Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра программного обеспечения информационных технологий

Л.А. Глухова, Е.П. Фадеева, Е.Е. Фадеева

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

ЛАБОРАТОРНЫЙ ПРАКТИКУМ
для студентов специальности I-40 01 01
«Программное обеспечение информационных технологий»
дневной формы обучения

В 4-х частях

Часть 2

Минск 2005

УДК 004.41 (075.8)
ББК 32.973-018.2 я 73
Г 55

Р е ц е н з е н т:
заведующий кафедрой информатики
Минского государственного высшего радиотехнического колледжа,
канд. техн. наук, доц. Ю.А. Скудняков

Глухова Л.А.

Г 55 Основы алгоритмизации и программирования: Лаб. практикум для студ. спец. I-40 01 01 «Программное обеспечение информационных технологий» дневной формы обуч. В 4 ч. Ч.2 / Л.А. Глухова, Е.П. Фадеева, Е.Е. Фадеева. – Мн.: БГУИР, 2005. – 52 с.

ISBN 985-444-796-0 (ч.2)

Вторая часть лабораторного практикума посвящена рассмотрению методов и алгоритмов сортировки совокупностей однотипных объектов. Рассмотрены два вида сортировок: сортировки массивов и сортировки файлов. Даны варианты индивидуальных заданий для выполнения лабораторных, практических и самостоятельных работ.

УДК 004.41 (075.8)
ББК 32.973-018.2 я 73

Первая часть издана в БГУИР в 2004 г.

ISBN 985-444-796-0(ч.2)
ISBN 985-444-616-6

СОДЕРЖАНИЕ

Введение……
1.Простые внутренние сортировки……
1.1. Сортировка вставками……..
1.1.1. Простые вставки……..
1.1.2. Вставки с барьерным элементом……
1.1.3. Вставки с бинарным поиском (бинарные вставки)……..
1.2. Сортировка методом выбора…….
1.3. Сортировка методом обмена……
1.3.1. Сортировка методом простого обмена (“пузырек”)…...……..…...
1.3.2. Шейкерная сортировка……
1.3.3. Модификации алгоритма сортировки “пузырьком”…….
1.3.4. Сортировка методом “плавающего пузырька”……
2. Улучшенные методы сортировки……..
2.1. Сортировка Шелла……
2.2. Быстрая сортировка (QuickSort)……...
2.3. Пирамидальная сортировка (HeapSort)…...……
3. Сортировка последовательностей (файлов)……..
3.1.Простое слияние……
3.2.Естественное слияние…….
Задания № 1……...
Задания № 2……...
Задания для самостоятельной работы……
Литература……
Приложение 1……...
Приложение 2……...
Приложение 3……..
Приложение 4……...
Приложение 5……...
Приложение 6……...
Приложение 7……...
Приложение 8……...
Приложение 9……...
Приложение 10……
Приложение 11……...
Приложение 12……
Приложение 13……
Приложение 14……

ВВЕДЕНИЕ

В общем случае под сортировкой понимают процесс упорядочения множества подобных информационных объектов в некотором определенном порядке с целью облегчения последующего поиска нужных элементов. Например, последовательность A из n элементов отсортирована в порядке возрастания (неубывания) значений элементов, если $a_1 \le \ldots \le a_i \le \ldots \le a_n$.

Различают два вида сортировок: сортировку массивов и сортировку файлов. Сортировку массивов также называют внутренней, т.к. все элементы массивов хранятся в быстрой внутренней памяти машины с прямым доступом, а сортировку файлов – внешней, т.к. их элементы хранятся в медленной, но более ёмкой внешней памяти.

При внутренней сортировке доступ к элементам может осуществляться в произвольном порядке. Напротив, при внешней сортировке доступ к элементам производится в строго определенной последовательности.

Имеется множество алгоритмов каждого из видов сортировки. Каждый алгоритм имеет свои достоинства, но в целом его оценка зависит от ответов, которые будут получены на следующие вопросы:

- с какой средней скоростью этот алгоритм сортирует информацию;
- какова скорость для лучшего и для худшего случаев;
- является ли естественным “поведение” алгоритма (т.е. возрастает ли скорость сортировки с увеличением упорядоченности массива);
- является ли алгоритм стабильным (т.е. выполняется ли перестановка элементов с одинаковыми ключами).

Для конкретного алгоритма большое значение имеет скорость сортировки. Скорость, с которой массив может быть упорядочен, зависит от числа сравнений и необходимых операций обмена. При этом необходимо учитывать, что операции обмена занимают большее время, чем операции сравнения.

Критериями оценки различных методов сортировки могут быть:

- количество сравнений;
- количество перестановок;
- время сортировки;
- требуемый объем памяти для сортировки;
- сложность алгоритмов сортировки.

Практически каждый алгоритм сортировки можно разбить на три компоненты:

- сравнение, определяющее упорядоченность пары элементов;
- перестановку, меняющую местами пару элементов;
- собственно сортирующий алгоритм, который осуществляет сравнение и перестановку элементов до тех пор, пока все элементы в последовательности не будут упорядочены.

Для решения задач сортировки разработан ряд алгоритмов. Тривиальным решением было бы найти наименьший элемент во всем массиве и поместить его в качестве первого элемента во вспомогательный массив. Затем найти наи-

меньший из оставшихся и поместить его в качестве второго элемента во вспомогательный массив. Однако при этом программе требуется памяти в два раза больше, чем занимает исходный массив. Далее рассматриваются методы сортировки, у которых нет указанного недостатка, т.е. сортировка осуществляется “по месту”.

Все методы внутренней сортировки можно разделить на две большие группы:

- прямые методы сортировки;
- улучшенные методы сортировки.

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

1) сортировка вставкой (включением);
2) сортировка обменом (“пузырьковая” сортировка);
3) сортировка выбором (выделением).

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

К улучшенным методам можно отнести следующие сортировки:

1) сортировку Шелла;
2) быструю сортировку (QuickSort);
3) пирамидальную сортировку (HeapSort).

В дальнейшем для простоты в примерах, иллюстрирующих методы сортировки, будут использоваться массивы целых чисел, а сортировка будет выполняться в порядке возрастания (неубывания).

1. ПРОСТЫЕ ВНУТРЕННИЕ СОРТИРОВКИ

1. 1. СОРТИРОВКА ВСТАВКАМИ

1. 1. 1. ПРОСТЫЕ ВСТАВКИ

Сущность метода сортировки вставками (данный метод называется еще методом прямого включения) легко пояснить на примере процесса тасования карточек с именами. Регистратор заносит каждое имя на карточку, а затем упорядочивает карточки по алфавиту, вставляя карточку в верхнюю часть стопки в подходящее место.

Аналогично происходит и сортировка в массиве. В каждый момент времени в нём можно выделить две части: отсортированную и неотсортированную. Элементы из неотсортированной части поочередно выбираются и вставляются в отсортированную часть так, чтобы не нарушить в ней упорядоченность элементов. На первом шаге в качестве отсортированной части массива принимается только один первый элемент, а в качестве неотсортированной части – все остальные.

Таким образом, алгоритм будет состоять из ***(n-1)*** – го прохода (где ***n*** – количество элементов массива), каждый из которых будет включать четыре действия:

- взятие очередного ***i-***го неотсортированного элемента и сохранение его в дополнительной переменной;
- поиск позиции ***j*** в отсортированной части массива, в которой присутствие взятого элемента не нарушит упорядоченности элементов;
- сдвиг элементов массива от ***j-***го до ***(i-1)***-го вправо, чтобы освободить позицию для вставки;
- вставка взятого элемента в найденную ***j***-ю позицию.

Пример: пусть дан исходный массив чисел ***A***={45, 55, 12, 42, 94, 18, 06, 67}. Как было сказано выше, при сортировке включением массив делится на две части: левая уже отсортирована, а правая еще нет. Сначала левая часть состоит из одного элемента {45}, а правая из всех остальных {55, 12, 42, 94, 18, 06, 67}.

Необходимо взять первый элемент из правой части, т.е. число 55 и поместить его в левую часть таким образом, чтобы левая часть стала упорядоченной. После этого левая отсортированная часть увеличится на один элемент, а правая на один элемент уменьшится. В примере 45 < 55, поэтому не надо производить перестановок элементов. Теперь массив будет поделен следующим образом: {45, 55} {12, 42, 94, 18, 06, 67}.

Далее вышеописанную процедуру нужно повторить с новыми правой и левой частями массива. Видно, что для того, чтобы поместить число 12 на нужное место, необходимо произвести перестановки элементов. Делается это следующим образом: значение 12 запоминается во временную переменную, а затем выполняется сравнение числа 12 с последним элементом левой части, т.е. с числом 55. Поскольку 12 < 55, то число 55 сдвигается на место числа 12. Затем

12 сравнивается с числом 45. Поскольку 12 < 45 , то число 45 помещается на место числа 55. Данная процедура повторяется до тех пор, пока не будет достигнут левый край массива, либо очередной элемент в левой части массива не окажется меньше или равен числу 12.

После этого число 12 помещается из временной переменной на освободившееся место (в данном случае на место числа 45). Теперь массив разбит следующим образом: {12, 45, 55} {42, 94, 18, 06, 67}. Аналогичные действия продолжаются до тех пор, пока в правой части, т.е. в неотсортированном массиве, еще есть элементы.

Все этапы сортировки представлены в табл. 1. Серым цветом выделена неотсортированная часть массива.

Таблица 1

Сортировка простыми вставками

Шаг	Элементы массива							
	45	**55**	**12**	**42**	**94**	**18**	**06**	**67**
1	**45**	**55**	**12**	**42**	**94**	**18**	**6**	**67**
2	**12**	**45**	**55**	**42**	**94**	**18**	**6**	**67**
3	**12**	**42**	**45**	**55**	**94**	**18**	**6**	**67**
4	**12**	**42**	**45**	**55**	**94**	**18**	**6**	**67**
5	**12**	**18**	**42**	**45**	**55**	**94**	**6**	**67**
6	**6**	**12**	**18**	**42**	**45**	**55**	**94**	**67**
7	**6**	**12**	**18**	**42**	**45**	**55**	**67**	**94**

Схема алгоритма сортировки представлена в прил. 1. Схема рассмотренного алгоритма и всех последующих выполнена в соответствии с ГОСТ 19.701-90 (представление циклов дано с помощью символа “Граница цикла”) [4].

Анализ сортировки методом прямого включения.

Сортировка вставкой имеет два преимущества.

Во-первых, она обладает естественным “поведением”, т. е. выполняется быстрее для упорядоченного массива и дольше всего выполняется, когда массив упорядочен в обратном направлении. Это делает сортировку вставкой полезной при работе с почти отсортированными массивами. Во-вторых, элементы с одинаковыми значениями не переставляются, т.е. сортировка стабильна.

Несмотря на то, что число сравнений может быть небольшим для определенных наборов данных, постоянные сдвиги массива требуют выполнения большого числа операций перестановок. Это является существенным недостатком метода.

Общее число сравнений ***(C)*** и перестановок ***(M)*** определяется выражениями [1]

$$C_{min} = n - 1; \qquad M_{min} = 3 * (n - 1);$$
$$C_{avg} = (n^2 + n - 2) / 4; \qquad M_{avg} = (n^2 + 9n - 10) / 4;$$

$$C_{max} = (n^2 + n - 4) / 4; \qquad M_{max} = (n^2 + 3n - 4) / 2.$$

где n – количество элементов в исходной последовательности;

C_{min} , C_{avg} , C_{max} – соответственно минимальное, среднее и максимальное значение сравнений;

M_{min} , M_{avg}, M_{max} – соответственно минимальное, среднее и максимальное значение перестановок.

1. 1. 2. ВСТАВКИ С БАРЬЕРНЫМ ЭЛЕМЕНТОМ

Данный метод сортировки аналогичен описанному выше за исключением одной детали, которая может в некоторой мере упростить алгоритм. Речь идет о барьерном элементе, который расположен перед отсортированной частью массива, т.е. слева, для этого в исходном массиве для него резервируется место. В позицию барьерного элемента на каждом шаге помещается текущий вставляемый элемент (первый в неотсортированной части).

Введение барьерного элемента позволяет отказаться от проверки условия выхода за пределы массива (слева), т. к. условие остановки при поиске места для элемента определяется следующим правилом: слева меньшие или равные вставляемому элементу, а справа строго большие его.

Схема алгоритма сортировки представлена в прил. 2.

Общее число сравнений и перестановок при использовании данного метода аналогично их числу при сортировке методом простых вставок.

1. 1. 3. ВСТАВКИ С БИНАРНЫМ ПОИСКОМ (БИНАРНЫЕ ВСТАВКИ)

Метод бинарных вставок является улучшением метода простых вставок. Отличие заключается в способе отыскания места вставки очередного элемента в отсортированную часть массива.

Остановимся подробнее на бинарном (двоичном, дихотомическом) поиске.

Рассмотрим последовательность, состоящую из n элементов, значения которых удовлетворяют условию

$$A_1 < A_2 < \ldots < A_n,$$

т.е. являются отсортированными.

При сравнении в такой последовательности искомого значения X и A_i может быть получен один из трех результатов:

- $X < A_i$ {элементы $A_i, A_{i+1}, \ldots, A_n$ исключаются из рассмотрения};
- $X = A_i$ {поиск завершен};
- $X > A_i$ {элементы $A_1, A_2, \ldots, A_i$ исключаются из рассмотрения}.

Из этого вытекает суть метода бинарного поиска: сравнить значение X со средним A_{mid} значением последовательности, в результате этого сравнения определить, в какой половине последовательности находится искомое значение, и снова применить ту же процедуру к выбранной половине последовательности.

Для поиска искомого значения понадобится выполнить порядка $\log_2 n$ сравнений. В результате искомое значение либо будет найдено, либо установлено, что его нет в последовательности.

В одной из наиболее популярных форм реализации метода поиска используются два указателя: ***L*** и ***R***, которые соответственно указывают на левую (***Left***) и правую (***Right***) границы поиска.

Рассмотрим последовательность шагов для реализации поиска элемента ***X*** в последовательности чисел, расположенных в порядке возрастания, а именно: $A_1<A_2<\ldots<A_n$.

Шаг 1. Инициализация

Установить ***L***=1, ***R***=N.

Шаг 2. Нахождение середины

На этом шаге известно, что если элемент ***X*** имеется в последовательности, то справедливо условие $A_L< X < A_R$. Если ***R*****<** ***L***, алгоритм завершается неудачно, т.е. искомый элемент не найден. В противном случае следует установить ***Mid*****=[*(L+R)* div 2]**, чтобы ***Mid*** соответствовало середине рассматриваемой части последовательности.

Шаг 3. Сравнение

Если $X<A_{mid}$, перейти к шагу 4; если $X>A_{mid}$, перейти к шагу 5; если $X=A_{mid}$, алгоритм успешно завершается и искомый элемент найден.

Шаг 4. Изменение R

Установить ***R=(Mid*–1*)*** и перейти к шагу 2.

Шаг 5. Изменение L

Установить ***L=(Mid*+1*)*** и перейти к шагу 2.

В табл. 2 представлен процесс бинарного поиска числа 653. Выделенные границы показывают зону поиска, а отмеченная серым цветом ячейка – A_{mid}. В данном случае поиск завершается после выполнения четырех сравнений.

Таблица 2

Бинарный поиск элемента 653

61	**87**	**154**	**170**	**275**	**426**	**503**	**509**	**512**	**612**	**653**	**677**	**703**	**765**	**897**	**908**
61	**87**	**154**	**170**	**275**	**426**	**503**	**509**	**512**	**612**	**653**	**677**	**703**	**765**	**897**	**908**
61	**87**	**154**	**170**	**275**	**426**	**503**	**509**	**512**	**612**	**653**	**677**	**703**	**765**	**897**	**908**
61	**87**	**154**	**170**	**275**	**426**	**503**	**509**	**512**	**612**	**653**	**677**	**703**	**765**	**897**	**908**

На первом шаге ***L***=1, ***R***=16, ***Mid***=8, т.е. указывает на элемент со значением 509.

На втором шаге ***L***=9, ***R***=16, ***Mid***=12, т.е. указывает на элемент со значением 677.

На третьем шаге ***L***=9, ***R***=11, ***Mid***=10, т.е. указывает на элемент со значением 612.

На четвертом шаге ***L***=11, ***R***=11, ***Mid***=11, т.е. указывает на элемент со значением 653. Поиск завершен.

Схема алгоритма сортировки представлена в прил. 3.

Анализ сортировки методом бинарной вставки.

Число сравнений при использовании данного метода фактически не зависит от начального порядка элементов из-за того, что при делении, выполняющемся при разбиении интервала поиска пополам, происходит отбрасывание дробной части, истинное число сравнений может оказаться на единицу больше. Все это приводит к тому, что в нижней части последовательности место включения отыскивается в среднем несколько быстрее, чем в верхней части. Фактически, если в исходном состоянии элементы расположены в обратном порядке, потребуется минимальное число сравнений, а если они упорядочены – максимальное число. Следовательно, можно говорить о неестественном "поведении" алгоритма поиска.

Улучшения, порожденные введением двоичного поиска, касаются лишь числа сравнений, а не числа необходимых перестановок чисел. Среднее число сравнений определяется по формуле [1]

$$c \approx n*(\log_2 n - log_2 e \pm 0,5),$$

где ***n*** – число элементов в сортируемой последовательности.

При этом число перестановок так и продолжает оставаться порядка n^2.

1. 2. СОРТИРОВКА МЕТОДОМ ВЫБОРА

Метод прямого выбора в некотором смысле противоположен методу прямой вставки. При прямой вставке на каждом шаге рассматривается только один очередной элемент исходной неотсортированной последовательности и все элементы готовой последовательности, среди которых отыскивается точка включения. При прямом выборе для поиска одного минимального элемента просматриваются все элементы исходной неотсортированной последовательности и найденный элемент помещается как очередной элемент в готовую последовательность.

Порядок шагов для сортировки методом прямого выбора можно представить следующим образом:

1) выбрать минимальный элемент из всего исходного массива и поместить его на первое место, а первый элемент – на место минимального;

2) просматривая подмассив от второго элемента до конца, найти минимальный элемент и поместить его на второе место, а второй элемент – на место минимального;

3) просматривая подмассив от третьего элемента до конца, найти минимальный элемент и поместить его на третье место, а третий элемент – на место минимального и т.д.

4) повторять операцию до предпоследнего ***(n-1)***-го элемента.

В табл. 3 приведен пример выполнения сортировки массива методом прямого выбора. Серым цветом выделена часть массива, в которой выполняется поиск минимального.

