МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Брянский государственный технический университет

Утверждаю
Ректор университета

_____О.Н. Федонин

«____»_____2020 г.

СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ

МЕТОДЫ ВНУТРЕННЕЙ СОРТИРОВКИ

Методические указания
к выполнению лабораторной работы №10
для студентов очной, очно-заочной и заочной форм обучения
по направлениям подготовки
230100 «Информатика и вычислительная техника»,
010500 «Математическое обеспечение и администрирование информационных систем»,
231000 «Программная инженерия»

Брянск 2020

УДК 006.91

Структуры и алгоритмы обработки данных. Методы внутренней сортировки [Текст] + [Электронный ресурс]: методические указания к выполнению лабораторной работы №10 для студентов очной, очно-заочной и заочной форм обучения по направлениям подготовки 230100 «Информатика и вычислительная техника», 010500 «Математическое обеспечение и администрирование информационных систем», 231000 «Программная инженерия». – Брянск: БГТУ, 2020. – 31 с.

Разработал:
канд. техн. наук, проф.
В.К. Гулаков
канд. техн. наук, доц.
А.О. Трубаков
канд. техн. наук, доц.
Е.О. Трубаков

Рекомендовано кафедрой «Информатика и программное обеспечение» БГТУ (протокол №1 от 13.09.13)

1. ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ

Целью лабораторной работы является приобретение навыков сортировки одномерных массивов с помощью алгоритмов простых сортировок.

2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Сортировка – процесс перегруппировки заданного множества объектов в определенном порядке. Часто при сортировке больших объемов данных нецелесообразно переставлять сами элементы, поэтому при решении задачи выполняют упорядочивание элементов по индексам. Индексы элементов выстраивают в такой последовательности, что соответствующие им значения элементов оказываются отсортированными по условию задачи.

Сортировка применяется для поиска элементов в упорядоченном множестве. Задача сортировки одна из основных в программировании.

Сортировка – упорядочивание набора однотипных данных по возрастанию или убыванию.

Чаще всего при сортировке данных только часть их используется в качестве ключа сортировки. ***Ключ сортировки*** – часть данных, определяющая порядок элементов. Таким образом, ключ участвует в сравнениях, но при обмене элементов происходит перемещение всей структуры данных. Так, в списке почтовой рассылки в качестве ключа может использоваться почтовый индекс, но сортируется весь адрес. При решении задач сортировок массивов ключ и данные совпадают.

Чтобы отсортировать данные, можно вызывать стандартную функцию *qsort*(), входящую в библиотеку C++. Однако различные методы сортировки обладают разными характеристиками. Несмотря на то, что некоторые методы сортировки могут быть в среднем лучше, чем другие, ни один из них не является идеальным для всех случаев.

Использование функции *qsort*() не является универсальным решением для всех задач сортировки. Во-первых, функцию общего назначения, такую как *qsort*(), невозможно применить во всех

случаях. Так, эта функция сортирует только массивы данных в памяти и не может сортировать данные, хранящиеся в связанных списках. Во-вторых, *qsort*() – параметризованная функция, благодаря чему она может обрабатывать широкий набор типов данных, но вследствие этого она работает более медленно, чем эквивалентная функция, рассчитанная на какой-то один тип данных. В-третьих, алгоритм быстрой сортировки, примененный в функции *qsort*(), может оказаться не самым эффективным алгоритмом сортировки в некоторых конкретных ситуациях.

Оценка алгоритмов сортировки

Существует множество различных алгоритмов сортировки. Все они имеют свои положительные и отрицательные стороны. Перечислим общие критерии оценки алгоритмов сортировки.

- *Скорость работы алгоритма сортировки*. Она непосредственно связана с числом сравнений и числом обменов, происходящих во время сортировки, причем обмены занимают больше времени. Сравнение происходит тогда, когда один элемент массива сравнивается с другим; обмен происходит тогда, когда два элемента меняются местами. Время работы одних алгоритмов сортировки увеличивается экспоненциально, а время работы других логарифмически зависит от числа элементов.
- *Время работы в лучшем и худшем случаях*. Оно имеет значение при анализе выполнения алгоритма, если одна из краевых ситуаций будет встречаться достаточно часто. Алгоритм сортировки зачастую имеет хорошее среднее время выполнения, но в худшем случае он работает очень медленно.
- *Поведение алгоритма сортировки*. Поведение алгоритма сортировки называется естественным, если время сортировки минимальное для упорядоченного списка элементов, увеличивается при увеличении степени неупорядоченности списка и максимально, когда элементы списка расположены в обратном порядке. Объем работы алгоритма оценивается числом производимых сравнений и обменов.

Различные методы сортировки массивов отличаются по быстродействию. Существуют простые методы сортировок, которые требуют порядка $n \cdot n$ сравнений, где n – число элементов массива и

быстрые сортировки, которые требуют порядка $n \cdot ln(n)$ сравнений. Простые методы сортировки удобны для объяснения принципов сортировок, так как имеют простые и короткие алгоритмы. Усложненные методы требуют меньшего числа операций, но сами операции более сложные, поэтому для небольших массивов простые методы более эффективны.

Простые методы сортировки можно разделить на три основные группы:

- сортировка методом «пузырька» (простого обмена);
- сортировка методом простого выбора (простой перебор);
- сортировка методом простого включения (сдвиг-вставка, вставками, вставка и сдвиг).

Сортировка методом «пузырька» (простого обмена)

Самый известный метод сортировки (*bubble sort*, сортировка методом «пузырька», или сортировка «пузырьком»). Его популярность объясняется интересным названием и простотой алгоритма сортировки.

Метод попарного сравнения элементов массива в литературе часто называют методом «пузырька», проводя аналогию с пузырьком, поднимающимся со дна бокала с газированной водой. При всплывании пузырек сталкивается с другими пузырьками и, сливаясь с ними, увеличивается в объеме. Чтобы аналогия стала очевидной, необходимо считать, что элементы массива расположены вертикально друг над другом, и их необходимо так упорядочить, чтобы они увеличивались сверху вниз.

Алгоритм сортировки методом «пузырька» состоит в повторяющихся проходах по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются до тех пор, пока на очередном проходе не окажется, что обмены больше не требуются, что означает – массив отсортирован. При проходе последовательности элемент, стоящий не на своём месте, «всплывает» до необходимой позиции (1).

