

**МИНОБРНАУКИ РОССИИ**

**федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технологический университет «СТАНКИН»**

**(ФГБОУ ВО «МГТУ «СТАНКИН»)**

Институт цифровых интеллектуальных систем  
Кафедра компьютерных систем управления

Направление подготовки: 15.03.04 Автоматизация технологических процессов и производств

**Отчет по лабораторной работе №1**

**по дисциплине «Программирование и Алгоритмизация»**

**Тема: «**Алгоритмы сортировки и поиска**»**

Выполнил:

студент гр. АДБ-22-06 \_\_\_\_\_\_\_\_\_\_\_ Голубев С.Г.

(дата) (подпись)

Принял:

к.т.н. \_\_\_\_\_\_\_\_\_\_\_ Бакулев Х.С.

(дата) (подпись)

**Москва 2023**

Оглавление

[Задание 1 «Алгоритмы сортировки» 3](#_Toc146819057)

[Суть задания 3](#_Toc146819058)

[Код программы 3](#_Toc146819059)

[Описание выполнения задания: 4](#_Toc146819060)

[Вывод 5](#_Toc146819061)

[Задание 2 «Алгоритмы поиска» 6](#_Toc146819062)

[Суть задания 6](#_Toc146819063)

[Код программы 6](#_Toc146819064)

[Описание выполнения задания: 7](#_Toc146819065)

[Вывод: 8](#_Toc146819066)

[Задание 3 «Телефонная книга» 9](#_Toc146819067)

[Суть задания 9](#_Toc146819068)

[Код программы 9](#_Toc146819069)

[Описание выполнения задания 11](#_Toc146819070)

[Вывод 11](#_Toc146819071)

[Приложения 12](#_Toc146819072)

[Приложение 1 12](#_Toc146819073)

[Приложение 2 13](#_Toc146819074)

[Приложение 3 14](#_Toc146819075)

[Приложение 4 15](#_Toc146819076)

[Приложение 5 16](#_Toc146819077)

[Приложение 6 17](#_Toc146819078)

[Приложение 7 18](#_Toc146819079)

[Приложение 8 19](#_Toc146819080)

[Приложение 9 20](#_Toc146819081)

[Приложение 10 21](#_Toc146819082)

# Задание 1 «Алгоритмы сортировки»

Суть задания: освоить разные способы сортировки массивов и определить/сравнить их эффективность(быстродействие).

Код программы:

#include <iostream>

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

#include <chrono>

using namespace std;

using namespace chrono;

int\* createArr(int size) {

int\* arr = new int[size];

srand(steady\_clock::now().time\_since\_epoch().count()); // инициализация системы псевдослучайных чисел для rand()

for (int i = 0; i < size; i++) {

arr[i] = rand(); // от 0 до 32767

}

return arr;

}

void printArr(int size, int\* arr) {

for (int i = 0; i < size; i++) {

cout << arr[i] << " ";

}

cout << endl;

}

void shakerSwap(int a[], int i) {

int buff = a[i];

a[i] = a[i - 1];

a[i - 1] = buff;

}

void shakerSort(int n, int a[])

{

int leftMark = 1;

int rightMark = n - 1;

while (leftMark <= rightMark)

{

for (int i = rightMark; i >= leftMark; i--

)

if (a[i - 1] > a[i]) shakerSwap(a, i);

leftMark++;

for (int i = leftMark; i <= rightMark;

i++)

if (a[i - 1] > a[i]) shakerSwap(a, i);

rightMark--;

}

}

void insertSort(int n, int a[]) {

int buff = 0, j;

for (int i = 1; i < n; i++)

{

buff = a[i];

for (j = i - 1;

j >= 0 && a[j] < buff; j--)

a[j + 1] = a[j];

a[j + 1] = buff;

}

}

int main() {

setlocale(LC\_ALL, "Russian");

int size;

cout << "Введите размер массива: ";

cin >> size;

int\* fromTrashToGold = createArr(size);

cout << "Созданный массив: ";

//printArr(size, fromTrashToGold);

cout << "Шейкер-сортировка: ";

time\_point<steady\_clock> start = steady\_clock::now();