Таблица 3

Сортировка прямым выбором

Шаг	Элементы массива							
	44	55	12	42	94	18	06	67
1	44	55	12	42	94	18	06	67
2	06	55	12	42	94	18	44	67
3	06	12	55	42	94	18	44	67
4	06	12	18	42	94	55	44	67
5	06	12	18	42	94	55	44	67
6	06	12	18	42	44	55	94	67
7	06	12	18	42	44	55	94	67
8	06	12	18	42	44	55	67	94

Схема алгоритма сортировки представлена в прил. 4.

Анализ сортировки методом прямого выбора.

При использовании метода прямого выбора число сравнений элементов в общем случае не зависит от начального порядка элементов. В этом смысле “поведение” указанного метода менее естественно, чем “поведение” прямого включения. Количество сравнений определяется по формуле [1]

$$C = (n^2 - n) / 2,$$

где n – число элементов в сортируемой последовательности.

Число перестановок минимально в случае изначально упорядоченных элементов и максимально, если первоначально элементы располагались в обратном порядке. Число перестановок определяется выражениями

$$M_{min} = 3 * (n - 1);$$
$$M_{avg} \approx n * (\ln n + g);$$
$$M_{max} = n^2 / 4 + 3 * (n - 1),$$

где $g = 0{,}577216\ldots$ – константа Эйлера;

M_{min}, M_{avg}, M_{max} – соответственно минимальное, среднее и максимальное значение перестановок.

Как правило, алгоритм сортировки с прямым выбором предпочтительнее метода прямой вставки. Однако если элементы вначале упорядочены или почти упорядочены, прямое включение будет оставаться несколько более быстрым.

1. 3. СОРТИРОВКА МЕТОДОМ ОБМЕНА

1. 3. 1. СОРТИРОВКА МЕТОДОМ ПРОСТОГО ОБМЕНА (“пузырек”)

Сущность этого метода отражена в его названии. Самые легкие элементы массива “всплывают”, а самые "тяжелые" – тонут. Алгоритмически это можно реализовать следующим образом. Необходимо просмотреть весь массив “снизу вверх” (т.е. от конца к началу) и менять стоящие рядом элементы в том случае, если “нижний” элемент меньше, чем “верхний”. Таким образом, наверх выталкивается самый “легкий” элемент всего массива. Затем эта последовательность операций повторяется для оставшихся неотсортированными ***(n-1)*** элементов (т.е. для тех элементов, которые лежат “ниже” отсортированных).

Пример: пусть имеется массив чисел ***A***={44, 23, 67, 11, 20, 79, 50}. Массив просматривается, начиная с конца. На первом шаге сравнивается последний ***n***-й и предпоследний ***(n-1)***-й элементы, т.е. 79 и 50. Т.к. 79>50, то элементы меняются местами и процедура сравнения выполняется над ***(n-1)-***м и ***(n-2)-***м элементами, т.е. 20 и 50. Т.к. 20<50, то элементы местами не меняются и т.д. После первого прохода массив будет иметь вид ***A***={11, 44, 23, 67, 20, 50, 79}. Элемент 11 находится на своем месте и его можно исключить из дальнейшего просмотра.

Аналогичные шаги повторяются до тех пор, пока массив не будет полностью отсортированным. При этом из просмотра исключаются элементы массива, уже занявшие свои места, т.е. на втором шаге из просмотра исключается 1-й элемент, на третьем шаге – 1- и 2-й элементы и т.д.

Вид массива после каждого шага представлен в табл. 4. Серым цветом выделена часть массива, исключаемая из просмотра.

Таблица 4

Сортировка методом простого обмена

Шаг	Элементы массива						
	44	**23**	**67**	**11**	**20**	**79**	**50**
1	**11**	**44**	**23**	**67**	**20**	**50**	**79**
2	**11**	**20**	**44**	**23**	**67**	**50**	**79**
3	**11**	**20**	**23**	**44**	**50**	**67**	**79**
4	**11**	**20**	**23**	**44**	**50**	**67**	**79**
5	**11**	**20**	**23**	**44**	**50**	**67**	**79**
6	**11**	**20**	**23**	**44**	**50**	**67**	**79**

Схема алгоритма сортировки представлена в прил. 5.

Анализ сортировки методом простого обмена.

Для данного метода худшим является случай, когда минимальный элемент массива стоит на последнем месте. В лучшем случае перестановок может не

потребоваться совсем (элементы первоначального массива уже упорядочены). Итак, для числа перестановок справедливы следующие выражения [1]:

$$M_{min} = 0;$$
$$M_{avg} = 3 * (n^2 - n) / 2;$$
$$M_{max} = 3 * (n^2 - n) / 4,$$

где n – число элементов в сортируемой последовательности,

$M_{min}, M_{avg}, M_{max}$ – соответственно минимальное, среднее и максимальное значение перестановок.

Число же сравнений постоянно и равно

$$C = (n^2 - n) / 2.$$

1. 3. 2. ШЕЙКЕРНАЯ СОРТИРОВКА

Легко увидеть, что в алгоритме сортировки “пузырьком” “легкие пузырьки” всплывают за один проход, а “тяжелые” – тонут за несколько. Такая асимметрия вызвала появление новой идеи сортировки, “пузырьком”, а именно: сортировать не в одну сторону, а поочередно в обе, т.е. на каждом шаге осуществляется проход как в одну сторону, так и в другую. Таким образом, на каждом шаге “легкий пузырек” всплывает на поверхность, а “тяжелый” – тонет.

Этот метод получил название шейкерной сортировки. Метод шейкерной сортировки поясняет табл. 5. Серым цветом выделена часть массива, исключаемая из просмотра.

Таблица 5

Шейкерная сортировка

Шаг	*Направление движения*	Элементы массива						
		44	**23**	**67**	**11**	**20**	**79**	**50**
1	*Справа-налево*	**11**	**44**	**23**	**67**	**20**	**50**	**79**
	Слева-направо	**11**	**23**	**44**	**20**	**50**	**67**	**79**
2	*Справа-налево*	**11**	**20**	**23**	**44**	**50**	**67**	**79**
	Слева-направо	**11**	**20**	**23**	**44**	**50**	**67**	**79**
3	*Справа-налево*	**11**	**20**	**23**	**44**	**50**	**67**	**79**
	Слева-направо	**11**	**20**	**23**	**44**	**50**	**67**	**79**

Схема алгоритма представлена в прил. 8.

Анализ шейкерной сортировки.

Анализ шейкерной сортировки довольно сложен. Минимальное число сравнений равно

$$C_{min} = (n–1).$$

Число же перестановок не меняется в сравнении с методом “пузырька”, потому скорость работы алгоритма возрастает незначительно.

1. 3. 3. МОДИФИКАЦИИ АЛГОРИТМА СОРТИРОВКИ “ПУЗЫРЬКОМ”

При практической реализации сортировки методом обмена приходится выполнять одиночные проходы ***(n-1)*** раз. Этого можно избежать, если перед проведением очередного прохода проверять, была ли перестановка элементов на предыдущем проходе. Для этого необходимо ввести вспомогательную переменную (“флажок-признак”), в которой отмечается, была ли выполнена хотя бы одна перестановка на предыдущем проходе. Очевидно, что если на предыдущем проходе не было перестановок, значит массив упорядочен и сортировку можно прекратить. Этот метод сортировки можно назвать “пузырьковой сортировкой с флажком”. Понятно, что этот метод сокращает время сортировки для частично упорядоченного массива (в том числе и для массива, который частично упорядочился в процессе сортировки).

Схема алгоритма “пузырьковой сортировки с флажком“ представлена в прил. 6.

Кроме того, возможность хранения не только признака перестановки, но и места этой последней перестановки, помеченное как новая граница, позволяет в дальнейшем делать проход только до этой границы. Этот метод сортировки можно назвать “пузырьковой сортировкой с быстрой границей”.

Схема алгоритма “пузырьковой сортировки с быстрой границей” представлена в прил. 7.

1. 3. 4. СОРТИРОВКА МЕТОДОМ “ПЛАВАЮЩЕГО ПУЗЫРЬКА”

Если на некотором шаге выполняется просмотр ***i-***го элемента и слева от него имеется уже упорядоченная последовательность элементов, то можно предположить, что в конечном счете ***i***-й элемент займет любую позицию от 1- до ***i***-й включительно. С учетом сказанного сущность метода “плавающего пузырька” заключается в следующем: выполняется движение “вперед”, т.е. к концу массива, до тех пор, пока не обнаружится нарушение сортирующего условия. После соответствующего обмена элементов фиксируется место остановки и начинается движение в обратном направлении, т.е. к началу массива до тех пор, пока выполняются необходимые обмены элементов. Если обменов при обратном движении уже нет, то движение “вперед” продолжается с места остановки. Описанная процедура повторяется до тех пор, пока не будет достигнут конец массива. Это соответствует состоянию массива, в котором все элементы отсортированы.

Пример: пусть имеется массив чисел A={23, 44, 67, 25, 79, 80, 11, 85, 34}. Сравниваются элементы 23 и 44. Поскольку 44>23, то выполняется переход к следующей паре элементов, а именно: 44 и 67. Поскольку 67>44, то выполняется переход к следующей паре элементов, а именно: 67 и 25. Поскольку 25<67, т.е. нарушено сортирующее условие, выполняется обмен элементов и начинает-

ся движение в обратном направлении, т.е. к началу массива. Сравниваются элементы 44 и 25. Поскольку 25<44, то выполняется обмен элементов и переход к следующей паре элементов (при движении к началу массива). Сравниваются элементы 23 и 25. Поскольку 25>23, то движение к началу массива закончено.

Процесс сортировки возвращается в точку остановки, т.е. на элемент 67, и просмотр пар элементов при движении в прямом направлении, т.е. к концу массива, продолжается. Следующая просматриваемая пара 67 и 79 и т.д.

Метод сортировки “плавающего пузырька” поясняет табл. 6. Серым цветом выделена глубина прохода по массиву при движении “назад”, т.е. к началу массива.

Таблица 6

Сортировка методом “плавающего пузырька”

Шаг	*Направление движения*	Элементы массива								
		23	**44**	**67**	**25**	**79**	**80**	**11**	**85**	**34**
1	*вперед*	**23**	**44**	**67**	**25**	**79**	**80**	**11**	**85**	**34**
	назад	**23**	**25**	**44**	**67**	**79**	**80**	**11**	**85**	**34**
2	*вперед*	**23**	**25**	**44**	**67**	**79**	**80**	**11**	**85**	**34**
	назад	**11**	**23**	**25**	**44**	**67**	**79**	**80**	**85**	**34**
3	*вперед*	**11**	**23**	**25**	**44**	**67**	**79**	**80**	**85**	**34**
	назад	**11**	**23**	**25**	**34**	**44**	**67**	**79**	**80**	**85**

Схема алгоритма сортировки представлена в прил. 9.

Анализ алгоритма сортировки “плавающий пузырек”.

Следует обратить внимание на то, что все перечисленные выше усовершенствования не влияют на число перемещений, они лишь сокращают число излишних двойных проверок. Поскольку обмен местами двух элементов – более дорогостоящая операция, чем сравнение элементов, поэтому очевидные улучшения метода не дают большого выигрыша. Общее число перестановок сортировки по-прежнему остается порядка n^2.

Такой анализ показывает, что “обменная” сортировка и ее небольшие усовершенствования представляют собой нечто среднее между сортировками с помощью вставок и с помощью выбора.

2. УЛУЧШЕННЫЕ МЕТОДЫ СОРТИРОВКИ

2. 1. СОРТИРОВКА ШЕЛЛА

Этот метод является улучшением прямого метода сортировки – сортировки вставками (см. подразд. 1.1). Основная идея этого метода заключается в том, чтобы в начале устранить массовый беспорядок в массиве, сравнивая далеко отстоящие друг от друга элементы. Постепенно интервал между сравниваемыми элементами уменьшается до единицы. Это означает, что на поздних стадиях

сортировка сводится просто к перестановкам соседних элементов (если, конечно, такие перестановки являются необходимыми).

На рис. 1–3 показаны схемы выполнения сортировки Шелла.

Пример: пусть дан исходный массив чисел A={6, 4, 3, 1, 7, 2, 8, 5}. Сначала сортируются все элементы, которые смещены друг от друга на четыре позиции, а именно: числа 6 и 7, 4 и 2, 3 и 8, 1 и 5 (рис. 1):

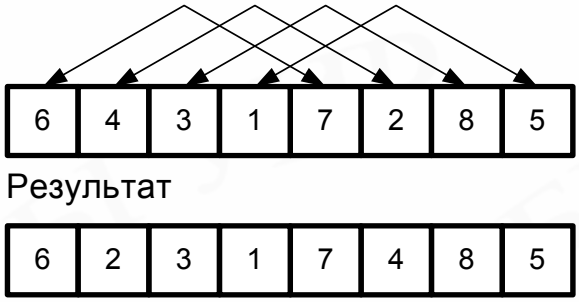


Рис.1. Сортировка Шелла (шаг 1)

Затем сортируются все элементы, которые смещены на две позиции (рис. 2):

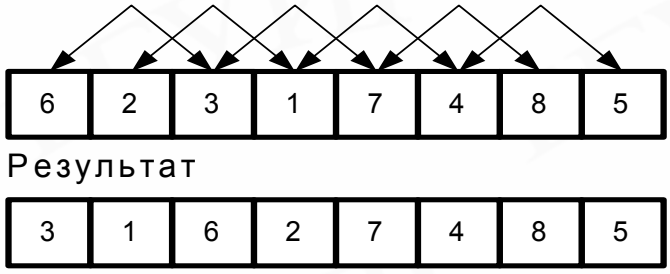


Рис.2. Сортировка Шелла (шаг 2)

И, наконец, упорядочиваются все соседние элементы (рис. 3):

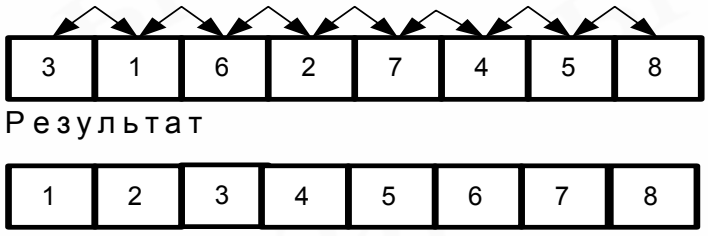


Рис.3. Сортировка Шелла (шаг 3)

Эффективность данного алгоритма объясняется тем, что при каждом проходе используется относительно небольшое число элементов или элементы массива уже находятся в относительном порядке, а упорядоченность увеличивается при каждом новом просмотре данных.

Схема алгоритма сортировки Шелла приведена в прил. 10.

Анализ алгоритма сортировки Шелла.

Расстояния между сравниваемыми элементами могут выбираться по-разному. Обязательным является лишь то, что последний шаг должен равняться единице. Доказано [2], что наиболее эффективна следующая последова-

тельность шагов (она записана в обратном порядке): 1,4,13,40,121…, где $h_{k-1} = 3h_k + 1; h_t = 1$ и $t = \lceil \log_3 n \rceil - 1$. Можно использовать и другую последовательность:1,3,7,15,31,…, где $h_{k-1} = 2h_k + 1, h_t = 1$ и $t = \lceil \log_2 n \rceil - 1$.

В вышеприведенных формулах: ***n***–размерность массива; ***t***–общее количество сортирующих шагов, зависящее от размерности массива; $h_t = 1$ последний сортирующий шаг; все предыдущие шаги до первого (самого большого) определяются по приведенным рекуррентным соотношениям.

Анализ сортировки Шелла показал, что время выполнения сортировки пропорционально $n^{1.2}$. Эта зависимость значительно лучше квадратичной зависимости n^2, которой подчиняются рассмотренные ранее простые алгоритмы сортировки.

2. 2. БЫСТРАЯ СОРТИРОВКА (QuickSort)

Быстрая сортировка (другое название QuickSort) является усовершенствованием метода сортировки обменами (см. подразд. 1.3). Суть метода состоит в следующем:

Шаг 1. Выбирается некоторый элемент ***X*** массива ***A,*** состоящего из ***n*** элементов, (в качестве такого барьерного элемента может быть выбран центральный элемент массива).

Шаг 2. Массив просматривается в прямом направлении (***i*** = 1, 2... и так далее до центрального элемента) с целью поиска в нем элемента ***A***[***i***], не меньшего, чем ***X***.

Шаг 3. Массив просматривается в обратном направлении (***j*** = ***n, n-1***,... и так далее до центрального элемента), но теперь ведется поиск элемента ***A***[***j***], не превосходящего ***X***.

Шаг 4. Производится перестановка элементов ***A***[***i***] и ***A***[***j***] местами.

Шаг 5. Шаги 1 – 4 повторяются, пока ***i < j.***

В результате таких действий слева от элемента ***X*** окажутся элементы, меньшие или равные ***X***, а справа – элементы, большие ***X.*** Пусть при этом элемент ***X*** попадет в позицию с номером ***k***, тогда массив будет иметь вид

***A*[*1*], *A*[*2*],…, *A*[*k-1*], *A*[*k*], *A*[*k+1*],…,*A*[*n*].**

Каждый из элементов ***A***[***1***], ***A***[***2***],…, ***A***[***k-1***] меньше или равен ***A***[***k***], а каждый из элементов ***A***[***k+1***],…,***A***[***n***] больше ***A***[***k***]. Отсюда можно сделать вывод, что элемент ***A***[***k***] стоит на своем месте. Исходный массив при этом разделился на две независимые неотсортированные части.

Для дальнейшей сортировки шаги 1–5 применяются для каждой из частей массива. И так до тех пор, пока в массиве не останутся подмассивы, состоящие из одного элемента, т.е. пока массив не будет отсортирован полностью.

Пример: пусть дан массив ***A***={0, 1, 6, 9, 5, 7, 3, 2, 4, 8}, состоящий из 10 целых чисел, проиндексированных в диапазоне [0]...[9].

Массив имеет нижнюю границу, равную 0 (***Low***), и верхнюю границу, равную 9 (***High***). Среднее значение индекса массива приходится на 4-й элемент (***Mid***). Поэтому первым центральным элементом является ***A***[***Mid***] = 5. Таким образом, все элементы массива ***A*** разбиваются на две части: ***Al*** и ***Ah***. Часть ***Al*** будет содержать элементы, меньшие или равные центральному. Часть ***Ah*** будет содержать элементы большие, чем центральный. Поскольку заранее известно, что центральный элемент в конечном итоге будет последним в ***Al***, мы временно передвигаем его в начало массива, меняя местами с ***A***[0] (или ***A***[***low***]). Это позволяет просматривать подмассив ***A***[1]...***A***[9] с помощью двух индексов: ***Up*** и ***Down***. Начальное значение ***Up*** соответствует индексу 1 (***Low***+1). Переменная ***Up*** адресует элементы подмассива ***Al***. Переменная ***Down*** адресует элементы подмассива ***Ah*** и имеет начальное значение 9 (***High***) (рис. 4).

