МИНИСТЕРСТВО науки и высшего ОБРАЗОВАНИЯ РОссИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

Институт №3 «Системы управления, информатика и электроэнергетика»

Кафедра № 304 «Вычислительные машины, системы и сети»

Программирование

Отчет по лабораторной работе №2

**Алгоритмы поиска**

Выполнили студенты группы M3О-211Б-21

Плоцкий Б.А.

Раужев Ю.М.

Проверила Дмитриева Е.А.

Москва 2022 г.

Оглавление

[Задание 5](#_Toc116257298)

[Общая блок-схема программы 6](#_Toc116257299)

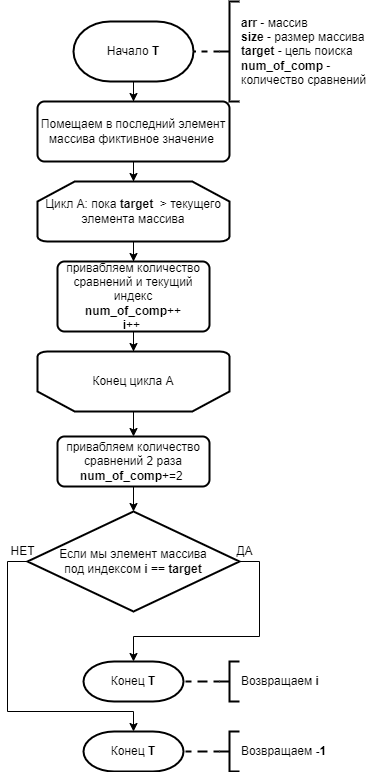
[Структурные схемы алгоритмов 7](#_Toc116257300)

[Функция gen\_arr\_draw\_table () 7](#_Toc116257301)

[Функция BLS() 10](#_Toc116257302)

[Функция SLS () 12](#_Toc116257303)

[Функция T () 15](#_Toc116257304)

[ 16](#_Toc116257305)

[Функция B () 17](#_Toc116257306)

[Код программы 20](#_Toc116257307)

[Тестирование 34](#_Toc116257308)

[Элемент в середине массива 34](#_Toc116257309)

[Результат программы 34](#_Toc116257310)

[График 35](#_Toc116257311)

[Элемент в начале массива. 36](#_Toc116257312)

[Результат программы 36](#_Toc116257313)

[График 36](#_Toc116257314)

[Элемент в конце массива. 37](#_Toc116257315)

[Результат программы 37](#_Toc116257316)

[График 37](#_Toc116257317)

[Тестирование программы 38](#_Toc116257318)

[Тестирование некорректных режимов 38](#_Toc116257319)

[Тест 1 38](#_Toc116257320)

[Тестирование корректных режимов 39](#_Toc116257321)

[Тест 1 39](#_Toc116257322)

[Вывод 41](#_Toc116257323)

# Задание

Для алгоритмов **BLS** и **SLS** в качестве входного массива использовать одну и ту же последовательность значений (функция **rand**( )).

Для алгоритмов **Т** и **В** – значения массива должны быть отсортированы по не убыванию, одна и та же последовательность чисел (можно использовать соответствующую функцию из первой лабораторной работы).

Оценить длительность поиска для различных значений размеров последовательностей (начиная с 10000 до 200000 элементов массива, провести измерения не менее, чем для 10 разных размерностей).

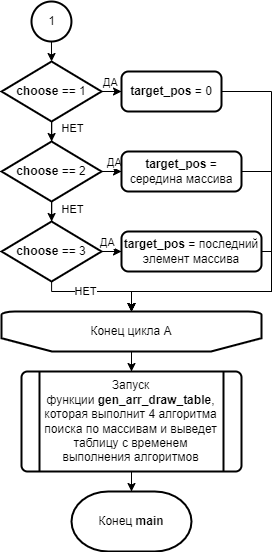
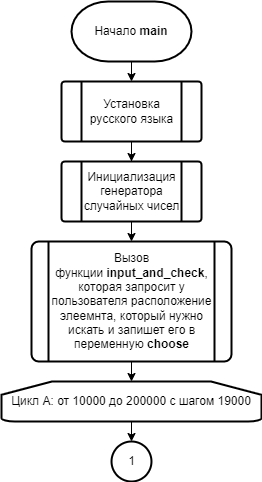
Для каждой размерности рассматриваются случаи нахождения ключа поиска в начале, в середине и в конце массива.

