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 методов поиска (Pascal/C)»

Выполнил: ст. группы ПВ-202
Аладиб язан

Проверил:

Кабалянц Петр Степанович
Маньшин Илья Михайлович

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

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

Задания к работе:

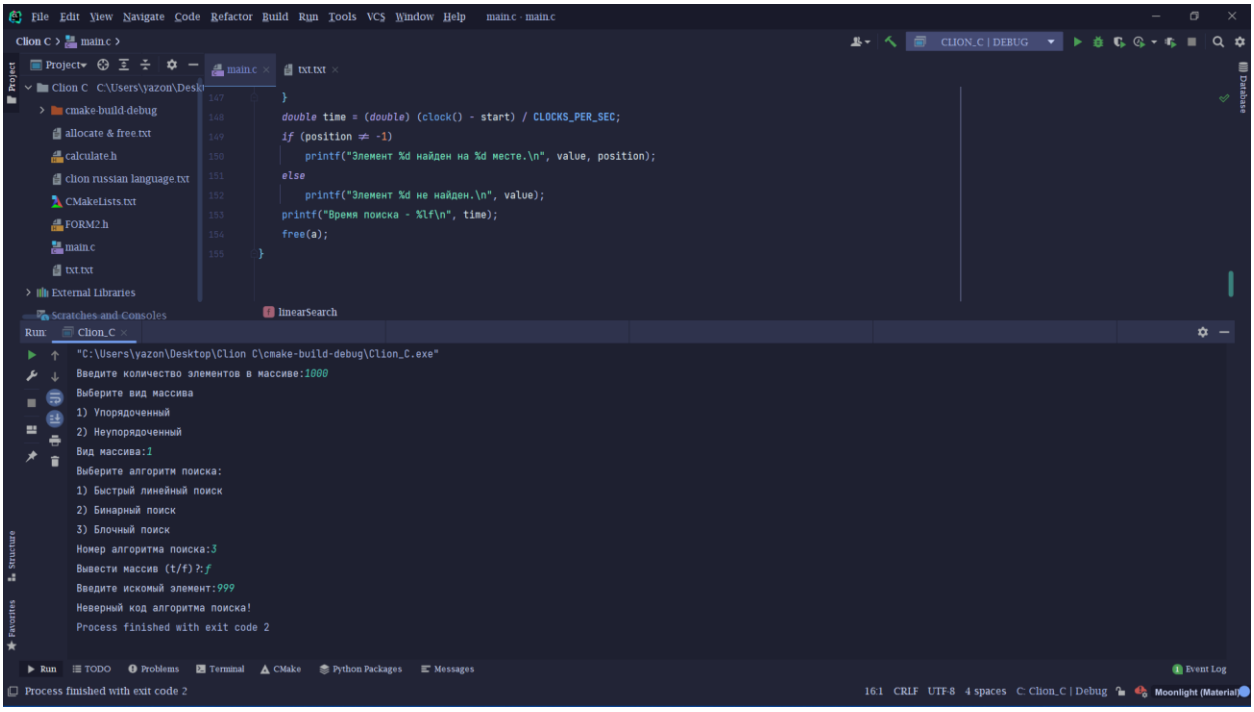
1. Изучить алгоритмы поиска:
    1) В неупорядоченном массиве:
        - Линейный
        - Быстрый линейный
2) В упорядоченном массиве:
    - Быстрый линейный
    - Бинарный
    - Блочный
2. Разработать и программно реализовать средство для проведения экспериментов по определению временных характеристик алгоритмов поиска.
3. Провести эксперименты по определению временных характеристик алгоритмов поиска. Результаты экспериментов представить в виде таблиц 12 и 13. Клетки таблицы 12 содержат максимальное количество операций сравнения при выполнении алгоритма поиска, а клетки таблицы 13 — среднее число операций сравнения.
4. Построить графики зависимости количества операций сравнения от количества элементов в массиве
5. Определить аналитическое выражение функции зависимости количества операций сравнения от количества элементов в массиве.
6. Определить порядок функций временной сложности алгоритмов Поиска.