Целью прохода является определение элементов для каждого подмассива.

Индекс ***Up*** перемещается вверх по массиву, ***Down*** – вниз. При изменении индекса ***Up*** ищется элемент массива ***A***[***Up***] больший, чем центральный. В этом месте просмотр останавливается, и найденный элемент готов к перемещению в верхний подмассив. Перед тем как это перемещение произойдет, индекс ***Down*** изменяется вниз по массиву для определения элемента, меньшего или равного центральному.

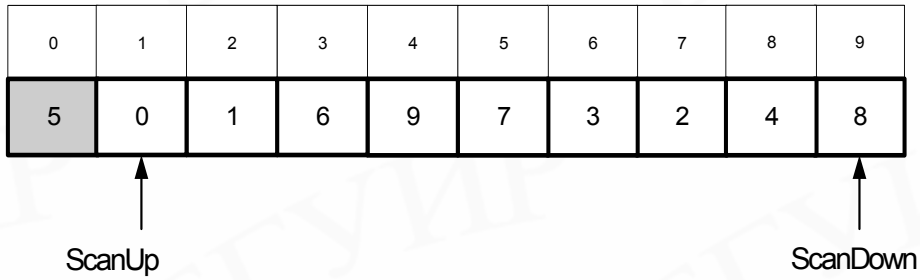


Рис. 4. Быстрая сортировка (шаг 1)

Таким образом, у нас есть два элемента, которые находятся не в тех подмассивах, и их можно менять местами:

Swap (***A***[***Up***], ***A***[***Down***]);

Этот процесс продолжается до тех пор, пока индексы ***Up*** и ***Down*** не зайдут друг за друга (***Up***=6, ***Down***=5). В этот момент ***Down*** оказывается в подмассиве ***Al***, элементы которого меньше или равны центральному. Это соответствует точке разбиения двух массивов и определяет окончательную позицию для центрального элемента. В соответствии с алгоритмом в примере меняются местами числа 6 и 4, 9 и 2, 7 и 3 (рис. 5).

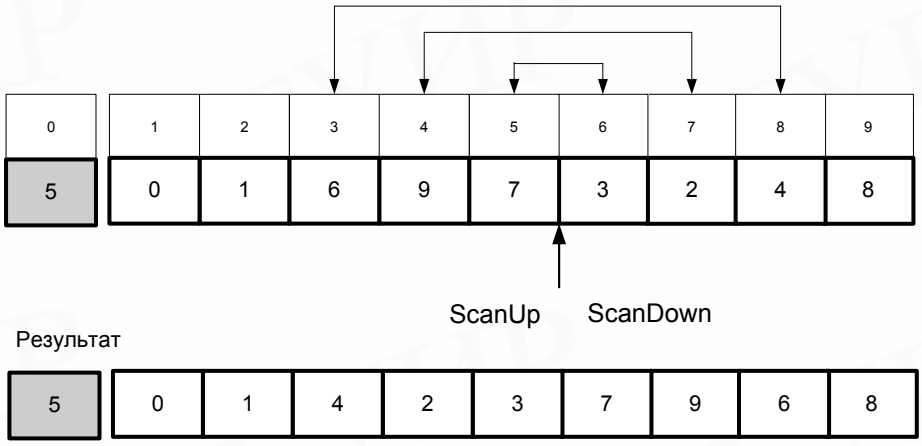


Рис. 5. Быстрая сортировка (шаг 2)

Затем происходит обмен значениями центрального элемента ***A***[0] с ***A***[***Down***]:

Swap(***A***[0], ***A***[***Down***]).

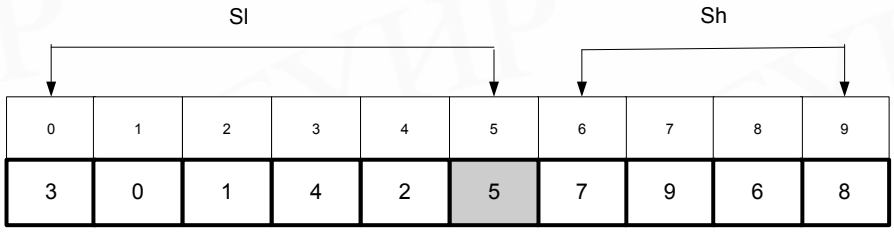


Рис. 6. Быстрая сортировка (шаг 3)

В результате появляются два подмассива: ***A***[0]…***A***[5] и ***A***[6]...***A***[9], причем все элементы первого подмассива меньше элементов второго, а последний элемент первого подмассива является его наибольшим элементом (рис. 6). Таким образом, можно считать, что после проделанных шагов подмассивы разделены элементом ***A***[5]=5. Поскольку оба подмассива являются такими же массивами, как и исходный, то к ним можно применить рассмотренный выше алгоритм. Применение одного и того же алгоритма к частям массива называется рекурсией.

Одним и тем же методом обрабатываются два подмассива: ***Al*** (***A***[0]… ***A***[4]) и ***Ah*** (***A***[6]…***A***[9]). Элемент ***A***[5] обрабатывать не надо, т.к. он уже находится на своем месте.

Тот же алгоритм применяется для каждого подмассива, разбивая эти подмассивы на меньшие части, пока в подмассиве не останется по одному элементу или пока подмассив не опустеет. Это соответствует состоянию массива, в котором все элементы отсортированы.

Анализ алгоритма быстрой сортировки.

Анализ эффективности быстрой сортировки достаточно труден. Будет лучше показать ее вычислительную сложность, подсчитав число сравнений при некоторых идеальных допущениях [1,2]. Пусть размерность исходного массива ***n*** является степенью двойки, ***n***=2^k (***k***=$\log_2 n$), а центральный элемент располагается точно посередине каждого массива и разбивает его на два подмассива

примерно одинаковой длины. При первом просмотре производится (***n***-1) сравнений. В результате создаются два подмассива размером (***n***/2). На следующей фазе обработка каждого подмассива требует примерно (***n***/2) сравнений. Общее число сравнений на этой фазе равно 2*(***n***/2)=***n***. На следующей фазе обрабатываются четыре подмассива, что требует 4*(***n***/4) сравнений, и т.д. Процесс разбиения прекращается после ***k*** проходов, когда получившиеся подмассивы содержат по одному элементу. Общее число сравнений приблизительно равно

$$n + 2(n/2) + 4(n/4) + ... + n(n/n) = n + n + ... + n = n * k = n * \log_2 n.$$

Таким образом, для массива общего вида вычислительная сложность быстрой сортировки равна (***n*log$_2$*n***). Идеальный случай, рассмотренный выше, фактически возникает тогда, когда массив уже отсортирован по возрастанию. Тогда центральный элемент попадает точно в середину каждого подмассива.

Наихудшим сценарием для быстрой сортировки будет тот, при котором центральный элемент все время попадает в одноэлементный подмассив, а все прочие элементы остаются во втором подмассиве. Это происходит тогда, когда центральным всегда является наименьший элемент (или наибольший). В этом случае на первом проходе производится ***n*** сравнений, а больший подмассив содержит (***n-1***) элементов. На следующем проходе этот подмассив требует (***n-1***) сравнений и дает подмассив из (***n-2***) элементов и т.д. Сложность сортировки для худшего случая пропорциональна n^2, т.е. не лучше, чем для сортировок вставками и выбором. Однако этот случай маловероятен на практике. В целом, средняя производительность быстрой сортировки выше, чем у всех рассмотренных ранее сортировок. Алгоритм быстрой сортировки (QuickSort) выбирается за основу в большинстве универсальных сортирующих утилит.

В прил. 11 дана схема алгоритма быстрой сортировки в нерекурсивной реализации. Для хранения границ подмассивов используется двумерный массив, первый столбец его хранит левые индексы подмассивов, а второй столбец – правые индексы.

В прил. 12 дана схема алгоритма быстрой сортировки в рекурсивной реализации.

В прил. 13 дана программа, написанная на языке Pascal, рекурсивной реализации быстрой сортировки.

2. 3. ПИРАМИДАЛЬНАЯ СОРТИРОВКА (HeapSort)

Метод пирамидальной сортировки (другое название HeapSort) базируется на сортировке прямым выбором (см. подразд. 1.2) и по сути является его усовершенствованием.

Метод сортировки с помощью прямого выбора основан на повторяющихся поисках наименьшего элемента среди ***n*** элементов, затем среди оставшихся (***n***-1) элементов и т.д. Усовершенствование прямого выбора заключается в том, что после каждого прохода сохраняется больше информации, чем просто иден-

тификация единственного минимального элемента. Например, сделав (***n***/2) сравнений, можно определить в каждой паре элементов наименьший. С помощью (***n***/4) сравнений – наименьший из пары уже выбранных наименьших и т.д. Проделав (***n***-1) сравнений, можно построить дерево выбора и идентифицировать его корень как искомый наименьший элемент.

Пусть ***A***[1]…***A***[***n***] – исходный массив. Сопоставим ему дерево, используя следующие правила:

1***)*** ***A***[1] – корень дерева;

2) если ***A***[***i***] элемент дерева и (2****i***) ≤ ***n***, то ***A***[2****i***] – "левый" потомок элемента ***A***[***i***];

3) если ***A***[***i***] элемент дерева и (2****i***+1) ≤ ***n***, то ***A***[2****i***+1] – "правый" потомок элемента ***A***[***i***].

Правила [1–3] определяют в массиве структуру дерева, причём глубина дерева не превышает ([$\log_2 \boldsymbol{n}$] + 1). Они же задают способ перемещения по дереву от корня к листьям (рис. 7). Движение в обратном направлении задаётся правилом 4:

4) если ***A***[***i***] – узел дерева и ***i*** > 1, то ***A***[***i*** div 2] – узел "предок" узла ***A***[***i***].

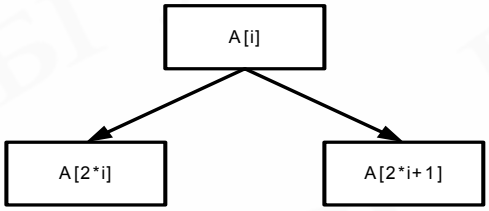


Рис. 7. Представление массива в виде дерева

Пример: пусть дан массив чисел ***A***={9, 2, 4, 7, 1, 6, 10, 8, 3, 5}.
На рис. 8 дано представление массива в виде бинарного дерева.

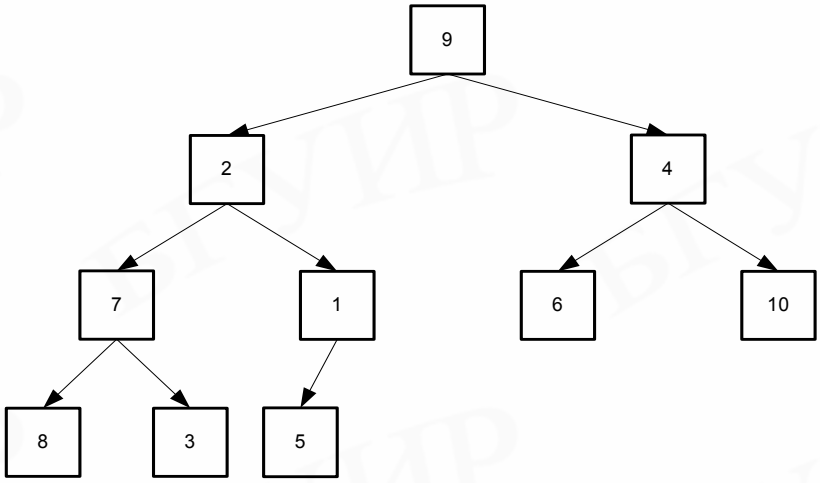


Рис. 8. Представление массива в виде бинарного дерева

Таким образом, при начальном проходе по массиву создается некоторая деревообразная структура.

Для сохранения информации вводится понятие пирамиды.

Пирамида определяется как последовательность элементов массива A_l, $A_{l+1}, .., A_N$, такая, что

$$A_i \le A_{2*i} \quad и \quad A_i \le A_{2*i+1} \; для \; i = l...n/2.$$

Построим для рассмотренного выше примера пирамиду, выполнив последовательно перестановки пар следующих элементов: 9 и 2, 9 и 1, 2 и 1, 9 и 5 (рис.9).

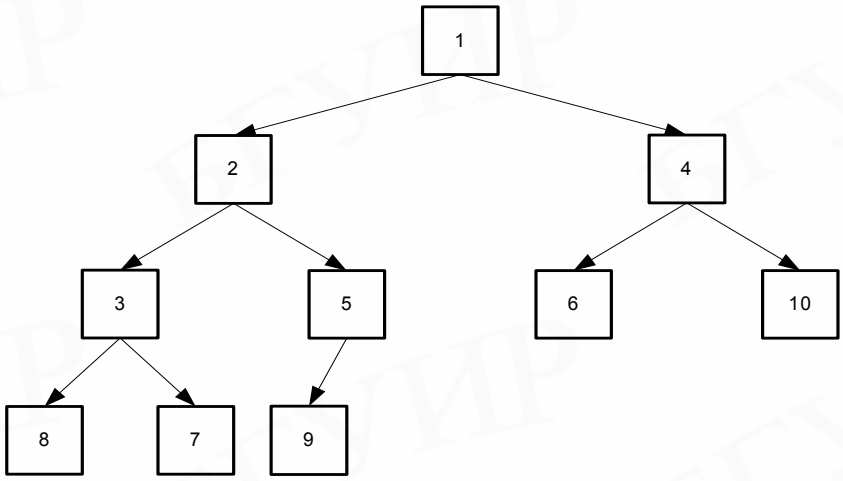


Рис. 9. Представление массива в виде пирамиды

В работе [1] описывается способ построения пирамиды “на том же месте”. Пирамида расширяется влево. Каждый раз добавляется и сдвигами ставится в надлежащую позицию новый элемент (процедура Sift в прил. 15). Процесс построения пирамиды из ***n*** элементов на том же самом месте можно записать на языке Pascal как

```
L:=(n div 2)+1;
While L>1 do
begin
   L:=L-1;
   Sift(L,n)
end;
```

Очевидно, что построение пирамиды приводит к выталкиванию в корень дерева наименьшего элемента. Поэтому для реализации пирамидальной сортировки осталось решить последнюю задачу – исключить из дальнейшего рассмотрения элемент, находящийся в корне. Этого можно добиться, обменивая последний элемент в массиве с наименьшим, т.е. находящимся в корне. Таким образом, в корень дерева помещается последний элемент массива и пирамида вновь строится для подмассива от 1-го элемента до предпоследнего. Элемент, передвинувшийся в корень дерева, вновь будет наименьшим элементом, и его можно исключить аналогичным способом, путем обмена с предпоследним элементом массива, и т.д.

После ***n*** таких шагов процесс сортировки заканчивается. Процедура построения пирамиды для подмассивов называется спуском или просеиванием элементов.

Обратим внимание, что на каждом из ***n*** шагов выбора требуется только $\log_2 n$ сравнений. Поэтому на весь процесс понадобится порядка ($n*\log_2 n$) элементарных операций плюс еще ***n*** шагов на построение дерева. Это весьма существенное улучшение не только прямого метода, требующего n^2 шагов, но и даже метода Шелла, где нужно $n^{1.2}$ шага.

Пример: пусть исходный массив представлен следующей совокупностью элементов ***A*** ={44, 55, 12, 42, 94, 18, 06, 67}, проиндексированных в диапазоне ***i***=[1]...[8]. Сначала в подмассив {94, 18, 06, 67} вставляется элемент 42 (***i***=4) путем сравнения с элементом 67 (***i***=8) (см. определение пирамиды). Затем в подмассив {42, 94, 18, 06, 67} вставляется элемент 12 (***i***=3) путем сравнения с элементом минимальным из пары {18, 06} (***i***=6 и ***i***=7 соответственно) и обмена с ним, т.е. с элементом 06. Затем в подмассив {06, 42, 94, 18, 12, 67} вставляется элемент 55 путем сравнения с элементом минимальным из пары {42, 94} (***i***=4 и ***i***=5) и обмена с элементом 42. Наконец, в подмассив {42, 06, 55, 94, 18, 12, 67} вставляется элемент 44 путем сравнения с элементом, минимальным из пары {42, 06} (***i***=2 и ***i***=3), и обмена с элементом 06 и далее сравнения его с элементом, минимальным из пары {18, 12} (***i***=6 и ***i***=7), и обмена с элементом 12. Если после выполнения этих действий представить результирующий массив в виде дерева (рис.10), то для каждого элемента дерева будет выполняться требование пирамиды, т.е. элемент 06 меньше элементов {42, 12}; элемент 42 меньше элементов {55, 94}; элемент 12 меньше элементов {18, 44}; элемент 55 меньше элемента 67.

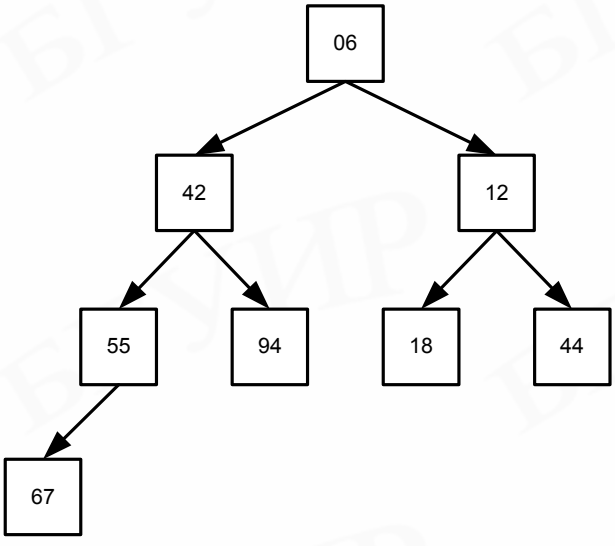


Рис.10. Представление массива в виде пирамиды

Табл. 7 иллюстрирует процесс построения пирамиды для рассмотренного выше примера. Темным цветом выделен вставляемый элемент. Светлым цветом выделана часть массива, представляемая как пирамида.