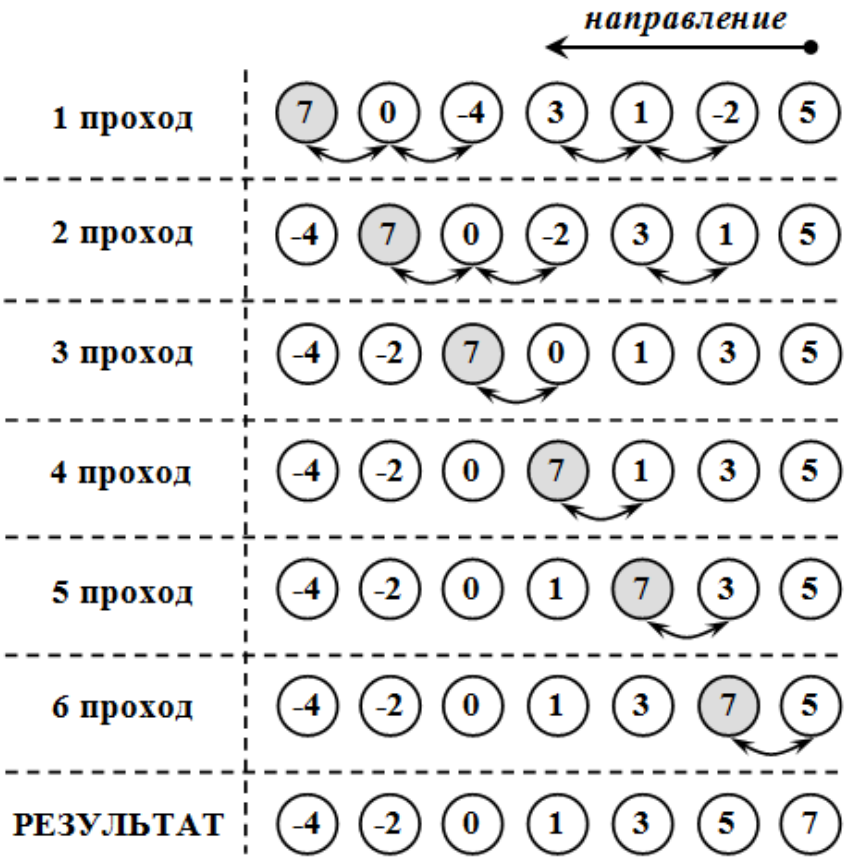


Рис. 1. Сортировка по неубыванию методом «пузырька»

Ниже показа програмная реализации алгоритма сортировки методом «пузырька».

```
//Описание функции сортировки методом "пузырька"
void BubbleSort (int k,int x[max]) {
  int i,j,buf;
  for (i=k-1;i>0;i--)
    for (j=0;j<i;j++)
    if (x[j]>x[j+1]) {
      buf=x[j];
      x[j]=x[j+1];
      x[j+1]=buf;
     }
}
```

В сортировке методом «пузырька»число сравнений всегда одно и то же, так как два цикла *for* повторяются указанное число раз независимо от того, был список изначально упорядочен или нет. Это значит, что в сортировке методом «пузырька» всегда выполняется

$\frac{n^2-n}{2}$ сравнений, где n – число сортируемых элементов. Формула выведена на том основании, что внешний цикл выполняется n–1 раз, а внутренний выполняется в среднем $n/2$ раз.

Сортировка методом «пузырька» имеет такую особенность: неупорядоченные элементы на «большом» конце массива занимают правильные положения за один проход, но неупорядоченные элементы в начале массива поднимаются на свои места очень медленно. Поэтому, вместо того чтобы постоянно просматривать массив в одном направлении, в последовательных проходах можно чередовать направления. Таким образом, элементы, значительно удаленные от своих положений, быстро станут на свои места. Данная версия сортировки пузырьком называется ***шейкер-сортировкой*** (*shaker sort* сортировка перемешиванием, сортировка взбалтыванием, сортировка встряхиванием), так как действия, производимые ею с массивом, напоминают взбалтывание или встряхивание. Ниже показана программная реализация алгоритма метода «шейкер-сортировки».

```
//Описание функции шейкер-сортировки
void Shaker(int k,int x[max]){
  int i,t;
  bool exchange;
  do {
    exchange = false;
    for(i=k-1; i > 0; --i) {
      if(x[i-1] > x[i]) {
        t = x[i-1];
        x[i-1] = x[i];
        x[i] = t;
        exchange = true;
      }
    }
    for(i=1; i < k; ++i) {
      if(x[i-1] > x[i]) {
        t = x[i-1];
        x[i-1] = x[i];
        x[i] = t;
        exchange = true;
      }
    }
```

```
  } while(exchange);
    //сортировать до тех пор, пока не будет обменов
}
```

Хотя метод «шейкер-сортировки» и является улучшенным методом по сравнению с сортировкой методом «пузырька», она по-прежнему имеет время выполнения порядка N^2. Это объясняется тем, что число сравнений не изменилось, а число обменов уменьшилось на относительно небольшую величину.

Сортировка методом простого выбора (простой перебор)

Это наиболее естественный метод сортировки. При сортировке методом простого выбора из массива выбирается элемент с наименьшим значением и обменивается с первым элементом. Затем из оставшихся n–1 элементов снова выбирается элемент с наименьшим ключом и обменивается со вторым элементом, и т.д. (рис. 2)

Алгоритм сортировки методом простого выбора:

1. Находим минимальное значение в текущей части массива.
2. Производим обмен этого значения со значением на первой неотсортированной позиции.
3. Сортируем хвост массива, исключив из рассмотрения уже отсортированные элементы.

Ниже приведена программная реализация алгоритма метода простого выбора.

```
//Описание функции сортировки методом простого выбора
void SelectionSort (int k,int x[max]) {
  int i,j,min,temp;
  for (i=0;i<k-1;i++) {
  //устанавливаем начальное значение минимального индекса
    min=i;
    //находим минимальный индекс элемента
    for (j=i+1;j<k;j++){
      if (x[j]<x[min])
        min=j;
      //меняем значения местами
      }
      temp=x[i];
      x[i]=x[min];
      x[min]=temp;
  }
```

```
}
```

Рис. 2. Сортировка по неубыванию методом простого выбора

Как и в сортировке пузырьком, внешний цикл выполняется n–1 раз, а внутренний – в среднем $n/2$ раз. Следовательно, сортировка простым выбором требует $\frac{n^2 - n}{2}$ сравнений. Таким образом, время выполнения сортировки простым выбором имеет сложность порядка n^2, из-за чего она считается очень медленной для сортировки большого числа элементов. Несмотря на то, что число сравнений в сортировке пузырьком и сортировке простым выбором одинаковое, в последней число обменов в среднем намного меньше.

Сортировка методом простого включения (сдвиг-вставка, вставками, вставка и сдвиг)

Хотя этот метод сортировки менее эффективен, чем сложные методы (такие как быстрая сортировка), у него есть ряд преимуществ:

- прост в реализации;

- эффективен на небольших наборах данных, на наборах данных до десятков элементов может оказаться лучшим;
- эффективен на наборах данных, которые частично отсортированы;
- это устойчивый метод сортировки (не меняет порядок элементов, которые отсортированы);
- может сортировать массив при его получении;
- не требует временной памяти, даже под стек.

