пока не отсортирована, для нахождения места для *i-*го элемента используется метод Шелла поиска элемента.
2. **При сортировке массива бинарными включениями всего будет произведено**
   1. *N⋅log2N сравнений.*
   2. *⋅log*2*N* сравнений.
   3. *log*2*(N/2)* сравнений.
   4. *N/2\*log*2*N* сравнений.
3. **Изменится ли количество пересылок в сортировке массива бинарными включениями по отношению к количеству сравнений**
   1. станет больше
   2. станет меньше
   3. *не изменится.*
4. **При сортировке массива методом бинарного включения внутренний цикл поиска с одновременным сдвигом следует разделить:**
   1. бинарным поиском находится позиция вставки, затем все элементы готовой последовательности, находящиеся левее этой позиции, сдвигаются вправо.
   2. бинарным поиском находится позиция вставки, затем все элементы готовой последовательности, находящиеся правее этой позиции, сдвигаются влево.
   3. *бинарным поиском находится позиция вставки, затем все элементы готовой последовательности, находящиеся правее этой позиции, сдвигаются вправо.*
   4. бинарным поиском находится позиция вставки, затем все элементы готовой последовательности, находящиеся левее этой позиции, сдвигаются влево.
5. **В чем состоит идея сортировки массива методом Шелла?**
   1. сортировке подвергаются не все подряд элементы последовательности, а только отстоящие друг от друга на определенном расстоянии большем h.
   2. сортировке подвергаются не все подряд элементы последовательности, а только отстоящие друг от друга на определенном расстоянии меньшем h.
   3. *сортировке подвергаются не все подряд элементы последовательности, а только отстоящие друг от друга на определенном расстоянии h.*
   4. сортировке подвергаются не все подряд элементы последовательности, а только h элементов.
6. **При сортировке массива методом Шелла на каждом шаге значение h изменяется, пока не станет (обязательно) равно**
   1. 3
   2. 2
   3. 0
   4. *1.*
7. **Если h=1, то алгоритм сортировки массива методом Шелла вырождается в**
   1. пирамидальную сортировку.
   2. *сортировку прямыми включениями.*
   3. сортировку слиянием.
   4. сортировку бинарного включения.
8. **При сортировке массива методом Шелла расстояния между сравниваемыми элементами могут изменяться по-разному. Обязательным является лишь то, что**
   1. *последний шаг должен равняться единице.*
   2. последний шаг должен равняться нулю.
   3. первый элемент равен последнему элементу.
   4. первый элемент равен предпоследнему элементу.
9. **Эффективность сортировки массива методом Шелла объясняется тем, что**
   1. при каждом проходе используется очень большое число элементов, упорядоченность увеличивается при каждом новом просмотре данных.
   2. при каждом проходе элементы массива не упорядочены, а упорядоченность увеличивается при каждом новом просмотре данных.
   3. *при каждом проходе используется относительно небольшое число элементов или элементы массива уже находятся в относительном порядке, а упорядоченность увеличивается при каждом новом просмотре данных.*
   4. при каждом проходе используется относительно небольшое число элементов или элементы массива уже находятся в относительном порядке, а упорядоченность увеличивается при каждом новом просмотре данных.
10. **Идея алгоритма сортировки массива прямым выбором заключается в том, что**
    1. на каждом шаге просмотру подвергается несортированная правая часть массива. В ней ищется очередной максимальный элемент. Если он найден, то производится его перемещение на место крайнего левого элемента несортированной правой части массива.
    2. на каждом шаге просмотру подвергается несортированная правая часть массива. В ней ищется очередной минимальный элемент. Если он не найден, то производится его перемещение на место крайнего левого элемента несортированной правой части массива.
    3. на каждом шаге просмотру подвергается несортированная правая часть массива. В ней ищется очередной минимальный элемент. Если он найден, то производится его перемещение на место крайнего правого элемента несортированной левой части массива.
    4. *на каждом шаге просмотру подвергается несортированная правая часть массива. В ней ищется очередной минимальный элемент. Если он найден, то производится его перемещение на место крайнего левого элемента несортированной правой части массива.*
11. **Если сортировку выбором применить для массива "bdac", то будут получены следующие проходы**
    1. первый проход: c d b a; второй проход: a b b c; третий проход: a b c d.
    2. *первый проход: a d b c; второй проход: a b d c; третий проход: a b c d.*
    3. первый проход: a d b c; второй проход: a c d b; третий проход: a b c d.
    4. первый проход: a d b c; второй проход: a b d c; третий проход: d b c a.
12. **Как и в сортировке массива пузырьковым методом в сортировке массива прямым выбором внешний цикл**
    1. выполняется n раз, а внутренний цикл выполняется n/2 раз.
    2. выполняется n-1 раз,  а внутренний цикл выполняется n раз.
    3. *выполняется n-1 раз,  а внутренний цикл выполняется n/2 раз.*
    4. выполняется n раз, а внутренний цикл выполняется n раз.
13. **Вставить во фрагмент кода сортировки массива прямым выбором, вместо знака «вопроса», верное неравенство:**

for i := 1 to n - 1 do

begin

min := i;

for j := i + 1 to n do

if **?** then

min := j;

t := a[i];

a[i] := a[min];

a[min] := t

end;

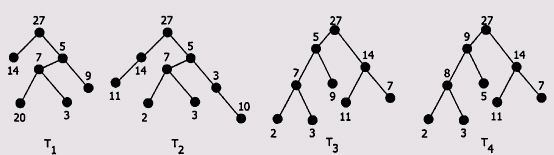
* 1. *a[min] > a[j].*
  2. a[min] = a[j].
  3. a[min] < a[j].
  4. a[min] <> a[j].