Таблица 7

Построение пирамиды

0-й шаг	**44**	**55**	**12**	**42**	**94**	**18**	**06**	**67**
1-й шаг (старт)	**44**	**55**	**12**	**42**	**94**	**18**	**06**	**67**
Вставка	**44**	**55**	**12**	**42**	**94**	**18**	**06**	**67**
2-й шаг (старт)	**44**	**55**	**12**	**42**	**94**	**18**	**06**	**67**
Вставка	**44**	**55**	**6**	**42**	**94**	**18**	**12**	**67**
3-й шаг (старт)	**44**	**55**	**6**	**42**	**94**	**18**	**12**	**67**
Вставка	**44**	**42**	**6**	**55**	**94**	**18**	**12**	**67**
4-й шаг (старт)	**44**	**42**	**6**	**55**	**94**	**18**	**12**	**67**
Вставка	**06**	**42**	**12**	**55**	**94**	**18**	**44**	**67**

Процесс собственно сортировки иллюстрируется табл. 8. Серым цветом помечены элементы дерева, исключенные из последующего просмотра, т.е. стоящие на своих местах.

Таблица 8

Сортировка массива

1-й шаг (старт)	**06**	**42**	**12**	**55**	**94**	**18**	**44**	**67**
Обмен	**67**	**42**	**12**	**55**	**94**	**18**	**44**	**06**
Просеивание	**12**	**42**	**18**	**55**	**94**	**67**	**44**	**06**
2-й шаг (старт)	**12**	**42**	**18**	**55**	**94**	**67**	**44**	**06**
Обмен	**44**	**42**	**18**	**55**	**94**	**67**	**12**	**06**
Просеивание	**18**	**42**	**44**	**55**	**94**	**67**	**12**	**06**
3-й шаг (старт)	**18**	**42**	**44**	**55**	**94**	**67**	**12**	**06**
Обмен	**67**	**42**	**44**	**55**	**94**	**18**	**12**	**06**
Просеивание	**42**	**55**	**44**	**67**	**94**	**18**	**12**	**06**
4-й шаг (старт)	**42**	**55**	**44**	**67**	**94**	**18**	**12**	**06**
Обмен	**94**	**55**	**44**	**67**	**42**	**18**	**12**	**06**
Просеивание	**44**	**55**	**94**	**67**	**42**	**18**	**12**	**06**
5-й шаг (старт)	**44**	**55**	**94**	**67**	**42**	**18**	**12**	**06**
Обмен	**67**	**55**	**94**	**44**	**42**	**18**	**12**	**06**
Просеивание	**55**	**67**	**94**	**44**	**42**	**18**	**12**	**06**
6-й шаг (старт)	**55**	**67**	**94**	**44**	**42**	**18**	**12**	**06**
Обмен	**94**	**67**	**55**	**44**	**42**	**18**	**12**	**06**
Просеивание	**67**	**94**	**55**	**44**	**42**	**18**	**12**	**06**
7-й шаг (старт)	**67**	**94**	**55**	**44**	**42**	**18**	**12**	**06**
Обмен	**94**	**67**	**55**	**44**	**42**	**18**	**12**	**06**
Просеивание	**94**	**67**	**55**	**44**	**42**	**18**	**12**	**06**

На рис.11–17 дано представление массива в виде дерева, иллюстрирующее последовательные сортирующие шаги. Исходный массив {06, 42, 12, 55, 94, 18, 44, 67} представляет пирамиду, на вершине которой находится минимальный в массиве элемент; выполняется обмен элементов 06 и 67 (рис. 13, слева), таким образом, минимальный элемент ставится в конец массива и в дальнейшем исключается из рассмотрения, а элемент 67 начинает просеиваться сквозь пирамиду для ее восстановления путем обмена с минимальным из пары {42, 12}, т.е. элементом 12, и далее путем обмена с минимальным из пары {18, 44}, т.е. эле-

ментом 18, и т.д. Результат просеивания представлен на рис.11 (справа). На вершину пирамиды поднялся очередной минимальный элемент. На рис.12 (слева) показан обмен минимального элемента 12 с предпоследним элементом дерева. В результате правило пирамиды нарушено и выполняется просеивание элемента через пирамиду для ее восстановления (см. рис.12, справа) и т.д.

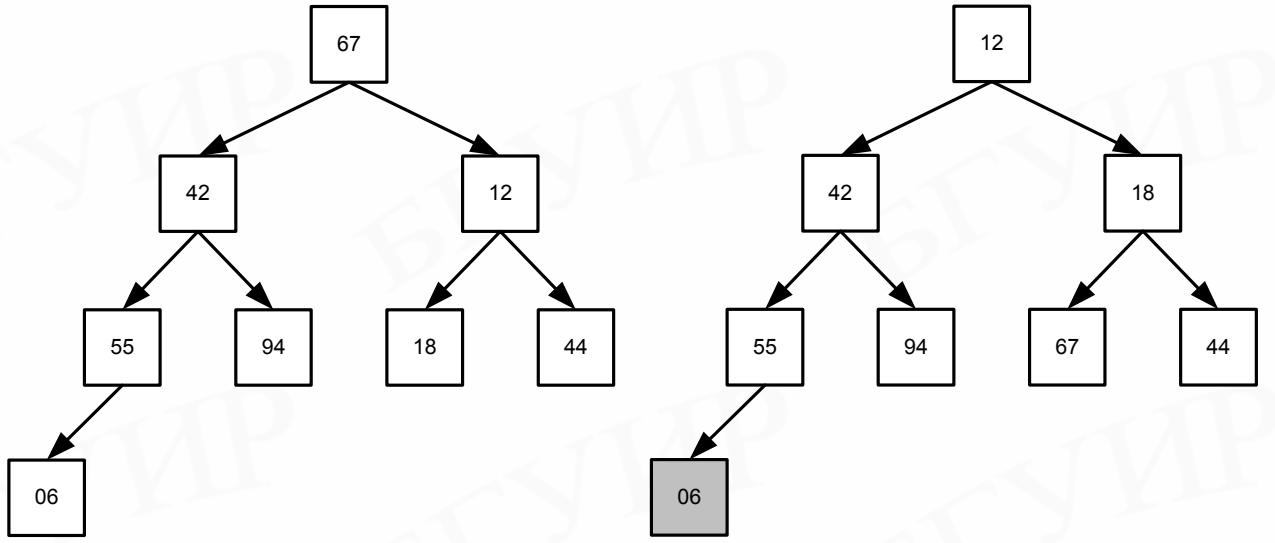


Рис.11. Сортировка массива (шаг 1: обмен, просеивание)

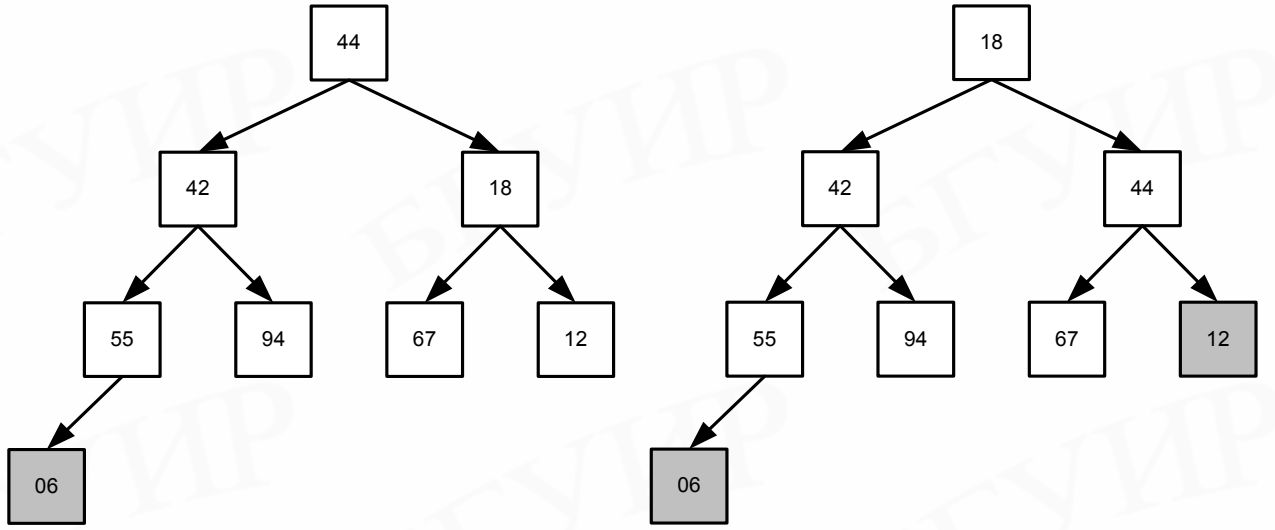


Рис.12. Сортировка массива (шаг 2: обмен, просеивание)

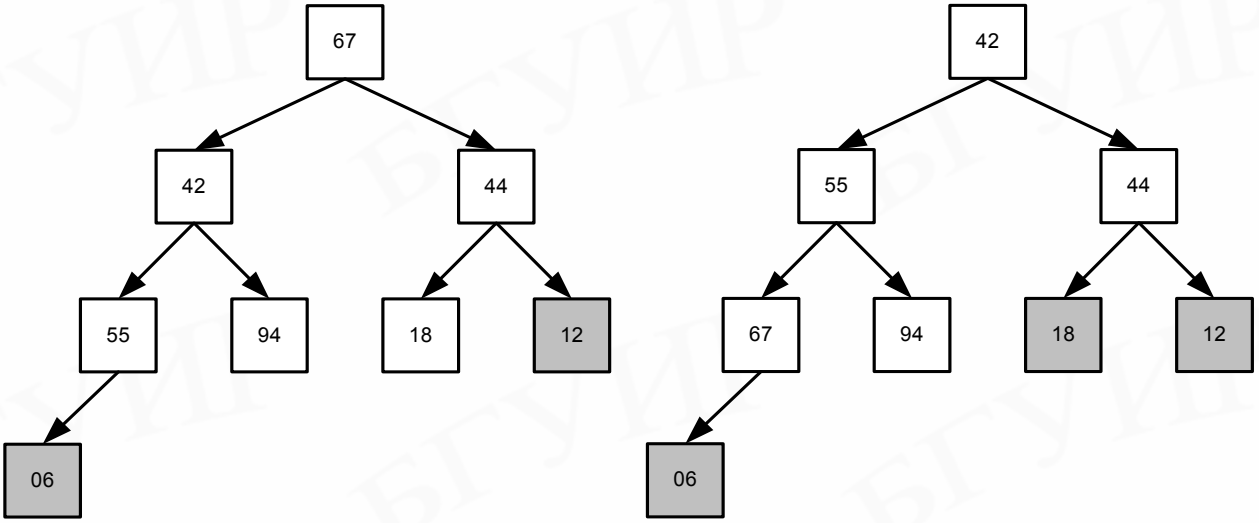


Рис.13. Сортировка массива (шаг 3: обмен, просеивание)

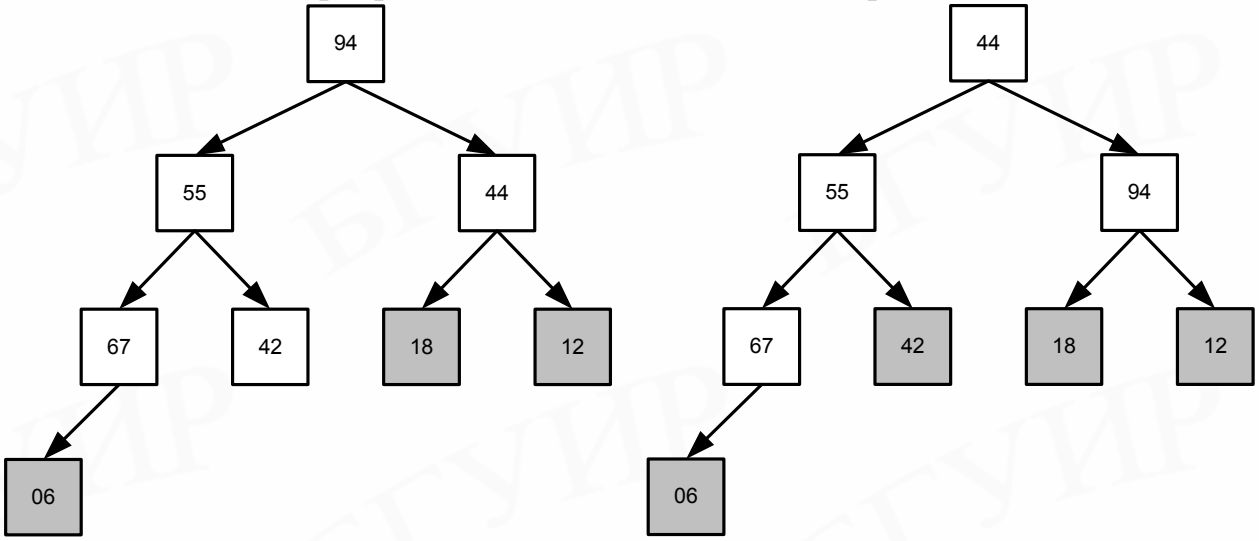


Рис.14. Сортировка массива (шаг 4: обмен, просеивание)

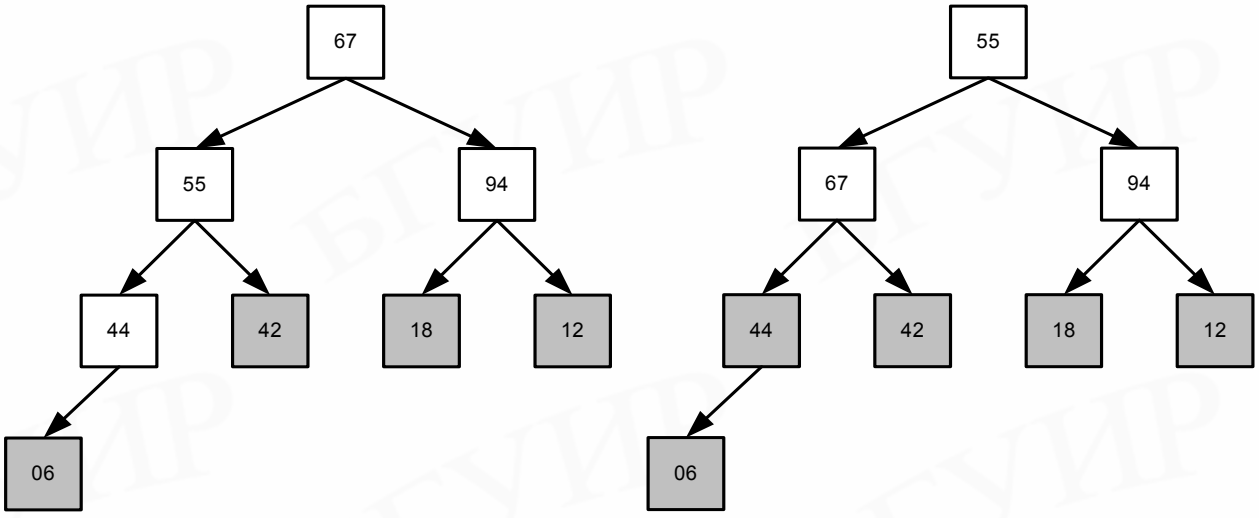


Рис.15. Сортировка массива (шаг 5: обмен, просеивание)

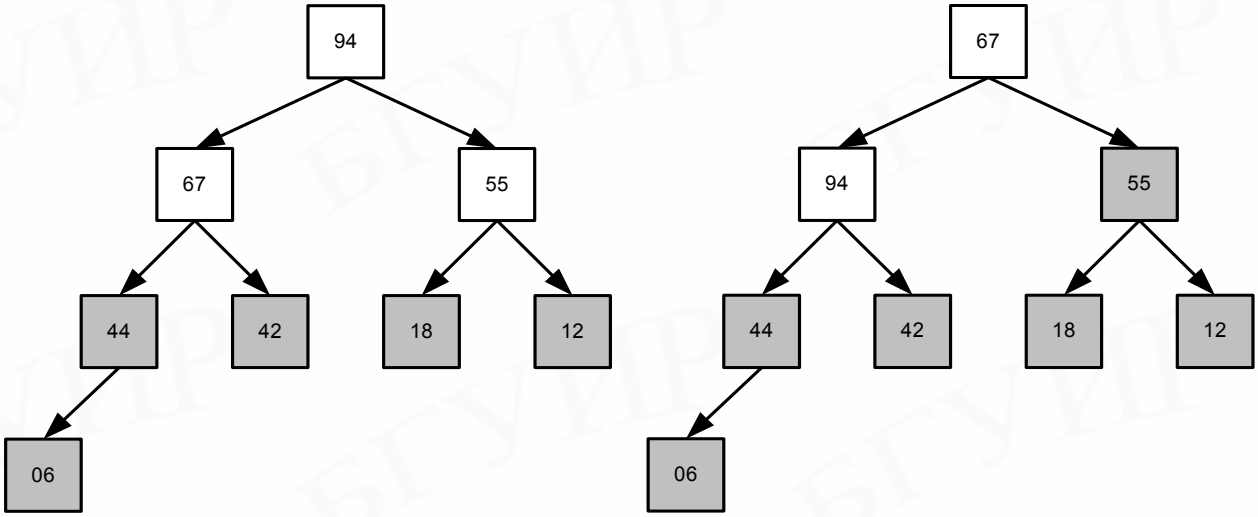


Рис.16. Сортировка массива (шаг 6: обмен, просеивание)

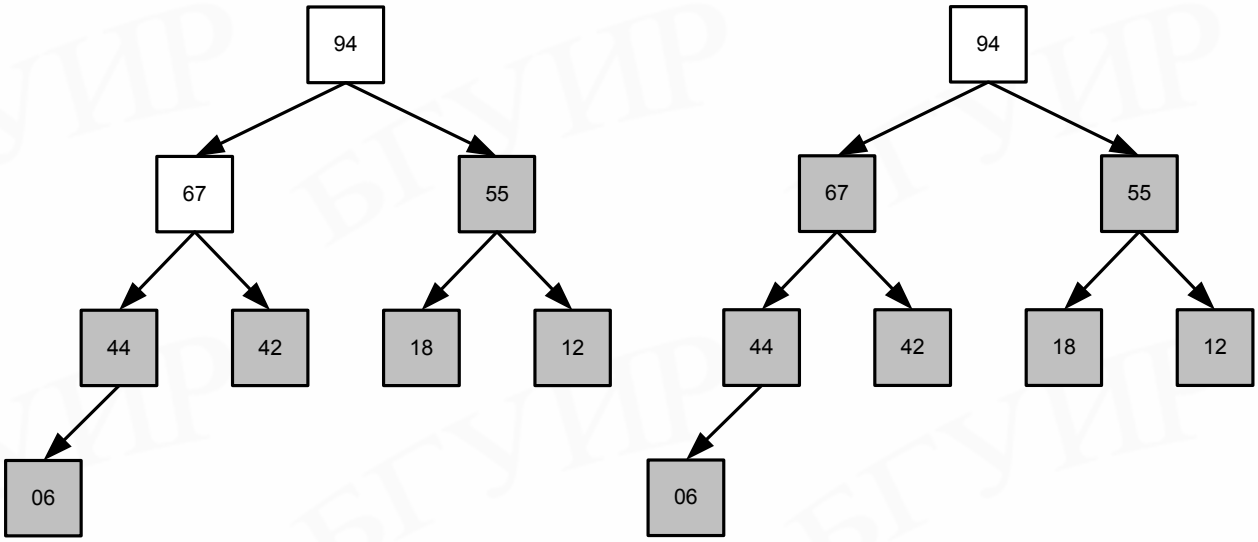


Рис.17. Сортировка массива (шаг 7: обмен, просеивание)

В рассмотренном примере сортировка массива выполняется по убыванию. Обратного порядка сортировки добиваются изменением направления “упорядочивающего отношения” в процедуре Sift. Схема алгоритма пирамидальной сортировки представлена в прил. 14.