//shakerSort(size, fromTrashToGold);

//printArr(size, fromTrashToGold);

cout << "Сортировка вставками: ";

insertSort(size, fromTrashToGold);

//printArr(size, fromTrashToGold);

time\_point<steady\_clock> fin = steady\_clock::now();

long dur = duration\_cast<milliseconds>(fin - start).count();

cout << dur;

delete[] fromTrashToGold;

return 0;

}

## Описание выполнения задания:

1. Создал в программе visual studio 2022 новое консольное приложение
2. Переименовал автоматический созданный файл, дублирующий имя проекта в “first.cpp” и удалил лишние строки/комментарии в нем.
3. Создал функцию для генерации массивов “createArr”
   1. Функция принимает входящее значение – размер массива
4. Создал функцию для вывода элементов массива “printArr”
   1. Функция принимает входящий массив и его размер
   2. Функция перебирает простым циклом счетчиком “for” все элементы пока массив не закончится, т.е. пока индекс элемента меньше размера массива
5. Реализовал функцию для алгоритма сортировки чисел по возрастанию согласно моему варианту “5” (Шейкер-сортировка)
   1. Использовал кусок кода из приложения А
6. Работаю с функцией main/проверяю работоспособность сортировки
   1. Создаю массив “fromTrashToGold”, вызвав функцию “createArr” с параметром 20
   2. Вывожу элементы массива фунцией “printArr”
   3. Сортирую массив шейкер сортировкой соответствующей функцией
   4. Вновь вывожу массив
   5. Проверяю отсортировались ли элементы по возрастанию? Да, отсортировались
7. Замеряю скорость выполнения сортировки с помощью разницы 2-х “time\_point”.
8. Получаемые замеры = 0. Проверяю результаты для 10 000 000 элементов. ПК не может их обработать в течение 10 мин….
9. Исходя из пункта (8), экспериментальным путём снижаю кол-во элементов в массиве до 100 000. Провожу 10 измерений, занося результаты в таблицу exel, считаю среднее время выполнения алгоритма (**25914,9 мс**)

См. [Приложение 1](#_Приложение_1)

1. Провожу аналогичные замеры для разного числа элементов в массиве, инструментами exel создаю график зависимости среднего времени выполнения от кол-ва элементов.

См. [Приложение 2](#_Приложение_2), [Приложение 3](#_Приложение_3)

1. Реализую функцию для алгоритма сортировки следующего за моим вариантом “6” (Сортировка простыми вставками)
   1. Использую готовую функцию из приложения А
   2. Меняю “a[j] > buff;” на “<” для сортировки по убыванию
2. Провожу замеры из пункта (10) для новой сортировки

См. [Приложение 4](#_Приложение_4), [Приложение 5](#_Приложение_5)

1. Пример выполнения

См. [Приложение 6](#_Приложение_6)

Вывод:

Узнал несколько новых сортировок и научился с ними работать.

В сравнении шейкер сортировки и сортировки простыми вставками, вторая явно выигрывает в скорости.

Хорошее поведение: массив уже упорядочен. В этом случае алгоритм будет выполнять только один проход в обе стороны массива без каких-либо обменов элементов. O(n)

Худшее поведение: массив в обратном порядке. O(n^2)

# Задание 2 «Алгоритмы поиска»

Суть задания: освоить разные способы поиска элементов в массивах и определить/сравнить их эффективность(быстродействие).

Код программы:

#include <iostream>

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

#include <chrono>

#include <algorithm>

using namespace std;

using namespace chrono;

int\* createArr(long size) {

int\* arr = new int[size];

srand(steady\_clock::now().time\_since\_epoch().count()); // инициализация си-стемы псевдослучайных чисел для rand()

for (int i = 0; i < size; i++) {

arr[i] = rand(); // от 0 до 32767

}

return arr;

}

void printArr(long size, int\* arr) {

for (int i = 0; i < size; i++) {

cout << arr[i] << " ";

}

cout << endl;

}

int enumSearch(int val, long size, int a[])

{

for (int i = 0; i < size; i++)

{

if (a[i] == val)

return i;

}

return -1;