На каждом шаге алгоритма метода простого включения выбираем один из элементов входных данных и вставляем его на требуемую позицию в отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. Выбор очередного элемента из исходного массива произволен; может использоваться практически любой метод выбора (рис. 3).

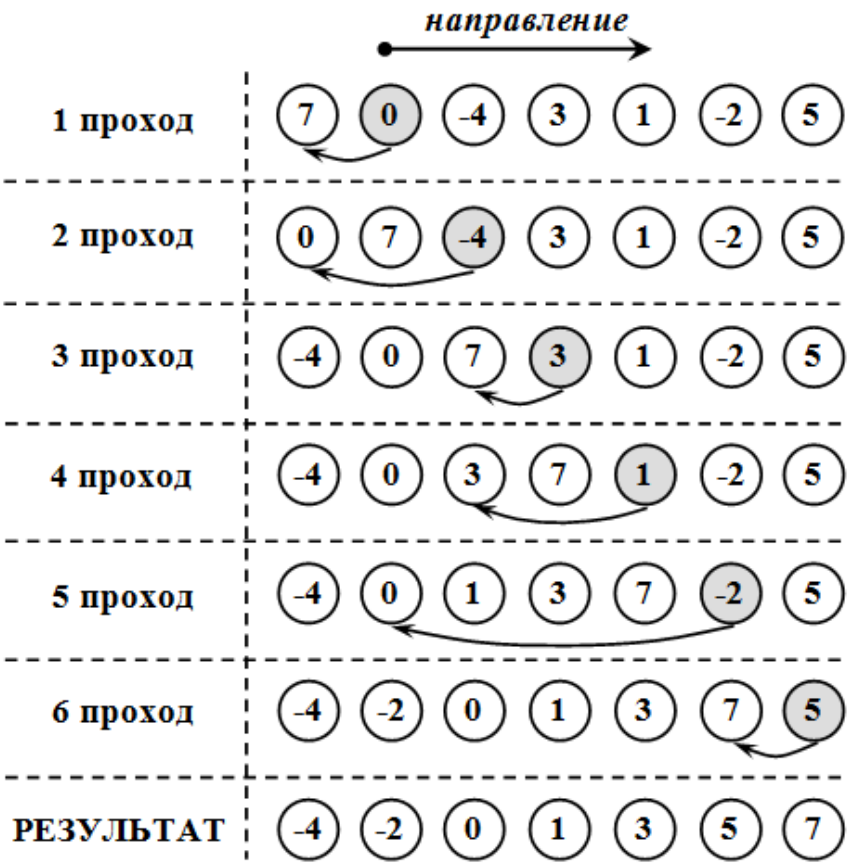


Рис. 3. Сортировка по неубыванию методом простого включения

Ниже приведена программная реализация алгоритма сортировки методом простого включения

```
//Описание функции сортировки методом простого включения
void InsertSort (int k,int x[max]) {
  int i,j, temp;
  for (i=0;i<k;i++) {
  //цикл проходов, i - номер прохода
    temp=x[i];
    //поиск места элемента
    for (j=i-1; j>=0 && x[j]>temp; j--)
      x[j+1]=x[j];//сдвигаем элемент вправо, пока не
дошли
      //место найдено, вставить элемент
      x[j+1]=temp;
  }
}
```

В отличие от сортировки методом пузырька и сортировки методом простого выбора число сравнений в сортировке методом простого включения зависит от изначальной упорядоченности списка. Если список отсортирован, число сравнений равно n–1; если не отсортирован, его производительность является величиной порядка n^2.

Основные понятия

Ключ сортировки – часть данных, определяющая порядок элементов.

Сортировка – упорядочивание набора однотипных данных по возрастанию или убыванию.

Сортировка методом «пузырька» – метод попарного сравнения элементов одномерного массива.

Сортировка методом простого включения – метод последовательного помещения элемента массива в отсортированную часть в соответствии с ключом сортировки.

Сортировка методом простого выбора – метод последовательного обмена минимального и первого элементов неотсортированной части массива.

Резюме

1. Задачи сортировок массивов имеют широкое прикладное значение.

2. Существует большое число методов сортировок массивов, различающихся трудоемкостью.

3. При оценке трудоемкости методов сортировки учитываются критерии: число сравнений и перестановок, время в лучшем и худшем случаях, естественность поведения.

4. К простым методам сортировок относятся: сортировка методом «пузырька», сортировка методом простого выбора, сортировка методом простого включения.

5. Простые методы сортировки эффективны на небольших объемах данных.

3. ТЕХНИКА БЕЗОПАСНОСТИ ПРИ ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

Меры безопасности при работе с электротехническими устройствами соответствуют мерам безопасности, принимаемым при эксплуатации установок с напряжением до 1000 В и разработанным в соответствии с «Правилами техники безопасности при эксплуатации электроустановок потребителей», утвержденными Главгосэнергонадзором 21 декабря 1984 г.

Студенту не разрешается приступать к выполнению лабораторной работы, если замечены какие-либо неисправности в лабораторном оборудовании.

Студент не должен прикасаться к токоведущим элементам электрооборудования, освещения и электропроводке, открывать двери электрошкафов и корпусов системных блоков, мониторов.

Студенту запрещается прикасаться к неизолированным или поврежденным проводам и электрическим устройствам, наступать на переносные электрические провода, лежащие на полу, самостоятельно ремонтировать электрооборудование и инструмент.

Обо всех замеченных неисправностях электрооборудования студент должен немедленно поставить в известность преподавателя или лаборанта.

При выполнении лабораторной работы необходимо соблюдать осторожность и помнить, что только человек, относящийся серьезно к своей безопасности, может быть застрахован от несчастного случая.

4. УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ

Каждую задачу решить в соответствии с изученными методами формирования, вывода и обработки данных очередей и стеков в языке C++. Обработку очередей или стеков выполнить на основе базовых алгоритмов: поиск, вставка элемента, удаление элемента, удаление всей динамической структуры. При объявлении списков выполнить комментирование используемых полей.

Каждое задание реализовать в соответствии с приведенными этапами:

- изучить словесную постановку задачи, выделив при этом все виды данных;
- сформулировать математическую постановку задачи;
- выбрать метод решения задачи, если это необходимо;
- разработать графическую схему алгоритма;
- записать разработанный алгоритм на языке C++;
- разработать контрольный тест к программе;
- отладить программу;
- составить отчет о лабораторной работе.

5. ЗАДАЧИ К ЛАБОРАТОРНОЙ РАБОТЕ

1. Отсортируйте по неубыванию методом «пузырька» одномерный целочисленный массив, заданный случайными числами на промежутке [-100; 100). Выведите на экран исходный и отсортированный массивы.