Для алгоритмов **BLS** и **SLS** кроме подсчета **времени,** необходимого для поиска, требуется определить сколько раз выполняются операции **сравнения** (сравнение ключа с элементом массива, а также в одном из этих двух алгоритмов добавляется подсчет сравнений при анализе индекса элемента массива в цикле… ).

Все результаты оформить в виде таблиц и графиков. На графиках **- только временные характеристики** поиска.

По результатам сделать выводы об эффективности того или иного алгоритма поиска.

# Общая блок-схема программы



# Структурные схемы алгоритмов

## Функция gen\_arr\_draw\_table ()

1. Назначение:

Сформировать массивы для алгоритмов, выполнит алгоритмы с этими массивами, нарисовать таблицу.

1. Прототип функции:

void gen\_arr\_draw\_table(

int size, //размер массива

int target // элемент поиска

);

1. Обращение:

gen\_arr\_draw\_table(i, target\_pos);

1. Описание параметров

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификаторы | Тип | Назначение | Входной/Выходной |
| size | int | Размер массива | Выходной |
| target | int | Элемент для поиска | Входной |

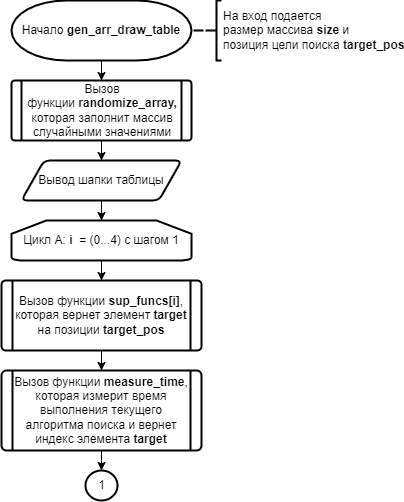


Рисунок 1. Структурная схема алгоритма функции gen\_arr\_draw\_table().

## Функция BLS()

1. Назначение:

Найти элемент в массиве

1. Прототип функции:

template<typename T>

T BLS(

T\* arr, // исходный массив

int size, // размер массива

T target, // цель поиска

int& num\_of\_comp // количество сравнений

)

1. Обращение

BLS(arr,size,target,number);

1. Описание параметров

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификаторы | Тип | Назначение | Входной/Выходной |
| arr | int\* | Массив | Входной |
| size | Int | Размер массива | Входной |
| target | int | Элемент поиска | Входной |
| num\_of\_comp | int& | Количество сравнений | Выходной |

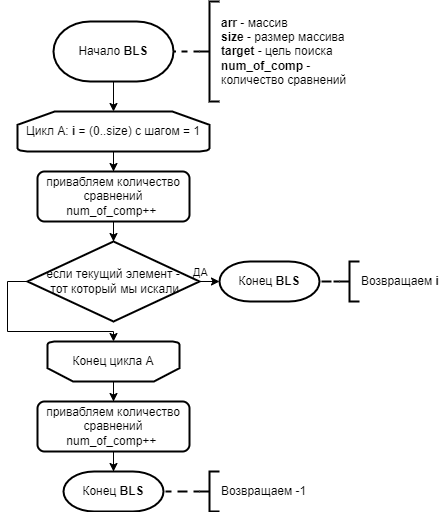


Рисунок 2. Структурная схема алгоритма функции BLS().

## Функция SLS ()

1. Назначение:

Найти элемент в массиве

1. Прототип функции:

template<typename T>

T SLS(

T\* arr, // исходный массив

int size, // размер массива

T target, // цель поиска

int& num\_of\_comp // количество сравнений

)

1. Обращение

SLS(arr,size,target,number);

1. Описание параметров

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификаторы | Тип | Назначение | Входной/Выходной |
| arr | int\* | Массив | Входной |
| size | Int | Размер массива | Входной |
| target | int | Элемент поиска | Входной |
| num\_of\_comp | int& | Количество сравнений | Выходной |

