МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №4

по дисциплине: Алгоритмы и структуры данных
тема: «Сравнительный анализ алгоритмов поиска (Pascal/C)»

Выполнил: студент группы ПВ-223
Мелехов Артём Дмитриевич

Проверил:
асс. Солонченко Роман Евгеньевич

Белгород 2023 г.

Лабораторная работа №4 «Сравнительный анализ алгоритмов поиска (Pascal/C)»

Цель работы: изучение алгоритмов поиска элемента в массиве и закрепление навыков в проведении сравнительного анализа алгоритмов.

Содержание отчета:

- Тема лабораторной работы;

- Цель лабораторной работы;

- Условия задач и их решение;

- Вывод.

Листинг программы:

Файл `searches/searches.c`

```c
//
// Created by Artyom on 19.10.2023.
//

#include "searches.h"

long long linear_search(const long long const *arr, const size_t
size, const long long x) {
    for (long long i = 0; i < size; i++)
        if (arr[i] == x)
            return i;

    return -1;
}

long long fast_linear_search(long long *arr, const size_t size,
const long long x) {
    arr[size] = x;

    long long i = 0;

    while (arr[i] != x)
        i++;

    return i != size ? i : -1;
}
```

```
long long fast_linear_search_for_a_sorted_array(long long *arr,
const size_t size, const long long x) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == x)
            return i;

        if (arr[i] > x)
            return -1;
    }

    return -1;
}

long long binary_search_in_a_subarray(long long *arr, long long
left, long long right,
                                      const long long x) {
    if (right >= left) {
        long long mid = left + (right - left) / 2;

        // Если элемент находится в середине
        if (arr[mid] == x)
            return mid;

        // Если элемент меньше, чем mid, то он может быть только
в левом подмассиве
        if (arr[mid] > x)
            return binary_search_in_a_subarray(arr, left, mid -
1, x);

        // В противном случае элемент может быть только в правом
подмассиве
        return binary_search_in_a_subarray(arr, mid + 1, right,
x);
    }

    // Возвращаем -1, если элемент не найден
    return -1;
}

long long binary_search(long long *arr, const size_t size, const
long long x) {
    return binary_search_in_a_subarray(arr, 0, size - 1, x);
}
```

```
long long block_search(long long *arr, const size_t size, const
long long x) {
    // Вычисляем размер блока для поиска
    long long block_size = sqrt(size);

    // Находим блок, в котором находится искомый элемент
    long long i;
    for (i = 0; i < size; i += block_size)
        if (arr[i] > x)
            break;

    // Выполняем линейный поиск в найденном блоке
    for (long long j = i - block_size; j < i; ++j)
        if (arr[j] == x)
            return j;

    // Возвращаем -1, если элемент не найден
    return -1;
}
```

Файл searches/searches.h

```
//
// Created by Artyom on 19.10.2023.
//

#ifndef ALGORITMS_AND_DS_SEARCHES_H
#define ALGORITMS_AND_DS_SEARCHES_H

#include <stdio.h>
#include <math.h>

// Линейный поиск элемента x в массиве arr размера size
long long linear_search(const long long const *arr, const size_t
size, const long long x);

// Быстрый линейный поиск элемента x в массиве arr размера size
long long fast_linear_search(long long *arr, const size_t size,
const long long x);

// Быстрый линейный поиск элемента x в отсортированном массиве
arr размера size. Массив должен
// быть упорядочен
long long fast_linear_search_for_a_sorted_array(long long *arr,
const size_t size, const long long x);

// Бинарный поиск элемента x в подмассиве массива arr от
элемента arr[left] до элемента arr[right].
// Подмассив должен быть упорядочен
long long binary_search_in_a_subarray(long long *arr, long long
left, long long right,
                                      const long long x);
```

```
// Бинарный поиск элемента x в массиве arr размера size. Массив должен быть упорядочен
long long binary_search(long long *arr, const size_t size, const long long x);

// Блочный поиск элемента x в массиве arr размера size. Массив должен быть отсортирован
long long block_search(long long *arr, const size_t size, const long long x);

#endif //ALGORITMS_AND_DS_SEARCHES_H
```

