МИНОБРНАУКИ РОССИИ

Орский гуманитарно-технологический институт (филиал)
федерального государственного бюджетного образовательного учреждения
высшего образования «Оренбургский государственный университет»
(Орский гуманитарно-технологический институт (филиал) ОГУ)

Кафедра программного обеспечения

Методические указания по выполнению и защите лабораторных работ по дисциплине «Б1.Д.В.1 Структуры и алгоритмы обработки данных»

Уровень высшего образования

БАКАЛАВРИАТ

Направление подготовки

09.03.01 Информатика и вычислительная техника

(код и наименование направления подготовки)

Программное обеспечение средств вычислительной техники и автоматизированных систем

(наименование направленности (профиля) образовательной программы)

Тип образовательной программы

Программа бакалавриата

Квалификация

Бакалавр

Форма обучения

Очная

Год начала реализации программы (набора)

2019

г. Орск 2018

Методические указания предназначены для обучающихся очной формы обучения направления подготовки 09.03.01 Информатика и вычислительная техника профилю Программное обеспечение средств вычислительной техники и автоматизированных систем по дисциплине «Б1.Д.В.1 Структуры и алгоритмы обработки данных»

Составитель _____ О.В. Подсобляева

Методические указания рассмотрены и одобрены на заседании кафедры программного обеспечения, протокол № 1 от «01» сентября 2018 г.

Заведующий кафедрой _____ Е.Е. Сурина

Согласовано:
Председатель методической комиссии по направлению подготовки 09.03.01 Информатика и вычислительная техника

_____ Е.Е.Сурина
«12» сентября 2018 г.

Пояснительная записка

В результате изучения дисциплины «Б1.Д.В.1 Структуры и алгоритмы обработки данных» у обучающихся должны быть сформированы знания, умения и навыки:

– изучить основные принципы объектно-ориентированного программирования; изучить реализацию этих принципов на языках C++ и Java;

– научиться писать программы на языках C++ и Java;

– научиться проектировать и разрабатывать объектно-ориентированные программы.

Одной из наиболее эффективных форм закрепления теоретических знаний и выработки навыков самостоятельной работы являются лабораторные занятия.

Целью проведения лабораторных занятий является:

- закрепление знаний студентов по основам проектной деятельности,

- формирование у студентов навыков использования современных технических средств и технологий для решения проектных и исследовательских задач.

Тематический план

Таблица 1 – Тематический план выполнения лабораторных работ по дисциплине «Б1.Д.В.1 Структуры и алгоритмы обработки данных» для обучающихся направления подготовки 09.03.01 Информатика и вычислительная техника профиль подготовки Программное обеспечение средств вычислительной техники и автоматизированных систем

Методические указания по выполнению и оформлению лабораторных работ

Лабораторные работы по дисциплине «Структуры и алгоритмы обработки данных» предполагают решение задач по темам, представленным в тематическом плане.

В практической работе должны быть выполнены все предусмотренные задания. В работе должна просматриваться логическая последовательность и взаимная увязка основных частей работы.

Рекомендуемая структура лабораторных работ:

1) цель практической работы;

2) задание в соответствии с выбранным вариантом;

3) теоретическая часть, включающая краткое изложение теоретических положений по теме практической работы, формулы для решения задания;

4) практическая часть, включающая решение задания по теме практической работы. Дополнительно для наглядности расчетный материал может быть представлен в виде таблиц, графиков;

5) выводы по практической работе;

6) список использованной литературы.

Лабораторные работы могут быть оформлены:

- машинописным текстом на листах формата А4.

Титульный лист оформляется на основе СТО 02069024. 101 – 2014 «РАБОТЫ СТУДЕНЧЕСКИЕ. Общие требования и правила оформления».

Работа защищается устно и принимается к зачету, если нет замечаний по ее выполнению и оформлению. При отсутствии зачтенных лабораторных работ студент не допускается к зачету по дисциплине «Б1.Д.В.1 Структуры и алгоритмы обработки данных».

4.3 Лабораторные работы

№ раздела	Наименование лабораторных работ	Кол-во часов
1	Динамические массивы в C++	4

№ раздела	Наименование лабораторных работ	Кол-во часов
1	Обработка файлов в С++	4
1	Указательный тип данных.	2
1	Динамический список.	4
1	Использование стека.	2
1	Другие виды списков: очередь, циклический список,.	4
1	Другие виды списков: двунаправленный список, дек	4
2	Рекурсивные функции. Бинарный поиск.	4
2	Бинарные деревья поиска.	2
3	Поиск на основе хеширования.	2
3	Разрешение коллизий хеширования.	2
	Итого:	34

4.4 Практические занятия (семинары)

№ раздела	Тема	Кол-во часов
4	Методы обменной сортировки.	4
4	Методы сортировки логарифмической скорости.	4
4	Сравнительный анализ методов сортировок.	2
5	Внешняя сортировка. Методы слияния.	4
5	Поиск во внешней памяти.	2
	Итого:	16

Лабораторная работа №1
Динамические массивы в С++

Варианты заданий

1. Сформировать одномерный массив Удалить из него элемент с заданным номером, добавить элемент с заданным номером;
2. Сформировать одномерный массив. Удалить из него элемент с заданным ключом, добавить элемент с заданным ключом;
3. Сформировать одномерный массив. Удалить из него К элементов, начиная с заданного номера, добавить элемент с заданным ключом;
4. Сформировать одномерный массив. Удалить из него элемент с заданным номером, добавить К элементов, начиная с заданного номера;
5. Сформировать одномерный массив. Удалить из него К элементов, начиная с заданного номера, добавить К элементов, начиная с заданного номера;
6. Сформировать двумерный массив. Удалить из него строку с заданным номером;
7. Сформировать двумерный массив. Удалить из него столбец с заданным номером;
8. Сформировать двумерный массив. Добавить в него строку с заданным номером;
9. Сформировать двумерный массив. Добавить в него столбец с заданным номером;
10. Сформировать двумерный массив. Удалить из него строку и столбец с заданным номером.
11. Сформировать двумерный массив. Добавить в него строку и столбец с заданным номером.
12. Сформировать двумерный массив. Удалить из него все строки, в которых встречается заданное число.
13. Сформировать двумерный массив. Удалить из него все столбцы, в которых встречается заданное число.

14. Сформировать двумерный массив. Удалить из него строку и столбец, на пересечении которых находится минимальный элемент.

15. Сформировать двумерный массив. Удалить из него строку и столбец, на пересечении которых находится максимальный элемент.

16. Сформировать массив строк. Удалить из него самую короткую строку.

17. Сформировать массив строк. Удалить из него самую длинную строку.

18. Сформировать массив строк. Удалить из него строку, начинающуюся на букву "а".