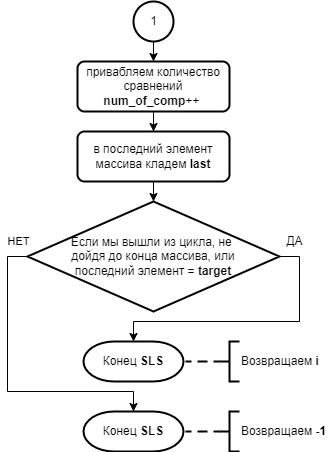
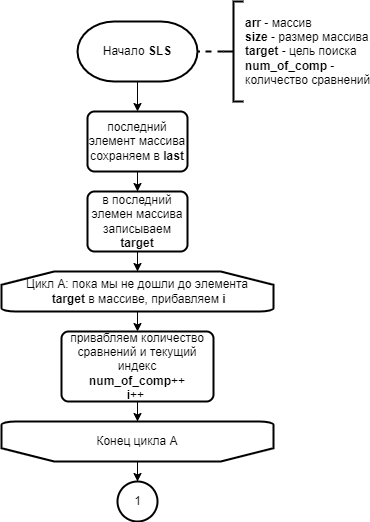


Рисунок 3. Структурная схема алгоритма функции SLS()

## Функция T ()

1. Назначение:

Найти элемент в массиве

1. Прототип функции:

template<typename Tt>

Tt T(

Tt\* arr, // исходный массив

int size, // размер массива

Tt target, // цель поиска

int& num\_of\_comp // количество сравнений

)

1. Обращение

T(arr,size,target,number);

1. Описание параметров

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификаторы | Тип | Назначение | Входной/Выходной |
| arr | int\* | Массив | Входной |
| size | Int | Размер массива | Входной |
| target | int | Элемент поиска | Входной |
| num\_of\_comp | int& | Количество сравнений | Выходной |

# C:\Users\bogda\AppData\Local\Microsoft\Windows\INetCache\Content.Word\all-T.drawio.png

## Функция B ()

1. Назначение:

Найти элемент в массиве

1. Прототип функции:

template<typename T>

B BLS(

T\* arr, // исходный массив

int size, // размер массива

T target, // цель поиска

int& num\_of\_comp // количество сравнений

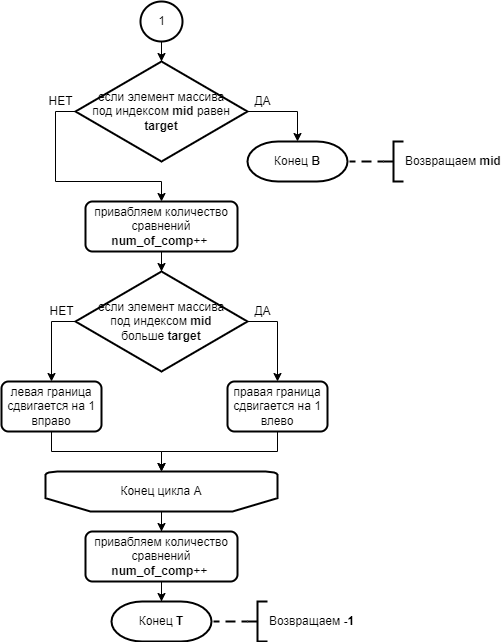
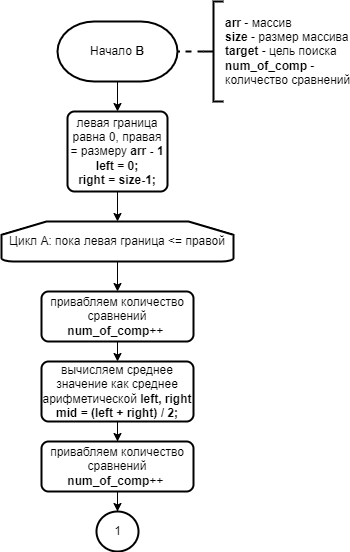
)

1. Обращение

B(arr,size,target,number);

1. Описание параметров

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификаторы | Тип | Назначение | Входной/Выходной |
| arr | int\* | Массив | Входной |
| size | Int | Размер массива | Входной |
| target | int | Элемент поиска | Входной |
| num\_of\_comp | int& | Количество сравнений | Выходной |