2. Отсортируйте по невозрастанию методом простого выбора одномерный вещественный массив, заданный случайными числами на промежутке [0; 50). Выведите на экран исходный и отсортированный массивы.

3. Отсортируйте по возрастанию методом простого включения одномерный целочисленный массив, заданный с клавиатуры различными числами. Выведите на экран исходный и отсортированный массивы.

4. Массив размером $2m + 1$, где m – натуральное число, заполнен случайным образом. Найдите в массиве медиану. Медианой называется элемент ряда, делящий его на две равные части: в одной

находятся элементы, которые не меньше медианы, в другой – не больше медианы.

5. Массив размером m, где m – натуральное число, заполнен случайным образом. Найдите в массиве моду. Модой называется элемент ряда, который встречается наиболее часто.

6. Разработать алгоритм и программу переразмещения чисел в некоторой заданной таблице таким образом, чтобы все отрицательные значения предшествовали положительным. Элементы не сортировать полностью; достаточно отделить отрицательные числа от положительных. Алгоритм должен производить минимальное число обменов.

7. Разработать алгоритм и программу слияния двух непересекающихся приоритетных очередей, представленных в виде левосторонних деревьев, в одну (в частности, если одна из данных очередей содержит всего один элемент, то алгоритм будет вставлять его в другую очередь).

8. Даны записи $R1$, ..., RN и $r1$, ..., rn, ключи которых различны и упорядочены, т.е. $K1<...<KM$ и $k1<...<kn$. Разработать алгоритм и программу слияния этих записей в один файл, в котором отсутствуют записи Ri первого файла, если во втором файле есть запись с таким же ключом.

9. Разработать алгоритм и программу сортировки слиянием списков основанную на трехпутевом слиянии.

10. Написать программы, реализующие алгоритмы сортировки выбором (простой выбор, выбор из дерева). Сравнить их эффективность (число сравнений и перестановок) для наилучшего и наихудшего вариантов последовательностей.

11. Написать программы, реализующие алгоритмы обменных сортировок (метод пузырька, параллельная сортировка Бэтчера). Сравнить их эффективность (число сравнений и перестановок) для наилучшего и наихудшего вариантов последовательностей.

12. Написать программы, реализующие алгоритмы обменных сортировок (быстрая сортировка; обменная поразрядная сортировка). Сравнить их эффективность (число сравнений и перестановок) для наилучшего и наихудшего вариантов последовательностей.

13. Написать программы, реализующие алгоритмы сортировки вставками (простые вставки, бинарные и двухпутевые вставки, вставки в список). Сравнить их эффективность (число сравнений и

перестановок) для наилучшего и наихудшего вариантов последовательностей.

14. Написать программы, реализующие алгоритмы сортировки вставками (метод Шелла, сортировка с вычислением адреса). Сравнить их эффективность (число сравнений и перестановок) для наилучшего и наихудшего вариантов последовательностей.

15. Разработать программы сортировки слияниями и пирамидальной сортировки. Сравнить их эффективность (число сравнений и перестановок) для наилучшего и наихудшего вариантов последовательностей.

16. Пусть дан массив $a1, ..., an$. Требуется переставить $a1$ и an так, чтобы вначале в массиве шла группа, больших того элемента, который в исходном массиве располагается на первом месте, затем сам этот элемент, потом группа элементов, меньших или равных ему. Число сравнений и перемещений, каждое в отдельности не должно превышать n-1.

17. Пусть файлы c и d с компонентами, являющимися действительными или целыми числами, упорядочены по не возрастанию компонент. Требуется собрать компоненты c и d в упорядоченном виде в файле f. Число сравнений не должно превосходить $p+q$, где p и q − число компонент в файлах c и d.

18. Вдоль линии расположены лунки и в каждой лунке лежит шар черного и белого цвета. Одним ходом разрешается менять местами два любых шара. Добиться того, чтобы сначала шли белые шары, а за ними черные. Если общее число лунок равно n, то для решения задачи достаточно сделать не более $n/2$ ходов.

19. Вдоль линии расположены лунки и в каждой лунке лежит красный, белый или синий шар. Одним ходом разрешается менять местами два любых шара. Добиться того, чтобы все красные шары шли первыми, все синие − последними, а белые − посередине. Если общее число лунок равно n, то для решения задачи достаточно сделать не более n-1 хода.

20. Выбрать метод и написать программу сортировки телефонного справочника по фамилиям в лексикографическом порядке.

21. Для упорядочивания пяти чисел достаточно семи сравнений. Это не простой факт, хотя он и допускает формулировку в виде задачи на взвешивания. Имеются весы без гирь и пять различных

грузов. На каждую из двух чашек весов можно класть по одному грузу. Расположить грузы по убыванию веса, использовав семь взвешиваний.

22. Реализуйте шейкер-сортировку (попеременный проход в пузырьковой сортировке в противоположных направлениях). Покажите, что если $x[i]$ и $x[i+1]$ не меняются местами при двух последовательных проходах, то они находятся в своих конечных положениях.

23. Придумайте и реализуйте алгоритм быстрой сортировки, аналогичный быстрой сортировке, в котором расщепление осуществляется на основании самого старшего разряда имен: если этот разряд единица, то имя пересылается в левую часть таблицы. Расщепления следующего уровня основываются на следующих разрядах имени. Этот метод известен как цифровая обменная сортировка. Проанализируйте лучший и худший случаи упорядочивания элементов для (а) числа перестановок, (б) числа просмотров разрядов, (в) максимальной глубины стека. Проанализируйте средние случаи для (а), (б) и (в), считая, что имена, которые нужно отсортировать − это числа 0, 1, 2, ..., (2^t) -1, расположенные в случайном порядке.

24. Проведите эксперимент, подтверждающий или опровергающий утверждение о том, что в среднем быстрая сортировка эффективнее пирамидальной.

25. Разработать алгоритм и программу лингвистической сортировки слов из текстового файла с указателем повторяемости слов. Подобрать тесты, провести подробный анализ работы алгоритма в лучшем, в среднем и в худшем случаях упорядочивания элементов, оценить ее эффективность и дать рекомендации по ее использованию.

26. Разработать алгоритм и программу сортировки методом выбора, когда таблица представлена в виде связанного списка. Подобрать тесты, провести подробный анализ работы алгоритма в лучшем, в среднем и в худшем случаях упорядочивания элементов, оценить ее эффективность и дать рекомендации по ее использованию.

27. Разработать алгоритм и программу сортировки на бинарных деревьях. Подобрать тесты, провести подробный анализ работы алгоритма в лучшем, в среднем и в худшем случаях упорядочивания

элементов, оценить ее эффективность и дать рекомендации по ее использованию.