19. Сформировать массив строк. Удалить из него строку, начинающуюся и заканчивающуюся на букву "а".

20. Сформировать массив строк. Удалить из него строку, начинающуюся и заканчивающуюся на одну и ту же букву.

21. Сформировать массив строк. Удалить из него строку с заданным номером.

22. Сформировать массив строк. Удалить из него К строк, начиная со строки с заданным номером.

23. Сформировать массив строк. Удалить из него одинаковые строки. Сформировать массив строк. Удалить из него К последних строк.

24. Сформировать массив строк. Удалить из него К первых строк.

25. Сформировать массив строк. Добавить в него строку с заданным номером.

Лабораторная работа №2
Обработка файлов

Задания:

1,14. Случайным образом создать таблицу пар целочисленных значений и записать её в текстовый файл в виде:

X Y

5 1

2 8

12 3 и т.д.

Считать из файла пары значений и в тех из них, где X>Y , поменять значения X и Y местами. Результат записать в другой текстовый файл такого же формата.

2,15. Ввести с клавиатуры попарно значения вещественного типа и записать их в текстовый файл в виде таблицы следующего формата:

X : Y

2.1 : 3.7

6.2 : 5.4 и т.д.

Считать из файла полученные пары значений и создать из них другой файл вида:

sin(x) : cos(y)

значение sin(2.1) : значение cos(3.7) и т.д.

3,16. Ввести с клавиатуры попарно значения вещественного типа и записать их в текстовый файл в виде таблицы следующего формата:

X : Y

2.1 : 3.7

6.2 : 5.4 и т.д.

Считать из файла полученные пары значений и создать из них другой файл вида:

tg(x) : ctg(y)

значение tg(2.1) : значение ctg(3.7) и т.д.

4,17. Случайным образом создать таблицу пар значений и записать её в текстовый файл в виде:

n * c

5 * m

7 * a

3 * q и т.д.

Считать из файла пары значений и создать из них другой текстовый файл вида

mmmmm

aaaaaaa

qqq

5,18. Случайным образом создать таблицу пар значений и записать её в текстовый файл в виде:

n * c

5 * 3.1

7 * 4.2

3 * 8.3 и т.д.

Считать из файла пары значений и создать из них другой текстовый файл вида произведений:

15.5

29.4

24.9

6,19. Случайным образом создать таблицу пар целочисленных значений и записать её в текстовый файл в виде:

X Y

5 1

2 8

12 3 и т.д.

Считать из файла пары значений и в тех из них, где X кратен Y , пометить строку таблицы:

X Y

5 1 ***

2 8

12 3 ***

в том же файле.

7,20. Случайным образом создать таблицу пар целочисленных значений и записать её в текстовый файл в виде:

X Y

5 8

2 1

1 3 и т.д.

Считать из файла пары значений и в тех из них, где X меньше Y , пометить строку таблицы:

X Y

5 8 ###

2 1

1 3 ###

в том же файле.

8,21. Случайным образом создать таблицу пар значений и записать её в текстовый файл в виде:

a b c

5.2 4.6 2.5

1.2 8.9 2.3 и т.д.

Считать из файла записанные данные и определить, можно ли построить треугольник с такими сторонами. Пометить соответствующие строки таблицы (в том же файле).

a b c

5.2 4.6 2.5 можно

1.2 8.9 2.3

9,22. Создать текстовый файл, содержащий вещественные значения. Считать из файла записанные данные и определить максимальное значение. Если оно находится в первой половине

файла, заменить его суммой последующих элементов, если во второй – суммой предыдущих элементов.

10,23. Случайным образом создать таблицу пар целочисленных значений и записать её в текстовый файл в виде:

X Y
5 25
1 3
49 7 и т.д.

Считать из файла пары значений и в тех из них, где X является точным квадратом Y или наоборот, найти сумму значений X и Y. Результат записать в другой текстовый файл в виде

X Y sum
5 25 30
1 3
49 7 56

11,24. Создать текстовый файл, содержащий целые числовые значения, например : 5 3 21 4 37 52 9 2. Считать из файла записанные данные и определить минимальное значение. Если оно кратно трем, заменить каждое третье значение файла нулем, если кратно пяти – заменить его суммой первого и последнего элементов.

12,25. Создать текстовый файл, содержащий целые числовые значения, например: 15 13 21 42 37 50 9 2. Считать из файла записанные данные и заменить нулем каждое значение файла, кратное минимальному числу.

13,26. Ввести с клавиатуры значения вещественного типа и записать их в текстовый файл в виде таблицы следующего формата:

X : Y : Z
2.1 : 3.7 : 0.9
6.2 : 5.4 : 4.2 и т.д.

Считать из файла полученные значения и создать из них другой файл вида:

sin(max{X,Y,Z}) : cos(min{X,Y,Z})

лабораторная работа №3,4
Динамический список. Использование стека.

Варианты задания

- 1. Создать список. Поменять местами максимальный и минимальный элементы.
- 2. Создать список. Удалить из него повторяющиеся элементы.
- 3. Создать два списка. Создать 3-й список, состоящий из элементов, которые есть как в первом, так и во втором списке.
- 4. Создать два списка. Создать 3-й список, объединяющий первый и второй списки. Удалить максимальный элемент в новом списке.
- 5. Создать список и отсортировать его по убыванию.
- 6. Создать два списка одинаковой длины. Произвести поэлементное вычитание из первого списка элементов второго списка.
- 7. Создать два списка. Создать 3-й список, состоящий из неповторяющихся элементов первых двух списков.
- 8. Создать список. Вставить в него после максимального элемента копию минимального элемента.
- 9. Создать список. После каждого отрицательного элемента вставить элемент, равный 0.
- 10. Создать список. Посчитать сумму всех элементов. Полученный результат вставить после минимального элемента списка.
- 11. Создать список. Удалить из него все отрицательные элементы.
- 12. Создать два списка. Создать 3-й список, состоящий из положительных элементов двух первых списков.

Лабораторная работа №5,6,7
Другие виды списков: очередь, циклический список,. Другие виды списков: двунаправленный список, дек Рекурсивные функции. Бинарный поиск.