Анализ метода пирамидальной сортировки HeapSort.

Метод пирамидальной сортировки не рекомендуется применять для массивов с небольшим числом элементов. Для больших же массивов сортировка HeapSort очень эффективна [1,2].

Проанализируем наихудший случай: для упорядочения массива из n элементов необходимо выполнить (n/2) сдвигающих шагов, они сдвигают элементы на $\log_2(n/2)$, $\log_2(n/2\text{-}1)$,..., $\log_2(n\text{-}1)$ позиций (логарифм округляется до следующего меньшего целого). Следовательно, фаза сортировки требует (n-1) сдвигов с самое большое $\log_2(n\text{-}1)$, $\log_2(n\text{-}2)$, ..., 1 перемещениями. Кроме того, нужно еще (n-1) перемещений для просеивания сдвинутого элемента на неко-

торое расстояние вправо. Эти соображения показывают, что даже в самом плохом из возможных случаев сортировка HeapSort потребует $\log_2 n$ шагов.

Очевидно, сортировка HeapSort "любит" начальные последовательности, в которых элементы более или менее отсортированы в обратном порядке. Поэтому "поведение" метода неестественно. При наличии обратного порядка расположения элементов в массиве фаза построения пирамиды не требует вообще каких-либо перемещений. Среднее число перемещений приблизительно равно ($n/2*\log_2 n$), причем отклонения от этого значения относительно невелики.

3. СОРТИРОВКА ПОСЛЕДОВАТЕЛЬНОСТЕЙ (файлов)

Алгоритмы сортировки, приведенные в предыдущем разделе, нельзя применять непосредственно для данных, которые из-за своих размеров не помещаются в оперативную память машины. В таком случае данные представляют собой файл (последовательный файл). Для файла характерно, что доступ к элементам производится в строго определенной последовательности. Поэтому для таких структур данных разработаны специальные методы сортировки.

Разумеется, в том случае, когда размер файла невелик и объём оперативной памяти достаточен для его размещения, можно прочитать файл в память, отсортировать полученный массив одним из методов внутренней сортировки, а затем отсортированный массив записать в файл. Однако это не является типичным случаем.

Далее мы рассмотрим общий случай, когда сортируемый файл имеет объем значительно больший, чем доступно оперативной памяти.

Наиболее важный из методов сортировки файлов – сортировка с помощью слияния. Слияние – это объединение нескольких последовательностей данных в одну упорядоченную последовательность методом повторяющегося выбора из доступных в данный момент элементов файла.

3. 1. ПРОСТОЕ СЛИЯНИЕ

Одна из сортировок на основе слияния называется простым слиянием. Она выполняется следующим образом:

Шаг 1. Исходная последовательность (файл) ***A*** разбивается на две половины: ***B*** и ***C***.

Шаг 2. Части ***B*** и ***C*** сливаются в последовательность ***A*** так, что каждая пара в последовательности ***A*** теперь является упорядоченной.

Шаг 3. Шаги 1–2 повторяются для вновь полученной последовательности ***A***. При этом упорядоченные пары переходят в упорядоченные четвёрки, восьмёрки и т.д. до тех пор, пока не будет упорядочена вся последовательность целиком.

Пример: пусть дана последовательность (файл) чисел: {6, 7, 2, 5, 9, 4, 1, 8}. Для сортировки необходимо использовать три ленты (рабочие файлы ***A,B,C***) (рис.18):

A	6	7	2	5	9	4	1	8
B								

Рис. 18. Сортировка простым слиянием (исходное состояние)

После первого разбиения получаются две последовательности (рис.19).

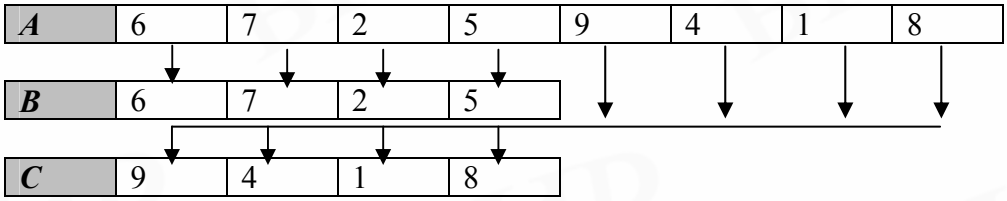


Рис. 19. Сортировка простым слиянием (фаза разделения)

Слияние одиночных компонент (т.е. упорядоченных последовательностей длиной в 1 элемент) в упорядоченные пары даёт упорядоченные пары, представленные на рис. 20.

Рис. 20. Сортировка простым слиянием (фаза слияния, шаг 1)

Полученная последовательность вновь делится пополам, но сливаются уже упорядоченные четвёрки (рис.21).

Рис. 21. Сортировка простым слиянием (фаза слияния, шаг 2)

Третье разделение и слияние приводят к отсортированной последовательности (рис.22).

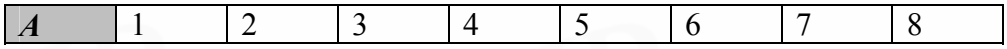


Рис. 22. Сортировка простым слиянием (фаза слияния, шаг 3)

Известно [1], что действия по однократной обработке всего набора данных называются фазой, а наименьший процесс, который периодически повторяется в ходе сортировки, называется проходом или этапом. В приведенном примере сортировка выполняется за три прохода, каждый из которых состоит из фазы разделения и фазы слияния. Для выполнения такой сортировки нужны три вспомогательных файла (ленты), поэтому она называется трехленточным слиянием.

Фазы разделения фактически не относятся к сортировке, т.к. в них элементы не переставляются. В этом смысле они непродуктивны, хотя и занимают половину всех операций по перезаписи. Если объединить разделение со слиянием, то от фаз разделения можно избавиться. Для этого вместо слияния в одну последовательность результаты слияния будут сразу распределяться по двум файлам (лентам), которые станут исходными для последующего прохода. В отличие от упомянутой двухфазной сортировки с помощью слияния такая сортировка будет называться однофазной. Она эффективнее, поскольку необходима только половина операций по перезаписи, однако для ее реализации необходим четвертый файл (лента).

Пример: пусть дан файл ***A***, рассмотренный в предыдущем примере (рис. 23).

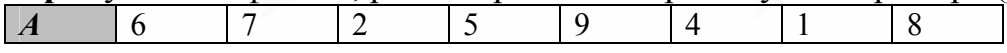


Рис. 23. Сортировка простым слиянием (исходное состояние)

При 1-м проходе выполняется первичное разделение файла ***A*** на файлы ***C*** и ***Д*** (рис.24).

A	6	7	2	5	9	4	1	8
B								
C	6	7	2	5				
Д	9	4	1	8				

Рис. 24. Сортировка простым слиянием (1-й проход)

На 2-м проходе выполняется слияние и разделение с упорядочением одиночных компонент в пары (файлы ***A*** и ***B***) (рис. 25).

A	6	9	1	2
B	4	7	5	8
C	6	7	2	5
Д	9	4	1	8

Рис. 25. Сортировка простым слиянием (2-й проход)

На 3-м проходе выполняется слияние и разделение с упорядочением пар компонент в четверки (файлы ***C*** и ***Д***) (рис.26).

A	6	9	1	2
B	4	7	5	8
C	4	6	7	9
Д	1	2	5	8

Рис. 26. Сортировка простым слиянием (3-й проход)

На 4-м проходе выполняется слияние с упорядочением четверок компонент в восьмерки (файлы ***A*** и ***B***) (рис.27). Для рассмотренного примера на этом шаге в файле ***A*** будет получен результат.

A	1	2	4	5	6	7	8	9

Рис. 27. Сортировка простым слиянием (4-й проход)

Анализ сортировки с помощью простого слияния.

Поскольку при каждом проходе число сливаемых компонент ***p*** удваивается (***p***=1,2,4,8,...) и сортировка заканчивается при ***p≥n***, то всего потребуется $\log_2 n$ проходов. На каждом проходе по определению копируются по одному разу все ***n*** элементов. Поэтому общее число пересылок равно (***n***$*\log_2 n$).

Число сравнений элементов ***C*** меньше числа перестановок ***M***, поскольку при копировании остатков никаких сравнений не производится. Однако сортировки слиянием используются при сортировке данных, хранящихся на внешних носителях, поэтому операция пересылки данных намного более трудоемкая по сравнению с операцией сравнения. Таким образом, детальный анализ числа сравнений не представляет практического интереса.

Метод сортировки слиянием сравним по производительности с быстрыми методами сортировки массивов. Однако минимальный объём памяти, необходимый для работы данного метода, составляет минимально (***2*n***) элементов, к

тому же высоки затраты на работу с индексами. Поэтому сортировка слиянием для массивов, т.е. для данных, находящихся в оперативной памяти, используется редко.

3. 2. ЕСТЕСТВЕННОЕ СЛИЯНИЕ

Метод простого слияния не учитывает особенности случая, при котором в файле есть длинные частично упорядоченные подпоследовательности. Размер сливаемых на ***k***-м шаге подпоследовательностей меньше или равен 2^k и не зависит от существования более длинных уже упорядоченных подпоследовательностей, которые можно было бы просто объединить. Фактически любые две упорядоченные подпоследовательности длиной ***m*** и ***n*** можно сразу сливать в одну последовательность из (***m+n***) элементов. Сортировка, при которой всегда сливаются две самые длинные из возможных подпоследовательностей, называется естественным слиянием.

Упорядоченные подпоследовательности называются сериями. В сортировке естественным слиянием объединяются максимальные упорядоченные серии, а не последовательности фиксированной заранее длины (p=1,2,4,8,….).

Пример: пусть дан файл ***A*** из 20 чисел (рис.28). Сортировка выполняется с использованием 4 вспомогательных файлов, т.е. на 4-х лентах (файлы ***A,B,C,Д***). На рис. 28 приведено исходное состояние файла (этап 0) и его состояния после каждого из проходов (этапы 1–4) с помощью естественного сбалансированного слияния (рис.29–32). Для полной сортировки файла нужно всего 4 прохода. Процесс сортировки заканчивается, как только число серий становится равным единице. Предполагается также, что в начальной последовательности есть по крайней мере одна непустая серия. Поэтому количество серий необходимо запоминать в отдельной переменной. Тогда признаком конца сортировки можно считать равенство этой переменной единице. На рис. 28–32 жирным шрифтом выделены серии элементов (упорядоченные подпоследовательности).

Этап 0:

A	**17**	**31**	**05**	**59**	**13**	**41**	**43**	**67**	**11**	**23**	**29**	**47**	**03**	**07**	**71**	**02**	**19**	**57**	**37**	**61**
B																				
C																				
Д																				

Рис. 28. Сортировка естественным слиянием (исходное состояние)

Этап 1:

A	**17**	**31**	**05**	**59**	**13**	**41**	**43**	**67**	**11**	**23**	**29**	**47**	**03**	**07**	**71**	**02**	**19**	**57**	**37**	**61**
B																				
C	**17**	**31**	**13**	**41**	**43**	**67**	**03**	**07**	**71**	**37**	**61**									
Д	**05**	**59**	**11**	**23**	**29**	**47**	**02**	**19**	**57**											

Рис. 29. Сортировка естественным слиянием (1-й проход)

Этап 2:

A	**05**	**17**	**31**	**59**	**02**	**03**	**07**	**19**	**57**	**71**										
B	**11**	**13**	**23**	**29**	**41**	**43**	**47**	**67**	**37**	**61**										
C	**17**	**31**	**13**	**41**	**43**	**67**	**03**	**07**	**71**	**37**	**61**									
Д	**05**	**59**	**11**	**23**	**29**	**47**	**02**	**19**	**57**											

Рис. 30. Сортировка естественным слиянием (2-й проход)

Этап 3:

A	**05**	**17**	**31**	**59**	**02**	**03**	**07**	**19**	**57**	**71**										
B	**11**	**13**	**23**	**29**	**41**	**43**	**47**	**67**	**37**	**61**										
C	**05**	**11**	**13**	**17**	**23**	**29**	**31**	**41**	**43**	**47**	**59**	**67**								
Д	**02**	**03**	**07**	**19**	**37**	**57**	**61**	**71**												

Рис. 31. Сортировка естественным слиянием (3-й проход)

Этап 4:

A	**02**	**03**	**05**	**07**	**11**	**13**	**17**	**19**	**23**	**29**	**31**	**37**	**41**	**43**	**47**	**57**	**59**	**61**	**67**	**71**
B																				
C	**05**	**11**	**13**	**17**	**23**	**29**	**31**	**41**	**43**	**47**	**59**	**67**								
Д	**02**	**03**	**07**	**19**	**37**	**57**	**61**	**71**												

Рис. 32. Сортировка естественным слиянием (4-й проход)

Анализ естественного слияния.

Если сливаются две последовательности, состоящие из ***n*** серий каждая, то результирующая последовательность содержит опять ровно ***n*** серий. Следовательно, при каждом проходе общее число серий уменьшается вдвое и общее число пересылок в самом плохом случае равно значению (***n****$\log_2$***n)*** (округленному к ближайшему большему целому), а в среднем даже меньше. Ожидаемое число сравнений значительно больше, поскольку кроме тех из них, которые необходимы для отбора элементов при слиянии, нужны ещё дополнительные сравнения между последовательными элементами каждого файла, необходимые для определения конца серии.

ЗАДАНИЯ № 1

Для заданного преподавателем варианта из таблицы провести сравнительный анализ методов сортировки массивов:

Вариант	Методы сортировки	Критерий
1	**Метод выбора** **Сортировка Шелла**	**а) Сравнения** **в) Перестановки**
2	**Метод «плавающего пузырька»** **Метод простых вставок**	**а) Сравнения** **в) Перестановки**
3	**Шейкерная сортировка** **Метод простых вставок (с барьером)**	**а) Сравнения** **в) Перестановки**
4	**Метод выбора** **Метод обмена**	**а) Сравнения** **в) Перестановки**

5	**Метод выбора** **Метод бинарных вставок**	**а) Сравнения** **в) Перестановки**
6	**Метод обмена с «флажком»** **Пирамидальная сортировка**	**а) Сравнения** **в) Перестановки**
7	**Шейкерная сортировка** **Быстрая сортировка**	**а) Сравнения** **в) Перестановки**
8	**Сортировка Шелла** **Пирамидальная сортировка**	**а) Сравнения** **в) Перестановки**
9	**Метод бинарных вставок** **Быстрая сортировка**	**а) Сравнения** **в) Перестановки**
10	**Сортировка Шелла** **Шейкерная сортировка**	**а) Сравнения** **в) Перестановки**
11	**Быстрая сортировка** **Сортировка Шелла**	**а) Сравнения** **в) Перестановки**
12	**Метод «плавающего пузырька»** **Метод бинарных вставок**	**а) Сравнения** **в) Перестановки**
13	**Шейкерная сортировка** **Метод бинарных вставок**	**а) Сравнения** **в) Перестановки**
14	**Метод выбора** **Метод «плавающего пузырька»**	**а) Сравнения** **в) Перестановки**
15	**Метод выбора** **Пирамидальная сортировка**	**а) Сравнения** **в) Перестановки**
16	**Метод обмена с «флажком»** **Быстрая сортировка**	**а) Сравнения** **в) Перестановки**
17	**Шейкерная сортировка** **Быстрая сортировка**	**а) Сравнения** **в) Перестановки**
18	**Сортировка Шелла** **Пирамидальная сортировка**	**а) Сравнения** **в) Перестановки**
19	**Метод простых вставок (с барьером)** **Быстрая сортировка**	**а) Сравнения** **в) Перестановки**
20	**Пирамидальная сортировка** **Шейкерная сортировка**	**а) Сравнения** **в) Перестановки**
21	**Метод бинарных вставок** **Метод простых вставок**	**а) Сравнения** **в) Перестановки**

Исследования провести для массивов следующих размерностей: *N*=100, 200, 300, 500, 1000, 2000; для типов массивов: случайный, сортированный, перевернутый.

Результаты расчетов свести в таблицу.

Лабораторную работу выполнять по следующей схеме:

Этап 1: обьявить три массива:

```
Type
    TM = Array [1..2000] of integer;
                                    Var
    M1: TM1;
    M2: TM2;
    M3: TM3;
```

для хранения 2 000 случайных целых чисел, расположенных в одном из указанных ниже порядков:

1) неотсортированные данные (случайный массив);
2) отсортированные данные (сортированный массив);
3) данные, отсортированные в обратном порядке (перевернутый массив).

Этап 2: разработать программы, реализующие методы сортировки (согласно заданию).

Этап 3: отсортировать в порядке «возрастания» массивы, выбирая соответственно из них подмассивы размерностей N=100, 200, 300, 500, 1 000, 2 000 элементов каждым из исследуемых методов сортировки. В процессе сортировки для каждого выбранного массива подсчитать количество сравнений элементов (или количество перестановок согласно заданию). Предварительно разработать структуру данных для хранения результатов подсчетов.

Этап 4: результаты подсчетов вывести на экран в виде таблицы:

Размерность массива	1-й метод сортировки		2-й метод сортировки	
	Количество экспе-риментальное	Количество тео-ретическое	Количество экспе-риментальное	Количество теоре-тическое
N=100				
⋮				
N=2000				

Этап 5: согласно полученным результатам сделать соответствующие выводы и занести их в отчет.