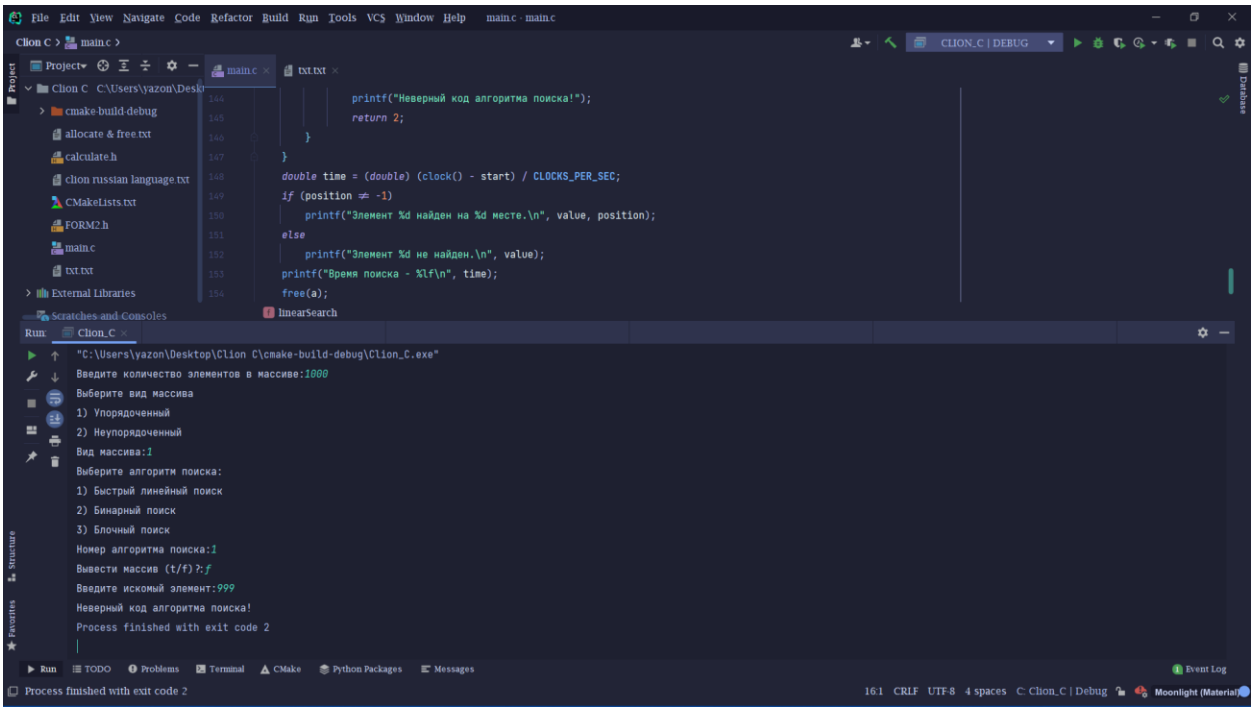
программы:

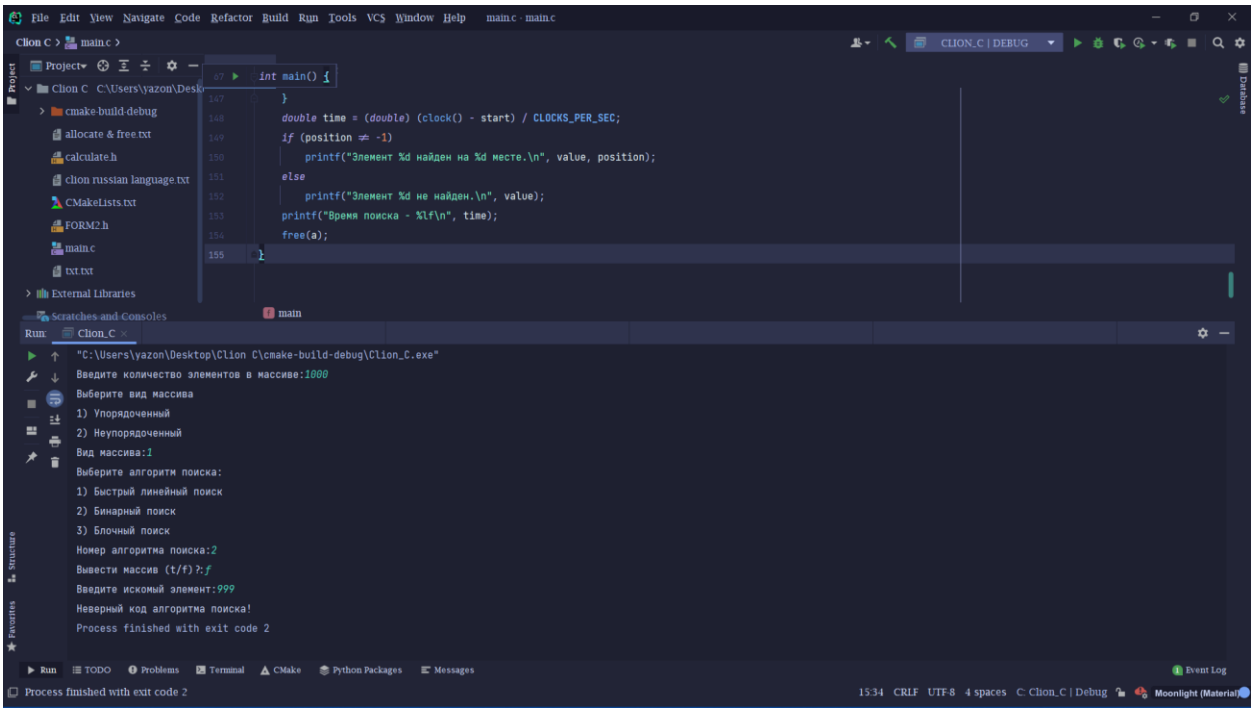
```
#include <stdio.h>
#include <time.h>
#include <math.h>
#include <aclapi.h>

// Линейный поиск
int linearSearch(const int *a, size_t len, int value) {
    int result = -1;
    for (size_t i = 0; i < len; ++i) {
        if (value == a[i]) {
            result = i;
            break;
        }
    }
    return result;
}
// Быстрый линейный поиск
int fastLinearSearch(int *a, size_t len, int value) {
    a[len] = value; //Надеюсь это ничего не испортит
    int i = 0;
    while (value != a[i]) {
        i++;
    }
    return i == len ? -1 : i;
}
// Бинарный поиск
int binarySearch(const int *a, size_t len, int value) {
    int leftBorder = 0, rightBorder = len - 1;
    while (leftBorder <= rightBorder) {
        int currentI = (rightBorder + leftBorder) / 2;
        if (value > a[currentI])
            leftBorder = currentI + 1;
        else if (value < a[currentI])
            rightBorder = currentI - 1;
        else
            return currentI;
    }
    return -1;
}
int blockSearch(const int *a, size_t len, int value) {
    int blockLen = (int) sqrt(len);
    //Проход по блокам
    int currentBlock = 0;
    for (int i = blockLen; i < len - blockLen; i += blockLen) {
        if (value < a[i]) {
            currentBlock = i;
            break;
        }
    }
    //Проверка последнего блока отдельно
    if (currentBlock == 0 && value <= a[len-1])
        currentBlock = len-1;
    //Если блок с элементом был найден
    if (currentBlock) {
        //линейный поиск элемента в блоке
        for (int i = currentBlock - blockLen; i < currentBlock; ++i) {
            if (a[i] == value)
                return i;
        }
    } else
        return -1;
```

```
}
void printArr(int *a, size_t len) {
    for (size_t i = 0; i < len; ++i) {
        printf("%d ", a[i]);
    } }
int main() {
    SetConsoleOutputCP(CP_UTF8);
    printf("Введите количество элементов в массиве:");
    size_t len;
    scanf("%d", &len);
    int *a = (int *) malloc(len * sizeof(int));
    printf("Выберите вид массива\n");
    printf("1) Упорядоченный\n");
    printf("2) Неупорядоченный\n");
    printf("Вид массива:");
    size_t arrTypeCode;
    scanf("%d", &arrTypeCode);
    if (arrTypeCode == 1) {
        for (size_t i = 0; i < len; ++i) {
            a[i] = i;
        }
    } else if (arrTypeCode == 2) {
        for (size_t i = 0; i < len; ++i) {
            a[i] = rand();
        }
    } else {
        printf("Неверный код типа массива!");
        return 1;
    }
    printf("Выберите алгоритм поиска:\n");
    if (arrTypeCode == 1) {
        printf("1) Быстрый линейный поиск\n");
        printf("2) Бинарный поиск\n");
        printf("3) Блочный поиск\n");
    } else {
        printf("1) Линейный поиск\n");
        printf("2) Быстрый линейный поиск\n");
    }
    printf("Номер алгоритма поиска:");
    size_t searchTypeCode;
    scanf("%d", &searchTypeCode);
    char outputFlag;
    printf("Вывести массив (t/f)?:");
    scanf(" %c", &outputFlag);
    if (outputFlag == 't') {
        printf("Массив:\n");
        printArr(a, len);
        printf("\n");
    }
    printf("Введите искомый элемент:");
    int value, position;
    scanf("%d", &value);
    clock_t start;
    if (arrTypeCode == 1) {
        switch (searchTypeCode) {
            case 1:
                start = clock();
                position = fastLinearSearch(a, len, value);
                break;
            case 2:
                start = clock();
                position = binarySearch(a, len, value);
                break;
```