1. **При сортировке массива методом прямого выбора в лучшем случае (когда исходный массив уже упорядочен) потребуется поменять местами только ?, а каждая операция обмена  требует три операции пересылки.**

**Вставьте вместо знака «вопрос» верный вариант.**

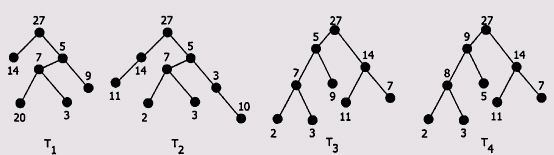
* 1. n-элементов.
  2. *(n-1)-элементов.*
  3. n/2-элементов.
  4. 2\*n-элементов.

1. **Алгоритм сортировки массива методом пирамидального выбора предназначен для сортировки последовательности чисел, которые**
   1. *являются отображением в памяти дерева специальной структуры — пирамиды.*
   2. являются отображением в памяти дерева специальной структуры — дерева.
   3. являются отображением в памяти пирамиды специальной структуры — пирамиды.
   4. являются отображением в памяти куста специальной структуры — дерева.
2. **На рисунке изображено несколько деревьев, из которых лишь одно** **является пирамидой.**



* 1. Т3.
  2. Т1.
  3. *Т4.*
  4. Т2.

1. **Пирамида, показанная на рисунке (одна из четырех), в памяти будет представлена следующим массивом:**



* 1. 3, 2, 7, 11, 5, 8, 14, 9, 27.
  2. 2, 3, 5, 7, 8, 9, 11, 14, 27.
  3. 27, 14, 11, 9, 8, 7, 5, 3, 2.
  4. *27, 9, 14, 8, 5, 11, 7, 2, 3.*

1. **На каждом из n шагов, требуемых для сортировки массива методом пирамидального выбора, нужно** 
   1. n\*log n (двоичных) сравнений.
   2. (log n)/2 (двоичных) сравнений.
   3. *log n (двоичных) сравнений.*
   4. 2 \* log n (двоичных) сравнений.
2. **Идея алгоритма сортировки массива прямым обменом заключается в том, что**
   1. если номер позиции большего из элементов больше номера позиции меньшего элемента, то меняем их местами.
   2. если номер позиции меньшего из элементов больше номера позиции большего элемента, то не меняем их местами.
   3. если номер позиции меньшего из элементов больше номера позиции большего элемента, то оставляем их на месте.
   4. *если номер позиции меньшего из элементов больше номера позиции большего элемента, то меняем их местами.*
3. **При использовании метода пузырьковой сортировки массива требуется самое большее**
   1. n проходов.
   2. *(n-1) проходов.*
   3. n/2 проходов.
   4. 2\*n проходов.
4. **При сортировке массива методом прямого обмена или методом пузырьковой сортировки после каждого прохода через таблицу может быть сделана проверка, были ли совершены перестановки в течение данного прохода. Если перестановок не было, то это означает, что**
   1. таблица не отсортирована и требует дальнейших проходов.
   2. таблица уже отсортирована и требует дальнейших проходов.
   3. *таблица уже отсортирована и дальнейших проходов не требуется.*
   4. таблица не отсортирована и дальнейших проходов не требуется.
5. **Сортировка массива пузырьковым методом обладает одной особенностью: расположенный не на своем месте в конце массива элемент**
   1. *достигает своего места за один проход.*
   2. достигает своего места за два прохода.
   3. достигает своего места за три прохода.
   4. достигает своего места за N проходов.
6. **Сортировка массива пузырьковым методом обладает одной особенностью: элемент, расположенный в начале массива** 
   1. очень быстро достигает своего места.
   2. *очень медленно достигает своего места.*
   3. не достигает своего места.
   4. достигает середины массива.
7. **В основе метода внутренней сортировки массива лежит  процедура слияния**
   1. *двух упорядоченных таблиц.*
   2. одной упорядоченной таблицы.
   3. трех упорядоченных таблиц.
   4. двух неупорядоченных таблиц.
8. **Сущность сортировки массива слиянием состоит в том, что упорядочиваемая таблица разделяется на равные группы элементов. Группы упорядочиваются, а затем** 
   1. попарно сливаются, образуя три новые группы, содержащие втрое больше элементов.
   2. попарно сливаются, образуя две новые группы, содержащие вдвое больше элементов.
   3. попарно сливаются, образуя новые группы, содержащие втрое больше элементов.
   4. *попарно сливаются, образуя новые группы, содержащие вдвое больше элементов.*

В предложенных тестах правильные ответы выделены курсивом.

Заключение**.**

В данной курсовой работе рассматриваются задачи сортировки, постановка задачи сортировки массивов. А также основная часть отведена рассмотрению методов: а именно, простые методы сортировки (сортировка с помощью прямого включения, сортировка с помощью прямого выбора, сортировка с помощью прямого обмена) и улученные методы сортировки массивов (метод Шелла, сортировка с помощью дерева, сортировка с помощью разделения). Предложен анализ к каждому из методов сортировки массивов. Разработаны тестовые задания по сортировкам массивов (30 заданий).

Используемая литература**:**

1. <http://www.books.everonit.ru>
2. <http://ru.wikipedia.org>
3. <http://khpi-iip.mipk.kharkiv.edu/library/>
4. <http://www.mgopu.ru/>
5. <http://www.compdoc.ru/prog/pascal/>
6. <http://www.cyberguru.ru/programming/>