Варианты индивидуальных заданий

1. Создать стек для целых чисел. Максимальный размер стека вводится с экрана. Создать функции для ввода и вывода элементов стека. Добавить 6 элементов. Удалить и вывести на экран 2 элемента.
2. Создать стек для целых (положительных и отрицательных) чисел. Максимальный размер стека вводится с экрана. Создать функции для ввода и вывода элементов стека. Ввести с экрана 6 элементов. При вводе чисел в стек попадают только отрицательные элементы. Вывести все элементы стека.
3. Создать стек для целых чисел. Максимальный размер стека вводится с экрана. Создать функции для ввода, вывода и определения размера стека. Ввести с экрана 6 элементов. Удалить 2 элемента. Вывести размер стека.
4. Создать стек для целых (положительных и отрицательных) чисел. Максимальный размер стека вводится с экрана. Создать функции для ввода, вывода и определения размера стека. Вводить с экрана числа, причем в стек должны добавляться поочередно положительные и отрицательные числа.
5. Создать стек для символов. Максимальный размер стека вводится с экрана. Создать функции для ввода и вывода элементов стека. Ввести эталонный символ. Вводить символы с экрана в стек до встречи эталонного. Вывести все элементы стека.
6. Создать стек для символов. Максимальный размер стека вводится с экрана. Создать функции для ввода и вывода элементов стека. Добавить символы с экрана в стек. После добавления 5-го символа перед добавлением удалять элемент из стека.
7. Создать стек для символов. Максимальный размер стека вводится с экрана. Создать функции для ввода, вывода и определения размера стека. Ввести эталонный символ. Вводить символы с экрана в стек до встречи эталонного. Вывести размер стека.
8. Создать стек для символов. Максимальный размер стека вводится с экрана. Создать функции для ввода и вывода элементов стека. Добавлять символы с экрана в стек. В случае совпадения вводимого символа с вершиной стека вытолкнуть его и распечатать ее.
9. Создать стек для символов. Максимальный размер стека вводится с экрана. Создать функции для ввода, вывода и определения размера стека. Вводить символы с экрана в стек. В случае совпадения вводимого символа с вершиной стека вывести размер стека.
10. Создать два стека для символов. Максимальный размер стеков вводится с экрана. Создать функции для ввода и вывода элементов стека. Вводить символы с экрана в первый стек. В случае совпадения вводимого символа с вершиной стека вводить во второй стек.
11. Создать два стека для символов. Максимальный размер стеков вводится с экрана. Создать функции для ввода и вывода элементов стека. Вводить символы с экрана в стеки поочередно.
12. Создать стек для символов и стек для чисел. Максимальный размер стеков вводится с экрана. Создать функции для ввода и вывода элементов стека. Вводить символы с экрана. Символ попадает в первый стек, а его численное представление – во второй.
13. Создать два стека для символов. Максимальный размер стеков вводится с экрана. Создать функции для ввода и вывода элементов стека. Вводить символы с экрана. Прописные буквы попадают в первый стек, строчные – во второй, остальные символы пропускаются.
14. Создать два стека для символов. Максимальный размер стеков вводится с экрана. Создать функции для ввода и вывода элементов стека. Вводить символы с экрана. Прописные буквы преобразуются в строчные и попадают в первый стек, строчные преобразуются в прописные и попадают во второй, остальные символы пропускаются.
15. Создать стек для целых чисел. Максимальный размер стека вводится с экрана. Создать функции для ввода и вывода элементов стека. Вводить символы с экрана. Числовое представление символа попадает в стек.

16. Создать стек для целых чисел. Максимальный размер стека вводится с экрана. Создать функции для ввода и вывода элементов стека. Добавить 6 элементов. Удалить и вывести на экран 2 элемента. Задачу решить с использованием механизма указателей.
17. Создать стек для целых (положительных и отрицательных) чисел. Максимальный размер стека вводится с экрана. Создать функции для ввода и вывода элементов стека. Ввести 6 элементов. При вводе чисел в стек попадают только отрицательные элементы. Вывести все элементы стека. Задачу решить с использованием механизма указателей.
18. Создать стек для целых чисел. Максимальный размер стека вводится с экрана. Создать функции для ввода, вывода и определения размера стека. Ввести с экрана 6 элементов. Удалить 2 элемента. Вывести размер стека. Задачу решить с использованием механизма указателей.
19. Создать стек для целых (положительных и отрицательных) чисел. Максимальный размер стека вводится с экрана. Создать функции для ввода, вывода и определения размера стека. Вводить с экрана числа, причем в стек должны добавляться поочередно положительные и отрицательные числа. Задачу решить с использованием механизма указателей.
20. Создать стек для символов. Максимальный размер стека вводится с экрана. Создать функции для ввода и вывода элементов стека. Ввести эталонный символ. Вводить символы с экрана в стек до встречи эталонного. Вывести все элементы стека. Задачу решить с использованием механизма указателей.
21. Создать стек для символов. Максимальный размер стека вводится с экрана. Создать функции для ввода и вывода элементов стека. Добавить символы с экрана в стек. После добавления 5-го символа перед добавлением следуюшего удалять элемент из стека. Задачу решить с использованием механизма указателей.
22. Создать стек для символов. Максимальный размер стека вводится с экрана. Создать функции для ввода, вывода и определения размера стека. Ввести эталонный символ. Вводить символы с экрана в стек до встречи эталонного. Вывести размер стека. Задачу решить с использованием механизма указателей.
23. Создать стек для символов. Максимальный размер стека вводится с экрана. Создать функции для ввода и вывода элементов стека. Добавлять символы с экрана в стек. В случае совпадения вводимого символа с вершиной стека вытолкнуть его и распечатать ее. Задачу решить с использованием механизма указателей.
24. Создать стек для символов. Максимальный размер стека вводится с экрана. Создать функции для ввода, вывода и определения размера стека. Вводить символы с экрана в стек. В случае совпадения вводимого символа с вершиной стека вывести размер стека. Задачу решить с использованием механизма указателей.
25. Создать два стека для символов. Максимальный размер стеков вводится с экрана. Создать функции для ввода и вывода элементов стека. Вводить символы с экрана в первый стек. В случае совпадения вводимого символа с вершиной стека вводить во второй стек. Задачу решить с использованием механизма указателей.
26. Создать два стека для символов. Максимальный размер стеков вводится с экрана. Создать функции для ввода и вывода элементов стека. Вводить символы с экрана в стеки поочередно. Задачу решить с использованием механизма указателей.
27. Создать стек для символов и стек для чисел. Максимальный размер стеков вводится с экрана. Создать функции для ввода и вывода элементов стека. Вводить символы с экрана. Символ попадает в первый стек, а его численное представление – во второй. Задачу решить с использованием механизма указателей.
28. Создать два стека для символов. Максимальный размер стеков вводится с экрана. Создать функции для ввода и вывода элементов стека. Вводить символы с экрана. Прописные буквы попадают в первый стек, строчные – во второй, остальные символы пропускаются. Задачу решить с использованием механизма указателей.
29. Создать два стека для символов. Максимальный размер стеков вводится с экрана. Создать функции для ввода и вывода элементов стека. Вводить символы с экрана. Прописные буквы преобразуются в строчные и попадают в первый стек, строчные преобразуются в прописные и