28. Разработать алгоритм и программу сортировки с учетом того факта, что все записи, начиная с некоторой и кончая последней, расположены в правильном порядке. Следовательно, эти записи не должны снова анализироваться. Подобрать тесты, провести подробный анализ работы алгоритма в лучшем, в среднем и в худшем случаях упорядочивания элементов, оценить ее эффективность и дать рекомендации по ее использованию.

29. Напишите программу сортировки методом пузырька так, чтобы очередные просмотры выполнялись в противоположных направлениях. (Эта сортировка известна как шейкер-сортировка.) Докажите, что если для двух элементов не выполняется транспозиция во время двух последующих просмотров в противоположных направлениях, то они находятся на своих окончательных позициях. Подобрать тесты, провести подробный анализ работы алгоритма в лучшем, в среднем и в худшем случаях упорядочивания элементов, оценить ее эффективность и дать рекомендации по ее использованию.

30. Сортировка методом подсчета выполняется следующим образом. Заводится некоторый массив *count* и элементу *count* (*i*) присваивается значение числа элементов, которые меньше или равны *x* (*i*). Затем элемент *x* (*i*) помещается в позицию *count* (*i*) в выходном массиве. (Остерегайтесь возможности существования одинаковых элементов.) Напишите программу для сортировки массива *x* размером *n*, используя этот метод. Подберите тесты, проведите подробный анализ работы алгоритма в лучшем, в среднем и в худшем случаях упорядочивания элементов, оцените ее эффективность и дайте рекомендации по ее использованию.

31. Предположим, что некоторый файл содержит целые числа в диапазоне от *a* до *b*, причем многие числа повторяются несколько раз. Сортировка методом распределяющего подсчета выполняется следующим образом. Заводится некоторый массив *number* размером *b*-*a*+1 и элементу *number* (*i*-*a*+1) присваивается значение, равное тому, сколько раз число i появляется в данном файле. Затем значения в файле переустанавливаются. Напишите программу для сортировки массива *x* размером *n* в диапазоне от *a* до *b*, используя этот метод. Подберите тесты, проведите подробный анализ работы алгоритма в лучшем, в среднем и в худшем случаях упорядочивания элементов,

оцените ее эффективность и дайте рекомендации по ее использованию.

32. Сортировка методом четных и нечетных транспозиций выполняется следующим образом. Выполните несколько раз просмотры файла. На первом просмотре сравнивайте *x* (*i*) с *x* (*i*+1) для всех нечетных *i*. На втором просмотре сравнивайте *x* (*i*) с *x* (*i*+1) для всех четных *i*. Каждый раз, когда *x* (*i*) > *x* (*i*+1) , выполняйте транспозицию этих двух элементов. Продолжайте эти просмотры пока файл не будет отсортирован.

а) Что является условием окончания этой сортировки?

б) Напишите программу, реализующую эту сортировку.

в) Какова оценка сложности этой сортировки в худшем, в среднем и лучшем случаях?

г) Сравнить эту сортировку с сортировкой методом пузырька.

33. Имеется *n* предметов весом *a*[*i*]. Необходимо их рассортировать по *m* посылкам, чтобы вес каждой посылки не превышал *p* кг, а число посылок было минимальным. Разработать алгоритм и программу решения этой задачи.

34. Имеется последовательность, состоящая из *n* чисел. Допустим, что можно складывать его элементы и сравнивать суммы. Сколько требуется сравнений для нахождения при этих условиях наибольшего элемента последовательности. Разработать алгоритм и программу решения этой задачи.

35. Модифицировать алгоритм быстрой сортировки так, чтобы использовалась сортировка «методом пузырька», если некоторый подмассив является небольшим. Определите, использовав реальные расчеты на ЭВМ, каким малым должен быть подмассив, чтобы эта смешанная стратегия была более эффективной, чем обычная «быстрая сортировка».

36. Рассмотрим следующий метод объединения массивов *a* и *b* в массив c. Выполним бинарный поиск элемента *b*(1) в массиве *a*. Если *b*(1) находится между *a*(*i*) и *a*(*i*+1), то выводятся элементы от *a*(1) до *a*(*i*) в массив c, затем выводится элемент *b* (1) в массив *c*. Далее выполните бинарный поиск элемента *b* (2) в подмассиве с элементами от *a*(*i*+1) до *a*(*la*) (где *la* является числом элементов в массиве *a*) и повторите процесс вывода. Повторите эту процедуру для каждого элемента в массиве *b*. Напишите программу, реализующую этот

метод. В каких случаях он наиболее эффективен и наоборот и почему?

37. Имеется файл, содержащий английские слова, разделенные пробелами. Необходимо отсортировать эти слова в следующем порядке:

- в начале располагаются все слова, состоящие из прописных букв;
- затем располагаются все слова, имеющие три последних символа равными *exe*, *com*, *bat*;
- и затем располагаются все слова, имеющие три последних символа *doc*, *txt*;
- затем располагаются все остальные слова.

В пределах каждой из указанных групп слова должны быть отсортированными в лексикографическом порядке. Оценить эффективность алгоритма.

38. Написать программу, которая сортируют файл, в начале применяя поразрядную сортировку для R старших значений цифр (R−заданная константа), а затем использует простые вставки для сортировки всего файла, что исключает излишние просмотры младших цифр и программу поразрядной сортировки в чистом виде для этих же файлов. Сравнить их эффективность (число сравнений и перестановок) для наилучшего среднего и наихудшего вариантов последовательностей.

39. Имеется последовательность состоящая из двузначных чисел. Трактуя каждое число как пару цифр, выполнить их упорядочивание, используя пирамидальную сортировку и карманную сортировку. Сравнить их эффективность (число сравнений и перестановок) для наилучшего среднего и наихудшего вариантов последовательностей.

40. Имеется множество слов, т.е. строк состоящих из букв а-я. Суммарная длина слов равна N. Алгоритмы с какой минимальной сложностью можно применить здесь. Написать программы, реализующие этот алгоритм, сравнить ее эффективность на примере другого метода.

Замечание: для слов одинаковой длины можно применить карманную сортировку, но предусмотреть и случай слов разной длины.

41. Дан массив x: array [1..*n*] *of integer*. Определить различне числа среди элементов этого массива. Выбрать два алгоритма (число действий должно быть порядка $n \cdot log(n)$.) разработать программы и оценить эффективность их при различных размерах массива.