}

int binarySearch(int val, long size, int

a[])

{

int lastIndex = size - 1;

int firstIndex = 0;

while (lastIndex >= firstIndex)

{

int middleIndex = (firstIndex +

lastIndex) / 2;

if (a[middleIndex] == val)

{

return middleIndex;

}

else

if (a[middleIndex] < val)

firstIndex = middleIndex + 1;

else

if (a[middleIndex] > val)

lastIndex = middleIndex - 1;

}

return -1;

}

int main() {

setlocale(LC\_ALL, "Russian");

long size = 1000000000;

int requiredNum;

int\* fromTrashToGold = createArr(size);

cout << "Значение, которое необходимо найти: ";

cin >> requiredNum;

// enumSearch

time\_point<steady\_clock> start1 = steady\_clock::now();

int enumResult = enumSearch(requiredNum, size, fromTrashToGold);

time\_point<steady\_clock> fin1 = steady\_clock::now();

long dur1 = duration\_cast<milliseconds>(fin1 - start1).count();

// Сортировка для бинарного поиска

time\_point<steady\_clock> start3 = steady\_clock::now();

sort(fromTrashToGold, fromTrashToGold + size);

time\_point<steady\_clock> fin3 = steady\_clock::now();

long dur3 = duration\_cast<milliseconds>(fin3 - start3).count();

// binarySearch

time\_point<steady\_clock> start2 = steady\_clock::now();

int binaryResult = binarySearch(requiredNum, size, fromTrashToGold);

time\_point<steady\_clock> fin2 = steady\_clock::now();

long dur2 = duration\_cast<milliseconds>(fin2 - start2).count();

// Результаты

if (enumResult != -1) {

cout << "Позиция значения в enumSearch: " << enumResult << endl;

}

else {

cout << "Значение не найдено в enumSearch" << endl;

}

if (binaryResult != -1) {

cout << "Позиция значения в binarySearch: " << binaryResult << endl;

}

else {

cout << "Значение не найдено в binarySearch" << endl;

}

cout << "Время выполнения enumSearch: " << dur1 << " мс" << endl;

cout << "Время выполнения binarySearch: " << dur2 << " мс" << endl;

cout << "Время выполнения sort: " << dur3 << " мс" << endl;

delete[] fromTrashToGold;

return 0;

}

## Описание выполнения задания:

1. В том же консольном проекте, на основе того же файла создаю файл “second.cpp”.
   1. Удаляю лишнее
   2. Из приложения Б копирую вставляю линейный перебор.
2. Чуть меняю main
   1. Запрашиваю через cin элемент позиция которого необходимо найти.
   2. Добавляю условия для вывода ответа
3. Проверяю работоспособность программы
4. Немного отклоняюсь от требуемого хода работы в методичке и реализую функцию для алгоритма поиска согласно варианту “1” (Бинарный поиск), посредством копирования кода из Приложения Б
5. Провожу последовательные замеры для линейного и бинарного поиска в одной программе
   1. Сначала замеряю время выполнения линейного поиска
   2. Далее сортиру массив функцией “sort” из библиотеки “algorithm” для бинарного поиска
   3. Замеряю бинарный поиск
   4. Вывожу время выполнения в консоль из которой заношу данные в таблицу exel

Результаты замеров: спустя 10 замеров на массивах из 1 00 000 000 элементов всегда получаю скорость выполнения 0 в обоих случаях. => Пришел к выводу, что имеет смысл проверять только те случаи, когда элементы не находятся в массиве. Тогда у бинарного поиска по прежнему время выполнения 0 мс, а вот у линейного результаты слудющие.

См [Приложение 7](#_Приложение_7), [Приложение 8](#_Приложение_8)

1. Пример выполнения программы

См. [Приложение 9](#_Приложение_9)

## Вывод:

Узнал несколько новых альтернатив линейному поиску и научился с ними работать.

В сравнении линейного и бинарного поисков, бинарный выигрывает в скорости. Особенно это заметно в случае, когда необходимого элемента нет в массиве.

Также на относительно небольшом объеме данных разница на мощном железе не заметна, а выполнение качественной тестировки затрудняет долгая генерацию массива.