попадают во второй, остальные символы пропускаются. Задачу решить с использованием механизма указателей.
30. Создать стек для целых чисел. Максимальный размер стека вводится с экрана. Создать функции для ввода и вывода элементов стека. Вводить символы с экрана. Числовое представление символа попадает в стек. Задачу решить с использованием механизма указателей.
31. Создать текстовый файл, содержащий текстовую и числовую информацию. Используя стек, создать другой текстовый файл, в котором числа были бы записаны в обратном порядке.
32. Создать текстовый файл, содержащий текстовую информацию. Используя стек, создать другой текстовый файл, в котором слова были бы записаны в обратном порядке.
33. Создать текстовый файл, содержащий некоторую информацию. Используя стек, создать другой текстовый файл, в котором строки были бы записаны в обратном порядке.
34. Создать текстовые файлы, содержащие один текстовую, а другой числовую информацию (количество слов и чисел должно быть одинаковым). Используя стек, создать другой текстовый файл, в котором числа и слова чередовались бы и были бы записаны в обратном порядке.
35. Создать текстовые файлы, содержащие один – текстовую, а другой – числовую информацию (количество слов и чисел должно быть одинаковым). Используя стек, создать другой текстовый файл, в котором числа и слова чередовались бы, а порядок чисел и слов был бы сохранен.
36. Создать текстовые файлы, содержащие один текстовую, а другой числовую информацию (количество слов и чисел может быть неодинаковым). Используя стек, создать другой текстовый файл, в котором числа и слова чередовались бы и были бы записаны в обратном порядке («лишние» числа или слова были бы записаны в конец файла).
37. В файле находится текст программы на Паскале. Используя стек, проверить правильность вложений циклов в этой программе.
38. В файле находится текст программы на Паскале. Используя стек, проверить правильность вложений операторных скобок (begin - end) в этой программе.
39. В файле записан текст, сбалансированный по круглым скобкам. Требуется для каждой пары соответствующих открывающей и закрывающей скобок напечатать номера их позиций в тексте, упорядочив пары номеров по возрастанию номеров позиций закрывающих скобок. Например, для текста a+(45-f(x)*(b-c)) надо напечатать 8 10, 12 16, 3
40. В текстовом файле без ошибок записано логическое выражение следующего вида: <лог.выр> :: = true | false | <лог.выр> and <лог.выр> | <лог.выр> or <лог.выр>. Используя стек, вычислить значение этого выражения с учетом общепринятого приоритета операций.
41. Написать программу слияния двух стеков, содержащих возрастающую последовательность целых положительных чисел, в третий стек так, чтобы его элементы располагались также в порядке возрастания.
42. Написать программу формирования стека, куда помещаются целые положительные числа, вводимые с клавиатуры (процесс ввода должен прекращаться, как только среди вводимых чисел появляется отрицательное число, после этого программа должна вывести на экран содержимое стека в том же порядке, в котором они были введены).

Лабораторная работа №8,9
Другие виды списков: очередь, циклический список. Другие виды списков: двунаправленный список, дек.

Для всех вариантов предварительно написать процедуру поиска
1. Опишите массив записей, содержащих фамилию абонента и номер его телефона. Запрограммируйте двоичный поиск в телефонном справочнике.
2. Индексом называется таблица, содержащая отсортированные значения некоторых ключей и их местоположение в массиве записей. Индексом пользуются для ускорения поиска в массиве (сам массив может быть неотсортирован). Запрограммируйте процедуру составления индекса и
2.1. Последовательного поиска при помощи индекса.
2.2. Бинарного поиска при помощи индекса.

3. Пусть задан в виде последовательности символов некоторый текст T, состоящий из слов и есть два списка из нескольких слов в виде двух массивов A и B. Написать программу, преобразующую текст T в текст S, путем замены каждого вхождения слова A[i] на соответствующее слово B[i].
4. В нашем примере поиск проводился в массиве, упорядоченном по возрастанию. Напишите процедуру бинарного поиска в массиве, упорядоченном по убыванию.
5. Дан массив из строк (например, фамилий). Отсортировать его по алфавиту и написать процедуру вставки новой фамилии после заданной так, чтобы алфавитный порядок не нарушился. Предусмотреть ситуацию, когда массив заполнен «до отказа» и вставка нового элемента невозможна.
6. Имеется железнодорожное расписание, содержащее номер рейса поезда, времена отправления и прибытия и станцию прибытия. Организовать поиск номера поезда, время отправления и прибытия, если задана станция.
7. Компания с целью определения спроса на свою продукцию организует некоторый опрос. Продукция – компакт-диски с записями шлягеров. Все опрашиваемые делятся на две группы в соответствии с полом. Каждый опрашиваемый должен назвать три песни, которые идентифицируются номерами от 1 до N (пусть N = 10). Файл с данными обрабатывается программой, которая должна печатать:
7.1. Список песен в порядке их популярности. Каждая строка содержит название песни и число упоминаний ее при опросе. Песни, которые ни разу не упоминались, в список не включаются.
7.2. Три наиболее популярные песни (хиты) и два списка (в соответствии с группами) с именами всех тех ответивших, которые поставили на первое место один из этих трех хитов.

Лабораторная работа №10, 11
Поиск на основе хеширования. Разрешение коллизий хеширования.

(4 часа)
Цель работы: Освоить на практике алгоритмы организации хеш-таблиц с открытой адресацией и хеш-таблиц с разрешением коллизий методом цепочек .
Домашнее задание:

1. Изучить алгоритмы организации хеш-таблиц с открытой адресацией, способы разрешения коллизий для таких таблиц , а так же алгоритмы поиска по ключу, добавления и удаления элемента.
2. Изучить организацию хеш-таблиц с разрешением коллизий методом цепочек.

Порядок выполнения работы.

1. Открыть проект Delphi **Structures**.
2. Добавить в управляющее главное меню пункт «Лабораторная работа №8», при выборе которого должно появляться окно модуля «Hesh» (модуль «Hesh» с формой добавить в проект).
3. Установить на форму модуля Hesh компоненты, обеспечивающие ввод исходных данных, управляющую кнопку (класса TButton или TBitBtn) и компоненты для вывода результатов на экране в соответствии с вариантом задания таблицы №8.1.
4. В обработчике события onClick управляющей кнопки на языке Object Pascal написать фрагмент программы для реализации алгоритма хеширования заданного ключа и дальнейшего поиска ключа в хеш-таблице в соответствии с вариантом задания.
5. Отладить обработчик на тестовых примерах и продемонстрировать работу приложения преподавателю.
6. произвести анализ запрограммированного алгоритма (по количеству сравнений).
7. Составить отчет и защитить работу преподавателю. В отчете обязательно представить блок-схему алгоритма решения задачи.