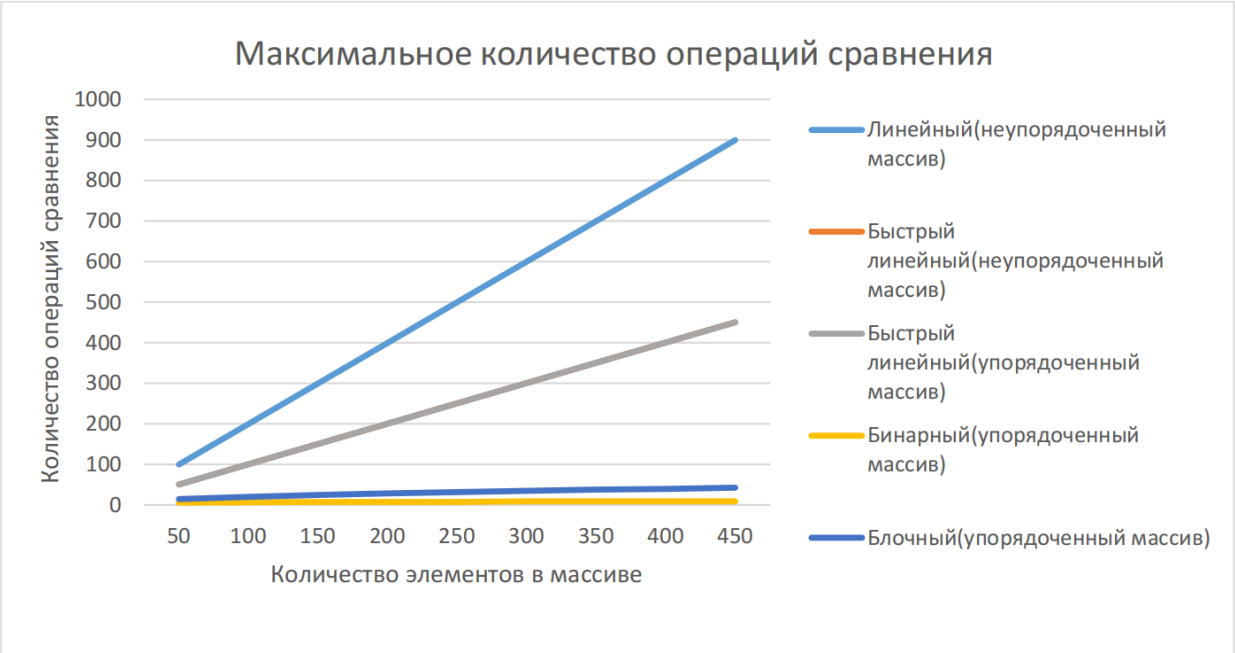
```
            case 3:
                start = clock();
                position = blockSearch(a, len, value);
                break;
            default:
                printf("Неверный код алгоритма поиска!");
                return 2;
        }
    } else {
        switch (searchTypeCode) {
            case 1:
                start = clock();
                position = linearSearch(a, len, value);
                break;
            case 2:
                start = clock();
                position = fastLinearSearch(a, len, value);
                break;
            default:
                printf("Неверный код алгоритма поиска!");
                return 2;
        }
    }
    double time = (double) (clock() - start) / CLOCKS_PER_SEC;
    if (position != -1)
        printf("Элемент %d найден на %d месте.\n", value, position);
    else
        printf("Элемент %d не найден.\n", value);
    printf("Время поиска - %lf\n", time);
    free(a);
}
```