Файл lab4/test_searches/test_searches.c

```
//
// Created by Artyom on 19.10.2023.
//

#include "test_searches.h"

void test_linear_search() {
    printf("Linear search test...\n");

    // Генерация рандомного массива из остатков от деления рандомного числа на 100
    size_t size = 30;
    long long *arr1 = (long long *) malloc(sizeof(long long) * size);
    generate_random_array(arr1, size);

    // Остаток не может быть отрицательным
    assert(linear_search(arr1, size, -1) == -1);
    free(arr1);
    printf("TEST 1 OK!\n");

    long long arr2[] = {88, 72, 95, 56, 1, 45, 95, 27, 6, 96,
                        95, 27, 92, 9, 66, 28, 87, 61, 40,
                        84, 76, 81, 35, 80, 49, 75, 29, 90,
                        74, 5};

    assert(linear_search(arr2, size, 1) == 4);
    printf("TEST 2 OK!\n");

    printf("\n");
}
```

```
void test_fast_linear_search() {
    printf("Fast linear search test...\n");

    // Генерация рандомного массива из остатков от деления
рандомного числа на 100
    size_t size = 30;
    long long *arr1 = (long long *) malloc(sizeof(long long) *
size);
    generate_random_array(arr1, size);

    // Остаток не может быть отрицательным
    assert(fast_linear_search(arr1, size, -1) == -1);
    free(arr1);
    printf("TEST 1 OK!\n");

    long long arr2[] = {88, 72, 95, 56, 1, 45, 95, 27, 6, 96,
                        95, 27, 92, 9, 66, 28, 87, 61, 40,
                        84, 76, 81, 35, 80, 49, 75, 29, 90,
                        74, 5};

    assert(fast_linear_search(arr2, size, 1) == 4);
    printf("TEST 2 OK!\n");

    printf("\n");
}

void test_fast_linear_search_for_a_sorted_array() {
    printf("Fast linear search for a sorted array test...\n");

    size_t size = 30;
    long long arr1[] = {1, 2, 3, 5, 7, 8, 10, 12, 14, 15,
                        16, 17, 18, 20, 21, 22, 23, 24,
                        25, 26, 27, 28, 29, 30, 31, 32,
                        33, 34, 35, 36};

    assert(fast_linear_search_for_a_sorted_array(arr1, size, -1)
== -1);

    printf("TEST 1 OK!\n");

    long long arr2[] = {1, 5, 6, 9, 27, 27, 28, 29, 35,
                        40, 45, 49, 56, 61, 66, 72, 74,
                        75, 76, 80, 81, 84, 87, 88, 90,
                        92, 95, 95, 95, 96};

    assert(fast_linear_search_for_a_sorted_array(arr2, size, 1)
== 0);
    printf("TEST 2 OK!\n");

    printf("\n");
}
```

```
void test_binary_search() {
    printf("Binary search test...\n");

    size_t size = 30;
    long long arr1[] = {1, 2, 3, 5, 7, 8, 10, 12, 14, 15,
                        16, 17, 18, 20, 21, 22, 23, 24,
                        25, 26, 27, 28, 29, 30, 31, 32,
                        33, 34, 35, 36};

    assert(binary_search(arr1, size, -1) == -1);

    printf("TEST 1 OK!\n");

    long long arr2[] = {1, 5, 6, 9, 27, 27, 28, 29, 35,
                        40, 45, 49, 56, 61, 66, 72, 74,
                        75, 76, 80, 81, 84, 87, 88, 90,
                        92, 95, 95, 95, 96};

    assert(binary_search(arr2, size, 1) == 0);
    printf("TEST 2 OK!\n");

    printf("\n");
}

void test_block_search() {
    printf("Block search test...\n");

    size_t size = 30;
    long long arr1[] = {1, 2, 3, 5, 7, 8, 10, 12, 14, 15,
                        16, 17, 18, 20, 21, 22, 23, 24,
                        25, 26, 27, 28, 29, 30, 31, 32,
                        33, 34, 35, 36};

    assert(block_search(arr1, size, -1) == -1);

    printf("TEST 1 OK!\n");

    long long arr2[] = {1, 5, 6, 9, 27, 27, 28, 29, 35,
                        40, 45, 49, 56, 61, 66, 72, 74,
                        75, 76, 80, 81, 84, 87, 88, 90,
                        92, 95, 95, 95, 96};

    assert(block_search(arr2, size, 1) == 0);
    printf("TEST 2 OK!\n");

    printf("\n");
}
```