ЗАДАНИЯ № 2

1. Дана матрица $\boldsymbol{X}$[10,10]. Упорядочить элементы строк матрицы по убыванию, а сами строки по возрастанию элементов 1-го столбца (использовать сортировку выбором).
2. Дана матрица $\boldsymbol{X}$[9,8]. Упорядочить элементы строк матрицы по возрастанию, а сами строки по возрастанию суммы элементов строк (использовать сортировку обменами).
3. Дана матрица $\boldsymbol{X}$[8,9]. Упорядочить элементы строк матрицы по неубыванию, а сами строки по убыванию максимальных элементов строк (использовать сортировку простыми вставками).
4. Дана матрица $\boldsymbol{X}$[8,10]. Упорядочить элементы строк матрицы по невозрастанию, а сами строки по возрастанию элементов 10-го столбца (использовать сортировку бинарными вставками).
5. Дана матрица $\boldsymbol{X}$[9,7]. Упорядочить элементы строк матрицы по неубыванию, а сами строки по убыванию минимальных элементов строк (использовать шейкерную сортировку).

6. Дана матрица ***X***[6,8]. Упорядочить элементы строк матрицы по невозрастанию, а сами строки по возрастанию произведения элементов строк (использовать сортировку «плавающий пузырек»).

7. Дана матрица ***X***[9,10]. Упорядочить элементы строк матрицы по возрастанию, а сами строки по возрастанию произведения четных элементов строк (использовать сортировку «пузырек с флажком»).

8. Дана матрица ***X***[8,7]. Упорядочить элементы строк матрицы по убыванию, а сами строки по убыванию модуля произведения нечетных элементов строк (использовать сортировку простыми вставками).

9. Дана матрица ***X***[9,7]. Упорядочить элементы строк матрицы по неубыванию, а сами строки по возрастанию произведения модулей четных элементов строк (использовать сортировку бинарными вставками).

10. Дана матрица ***X***[10,7]. Упорядочить элементы строк матрицы по убыванию, а сами строки в соответствии с ростом характеристик строк (использовать быструю сортировку) (характеристикой строки матрицы называется сумма ее положительных четных элементов).

11. Дана матрица ***X***[7,9]. Упорядочить элементы столбцов матрицы по убыванию, а сами столбцы по возрастанию элементов 1-й строки (использовать сортировку выбором).

12. Дана матрица ***X***[8,5]. Упорядочить элементы столбцов матрицы по невозрастанию, а сами столбцы по невозрастанию сумм элементов столбцов (использовать сортировку обменами).

13. Дана матрица ***X***[8,10]. Упорядочить элементы столбцов матрицы по неубыванию, а сами столбцы по возрастанию максимальных элементов столбцов (использовать сортировку простыми вставками).

14. Дана матрица ***X***[10,9]. Упорядочить элементы столбцов матрицы по возрастанию, а сами столбцы по невозрастанию элементов 10-й строки (использовать сортировку бинарными вставками).

15. Дана матрица ***X***[7,8]. Упорядочить элементы столбцов матрицы по убыванию, а сами столбцы по неубыванию минимальных элементов столбцов (использовать шейкерную сортировку).

16. Дана матрица ***X***[6,8]. Упорядочить элементы столбцов матрицы по невозрастанию, а сами столбцы по возрастанию произведения элементов столбцов (использовать сортировку «плавающий пузырек»).

17. Дана матрица ***X***[9,10]. Упорядочить элементы столбцов матрицы по возрастанию, а сами столбцы по возрастанию произведения четных элементов столбцов (использовать сортировку «пузырек с флажком»).

18. Дана матрица ***X***[8,7]. Упорядочить элементы столбцов матрицы по убыванию, а сами столбцы по убыванию модуля произведения нечетных элементов столбцов (использовать сортировку вставками).

19. Дана матрица ***X***[9,7]. Упорядочить элементы столбцов матрицы по неубыванию, а сами столбцы по возрастанию произведения модулей нечетных элементов столбцов (использовать сортировку бинарными вставками).

20. Дана матрица ***X***[10,7]. Упорядочить элементы столбцов матрицы по неубыванию, а сами столбцы в соответствии с ростом характеристик (использо-

вать быструю сортировку) (характеристикой столбца матрицы называется сумма модулей ее отрицательных нечетных элементов).

21.Дана матрица ***X***[7,10]. Упорядочить элементы столбцов матрицы по невозрастанию, а сами столбцы в соответствии с ростом модулей минимальных элементов столбцов (использовать сортировку бинарными вставками).

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. В массиве ***A***[100], содержащем целые положительные числа, найти сумму максимального количества чисел, при этом их произведение не должно превышать число 300.

2. Числа в массиве ***X***[10] расположены по возрастанию. Включить в эту последовательность число ***B***, не нарушая упорядоченности элементов.

3. Дан массив ***X*** из ста чисел. Сформировать массив, в котором сначала записаны положительные числа, а затем отрицательные (массив сформировать на месте исходного).

4. Дан массив ***X***[100]. Все положительные элементы массива расположить в порядке возрастания, отрицательные – в порядке убывания, нулевые оставить без изменения. Относительного расположения отрицательных и положительных элементов не менять.

5. Дан массив ***X***[100]. Отсортировать четные элементы массива по возрастанию.

6. Дан массив ***X***[100]. Отсортировать элементы массива, стоящие на нечетных позициях по неубыванию.

7. В массиве ***X***[100] выбрать без повторения все числа, встречающиеся более одного раза.

8. В целочисленном массиве ***B***[50] найти число, повторяющееся максимальное количество раз. Если таких чисел несколько, то вывести на печать их все.

9. Дан массив целых чисел ***X***[100]. Определить, сколько различных чисел входит в этот массив.

10.Получить упорядоченный по убыванию массив ***C***[50] путем слияния упорядоченных по убыванию ***A***[30] и ***B***[20]. (Массив ***C*** сформировать непосредственно из массивов ***A*** и ***B***)

11.В массиве ***B***[100] записаны целые числа от 1 до 25. Упорядочить элементы массива в порядке убывания частоты встречаемости.

12.В массиве ***B***[100] записаны любые целые числа. Упорядочить элементы массива в порядке возрастания частоты встречаемости. (Если в массиве есть несколько элементов одинаковой частоты встречаемости, то группы упорядочить по убыванию абсолютных значений элементов).

13.Расположить элементы строк матрицы ***X***[5,7] в порядке возрастания элементов.

14.Расположить элементы столбцов матрицы ***X***[4,9] в порядке неубывания абсолютных значений элементов.

15.Расположить элементы строк матрицы ***X***[5,7] в порядке возрастания элементов, если номера строк четные, и в порядке убывания элементов, если номера строк нечетные.

16.Расположить элементы побочной диагонали матрицы ***X***[7,7] по убыванию.

17.В матрице ***A***[10,10] расположить элементы строк, находящиеся выше главной диагонали, по возрастанию.

18.Дана матрица ***X***[5,5]. Упорядочить строки матрицы по возрастанию элементов первого столбца.

19.Дана матрица ***X***[11,11]. Упорядочить элементы матрицы по спирали, начиная с центра (элемент ***X***[6,6]).

20. Дан целочисленный массив ***X***[15,15]. Каждая строка массива упорядочена по неубыванию. Найти числа, встречающиеся во всех строках.

ЛИТЕРАТУРА

1. Вирт Н. Алгоритмы и структуры данных. – М.: Мир, 1989.
2. Кнут Д. Искусство программирования для ЭВМ:т.2. – М.: Мир, 1978.
3. Ахо А., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы. /Пер. с англ.: – М. : Изд. дом “Вильямс”, 2003.
4. ГОСТ 19.701-90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения.
5. Глухова Л.А., Фадеева Е.П., Фадеева Е.Е., Болтак С.В. Лабораторный практикум по курсу "Основы алгоритмизации и программирования" для студентов специальности Программное обеспечение информационных технологий. В 4 ч.: Ч.1 – Мн.: БГУИР, 2004.

Схема алгоритма сортировки простыми вставками

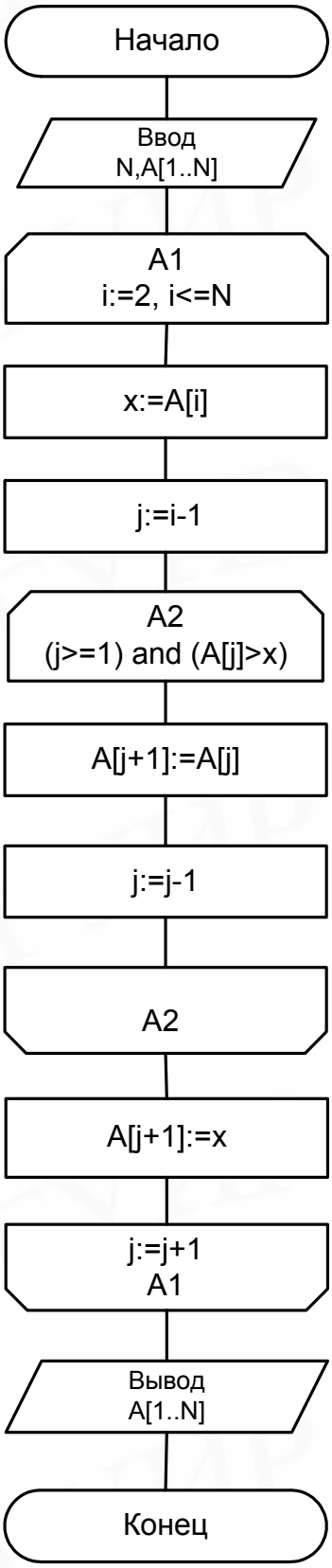


Схема алгоритма сортировки простыми вставками с барьерным элементом

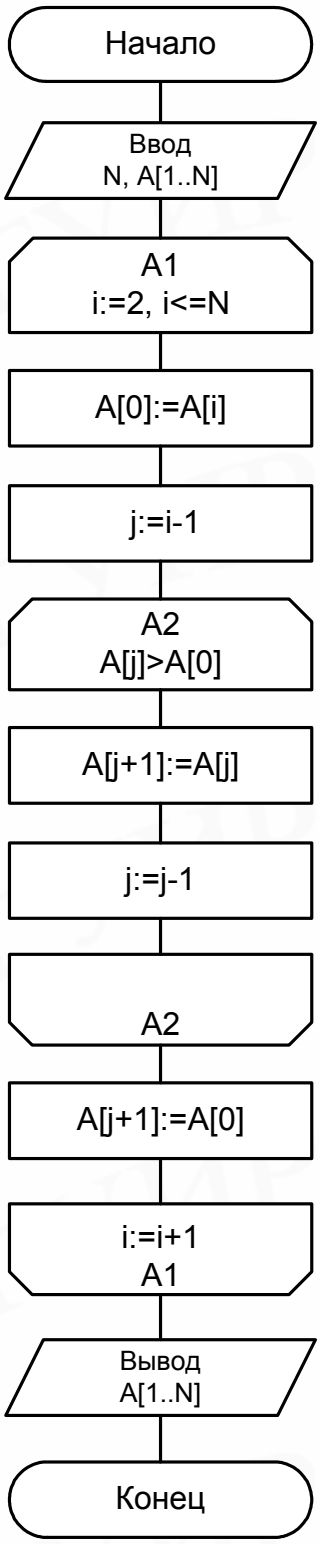


Схема алгоритма бинарных вставок

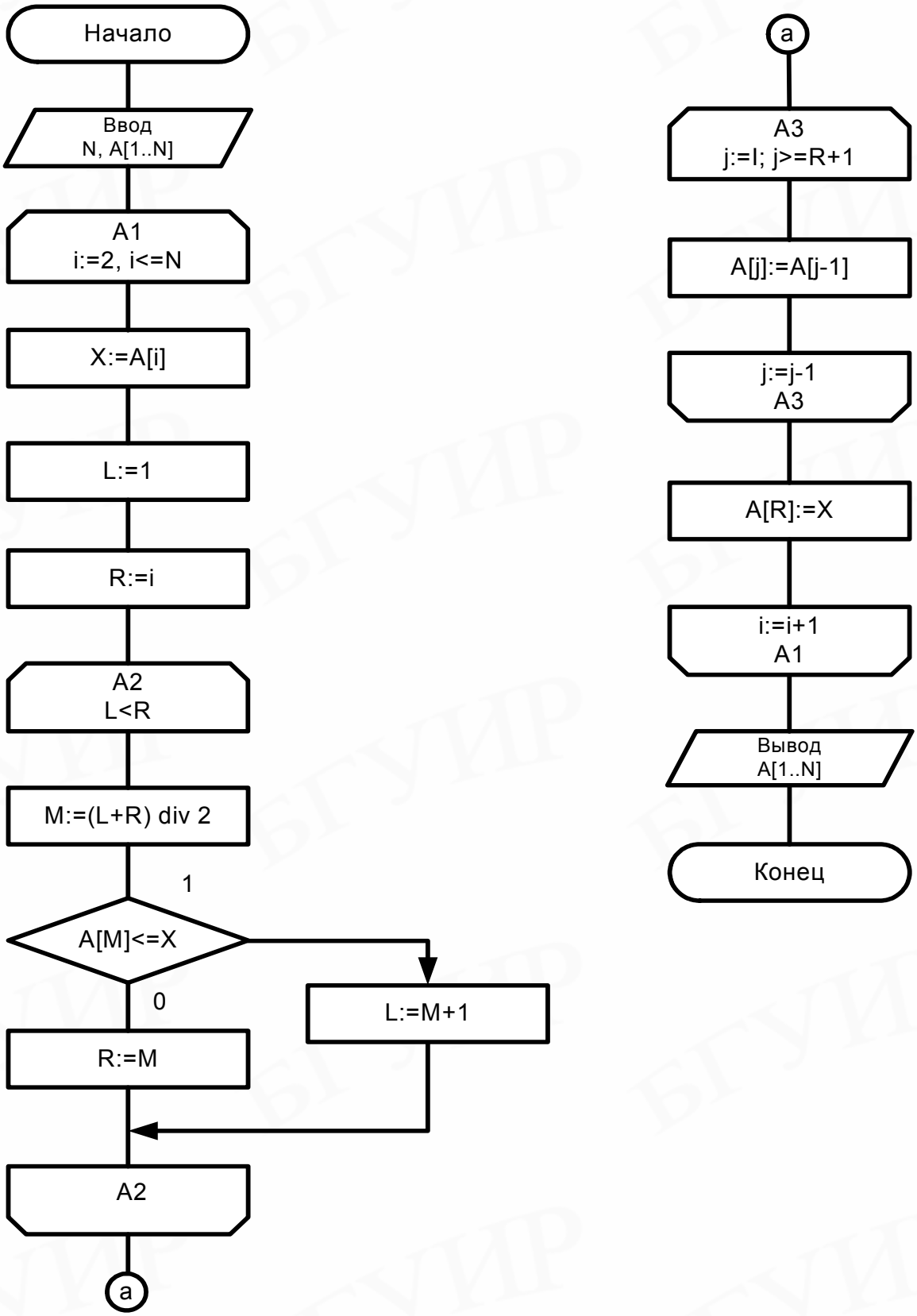


Схема алгоритма сортировки выбором

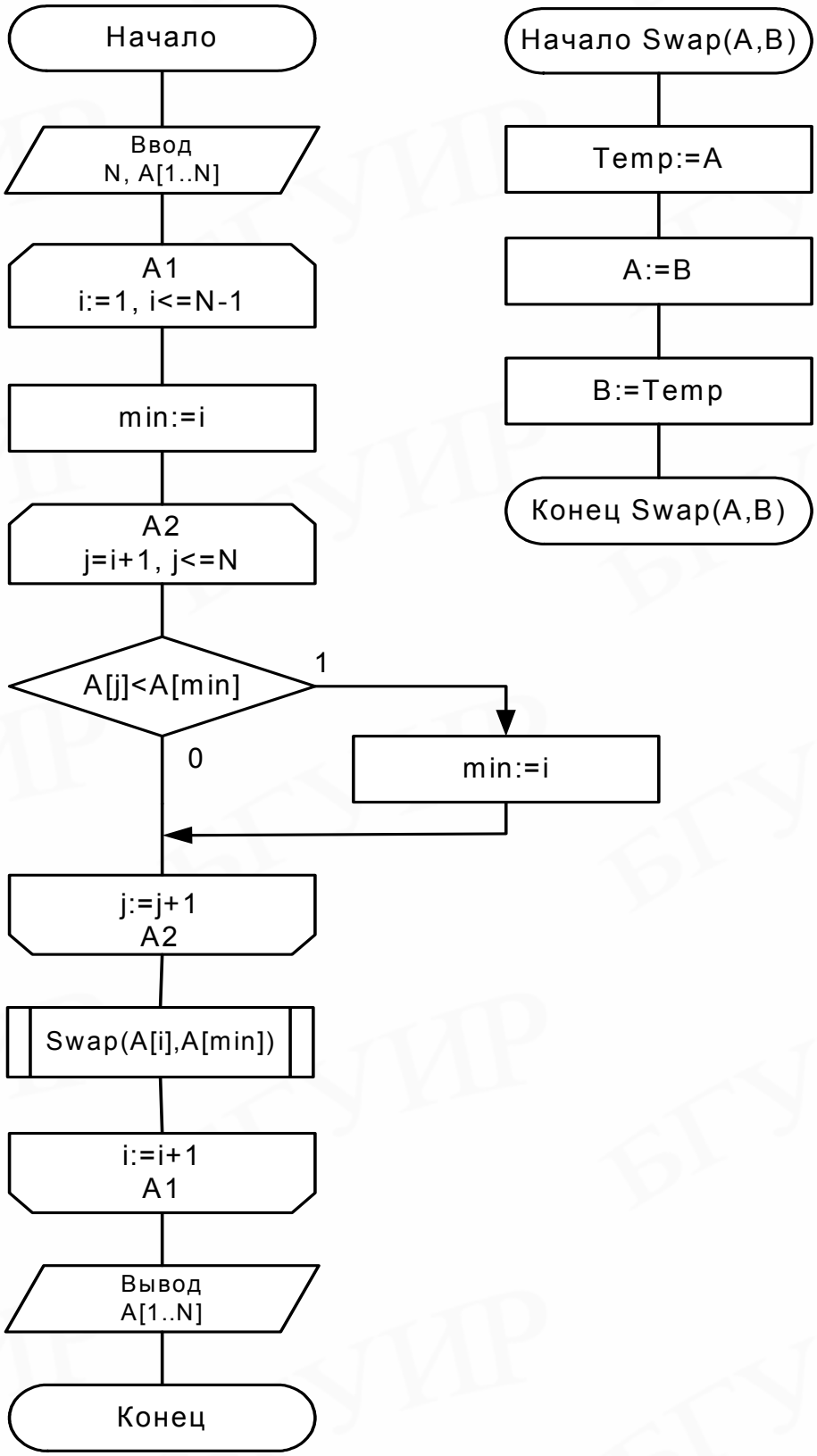
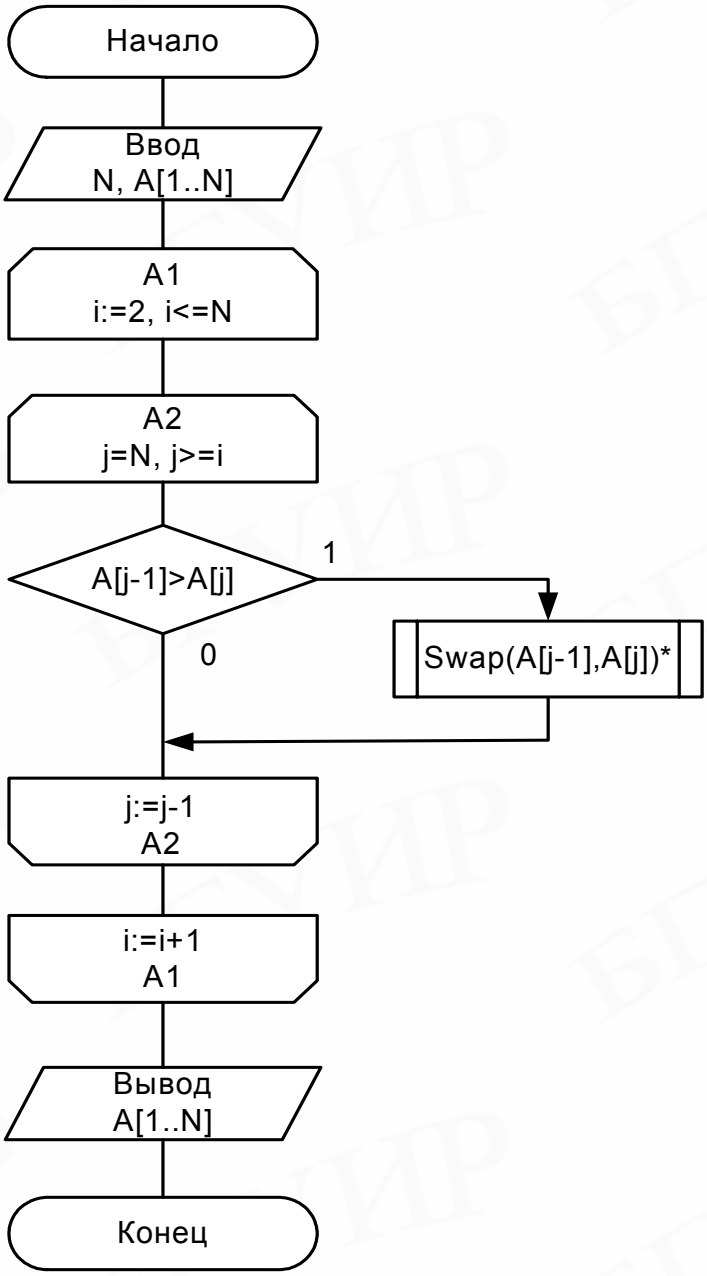