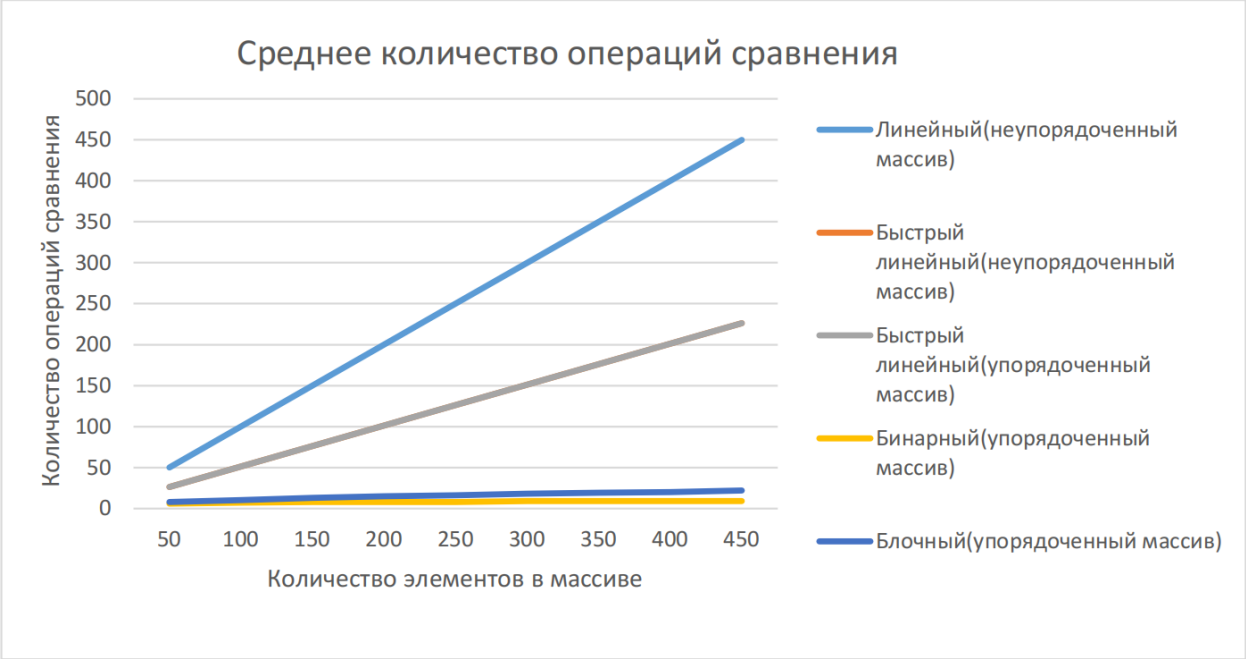



File Edit View Navigate Code Refactor Build Run Tools VCS Window Help main.c - main.c
Clion C > main.c >
CLION_C | DEBUG
Project
Clion C C:\Users\yazon\Desk
cmake-build-debug
allocate & free.txt
calculate.h
clion russian language.txt
CMakeLists.txt
FORM2.h
main.c
txt.txt
External Libraries
Scratches and Consoles
main.c
txt.txt
printf("Неверный код алгоритма поиска!");
return 2;
}
}
double time = (double) (clock() - start) / CLOCKS_PER_SEC;
if (position != -1)
printf("Элемент %d найден на %d месте.\n", value, position);
else
printf("Элемент %d не найден.\n", value);
printf("Время поиска - %lf\n", time);
free(a);
linearSearch
Run: Clion_C
"C:\Users\yazon\Desktop\CLion C\cmake-build-debug\CLion_C.exe"
Введите количество элементов в массиве:1000
Выберите вид массива
1) Упорядоченный
2) Неупорядоченный
Вид массива:1
Выберите алгоритм поиска:
1) Быстрый линейный поиск
2) Бинарный поиск
3) Блочный поиск
Номер алгоритма поиска:1
Вывести массив (t/f)?:f
Введите искомый элемент:999
Неверный код алгоритма поиска!
Process finished with exit code 2
Run TODO Problems Terminal CMake Python Packages Messages
Event Log
Process finished with exit code 2
16:1 CRLF UTF-8 4 spaces C: Clion_C | Debug
Moonlight (Material)


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help main.c - main.c
Clion C > main.c >
CLION_C | DEBUG
int main() {
}
double time = (double) (clock() - start) / CLOCKS_PER_SEC;
if (position != -1)
printf("Элемент %d найден на %d месте.\n", value, position);
else
printf("Элемент %d не найден.\n", value);
printf("Время поиска - %lf\n", time);
free(a);
}
main
Run: Clion_C
"C:\Users\yazon\Desktop\CLion C\cmake-build-debug\CLion_C.exe"
Введите количество элементов в массиве:1000
Выберите вид массива
1) Упорядоченный
2) Неупорядоченный
Вид массива:1
Выберите алгоритм поиска:
1) Быстрый линейный поиск
2) Бинарный поиск
3) Блочный поиск
Номер алгоритма поиска:2
Вывести массив (t/f)?:f
Введите искомый элемент:999
Неверный код алгоритма поиска!
Process finished with exit code 2
Run TODO Problems Terminal CMake Python Packages Messages
Event Log
Process finished with exit code 2
15:34 CRLF UTF-8 4 spaces C: Clion_C | Debug
Moonlight (Material)

Графики зависимостей функции временной сложности :





Вывод

Я изучил алгоритмы поиска элемента в массиве и закрепил навыки проведения сравнительного анализа алгоритмов. В ходе сравнительного анализа алгоритмов поиска я пришел к выводу, что поиск в упорядоченном массиве выполняется быстрее. Самым быстрым алгоритмом поиска является бинарный поиск.