Файл lab4/test_searches/test_searches.h

```
//
// Created by Artyom on 19.10.2023.
//

#ifndef ALGORITMS_AND_DS_TEST_SEARCHES_H
#define ALGORITMS_AND_DS_TEST_SEARCHES_H

#include <assert.h>

#include "../../searches/searches.h"
#include "../../standart_functions/standart_functions.h"
#include "../../sorts/sorts.h"

void test_linear_search();

void test_fast_linear_search();

void test_fast_linear_search_for_a_sorted_array();

void test_binary_search();

void test_block_search();

#endif //ALGORITMS_AND_DS_TEST_SEARCHES_H
```

Файл lab4/searches_for_tcf/searches_for_tcf.c

```
//
// Created by Artyom on 19.10.2023.
//

#include "searches_for_tcf.h"

long long linear_search_for_tcf(long long *arr, const size_t
size, const long long x) {
    long long number_of_comparisons = 0;

    for (long long i = 0; i < size; i++) {
        number_of_comparisons += 2;

        if (arr[i] == x)
            return number_of_comparisons;
    }

    return number_of_comparisons + 1;
}
```

```
long long fast_linear_search_for_tcf(long long *arr, const
size_t size, const long long x) {
    arr[size] = x;

    long long i = 0;
    long long number_of_comparisons = 1;

    while (arr[i] != x) {
        number_of_comparisons++;
        i++;
    }

    return number_of_comparisons + 1;
}

long long fast_linear_search_for_a_sorted_array_for_tcf(long
long *arr, const size_t size,
                                                        const
long long x) {
    long long number_of_comparisons = 1;

    for (int i = 0; i < size; i++) {
        if (arr[i] == x)
            return number_of_comparisons + 2;

        if (arr[i] > x)
            return number_of_comparisons + 3;

        number_of_comparisons += 3;
    }

    return number_of_comparisons;
}
```

```
long long binary_search_in_a_subarray_for_tcf(long long *arr,
long long left, long long right,
                                              const long long x)
{
    long long max_index = right;
    long long number_of_comparisons = 1;

    while (right - left > 1) {
        long long middle = left + (right - left) / 2;

        number_of_comparisons += 2;

        if (arr[middle] > x)
            right = middle;
        else
            left = middle;
    }

    return number_of_comparisons + 1;
}

long long binary_search_for_tcf(long long *arr, const size_t
size, const long long x) {
    return binary_search_in_a_subarray_for_tcf(arr, -1, size,
x);
}

long long block_search_for_tcf(long long *arr, const size_t
size, const long long x) {
    long long number_of_comparisons = 1;

    if (arr[0] > x)
        return number_of_comparisons;

    long long block = sqrt(size);
    long long i = 0;

    number_of_comparisons++;

    while (i < size) {
        number_of_comparisons += 2;

        if (arr[i] > x) {
            break;
        }

        i += block;
    }

    return number_of_comparisons +
binary_search_in_a_subarray_for_tcf(arr, i - block - 1, i, x);
}
```