Таблица 1

№ вар.	Текст задачи
1.	Организовать хеш-таблицу с открытой адресацией, используя хеш-функцию h(k)=trunc(M*Frac(k*d)), где d=(sqrt(5)-1)/2, M - размер хеш-таблицы. Организовать процедуру поиска по ключу в этой хеш-таблице. Результат поиска - номер ячейки с найденным ключом или (-1).
2.	Организовать хеш-таблицу с открытой адресацией, используя процедуру поиска и вставки по ключу. Для формирования хеш-адреса использовать хеш-функцию универсального хеширования и процедуру линейного исследования для разрешения коллизии.
3.	Организовать хеш-таблицу с открытой адресацией, используя процедуру поиска и вставки по ключу. Для формирования начального хеш-адреса использовать метод деления, а затем при возникновении коллизии процедуру квадратичного исследования.
4.	Построить хеш-таблицу с открытой адресацией, используя двойное хеширование, как способ открытой адресации. Хеш-функция $h_1(k)$ и $h_2(k)$ организовать методом деления. Формирование таблицы - с помощью процедуры поиска и вставки по ключу.
5.	Организовать хеш-таблицу, используя хеш-функцию h(k) по методу умножения для формирования хеш-адреса. разрешение коллизий - методом внешних цепочек. а) Написать процедуру поиска и вставки по ключу. б) Написать процедуру удаления ключа.
6.	Организовать хеш-таблицу с помощью идеального хеширования - двухуровневая схема с универсальным хешированием на каждом уровне. Запрограммировать процедуру поиска и вставки по ключу.
7.	Организовать программно хеширование 100 записей в таблицу. состоящую из 20 ссылок на линейные списки (стеки), первоначально пустые. записи имеют неотрицательные целые ключи k<50, формируемые случайным образом. Хеш-функцию организовать методом умножения в отдельной Function.
8.	Организовать программно хеширование 50 записей в таблицу. состоящую из 10 ссылок на линейные списки (очереди), первоначально пустые. записи имеют неотрицательные целые ключи k<30, формируемые случайным образом. Хеш-функцию организовать методом деления в отдельной Function

Практические работы №1-5
Методы обменной сортировки. Методы сортировки логарифмической скорости. Сравнительный анализ методов сортировок. Внешняя сортировка. Методы слияния. Поиск во внешней памяти.

Простые алгоритмы сортировки массивов

К простым алгоритмам принято относить следующие три очевидных и несложных в реализации алгоритма, не отличающихся, к сожалению, высокой эффективностью.

Алгоритм пузырька (BubbleSort)

Идея алгоритма заключается в следующем. Сравним элементы массива с индексами 1 и 2. Если первый больше второго, то поменяем эти элементы местами. Затем таким же образом сравним (и, если нужно, переставим) элементы с индексами 2 и 3, потом 3 и 4 и т.д. После сравнения элементов (**N-1**) и **N** первый проход алгоритма завершается. Можно гарантировать, что после этого прохода максимальный элемент массива находится на последнем месте (т.е. имеет индекс **N**). На втором проходе сравниваем пары 1 и 2, 2 и 3, ... (**N-2**) и (**N-1**). Далее аналогично. После **N-1** прохода все элементы займут свои законные места.

Можно попытаться сократить число проходов, отмечая с помощью специального флажка, были ли сделаны какие-либо перестановки на очередном проходе. Если ни одной перестановки за весь проход не было, то массив, очевидно, уже отсортирован и работу алгоритма можно завершить досрочно. Однако такое усовершенствование редко дает заметный выигрыш. Нетрудно показать, что для массива, заполненного случайным образом, с вероятностью 75% все равно будут выполнены все проходы алгоритма.

Ненамного лучшие результаты дает и другая модификация алгоритма пузырька, называемая «шейкер-сортировкой». Она отличается тем, что на нечетных проходах пузырек проходит слева направо (индекс в цикле возрастает), а на четных – справо налево (индекс убывает). Таким образом, после первых двух проходов на свои места станут максимальный и минимальный элементы массива.

Алгоритм простого выбора (SelectionSort)

Идея этого алгоритма еще проще. Найдем минимальный элемент массива и поменяем его местами с первым элементом. Затем повторим ту же процедуру, начиная со второго элемента массива, затем начиная с третьего и т.д. После **N-1** проходов все элементы станут на места.

Алгоритм простых вставок (InsertionSort)

Идея заключается в следующем. Пусть к некоторому моменту работы алгоритма первые **k** элементов массива уже отсортированы, т.е. расположены по возрастанию. На очередном проходе постараемся добиться, чтобы стали отсортированными (**k+1**) элементов. Для этого запомним значение элемента (**k+1**) в рабочей переменной **R** и будем сравнивать **R** со значениями элементов **k**, (**k-1**), (**k-2**) и т.д. Если значение сравниваемого элемента больше **R**, то этот элемент перемещается на одну позицию правее. Сравнения продолжаются, пока не будет найдено место, куда должен быть помещен элемент **R** (это случится либо когда очередной сравниваемый элемент меньше или равен **R**, либо когда мы дойдем до начала массива).

Таким образом, на очередном проходе отсортированная часть массива удлиняется на 1 элемент. Начав со значения **k=1**, можно за **N-1** проход отсортировать весь массив.

Алгоритм простых вставок можно улучшить, если выбирать место для вставки **k+1**-го элемента не последовательным просмотром элементов от **k** до **1**, а бинарным поиском (т.е. сравнить **R** с элементом **j := (k+1) div 2**, затем продолжить поиск на одном из интервалов **[1..j-1]** или **[j+1..k]** и т.д.). Этот подход, называемый алгоритмом бинарных вставок, позволяет существенно сократить число сравнений, но, к сожалению, не влияет на число перестановок.

Сравнение простых алгоритмов

Все три вышеописанных алгоритма и все их модификации имеют как максимальную, так и среднюю оценку $\mathbf{O(N^2)}$, а потому используются только в случае небольших массивов или отсутствия жестких требований к времени сортировки.

Эксперименты показывают, что алгоритм пузырька является обычно наиболее медленным из трех, а алгоритмы выбора и вставок дают примерно одинаковое время сортировки.

В некоторых случаях можно ожидать, что исходный массив окажется «почти отсортированным», т.е. лишь небольшое число элементов будет нарушать порядок. Такое бывает, например, если массив ранее уже был отсортирован, но потом данные в нем подверглись небольшим модификациям. В этом случае вне конкуренции оказываются алгоритмы вставок, которые показывают очень высокую скорость на почти отсортированных массивах. Эта особенность алгоритмов вставок используется в описанном ниже алгоритме Шелла (п. 0).