Нужно отметить, что для реализации бинарного поиска необходима предварительная сортировка массива, что забирает гораздо больше времени, чем весь линейный поиск и усложняет код.

# Задание 3 «Телефонная книга»

Суть задания: Создать список из фамилий и соответствующих им телефонных номеров формата «0-00», где 0 — случайная цифра, и номеров кабинетов (номера не повторяются). Реализовать функции сортировки по фамилиям и по номерам кабинетов, а также функцию поиска номера по фамилии или кабинету.

Код программы:

#include <iostream>

#include <string>

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

#include <chrono>

#include <set>

using namespace std;

using namespace chrono;

struct Record {

string surname;

string phone;

int roomNumber;

};

const int MAX\_RECORDS = 10;

set<int> usedRoomNumbers;

int generateRoomNumber() {

int roomNumber;

do {

roomNumber = rand() % 100 + 1;

} while (usedRoomNumbers.count(roomNumber) > 0);

usedRoomNumbers.insert(roomNumber);

return roomNumber;

}

Record generateRandomRecord(int i) {

Record record;

record.surname = "Surname" + to\_string(i);

record.phone = to\_string(rand() % 10) + "-" + to\_string(rand() % 10) + to\_string(rand() % 10);

record.roomNumber = generateRoomNumber();

return record;

}

void printRecord(const Record& record) {

cout << "Имя: " << record.surname << " Телефон: " << record.phone << " Номер комнаты: " << record.roomNumber << endl;

}

void shakerSwap(Record records[], int i) {

Record buff = records[i];

records[i] = records[i - 1];

records[i - 1] = buff;

}

void sortBySurname(Record records[], int size) {

int leftMark = 1;

int rightMark = size - 1;

while (leftMark <= rightMark) {

for (int i = rightMark; i >= leftMark; i--) {

if (records[i - 1].surname > records[i].surname) {

shakerSwap(records, i);

}

}

leftMark++;

for (int i = leftMark; i <= rightMark; i++) {

if (records[i - 1].surname > records[i].surname) {

shakerSwap(records, i);

}

}

rightMark--;

}

}

void sortByRoomNumber(Record records[], int size) {

int leftMark = 1;

int rightMark = size - 1;

while (leftMark <= rightMark) {

for (int i = rightMark; i >= leftMark; i--) {

if (records[i - 1].roomNumber > records[i].roomNumber) {

shakerSwap(records, i);

}

}

leftMark++;

for (int i = leftMark; i <= rightMark; i++) {

if (records[i - 1].roomNumber > records[i].roomNumber) {

shakerSwap(records, i);

}

}

rightMark--;

}

}

void searchRecord(Record records[], int size, string& key) {

bool found = false;

for (int i = 0; i < size; i++) {

if (records[i].surname == key or to\_string(records[i].roomNumber) == key) {

printRecord(records[i]);

found = true;

}

}

if (!found) {

cout << "Запись не найдена" << endl;

}

}

int main() {

int c;

setlocale(LC\_ALL, "Russian");

srand(steady\_clock::now().time\_since\_epoch().count());

Record records[MAX\_RECORDS];

for (int i = 0; i < MAX\_RECORDS; i++) {

records[i] = generateRandomRecord(i);

}

cout << "Введите команду:" << endl;

cout << "1 - Сортировка по фамилиям" << endl;

cout << "2 - Сортировка по комнатам" << endl;

cout << ">";

cin >> c;

if (c == 1) {

sortBySurname(records, MAX\_RECORDS);

cout << "Сортировка по фамилиям:" << endl;

}

else if (c == 2) {

sortByRoomNumber(records, MAX\_RECORDS);

cout << "Сортировка по комнатам:" << endl;

}

else {

cout << "Неверное значение!" << endl;

cout << "..." << endl;

cout << "Вывод всех записей" << endl;

}

for (int i = 0; i < MAX\_RECORDS; i++) {

printRecord(records[i]);

}

cout << "Введите фамилию или номер комнаты для поиска: ";

string searchKey;

cin >> searchKey;

cout << "Результаты поиска:" << endl;

searchRecord(records, MAX\_RECORDS, searchKey);

return 0;