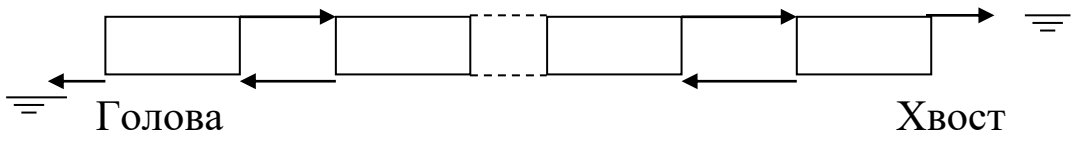
42. Выбрать два алгоритма сортировки, имеющие сложность $O(n*log(n))$, разработать программы и проверить их на различного типа данных.

43. Написать программу сортировки числовой последовательность типа 8, 4, 1, 9, 2, 1, 7, 4 посредством выбора. Отображайте состояние списка после каждого прохода. Предусмотреть сортировку символьной последовательности типа V, B, L, A, Z, Y, C, H, S, S, B, H.

44. Написать программу сортировки числовой последовательность типа 8, 4, 1, 9, 2, 1, 7, 4 методом вставки. Отображайте состояние списка после каждого прохода. Предусмотреть сортировку символьной последовательности типа V, B, L, A, Z, Y, C, H, S, S, B, H.

45. Массив *A* отсортировать включением его элементов в двусвязный список. Вставьте элемент в текущую точку и перемещайте его вперед по списку, если новый элемент больше текущего, или назад, если меньше. Напишите функцию *DoubleSort*, реализующую этот метод сортировки.

```
template <class T>
void DoubleSort  (T a[],  int n) ;
```



Оцените алгоритмическую сложность метода сортировки. Рассмотрите наилучший, наихудший и средний случаи.

46. Реализуйте следующий алгоритм:

Разбить *n*-элементный список пополам. Отсортировать каждую половину с помощью сортировки выбором, а затем слить обе половины.

а) Проанализируйте сложность этой сортировки.

б) Используйте этот алгоритм для сортировки 20000 случайных чисел и измерьте время выполнения программы.

в) Запустите ту же программу, но использующую обычную сортировку посредством выбора. Какая версия работает быстрее?

47. Дана структура

```
struct TwoKey
{
 int primary; int secondary;
 };
```

Создайте массив из 100 записей этого типа. Поле primary содержит случайное число в диапазоне 0..9, а поле *secondary* – в диапазоне 0..100. Модифицируйте алгоритм сортировки вставками для упорядочения по двум ключам. Новый алгоритм должен производить сортировку по вторичному ключу для каждого фиксированного значения первичного ключа. Отсортируйте с его помощью ранее созданный массив. Распечатайте массив в формате *primary* (*secondary*).

48. В этом упражнении разрабатывается простая программа орфографического контроля. Имеется файл *words*, который содержит 500 наиболее часто употребляющихся слов. Прочитайте этот файл и вставьте все имеющиеся там слова в хеш-таблицу. Прочитайте текстовый документ и разбейте его на отдельные слова с помощью следующей несложной функции:

```
    // извлечь слово, начинающееся с буквы и состоящее из
букв и цифр
    int GetWord (ifstreams fin, char w[{})
    {
    char c;
    int i = 0;
    // пропустить все не буквы
    while (fin.get(c) ss lisalpha(c));
    // возвратить 0 по окончании файла
    if (fin.eof())
    return 0;
    // записать первую букву слова
    w[i++] =c;
    // собрать буквы и цифры. Завершить слово нулем
    while (fin.get(c) ss (isalpha(c) || isdigit(c)))
```

```
    w[i++]=c;
    w[i]='\0';
    return 1;
    char c;
    int i - 0;
    // пропустить все не буквы
    while (fin,get(c) ss lisalpha(c));
    // возвратить 0 по окончании файла if (fin.eof())
return 0;
    // записать первую букву слова w[i++] =c;
    // собрать буквы и цифры. Завершить слово нулем while
(fin.get(c) ss (isalpha(c) II isdigit(c)))
    c; '\0';
    return 1;
    }
```

49. Построить алгоритм и разработать программу сортировки методом двухпутевого слияния, учитывающие степень упорядочения, которое существует в исходной таблице.

50. Построить алгоритм и разработать программу двухпутевой сортировки слиянием, используя методы связанного распределения.

51. Дан массив целых чисел. Найти количество различных чисел среди элементов этого массива. Выбрать два алгоритма (число действий должно быть порядка $n*log(n)$.) разработать программы и оценить эффективность их при различных размерах массива.

52. Разработайте программу сортировки числовой последовательности методом простого выбора, а затем символьной последовательности. Отображайте состояние последовательности после каждого прохода. Дайте анализ эффективности

53. Разработайте программу сортировки методом простых вставок числовой последовательности, а затем символьной последовательности. Отображайте состояние последовательности после каждого прохода. Оцените эффективност программы.

54. Упорядочить массив записей по их ключам. Записи занимают несколько слов памяти, а ключи короткие. В этом случае лучше составить таблицу ключей и упорядочивать ключи, а затем по ним расположить записи. Перестановку записей по отсортированным ключам осуществлять на том месте памяти, где располагается исходный массив записей (т.е. нельзя заводить вспомогательный массив).

55. Составить программу обработки сведений о студентах. Сведения заданы в файле и сожержат следующую информацию о каждом студенте: фамилию, имя, отчество, индекс группы и результаты сдачи сессии; оценки по четырем предметам. Вывести фамилии отличников и фамилии неуспевающих студентов.

56. Дана последовательность $a_1,a_2,\ldots,a_n$ и перестановка $p(1),p(2),\ldots,p(n)$. Составьте программу, которая располагала бы эту последовательность в порядке $a_{p(1)},a_{p(2)},\ldots,a_{p(n)}$ без использования вспомогательных массивов.

57. В памяти ЭВМ хранятся списки номеров телефонов и фамилий абонентов, упорядоченных в алфавитном порядке. Составить программу, обеспечивающую быстрый поиск фамилии абонента по номеру телефона.

58. Фамилии участников соревнований по фигурному катанию после короткой программы расположены в порядке, соответствующем занятому месту. Составить список участников в порядке их стартовых номеров для произвольной программы (участники выступают в порядке, обратном занятым местам).

59. Результаты соревнований фигуристов по одному из видов многоборья представлены оценками судей в баллах (от 0 до 6). По результатам оценок судьи определяется место каждого участника у этого судьи. Места участников определяются далее по сумме мест, которые каждый участник занял у всех судей. Составить программу, определяющую по исходной таблице оценок фамилии и сумму мест участников в порядке занятых ими мест. Число участников не более 15, число судей не более 10.

60. При поступлении в университет абитуриенты, получившие двойку на первом экзамене, ко второму не допускаются. Считая фамилии абитуриентов и их оценки после первого экзамена исходными данными, составить список абитуриентов, допущенных к первому экзамену.