Файл lab4/searches_for_tcf/searches_for_tcf.h

```c
//
// Created by Artyom on 19.10.2023.
//

#ifndef ALGORITMS_AND_DS_SEARCHES_FOR_TCF_H
#define ALGORITMS_AND_DS_SEARCHES_FOR_TCF_H

#include <stdio.h>
#include <math.h>

// Возвращает количество операций сравнения проведённых в функции linear_search
long long linear_search_for_tcf(long long *arr, const size_t size, const long long x);

// Возвращает количество операций сравнения проведённых в функции fast_linear_search
long long fast_linear_search_for_tcf(long long *arr, const size_t size, const long long x);

// Возвращает количество операций сравнения проведённых в
// функции fast_linear_search_for_a_sorted_array_for_tcf
long long fast_linear_search_for_a_sorted_array_for_tcf(long long *arr, const size_t size,
                                                        const long long x);

// Возвращает количество операций сравнения проведённых в функции binary_search_in_a_subarray
long long binary_search_in_a_subarray_for_tcf(long long *arr, long long left, long long right,
                                              const long long x);

// Возвращает количество операций сравнения проведённых в функции binary_search
long long binary_search_for_tcf(long long *arr, const size_t size, const long long x);

// Возвращает количество операций сравнения проведённых в функции block_search
long long block_search_for_tcf(long long *arr, const size_t size, const long long x);

#endif //ALGORITMS_AND_DS_SEARCHES_FOR_TCF_H
```

Файл lab4/time_complexity_function_for_searches/time_complexity_function_for_searches.c

```c
//
// Created by Artyom on 19.10.2023.
//

#include "time_complexity_function_for_searches.h"

void check_time_for_searches(long long (*sort_func)(long long *, size_t, long long),
                             void (*generate_func)(int *, size_t), size_t size,
                             char *experiment_name, long long name_search) {
    static size_t run_counter = 1;
    static int inner_buffer[100000000];

    generate_func(inner_buffer, size);

    // Если вызывается поиск, который требует отсортированности массива, то массив будет сортироваться
    if (name_search == 2 || name_search == 3 || name_search == 4)
        qsort(inner_buffer, size, sizeof(int), compare_ints);

    printf("Run #%zu | ", run_counter++);
    printf("Name: %s\n", experiment_name);

    long long count_comparison = sort_func(inner_buffer, size, -1);

    printf("Status: ");

    printf("OK! Count comparison %lld\n\n", count_comparison);

    char filename[256];

    sprintf(filename, "data/%s.csv", experiment_name);

    FILE *f = fopen(filename, "a");

    if (f == NULL) {
        printf("File open error %s", filename);

        exit(1);
    }

    fprintf(f, "%llu; %lld\n", size, count_comparison);

    fclose(f);
}
```

```
void time_experiment_for_searches() {
    search_function searches[] = {
            {linear_search_for_tcf,      "linear_search"},
            {fast_linear_search_for_tcf, "fast_linear_search"},
            {fast_linear_search_for_a_sorted_array_for_tcf,

"fast_linear_search_for_a_sorted_array_for_tcf"},
            {binary_search_for_tcf,      "binary_search"},
            {block_search_for_tcf,       "block_search"},
    };
    const unsigned FUNCS_N = ARRAY_SIZE(searches);
    generation_function generation[] = {
            {generate_random_array, "random"}
    };
    const unsigned CASES_N = ARRAY_SIZE(generation);

    for (size_t size = 5; size <= 45; size += 5) {
        printf("************************************\n");
        printf("Size: %llu\n", size);

        for (size_t i = 0; i < FUNCS_N; i++)
            for (size_t j = 0; j < CASES_N; j++) {
                static char filename[128];

                sprintf(filename, "%s_%s_time",
searches[i].name, generation[j].name);

                check_time_for_searches(searches[i].search,
generation[j].generate,
                                        size, filename, i);
            }

        printf("\n");
    }
}
```