}

Описание выполнения задания:

1. Первое, что я сделал – это создал структуру записи, состоящую из полей фамилии, телофона и номера комнаты.
2. Функция создания записей, в которой автоматически генерацией заполняются поля
3. Из этой функции я вынес отдельную функцию генерации номера комнаты, для того чтобы он не повторялся. Реализовал это с помощью set и цикла while (генерирует, пока в set не будет такого числа)
4. Далее написал две функции сортировки из предыдущих заданий
5. Написал функцию линейного поиска
6. Пример выполнения программы

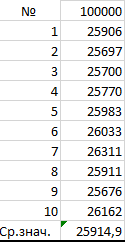
См. [Приложение 10](#_Приложение_10)

Вывод:

Индивидуальное задание закрепило полученные знания работы с поиском и сортировкой и наглядно показало, для каких задач эти навыки пригодятся.

# Приложения

## Приложение 1



## Приложение 2

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | Количество элементов в массиве | | | | | | | |
| 100000 | 75000 | 50000 | 25000 | 10000 | 7500 | 5000 | 1000 |
| 1 | 25906 | 14452 | 6547 | 1604 | 250 | 140 | 61 | 2 |
| 2 | 25697 | 14457 | 6471 | 1608 | 251 | 142 | 60 | 2 |
| 3 | 25700 | 14452 | 6454 | 1604 | 255 | 139 | 60 | 2 |
| 4 | 25770 | 14453 | 6414 | 1607 | 250 | 138 | 60 | 2 |
| 5 | 25983 | 14504 | 6464 | 1615 | 252 | 138 | 61 | 2 |
| 6 | 26033 | 14413 | 6416 | 1605 | 247 | 138 | 61 | 2 |
| 7 | 26311 | 14484 | 6433 | 1604 | 249 | 142 | 60 | 2 |
| 8 | 25911 | 14445 | 6487 | 1595 | 250 | 139 | 61 | 2 |
| 9 | 25676 | 14488 | 6441 | 1605 | 249 | 139 | 61 | 2 |
| 10 | 26162 | 14461 | 6445 | 1609 | 255 | 141 | 61 | 2 |
| Ср.знач. | 25914,9 | 14460,9 | 6457,2 | 1605,6 | 250,8 | 139,6 | 60,6 | 2 |