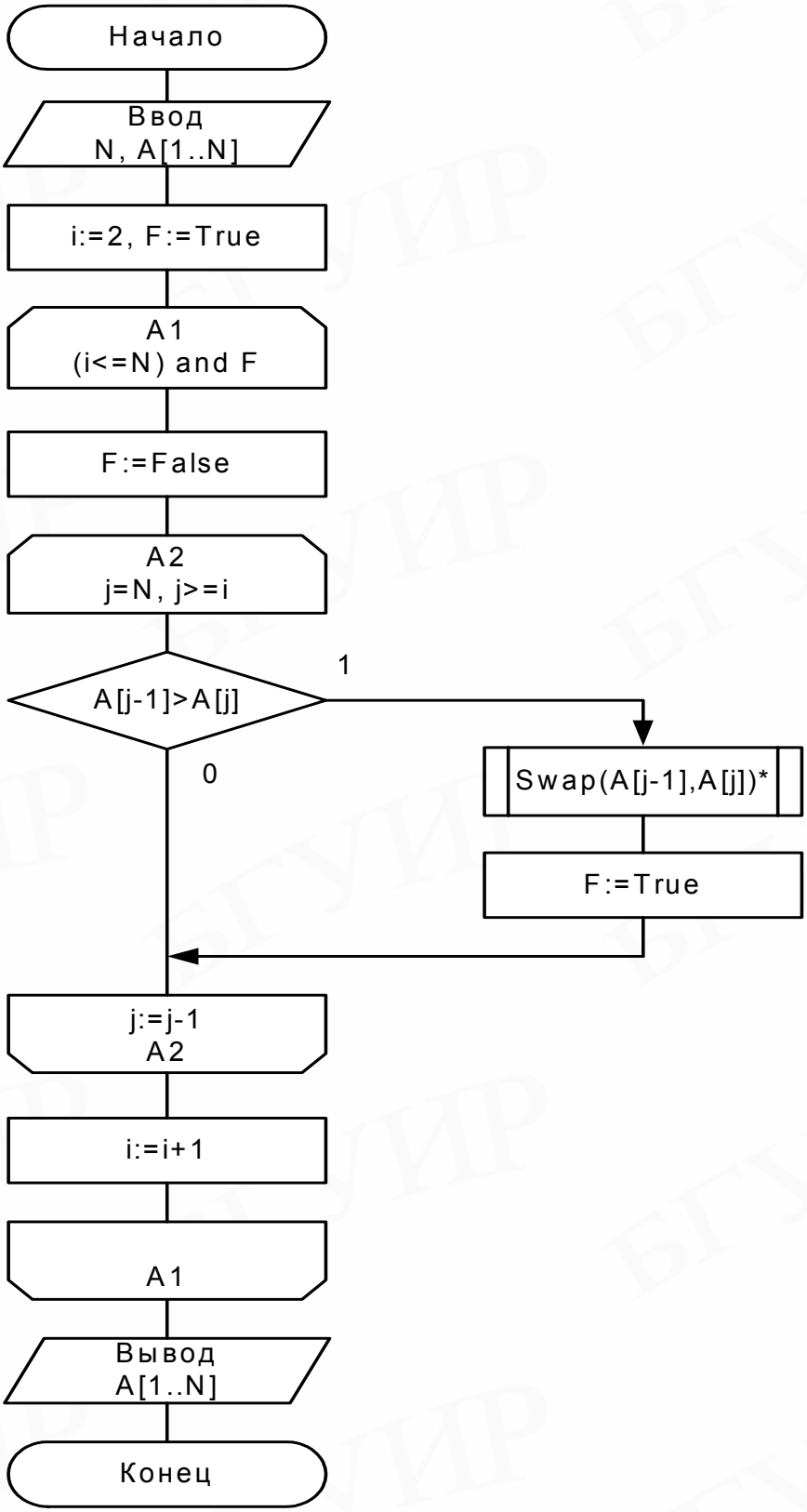
Схема алгоритма сортировки обменами (“пузырек”)



---

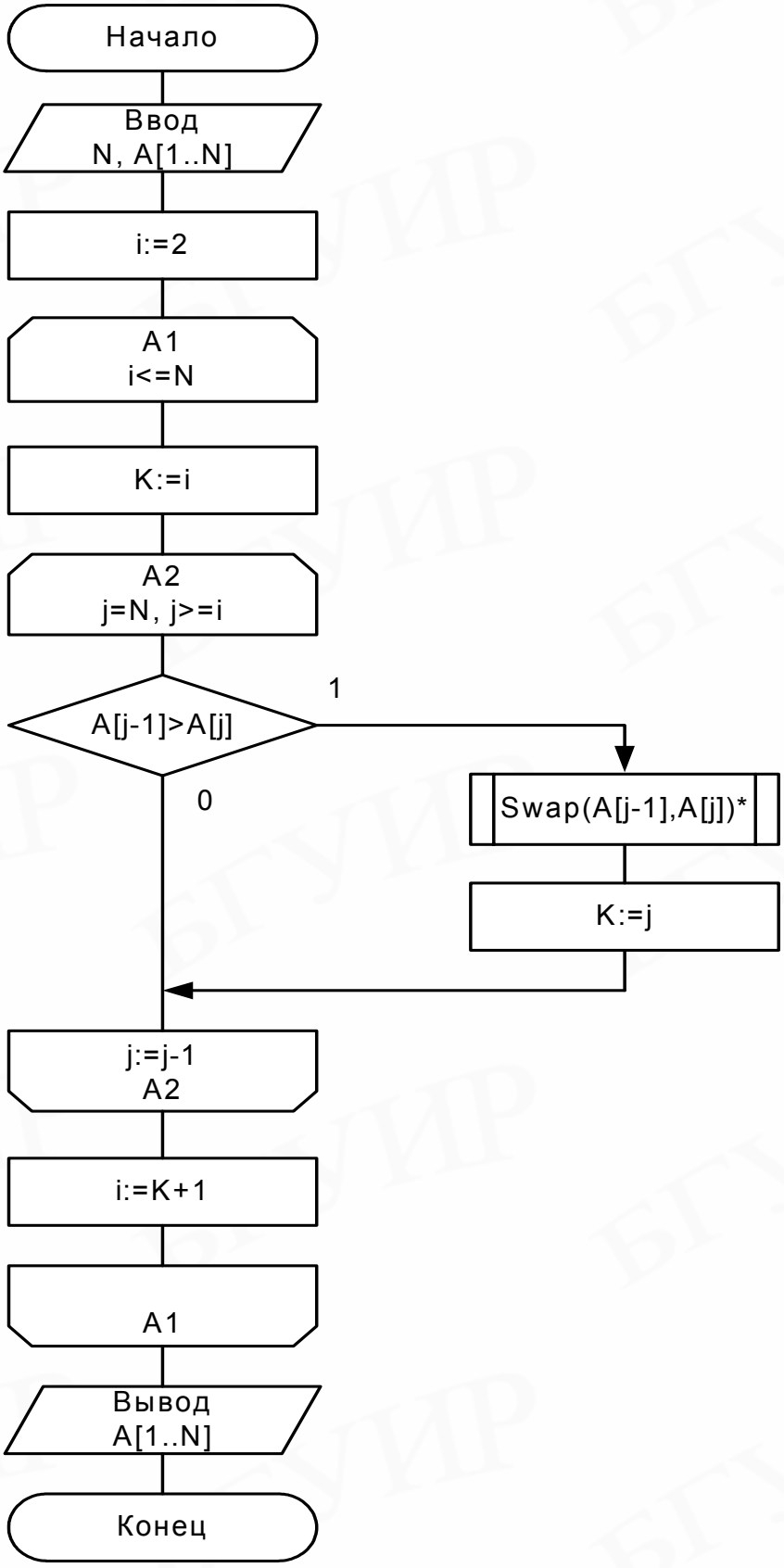
* Схему подпрограммы обмена элементов Swap см. в Приложении 4.

Схема алгоритма сортировки обменами с флажком



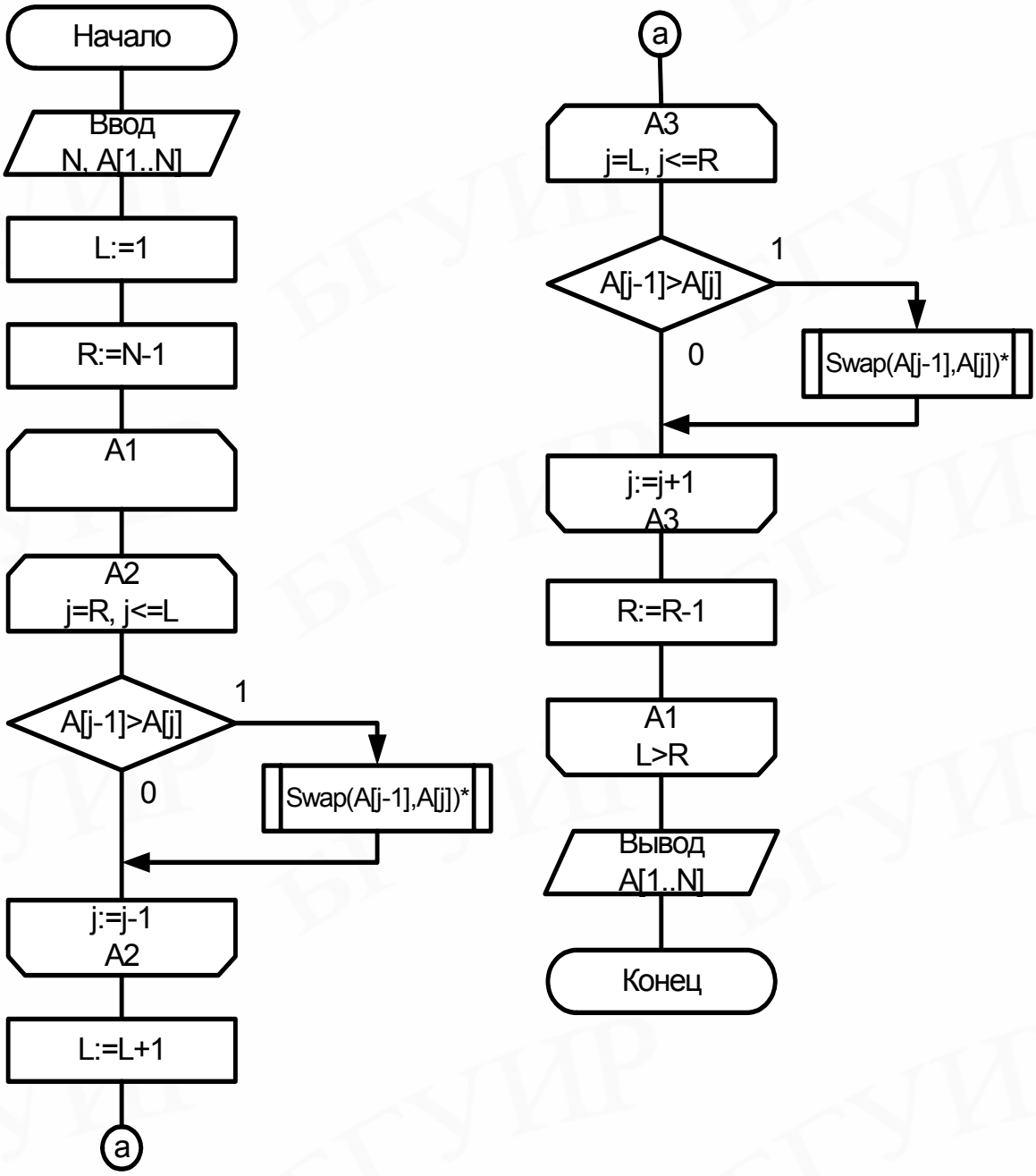
- Схему подпрограммы обмена элементов Swap см. в Приложении 4.

Схема алгоритма сортировки обменами с “быстрой границей”



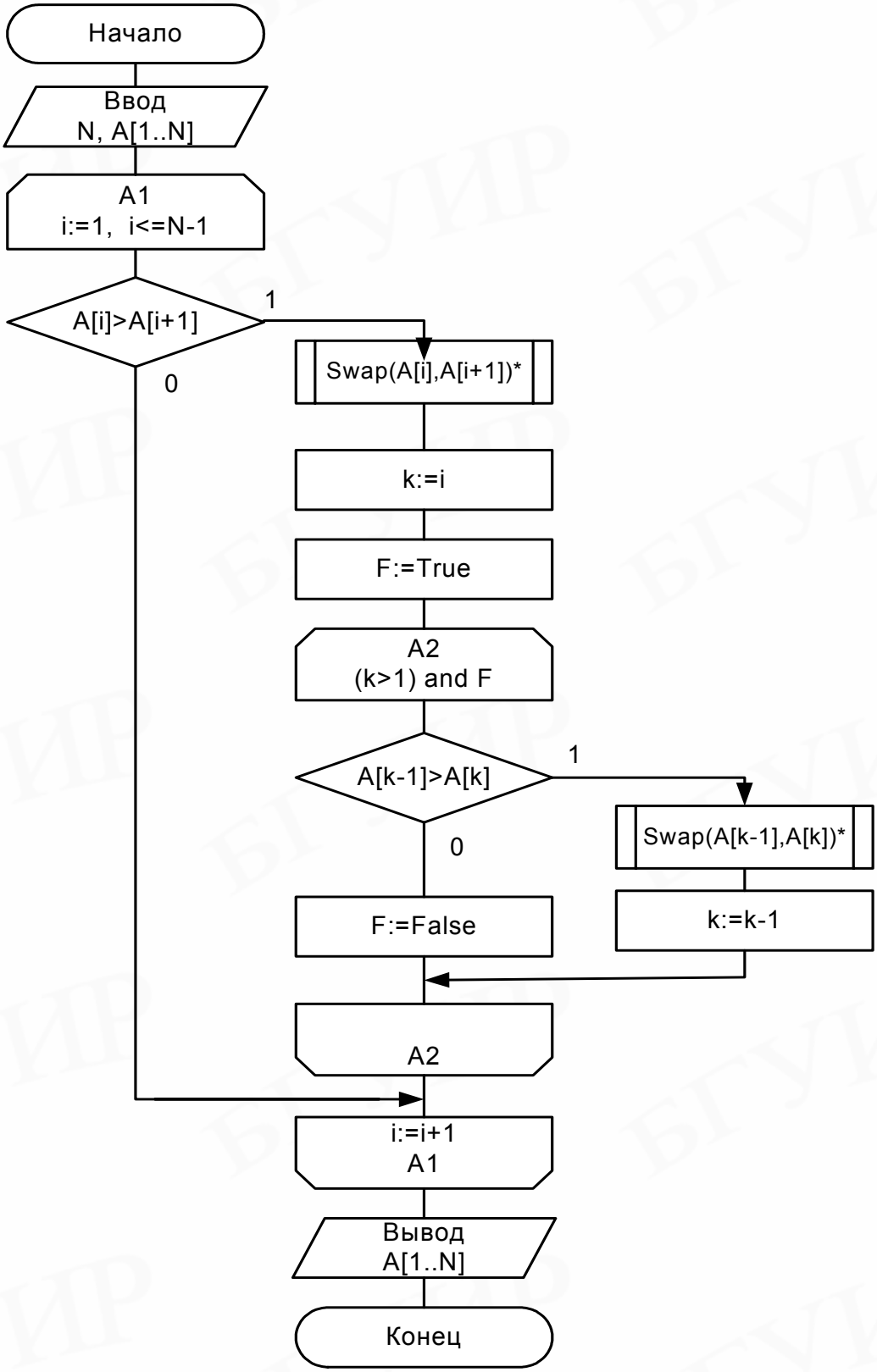
- Схему подпрограммы обмена элементов Swap см. в Приложении 4.