Файл lab4/time_complexity_function_for_searches/

time_complexity_function_for_searches.h

```c
//
// Created by Artyom on 19.10.2023.
//

#ifndef ALGORITMS_AND_DS_TIME_COMPLEXITY_FUNCTION_FOR_SEARCHES_H
#define ALGORITMS_AND_DS_TIME_COMPLEXITY_FUNCTION_FOR_SEARCHES_H

#include <stdio.h>
#include <stdlib.h>

#include "../searches_for_tcf/searches_for_tcf.h"
#include "../../standart_functions/standart_functions.h"
#include "../../sorts/sorts.h"

#define ARRAY_SIZE(arr) (sizeof(arr) / sizeof((arr)[0]))

typedef struct search_function {
    long long (*search)(long long *, size_t, long long);

    char *name;
} search_function;

typedef struct generation_function {
    void (*generate)(int *, size_t);

    char *name;
} generation_function;

void time_experiment_for_searches();

#endif
//ALGORITMS_AND_DS_TIME_COMPLEXITY_FUNCTION_FOR_SEARCHES_H
```

Файл main.c

```c
#include <stdio.h>

#include "lab4/test_searches/test_searches.h"
#include
"lab4/time_complexity_function_for_searches/time_complexity_func
tion_for_searches.h"

// Здесь происходит запуск последней выполненной лабораторной
работы
int main() {
    test_linear_search();
    test_fast_linear_search();
    test_fast_linear_search_for_a_sorted_array();
    test_binary_search();
    test_block_search();

    time_experiment_for_searches();

    return 0;
}
```

Временные характеристики алгоритмов (при поиске элемента, которого точно нет в массиве (-1))

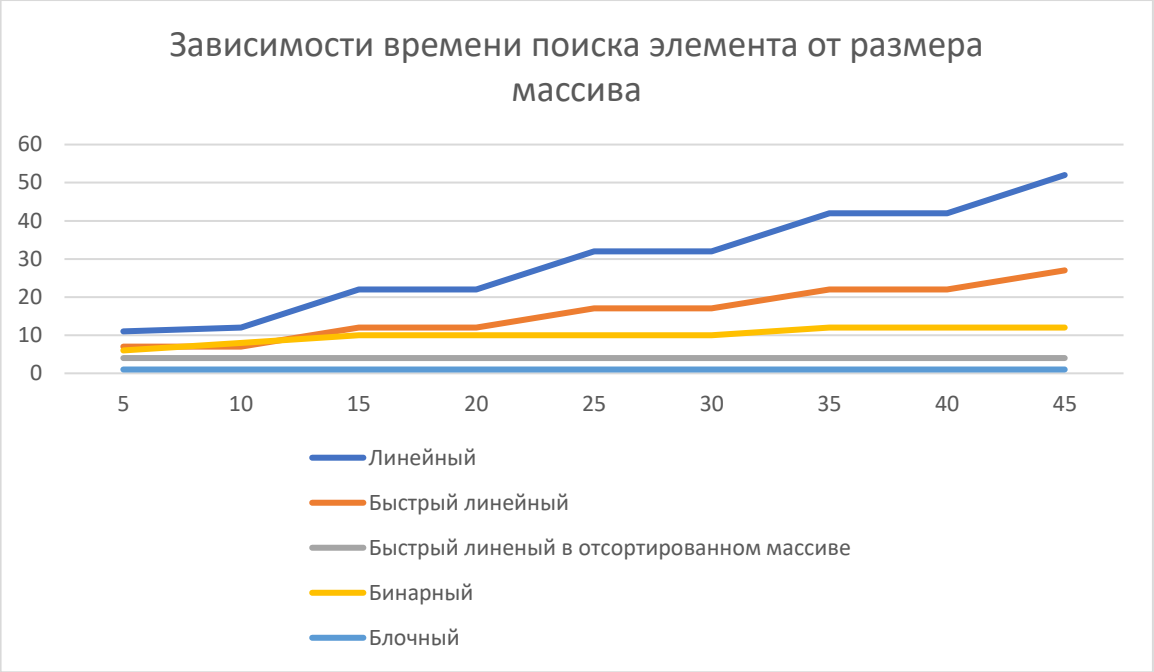
Поиск	Количество элементов в массиве								
	5	10	15	20	25	30	35	40	45
Линейный	11	12	22	22	32	32	42	42	52
Быстрый линейный	7	7	12	12	17	17	22	22	27
Быстрый линейный на отсортированном массиве	4	4	4	4	4	4	4	4	4
Бинарный	6	8	10	10	10	10	12	12	12
Блочный	1	1	1	1	1	1	1	1	1