61. Сортировка массива $X[n]$ методом подсчета выполняется следующим образом. Заводится некоторый массив count и элементу $count[i]$ присваивается значение числа элементов, которые меньше или равны $x[i]$. Затем элемент $x[i]$ помещается в позицию $count[i]$ в выходном массиве. (Однако остерегайтесь возможности существования одинаковых элементов). Составьте программу для сортировки массива $X[n]$, используя этот метод.

62. Предположим, что некоторый массив содержит целые числа в диапазоне от *a* до *b*, причем многие числа повторяются несколько раз. Сортировка методом распределяющего подсчета выполняется следующим образом. Заводится некоторый массив *number* размером *b*-*a*+1 и элементу *number*(*i*-*a*+1) присваивается значение, равное тому, сколько раз число *i* появляется в данном массиве. Затем значения в данном массиве соответственно переустанавливаются. Составьте программу для сортировки массива *X*[*n*], содержащего числа в диапазоне от *a* до *b*, используя этот метод.

63. Пусть *A*=(*a*1, *a*2...,*an*), *B*(*b*1, *b*2...,*bn*), *C*(*c*1, *c*2...,*cn*) – три конечные последовательности натуральных чисел. Допустим, что каждая отдельно взятая последовательность упорядочена по возрастанию. Объедините три последовательности в одну последовательность $D(d_1, d_2...,d_{n+m+p})$, являющуюся списком всех чисел из *A, B* и *C*, такую, что $d_1 \le d_2...\le d_{n+m+p}$. Обязательным условием является то, что слияние должно быть выполнено за *m*+*n*+*p* действий, т.е. число сравнений должно быть не больше *m*+*n*+*p*.

64. Есть набор из *n* кеглей, который можно описать с помощью массива *T*[*n*]. Устройство, управляемое ЭВМ, умеет выполнять операцию «Переставить (*i*,*j*) {1≤*i*≤*j*≤*n*}». Это значит, что устройство берет кеглю *T*[*i*] и кеглю *T*[*j*] и переставляет их местами. Кроме того, оптический датчик этого устройства может выполнять проверку цвета кегли *T*[*i*]: «Цвет (*i*) {1≤*i*≤*n*}». Возможные цвета: голубой, белый и красный. Требуется написать программу, реорганизующую массив так, чтобы кегли располагались как на французском флаге. (На французском флаге цвета следуют в порядке: голубой, белый, красный.)

65. Написать программу сортировки однонаправленного списка записей, содержащих фамилию студента, его год рождения, индекс группы и время забега. Сортировка должна выполняться по ключу «фамилия» в лексико-графическом порядке.

Пример решения задач

Написать программу пирамидальной сортировки. Подобрать тесты, провести подробный анализ в лучшем, в среднем и в худшем

случаях, оценить ее эффективность и дать рекомендации по ее использованию.

Анализ пирамидальной сортировки. С первого взгляда неочевидно, что этот метод сортировки дает хорошие результаты. Ведь элементы с большими ключами вначале просеиваются влево, прежде чем, наконец, окажутся справа. Действительно, эта процедура не рекомендуется для такого небольшого числа элементов, как восемь элементов последовательности. Однако для большого числа элементов (*n*) пирамидальная сортировка оказывается очень эффективной, и чем больше *n* тем она более эффективна − даже по сравнению с сортировкой Шелла.

В худшем случае необходимы *n*/2 шагов, которые просеивают элементы через *log* (*n*/2), *log* (*n*/2-1) ,.., *log* (*n*-1) позиций (здесь берется целая часть логарифма по основанию 2). Следовательно, на фазе сортировки происходит *n*-1 просеиваний с самое большее *log* (*n*-1), *log* (*n*-2) , .., 1 пересылками. Кроме того, требуется *n*-1 пересылок для того, чтобы отложить просеянный элемент вправо. Отсюда видно, что пирамидальная сортировка требует *n·log* (*n*) шагов даже в худшем случае. Такие отличные характеристики для худшего случая − одно из самых выгодных качеств пирамидальной сортировки.

Не совсем ясно, в каких случаях можно ожидать наименьшую или наибольшую эффективность сортировки. Однако для пирамидальной сортировки, видимо, больше всего подходят случаи, когда элементы более или менее рассортированы в обратном порядке, т.е. для нее характерно неестественное поведение. Очевидно, что при обратном порядке фаза построеня пирамиды не требует никаких пересылок. Для восьми элементов минимальное и максимальное число пересылок дают следующие исходные последовательности:

Mmin = 13 для последовательности
94 67 44 55 12 42 18 6
Mmax = 24 для последовательности
18 42 12 44 6 55 67 94

Среднее число пересылок равно приблизительно 1/2*n·log n* и отклонения от этого значения сравнительно малы.

Реализация алгоритма пирамидальной сортировки на Си

```
#include <stdio.h>
#include <conio.h>

const int n=11;

struct item
{
    int key;
};

void HeapSort(item a[])
{
    int l,r;
    item x;
    void sift(item [],int,int);
    l=(n-1)/2+1;
    r=n-1;
    while(l>1)
    {
        l--;
        sift(a,l,r);
    }
    while(r>1)
    {
        x=a[1];
        a[1]=a[r];
        a[r]=x;
        r--;
        sift(a,l,r);
    }
}
void main(void)
{
    item a[n];
    clrscr();
    printf("Введите %d целых:\n",n-1);
    for(int i=1;i<n;i++)
        scanf("%d",&a[i].key);
    HeapSort(a);
    for(i=1;i<n;i++)
        printf("%d ",a[i].key);
    getch();
```

```
}

void sift(item a[],int l,int r)
{
     int i,j;
     item x;
     i=l;
     j=2*i;
     x=a[i];
     while(j<=r)
     {
          if(j<r)
               if(a[j].key < a[j+1].key) j++;
          if(x.key>=a[j].key) goto L1;
          a[i]=a[j];
          i=j;
          j=2*i;
     }
L1:  a[i]=x;
}
```

Реализация алгоритма пирамидальной сортировки на Pascal

```
 program sort;
 const n=10;
 type index=0..n;
    item=record key: integer end;
 var a: array[1..n] of item;
     ii: integer;
 procedure heapsort;
    var l, r: index; x: item;
    procedure sift;
    label 13;
    var i, j: index;
    begin
    i: =l; j: =2*i; x: =a[i];
    while j<=r do
    begin if j<r then
    if a[j].key<a[j+1].key then j: =j+1;
    if x.key>=a[j].key then goto 13;
    a[i]: =a[j]; i: =j; j: =2*i;
    end;
    13: a[i]: =x
    end;
    begin l: = (ndiv 2) +1; r: =n;
```

```
    while l>1 do
    begin l: =l-1; sift
    end;
    while r>1 do
    begin x: =a[1]; a[1]: =a[r]; a[r]: =x;
    r: =r-1; sift
    end

    end{heapsort};
begin for ii:=1 to n do readln (a[ii].key) ;
heapsort; for ii: = 1 to n do write (a[ii].key, ' ');
end.
```

6. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. От чего зависит производительность сортировки?
2. Чем отличаются алгоритмы внутренней сортировки от алгоритмов внешней сортировки?
3. Каковы общие критерии оценки эффективности сортировки?
4. Какие методы оценки эффективности сортировки Вы знаете?
5. В каких случаях алгоритм сортировки простыми вставками эффективен?
6. Какова сложность алгоритма сортировки простыми вставками?
7. Какова максимальная сложность самого неэффективного алгоритма сортировки?
8. Какова сложность самого эффективного алгоритма сортировки?
9. Чем отличается сортировка списка от сортировки массива?
10. В каких случаях сортировка со сложностью $O(n^2)$ может быть эффективной?
11. Какой из методов сортировки вставками наиболее эффективен?
12. Какой из обменных методов сортировки наиболее эффективен для больших последовательностей?
13. В каких случаях наиболее эффективна параллельная сортировка Бэтчера?
14. Какова сложность поразрядной сортировки?
15. Когда поразрядная сортировка наиболее эффективна?

16. Какие сортировки выбором наиболее эффективны для больших последовательностей?

17. Для каких структур данных применима пирамидальная сортировка?

18. Какие методы сортировки требуют меньше всего сравнений?

19. Существует ли оптимальные методы сортировка?

20. Какие алгоритмы внешней сортировки Вы знаете?

21. Чем отличаются алгоритмы внешней сортировки?

22. Объясните, почему сортировка методом простого выбора более эффективна, чем сортировка методом пузырька?

23. Какой из основных алгоритмов сортировки (выбором, вставками или пузырьковый) наиболее эффективен для обработки отсортированного списка? А если этот список отсортирован в обратном порядке?

24. Почему существует большое число алгоритмов сортировок?

25. С какой целью используются простые сортировки, если они характеризуются малой эффективностью?

26. Чем отличается принцип сортировки по неубыванию (невозрастанию) от сортировки по возрастанию (убыванию)?

27. На каких наборах исходных данных проявляется эффективность алгоритмов простых сортировок по сравнению друг с другом?

28. В чем заключается улучшение метода шейкер-сортировки по сравнению сортировкой методом пузырька?

7. СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Кнут, Д. Искусство программирования для ЭВМ Т.1, Основные алгоритмы / Д. Кнут – М.: Вильямс 2000. – 215 с.

2. Вирт, Н. Алгоритмы и структуры данных / Н. Вирт – М.: МИР, 1989. – 363 с.

3. Кнут, Д. Искусство программирования для ЭВМ Т. 3. Сортировка и поиск / Д. Кнут – М: Вильямс, 2000. – 245 с.

4. Пападимитриу, Х. Комбинаторная оптимизация. Алгоритмы и сложность: пер. с англ. / Х. Пападимитриу, К.М. Стайглиц. – М.: Мир 1985. – 512 с.

5. Топп, У. Структуры данных в С ++: пер. с англ. / У. Топп, У. Форд. – М.: БИНОМ, 1999. – 816 с.

6. Кормен, Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест. – М: МЦНМО, 1999. –960 с.

7. Гэри, М. Вычислительные машины и труднорешаемые задачи / М. Гэри, Д. Джонсон. – М.: Мир, 1982. – 416 с.

8. Ахо, А. В. Структуры данных и алгоритмы / А. В. Ахо, Д. Э. Хопкрофт, Д. Д. Ульман. – М.: Вильямс, 2000. – 384 с.

9. Кубенский, А. А. Создание и обработка структур данных в примерах на Java / А. А.Кубенский. – СПб.: БХВ-Петербург, 2001. – 336 с.

10. Седжвик, Р. Фундаментальные алгоритмы на С++. Анализ. Структуры данных. Сортировка. Поиск: пер. с англ. /Р. Седжвик. – К.: ДиаСофт, 2001. – 688 с.

11. Хэзфилд, Р. Искусство программирования на С. Фундаментальные алгоритмы, структуры данных и примеры приложений. Энциклопедия программиста: пер. с англ. / Р. Хэзфилд, Л. Кирби [и др.]. – К.: ДиаСофт, 2001. – 736 с.

12. Мейн, М. Структуры данных и другие объекты в С++: пер. с англ. / М .Мейн, У. Савитч. – 2-е изд.– М.: Вильямс, 2002. – 832 с.

13. Хусаинов, Б.С. Структуры и алгоритмы обработки данных. Примеры на языке Си (+ CD): учеб. Пособие / Б.С. Хусаинов. – М.: Финансы и статистика, 2004. – 464 с.

14. Макконнелл, Дж. Основы современных алгоритмов / Дж. Макконнелл. – 2-е изд. – М.: Техносфера, 2004. – 368с.

15. Гудман, С.Введение в разработку и анализ алгоритмов / С. Гудман, С.Хидетнием. – М.: Мир, 1981. – 206 с.

16. Ахо, А. Построение и анализ вычислительных алгоритмов / А. Ахо, Дж.Хопкрофт, Дж.Ульман. – М.: МИР, 1979. – 329 с.

17. Сибуя, М. Алгоритмы обработки данных / М. Сибуя, Т. Ямато. – М.:МИР, 1986. – 473 с.

18. Лэнгсам, Й.Структуры данных для персональных ЭВМ / Й. Лэнгсам, М. Огенстайн, А. Тененбаум. – М.: МИР, 1989. – 327 с.

19. Гулаков, В.К. Деревья: алгоритмы и программы / В.К. Гулаков. – М.: Машиностроение-1, 2005. – 206 с.

Структуры и алгоритмы обработки данных. Методы внутренней сортировки: методические указания к выполнению лабораторной работы №10 для студентов очной, очно-заочной и заочной форм обучения по направлениям подготовки 230100 «Информатика и вычислительная техника», 010500 «Математическое обеспечение и администрирование информационных систем», 231000 «Программная инженерия»

ВАСИЛИЙ КОНСТАНТИНОВИЧ ГУЛАКОВ

Научный редактор В.В. Конкин
Редактор издательства Л.Н. Мажугина
Компьютерный набор В.К. Гулаков

Темплан 2013 г., п.210

Подписано в печать Формат 1/16 Бумага офсетная. Офсетная печать. Усл.печ.л. 1,8 Уч.-изд.л. 1,8 Тираж 40 экз. Заказ Бесплатно

Издательство Брянского государственного технического университета
241035, Брянск, бульвар им.50-летия Октября, 7, БГТУ, тел. 58-82-49
Лаборатория оперативной полиграфии БГТУ, ул. Институтская, 16