Схема шейкерной сортировки



- Схему подпрограммы обмена элементов Swap см. в Приложении 4.

Схема алгоритма сортировки “плавающий пузырек”



- Схему подпрограммы обмена элементов Swap см. в Приложении 4.

Схема алгоритма сортировки Шелла

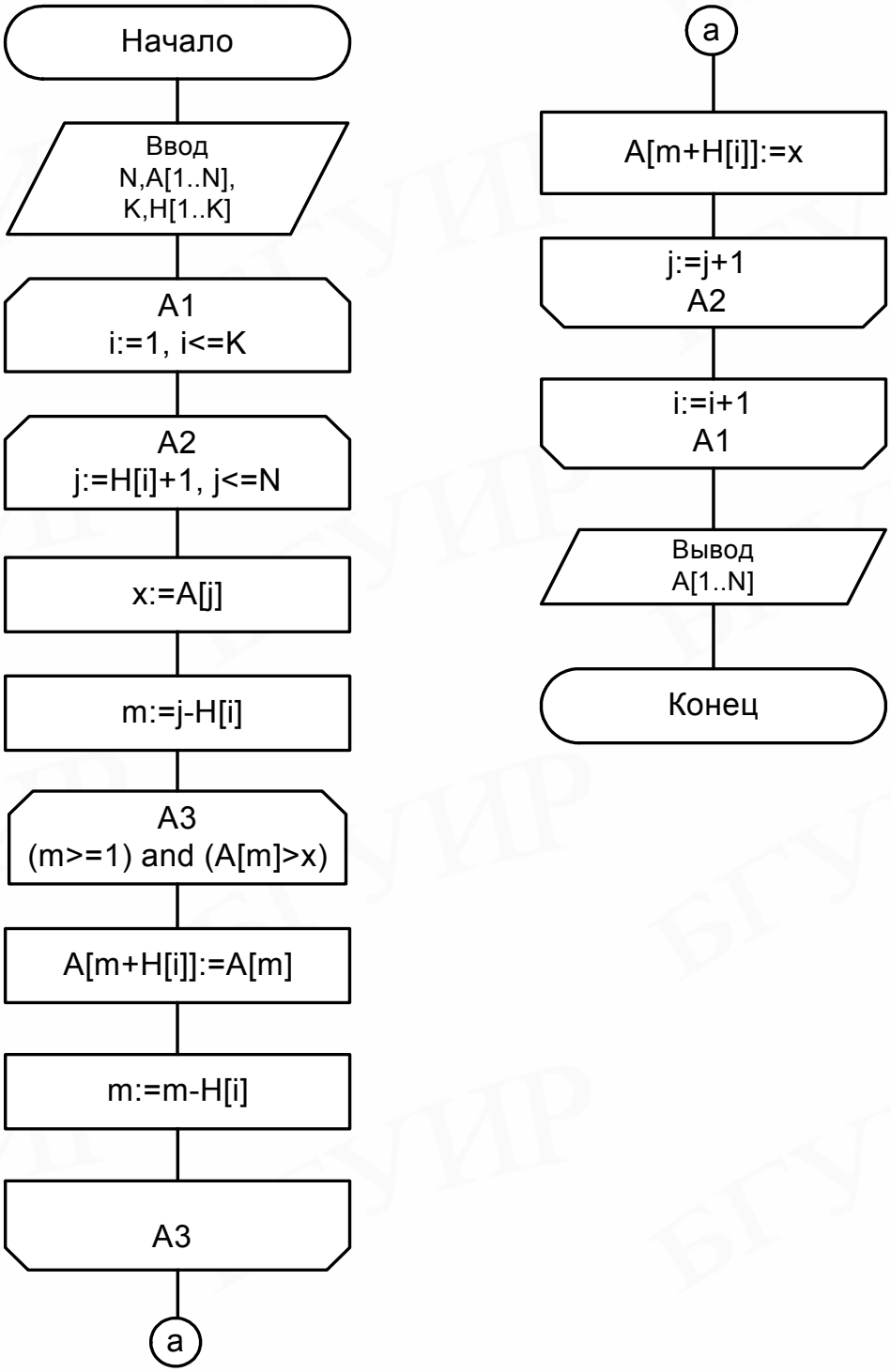
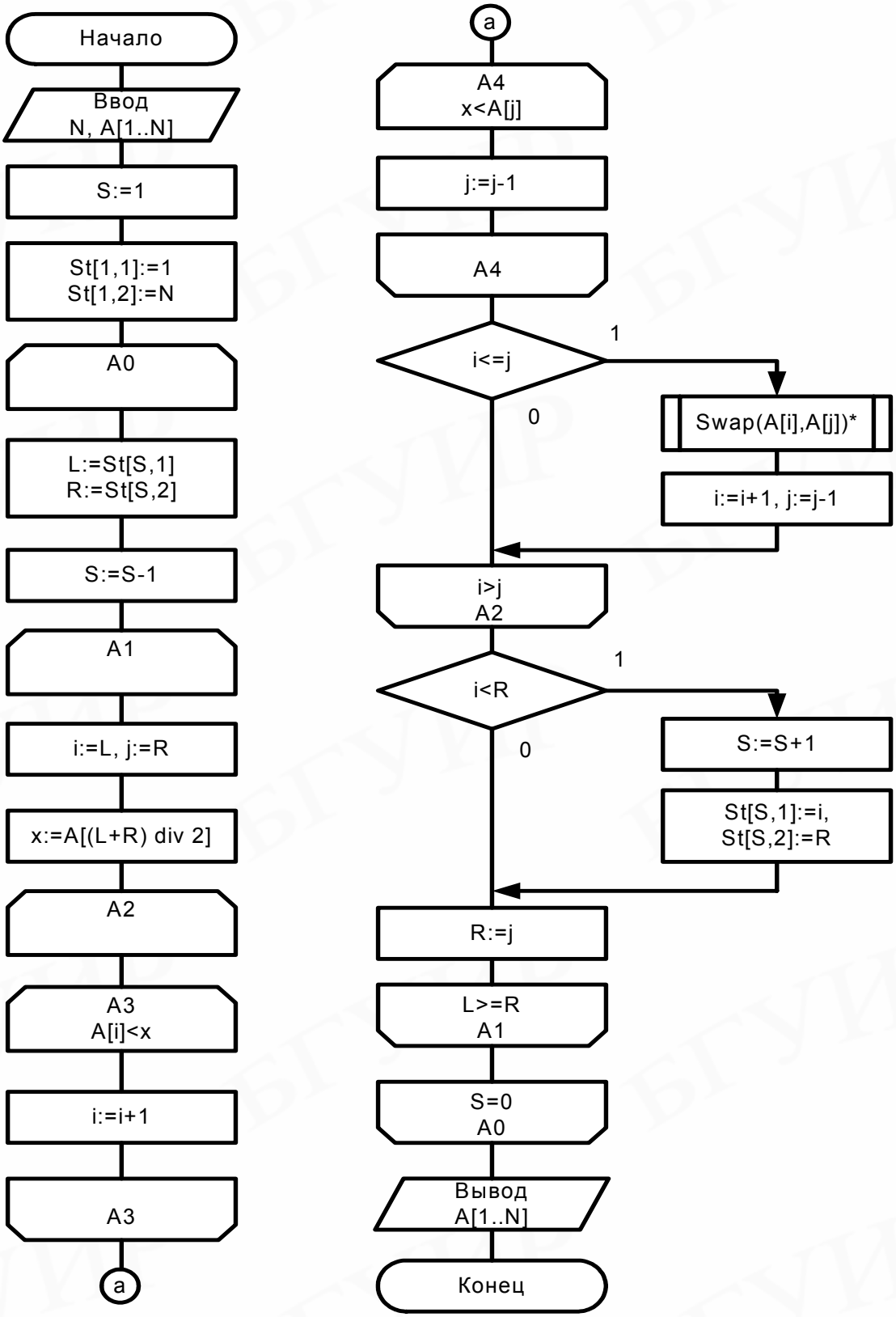
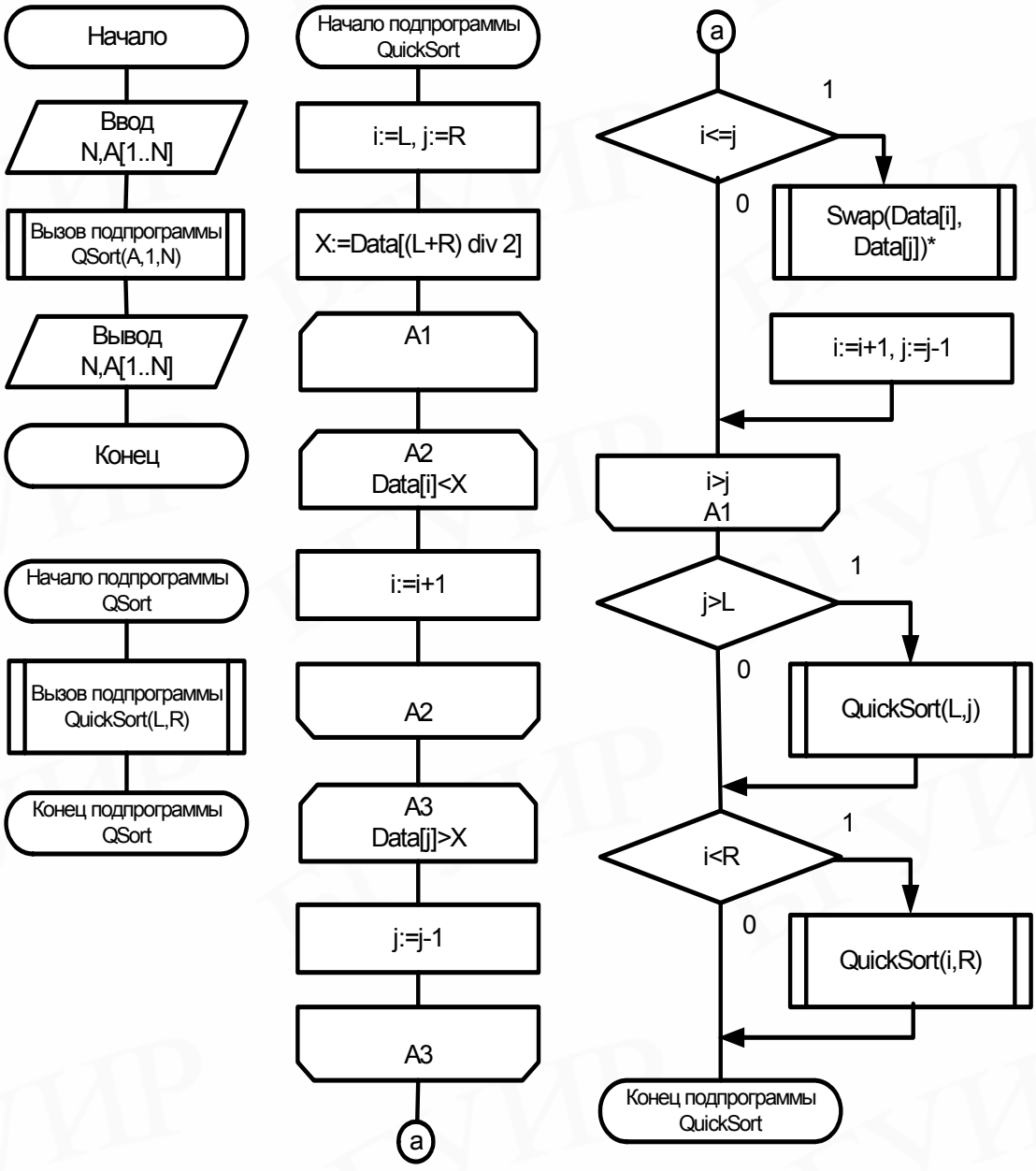


Схема алгоритма быстрой сортировки (нерекурсивная реализация)



* Схему подпрограммы обмена элементов Swap см. в Приложении 4.

Схема алгоритма быстрой сортировки
(рекурсивная реализация)



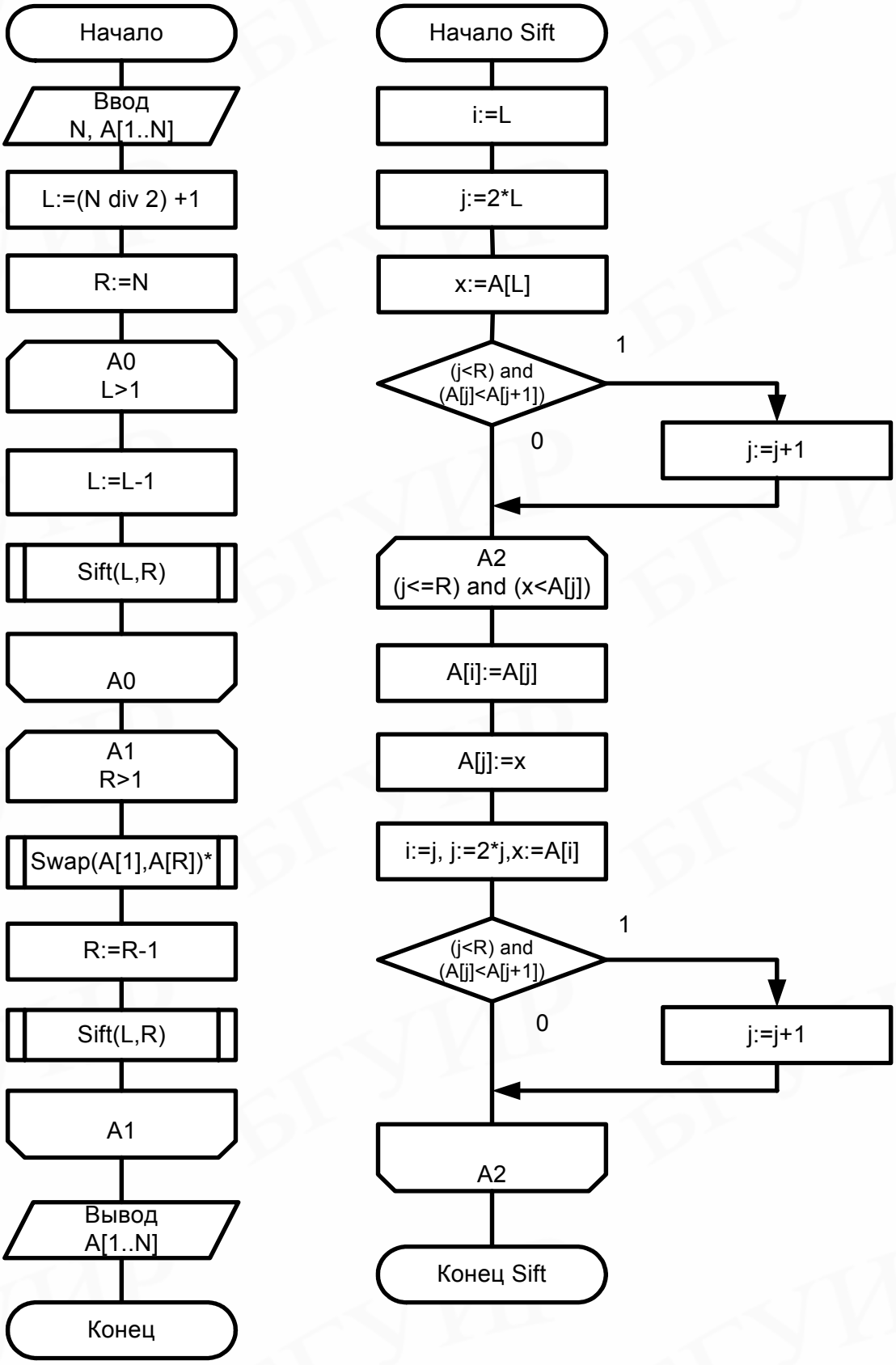
---

* Схему подпрограммы обмена элементов Swap см. в Приложении 4.

Программа быстрой сортировки (рекурсивная реализация)

```
Program Demo;
Uses
  Crt;
Const
   N=1000;
Type
   List=Array[1..N] of Integer;
Var
   Mas:Tlist;
   I:Integer;
Procedure Qsort(Var Data:TList; L,R: Integer);
Procedure QuickSort(L,R: Integer);     {рекурсивная  подпрограмма-процедура}
   I,J,X,Y:Integer;
begin
  I:=L;
  J:=R;
  X:=Data[(L+R) div 2];
  Repeat
     While Data[I]<X do Inc (I);
     While Data[J]>X do Dec (J);
     If I<=J then
     begin
        Y:=Data[I];
        Data[I]:=Data[J];
        Data[J]:=Y;
        Inc (I);
        Dec (J);
    end
  until I>J;
  If J>L then QuickSort(L,J);
  If I<R then QuickSort(I,R);
end;
begin
   Sort (L,R);
end;
BEGIN
    For I:=1 to N do Mas[I]:=Random(N);          {Заполнение массива данными}
    For I:=1 to N do Write(Mas[I],' ');              {Вывод исходного массива}
    Writeln;
    QSort(Mas,1,N);              {Вызов рекурсивной подпрограммы-процедуры}
    For I:=1 to N do Write (Mas[I],' ');         {Вывод результирующего  массива}
END.
```

Схема алгоритма пирамидальной сортировки



* Схему подпрограммы обмена элементов Swap см. в Приложении 4.

Св. план 2005, поз. 50

Учебное издание

Глухова Лилия Александровна,
Фадеева Елена Павловна,
Фадеева Елена Евгеньевна

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

ЛАБОРАТОРНЫЙ ПРАКТИКУМ
для студентов специальности I-40 01 01
«Программное обеспечение информационных технологий»
дневной формы обучения

В 4-х частях

Часть 2

Редактор Т.П. Андрейченко
Корректор Н.В. Гриневич

Подписано в печать 07.06.2005	Формат 60x84 1/16.	Бумага офсетная.
Гарнитура «Таймс».	Печать ризографическая.	Усл. печ. л.3,26.
Уч.-изд. л. 2,0.	Тираж 100 экз.	Заказ 70.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
Лицензия на осуществление издательской деятельности №02330/0056964 от 01.04.2004.
Лицензия на осуществление полиграфической деятельности №02330/0131518 от 30.04.2004.
220013, Минск, П. Бровки, 6