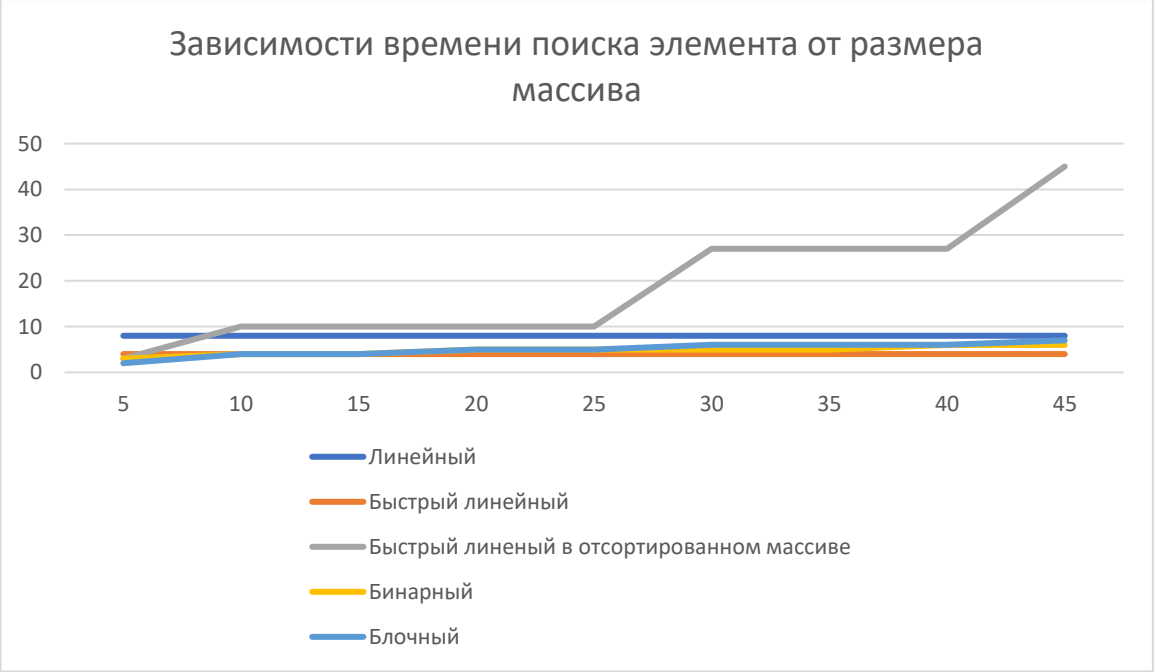
2 9 9 9 9 26 26 26 44

Временные характеристики алгоритмов (при поиске элемента, который при генерации массива находится под индексом 3)

Поиск	Количество элементов в массиве								
	5	10	15	20	25	30	35	40	45
Линейный	8	8	8	8	8	8	8	8	8
Быстрый линейный	4	4	4	4	4	4	4	4	4
Быстрый линейный на отсортированном массиве	3	10	10	10	10	27	27	27	45
Бинарный	3	4	4	5	5	5	5	6	6
Блочный	2	4	4	5	5	6	6	6	7

Графики зависимости ФВС





Порядок ФВС

Поиски	Порядок ФВС
Линейный	$O(n)$
Быстрый линейный	$O(n)$
Быстрый линейный в отсортированном массиве	$O(n)$
Бинарный	$O(\log(n))$
Блочный	$O(\sqrt{n})$

Вывод: в ходе выполнения лабораторной работы были изучены алгоритмы поиска элемента в массиве и закреплены навыки в проведении сравнительного анализа алгоритмов.