Усовершенствованные алгоритмы сортировки массивов

Усложнение алгоритмов позволяет значительно повысить эффективность сортировки больших массивов. Перечислим наиболее известные алгоритмы этого класса.

Алгоритм Шелла

Разобьем элементы сортируемого массива на **h** цепочек, каждая из которых состоит из элементов, отстоящих друг от друга на расстояние **h** (здесь **h** – произвольное натуральное число). Первая цепочка будет содержать элементы с индексами **1**, **h+1**, **2h+1**, **3h+1** и т.д., вторая – **2**, **h+2**, **2h+2** и т.д., последняя цепочка – **h**, **2h**, **3h** и т.д. Отсортируем каждую цепочку как отдельный массив, используя для этого метод простых вставок. Затем выполним все вышеописанное для ряда убывающих значений **h**, причем последний раз – для **h=1**.

Очевидно, массив после этого окажется отсортированным. Неочевидно, что все проходы при **h>1** не были пустой тратой времени. Тем не менее, оказывается, что дальние переносы элементов при больших **h** настолько приближают массив к отсортированному состоянию, что на последний проход остается очень мало работы. Эксперименты показывают, что для больших значений **N** оценка среднего времени работы алгоритма примерно $\mathbf{O(N^{1.26})}$. Это значительно лучше, чем $\mathbf{O(N^2)}$ для простых алгоритмов.

Большое значение для эффективности алгоритма Шелла имеет удачный выбор убывающей последовательности значений **h**. Желательно, чтобы при соседних значениях **k** значения $\mathbf{h_k}$ не были кратны друг другу. В литературе обычно рекомендуется использовать одну из двух последовательностей: $\mathbf{h_{k+1} = 3h_k+1}$ или $\mathbf{h_{k+1} = 2h_k+1}$. В обоих случаях в качестве начального $\mathbf{h_k}$ выбирается такое значение из последовательности, при котором все сортируемые цепочки имеют длину не меньше 2. Чтобы воспользоваться, например, первой из этих формул, надо сначала положить $\mathbf{h_1:=1}$, а затем в цикле увеличивать значение **h** по формуле $\mathbf{h_{k+1}:=3*h_k+1}$, пока для очередного $\mathbf{h_k}$ не будет выполнено неравенство $\mathbf{h_k >= (N–1)\ div\ 3}$. Это значение $\mathbf{h_k}$ следует использовать на первом проходе алгоритма, а затем можно получать следующие значения по обратной формуле: $\mathbf{h_{k-1} := (h_k-1)\ div\ 3}$, вплоть до $\mathbf{h_1}$=1.

Более сложными формулами определяется последовательность Седжвика:

$$h_k = \begin{cases} 9\cdot 2^k - 9\cdot 2^{k/2} + 1, & \textit{если } k \textit{ четно}; \\ 8\cdot 2^k - 6\cdot 2^{(k+1)/2} + 1, & \textit{если } k \textit{ нечетно}. \end{cases}$$

Доказано, что при выборе $\mathbf{h_k}$ по Седжвику среднее время работы алгоритма есть $\mathbf{O(N^{7/6})}$, а максимальное – $\mathbf{O(N^{4/3})}$.

На практике удобно раз и навсегда вычислить достаточное количество членов последовательности Седжвика (вот они: 1, 5, 19, 41, 109, 209, 505, 929, 2161, 3905, 8929, 16001, 36289, 64769, 146305, 260609, 587521, 1045505, ...), затем по заданному **N** выбрать такое **k**, при котором $\mathbf{h_k >= (N–1)\ div\ 3}$, а далее в цикле выбирать значения последовательности $\mathbf{h_k}$ по убыванию **k**.

Алгоритм быстрой сортировки (QuickSort)

В основе этого алгоритма – операция ***разделения массива***. Пусть **x** – некоторый произвольно выбранный элемент сортируемого массива (разделяющий элемент). Операция разделения имеет целью переставить элементы массива таким образом, чтобы сначала шли элементы, меньшие или равные **x** (не обязательно в порядке возрастания), а затем элементы, большие или равные **x**. При этом совершенно неважно, где именно окажется после разделения сам элемент **x**.

Чтобы выполнить разделение, начнем просматривать элементы массива слева направо и остановимся на первом из элементов, большем или равном **x**. Пусть индекс этого элемента – **i**.

Аналогичным образом, начав с правого конца массива, остановимся на самом правом элементе, меньшем или равном **x** (индекс элемента – **j**). Поменяем местами элементы **i** и **j**, а затем продолжим идти вправо от **i** и влево от **j**, меняя местами пары элементов, стоящих не на месте. Разделение окончится, когда будет **i>j**.

Алгоритм быстрой сортировки можно теперь описать таким образом. Возьмем произвольный элемент массива **x** и выполним для него операцию разделения. Если левый и/или правый отрезки массива содержат более одного элемента, то выполним для каждого из этих отрезков ту же операцию, что и для всего массива (т.е. выберем произвольный элемент, выполним разделение и т.д.).

Организовать вышеописанное многократное выполнение операции разделения проще всего с помощью рекурсивного определения процедуры сортировки. Нерекурсивное описание алгоритма оказывается сложнее, т.к. при необходимости сортировать два отрезка массива, образовавшиеся после разделения, приходится явно сохранять в стеке один из отрезков (точнее, два граничных значения его индексов), пока сортируется другой. В то же время нерекурсивный вариант обычно оказывается более эффективным и, что немаловажно, он позволяет уменьшить используемую глубину стека. Для этого следует всегда из двух отрезков массива, подлежащих сортировке, заносить в стек более длинный. При этом можно гарантировать, что максимальное количество отрезков, одновременно находящихся в стеке, не может превысить $\log_2(N)$.

На эффективность алгоритма быстрой сортировки влияет также выбор разделяющего элемента **x**. Хотя теоретически **x** может выбираться произвольным образом, легко видеть, что выбор в качестве **x** первого или последнего элемента приводит к очень плохим результатам, если исходный массив оказывается уже сортированным или близким к сортированному. В связи с этим принято выбирать в качестве **x** либо элемент из середины массива, либо элемент со случайно выбранным индексом. Иногда используют чуть более сложное правило: взять первый элемент массива, последний элемент, элемент из середины массива и выбрать в качестве **x** средний по величине из этих трех.

Алгоритм **QuickSort** имеет оценку среднего времени $O(N \cdot \log(N))$ и на практике оказывается быстрее всех прочих, однако в худшем случае он может потребовать времени порядка $O(N^2)$.