## Приложение 3

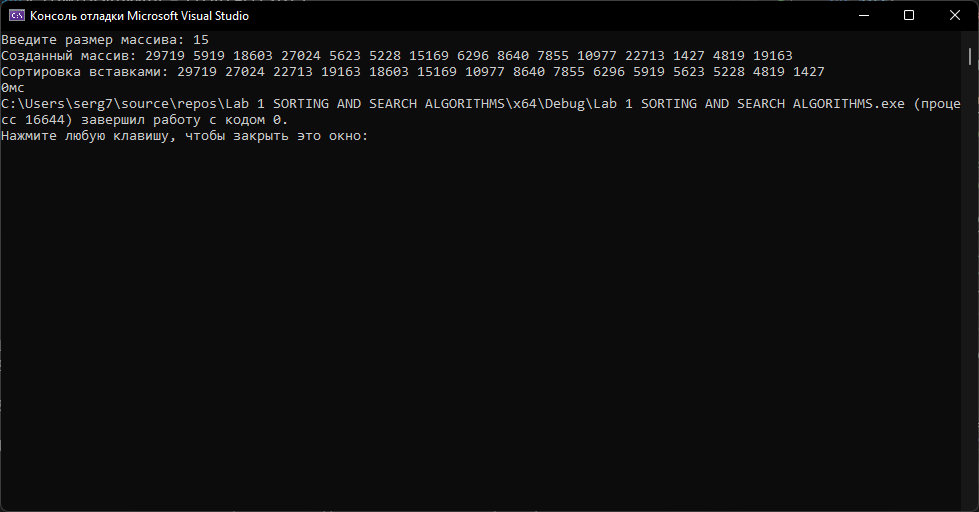


## Приложение 4

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | Количество элементов в массиве | | | | | | | |
| 100000 | 75000 | 50000 | 25000 | 10000 | 7500 | 5000 | 1000 |
| 1 | 4955 | 2770 | 1237 | 312 | 51 | 28 | 12 | 0 |
| 2 | 4931 | 2768 | 1249 | 312 | 50 | 28 | 12 | 0 |
| 3 | 4980 | 2795 | 1229 | 310 | 51 | 28 | 13 | 0 |
| 4 | 4932 | 2797 | 1240 | 311 | 51 | 28 | 12 | 0 |
| 5 | 4943 | 2778 | 1224 | 315 | 50 | 28 | 13 | 0 |
| 6 | 4981 | 2787 | 1229 | 312 | 49 | 29 | 13 | 0 |
| 7 | 4942 | 2777 | 1230 | 316 | 52 | 28 | 12 | 0 |
| 8 | 4947 | 2786 | 1237 | 315 | 51 | 28 | 12 | 0 |
| 9 | 5002 | 2768 | 1238 | 314 | 51 | 30 | 12 | 0 |
| 10 | 4938 | 2802 | 1243 | 313 | 50 | 28 | 12 | 0 |
| Ср.знач. | 4955,1 | 2782,8 | 1235,6 | 313 | 50,6 | 28,3 | 12,3 | 0 |

## Приложение 5

## Приложение 6

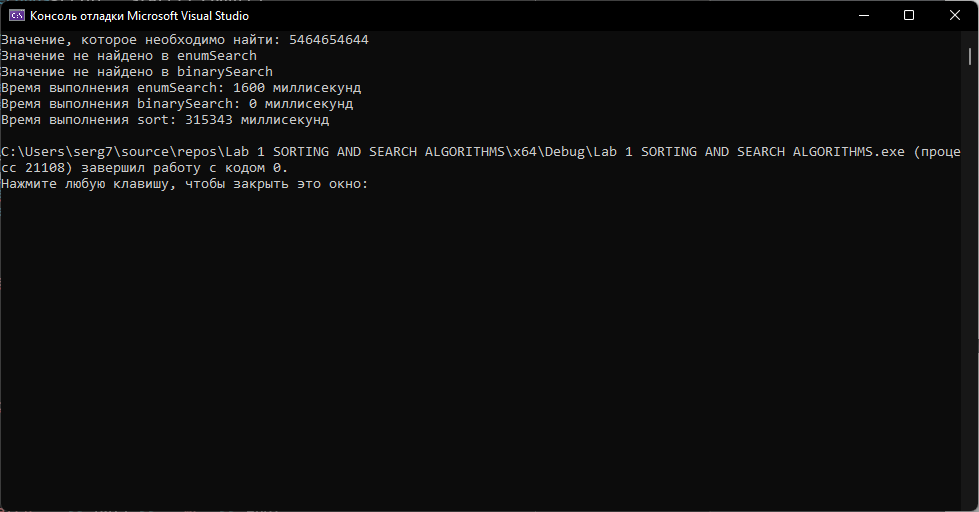


## Приложение 7

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Количество элементов в массиве | | | | | |
| 1000000000 | 150000000 | 100000000 | 50000000 | 10000000 | 750000 |
| 1 | 1559 | 231 | 153 | 78 | 20 | 2 |
| 2 | 1551 | 231 | 152 | 77 | 18 | 1 |
| 3 | 1556 | 230 | 155 | 76 | 20 | 2 |
| 4 | 1550 | 228 | 153 | 76 | 23 | 2 |
| 5 | 1551 | 230 | 153 | 78 | 16 | 2 |
| 6 | 1579 | 232 | 155 | 78 | 24 | 1 |
| 7 | 1553 | 231 | 154 | 76 | 16 | 1 |
| 8 | 1560 | 225 | 155 | 76 | 17 | 3 |
| 9 | 1554 | 240 | 154 | 77 | 15 | 1 |
| 10 | 1556 | 226 | 153 | 76 | 19 | 2 |
| Ср.знач. | 1556,9 | 230,4 | 153,7 | 76,8 | 18,8 | 1,7 |

## Приложение 8

## Приложение 9



## Приложение 10