В некоторых случаях может оказаться разумным скомбинировать **QuickSort** с алгоритмом пузырька. Дело в том, что достаточно громоздкая процедура разделения выполняет очень мало полезной работы в конце, когда разделяемые отрезки массива становятся короткими. Можно не разделять отрезки, длина которых не больше некоторого **L**. После этого можно «досортировать» массив, выполнив **L-1** проход «пузырька». При этом нет необходимости организовывать отдельные проходы «пузырька» для множества коротких массивов, проще пропустить «пузырек» по всему сортируемому массиву. Практика показывает, что разумные значения **L** не превышают 3 – 4.

Алгоритм пирамидальной сортировки (HeapSort)

В основе этого алгоритма лежит понятие ***пирамиды***.

Массив $A_1, A_2, \ldots A_N$ называется пирамидой, если:

$A_k \geq A_{2k}$ для всех **k** таких, что $2k \leq N$;

$A_k \geq A_{2k+1}$ для всех **k** таких, что $2k+1 \leq N$.

Пирамиду можно рассматривать как представленное в виде массива двоичное дерево, у которого значение, связанное с вершиной-родителем, не меньше, чем значения, связанные с сыновьями. При этом A_1 – корень дерева, A_2 и A_3 – его сыновья, A_4, … A_7 – внуки и т.д.

Алгоритм сортировки состоит из двух фаз: построения пирамиды и преобразования пирамиды в отсортированный массив. В основе каждой из фаз лежит операция ***просеивания*** элементов через пирамиду.

Суть операции просеивания заключается в следующем. Предположим, имеется «почти правильная» пирамида, лишь для одного из элементов которой A_k, возможно, нарушены приведенные выше неравенства. Чтобы восстановить правильность пирамиды, следует сначала сравнить между собой сыновей A_k, т.е. элементы A_{2k} и A_{2k+1}. Обозначим номер большего из этих элементов буквой **r**. Теперь следует сравнить A_r с отцом (элементом A_k) и, если неравенство

$A_k \geq A_r$ нарушено, то поменять местами A_k и A_r. Теперь оба неравенства для A_k будут выполняться, однако могут оказаться нарушенными аналогичные неравенства для A_r. Поэтому следует положить **k:=r** и повторять циклически те же действия, пока не окажется, что на очередной итерации элемент A_k либо уже не имеет сыновей, либо для него выполнены требуемые неравенства. На этом просеивание элемента A_k заканчивается.

Если представить пирамиду как двоичное дерево, то просеивание будет выглядеть как «падение» элемента A_k вдоль одной из ветвей дерева, пока он не займет подобающее ему место.

Построение пирамиды начинается с просеивания элемента, индекс которого **k := N div 2** (это самый правый из элементов, имеющих сыновей в дереве). После просеивания этого элемента отрезок, начиная с индекса **k** и до конца массива, будет соответствовать требованиям к пирамиде (потому что для элементов с номерами, большими **k**, никаких неравенств проверять не надо, у них нет сыновей!). Далее уменьшаем значение **k** на единицу и опять выполняем просеивание. Построение пирамиды будет завершено, когда будет просеян элемент A_1. Отметим, что первый элемент построенного массива-пирамиды – это его наибольший элемент.

Фаза преобразования пирамиды в отсортированный массив начинается с обмена значениями первого (наибольшего) элемента пирамиды с последним элементом. При этом наибольший элемент становится на свое окончательное место и не считается далее частью пирамиды (т.е. **N := N-1**). Сама пирамида может оказаться испорченной за счет замены корня. Чтобы восстановить пирамиду, следует таким же образом, как при построении пирамиды, еще раз просеять только новый корневой элемент A_1. Когда пирамида (уменьшившаяся на один элемент) восстановлена, ее корневой элемент – наибольший среди элементов пирамиды. Он опять обменивается с последним ее элементом, пирамида уменьшается еще на один элемент и т.д., пока пирамида не выродится в один элемент, за которым будут следовать отсортированные элементы массива.

Для алгоритма **HeapSort** имеются гарантированные оценки как максимального, так и среднего времени работы порядка **O(N·log(N))**, однако эксперименты показывают, что этот алгоритм обычно работает несколько медленнее, чем **QuickSort**, а в некоторых случаях уступает и алгоритму Шелла.

Алгоритмы слияния

Алгоритмы этой группы раньше широко использовались для сортировки последовательных файлов на магнитной ленте, однако и в задачах внутренней сортировки алгоритмы слияния показывают хорошую производительность. Их существенным недостатком является потребность во вспомогательном массиве того же размера, что и сортируемый массив.

Разнообразные алгоритмы слияния основываются на процедуре слияния двух уже отсортированных массивов в один отсортированный массив. Такая процедура реализуется легко и эффективно, за один проход по массивам. Рассмотрим два наиболее известных алгоритма сортировки путем многократных слияний.

Алгоритм простого слияния основан на следующем рассуждении. Пусть дан массив из **N** элементов. Каждый элемент исходного массива можно формально рассматривать как подмассив длины 1, причем, конечно, отсортированный. Сгруппируем попарно подмассивы из элементов с индексами 1 и 2, 3 и 4 и т.д. Будем сливать эти пары, записывая результат слияния во вспомогательный массив. (Возможен также другой вариант слияния: элементы 1 и **(N div 2)+1**, 2 и **(N div 2)+2** и т.п. Иногда это удобнее с точки зрения программирования.) В результате во вспомогательном массиве образуется **(N div 2)** сортированных подмассивов длины 2, а при нечетном **N** – еще один подмассив длины 1. Повторим операцию слияния, считая теперь источником вспомогательный массив, а приемником – исходный, и сливая подмассивы длины 2 в подмассивы длины 4 (последний подмассив опять может получиться неполным). Повторяем слияния до тех пор, пока не получится один массив длины **N**. Если для этого потребуется нечетное число проходов, то результат окажется во вспомогательном массиве и его нужно будет еще перенести в исходный массив.

Алгоритм естественного слияния исходит из того, что в исходном массиве почти наверняка уже есть отрезки, где элементы идут по возрастанию. Назовем такие отрезки ***сериями***, допуская в том числе «серии» длиной 1. Проход алгоритма состоит из двух фаз. На фазе распределения из

массива выделяются серии, причем нечетные по счету серии записываются во вспомогательный массив от его начала, а четные – от конца по убыванию индекса. На фазе слияния сливаются пары серий, одна от начала вспомогательного массива, другая от конца, и результат записывается в исходный массив от его начала. Проходы повторяются, пока не останется одна серия. При реализации этого алгоритма следует учитывать, что число сливаемых серий на фазе слияния может оказаться меньше, чем число серий, записанных на фазе распределения. Такое возможно, если две серии (например, первая и третья) сами сольются в одну, т.е. если последний элемент первой серии не больше, чем первый элемент третьей серии. Из-за этого при слиянии количество нечетных и четных серий может оказаться различным. Лишние серии, оставшиеся без пары, просто переписываются из вспомогательного массива в исходный.

Оба описанных выше алгоритма слияния имеют оценку **O(N·log(N))** как для среднего, так и для максимального времени выполнения.

Экспериментальная оценка производительности алгоритма

Оценка производительности алгоритма имеет первостепенное значение при определении пригодности этого алгоритма для решения конкретных типов задач. Важнейшей частью данной работы является сбор статистики о времени работы алгоритмов при сортировке различных массивов данных.

Трудоемкость алгоритма может измеряться либо в единицах времени, либо в единицах числа операций, наиболее характерных для алгоритмов данного типа. Для алгоритмов сортировки обычно измеряют число операций сравнения элементов массива и отдельно – число операций присваивания элементов. Остальными операциями – инициализацией и изменением счетчиков циклов, проверками окончания и т.п. – можно пренебречь, поскольку их число либо значительно меньше, чем числа сравнений и присваиваний, либо, в крайнем случае, прямо пропорционально им.

Поскольку для большей части алгоритмов трудоемкость может сильно зависеть от более удачных или менее удачных для данного алгоритма данных, для экспериментальной оценки трудоемкости чаще всего используют случайные данные. Результаты одного прогона алгоритма на случайном массиве данных не несут надежной информации, однако при увеличении числа прогонов с усреднением результатов обычно удается получить достаточно объективную оценку алгоритма. Эти вопросы подробно исследуются в теории математической статистики, здесь же можно лишь отметить, что случайная ошибка определения трудоемкости убывает обратно пропорционально квадратному корню из числа экспериментов.

Оценка трудоемкости непосредственно в единицах времени тоже является достаточно полезной, поскольку показывает, чего можно ждать от использования конкретной программы сортировки.

ОС Windows предоставляет несколько различных API-функций для измерения системного времени. Однако наиболее простая из этих функций **GetTickCount** дает достаточно низкую точность (10 мс для версий от Windows XP и выше). При этом для получения достаточно малой относительной ошибки измерений требуется, чтобы измеряемые интервалы были, по крайней мере, не меньше секунды. Таким образом, доверять можно только таким оценкам времени сортировки, которые получены для больших массивов данных, сортировка которых требует нескольких секунд. Учитывая высокую эффективность усовершенствованных алгоритмов сортировки и большую производительность современных процессоров, размеры массивов должны при этом измеряться, по крайней мере, миллионами элементов, что не всегда удобно. Можно использовать функцию **QueryPerformanceCounter**, однако она, обеспечивая высокую точность измерения времени, не позволяет отделить время данного процесса от затрат времени системой и другими приложениями.

Наиболее подходящим средством измерения времени работы алгоритма представляются API-функции **GetProcessTimes** и **GetThreadTimes**. Эти функции позволяют получить чистое процессорное время, затраченное, соответственно, всем процессом или только одной из его нитей. При этом имеется возможность отделить время работы в режиме задачи (т.е. время выполнения

прикладного кода) от времени работы в режиме системы (выполнения API-функций и т.п.). Номинальная точность измерения времени – 100 нс.

Выполнение задания

В данной лабораторной работе требуется запрограммировать и протестировать алгоритмы внутренней сортировки, указанные в варианте задания.

В каждом варианте задания требуется запрограммировать один или два (в зависимости от сложности) алгоритма сортировки и проверить их работу на ряде тестовых примеров.

В качестве тестовых массивов следует использовать:

массив из 1000 первых натуральных чисел в порядке возрастания (т.е. пример уже отсортированного массива);

массив из 1000 первых натуральных чисел в порядке убывания (т.е. пример массива, отсортированного «наоборот»);

не менее 4 сгенерированных массивов псевдослучайных чисел разного размера, от 100 до 100000 элементов (максимальный размер массива может быть изменен в зависимости от производительности процессора).

Таблица результатов должна для каждого алгоритма и для каждого тестового массива включать время работы алгоритма, число выполненных сравнений и присваиваний элементов массива.

По крайней мере, для одного не очень большого примера следует для визуальной проверки правильности работы алгоритма выдать на экран весь сортируемый массив до и после сортировки.

Отчет о работе должен включать:

постановку задачи, вариант задания, список бригады;

исходные тексты разработанных процедур сортировки (не обязательно полный текст всей программы);

итоговую таблицу результатов тестирования;

выводы, которые можно сделать по этим результатам.

Отчет и программа представляются также в электронном виде.

Варианты заданий

Запрограммировать и протестировать следующие алгоритмы, описанные в разделе **Ошибка! Источник ссылки не найден.**:

1. Нерекурсивный вариант алгоритма QuickSort со случайным выбором разделяющего
2. элемента и с выбором среднего по величине из трех элементов;
3. Алгоритм простого выбора и алгоритм HeapSort;
4. Комбинация QuickSort и пузырька для 2-3 различных значений L;
5. Алгоритм бинарных вставок и алгоритм Шелла;
6. Алгоритм простых вставок и алгоритм естественного слияния;
7. Алгоритм шейкер-сортировки и алгоритм простого слияния.

Рекомендуемая литература

Основная литература

1. Лафоре, Р. Струтктуры и алгоритмы обработки данных в C++ [Текст] / Р. Лафоре.- 4-е изд.. - Санкт-Петербург : Питер, 2014. - 928 с. : ил. - (Классика Komputer Science) - ISBN 978-5-496-00353-7. (20)

2. Васильев, А. Н. Java. Струтктуры и алгоритмы обработки данных [Текст] : для магистров и бакалавров. Базовый курс по объектно-ориентированному программированию / А. Н. Васильев. - Санкт-Петербург : Питер, 2014. - 400 с. - (Учебное пособие) - ISBN 978-5-496-00044-4. (15)

Дополнительная литература

1. Павловская, Т.А. С++. Струткуры и алгоритмы обработки данных: практикум / Павловская, Т.А. . - СПб. : Питер, 2006. - 265с. : ил. (5)

2. Кирнос, В.Н. Информатика II. Основы алгоритмизации и программирования на языке С++ : учебно-методическое пособие / В.Н. Кирнос ; Министерство образования и науки Российской Федерации, Томский Государственный Университет Систем Управления и Радиоэлектроники (ТУСУР). - Томск : Эль Контент, 2013. - 160 с. : ил.,табл., схем. - ISBN 978-5-4332-0068-5 ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=208651.

Периодические издания

1. Журнал «Вестник компьютерных и информационных технологий »
2. Журнал «Информационные технологии и вычислительные системы»
3. Журнал «Стандарты и качество»
4. Журнал «Прикладная информатика»