# Код программы

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\  
\*                    кафедра № 304 2 курс 3 семестр информатика         \*  
\*-----------------------------------------------------------------------\*  
\*   Project type : solution                                             \*  
\*   Project name : LW2                                                  \*  
\*   File name    : LW2.cpp                                              \*  
\*   Language     : c/c++                                                \*  
\*   Programmers  : Плоцкий Б.А. Раужев Ю. М.                            \*  
\*   Created      :  4/10/22                                             \*  
\*   Last revision:  8/10/22                                             \*  
\*   Comment(s)   :                                                      \*  
\*                                                                       \*  
\*   Для алгоритмов BLS и SLS в качестве входного массива использовать   \*  
\*   одну и ту же последовательность значений (функция rand( )).         \*  
\*                                                                       \*  
\*   Для алгоритмов Т и В – значения массива должны быть отсортированы   \*  
\*   по не убыванию, одна и та же последовательность чисел (можно        \*  
\*   использовать соответствующую функцию из первой лабораторной работы).\*  
\*                                                                       \*  
\*   Оценить длительность поиска для различных значений размеров         \*  
\*   последовательностей (начиная с 10000 до 200000 элементов массива,   \*  
\*   провести измерения не менее, чем для 10 разных размерностей).       \*  
\*   Для каждой размерности рассматриваются случаи нахождения ключа      \*  
\*   поиска в начале, в середине и в конце массива.                      \*  
\*                                                                       \*  
\*   Для алгоритмов BLS и SLS кроме подсчета времени, необходимого для   \*  
\*   поиска, требуется определить сколько раз выполняются операции       \*  
\*   сравнения (сравнение ключа с элементом массива, а также в одном     \*  
\*   из этих двух алгоритмов добавляется подсчет сравнений при анализе   \*  
\*   индекса элемента массива в цикле… ).                                \*  
\*                                                                       \*  
\*   Все результаты оформить в виде таблиц и графиков. На графиках -     \*  
\*   только временные характеристики поиска.                             \*  
\*   По результатам сделать выводы об эффективности того или иного       \*  
\*   алгоритма поиска.                                                   \*  
\*                                                                       \*  
\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
#include <iostream>  
#include <ctime>  
#include <cmath>  
#include <iomanip>  
#include <chrono>****using******namespace****std;  
   
// тип данных NS  
typedef chrono::nanoseconds NS;  
   
// нужна ли печать массива в консоль  
#define NEED\_PRINT false  
   
// минимальный и максимальный  
// размер массива  
#define MIN\_ARR\_SIZE 10000  
#define MAX\_ARR\_SIZE 200000  
   
// количество таблиц в консоли  
#define NUMB\_OF\_TABLES 10  
   
// минимальное и максимальное значение  
// для чисел в массиве  
#define MIN\_VALUE MIN\_ARR\_SIZE \* 2  
#define MAX\_VALUE MAX\_ARR\_SIZE \* 2  
   
// заполнение len элементов элементом symb  
#define OUT\_W(symb, len) fixed << setfill(symb) << setw(len)  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*              П Р О Т О Т И П Ы   Ф У Н К Ц И Й                \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// упорядочная функция по возрастанию  
template<typename T>****void****f1(T\* arr,****int****size,****int****max = MAX\_VALUE,****int****step = 1);  
   
// изменение размера динамического массива  
template<typename T>****void****resize\_arr(T\*& arr,****int****& size,****int****inc = 1);  
   
// возвращение случайного значения из  
// интервала (min, max)  
template<typename T>  
T rand\_num(T min, T max);  
   
// рандомизация значений массива  
template<typename T>****void****randomize\_array(T\* arr,****int****size);  
   
// печать массива в поток  
template<typename T>****void****print\_arr(T\* arr,****int****size, ostream& stream = cout);  
   
// ввод и проверка значений  
template<typename T>  
T input\_and\_check(T \_min, T \_max,****const******char****\* welcome\_str,****const******char****\* err\_str);  
   
// алгоритм Better Linear Search  
template<typename T>  
T BLS(  
    T\* arr,         // исходный массив****int****size,           // размер массива  
    T target,           // цель поиска****int****& num\_of\_comp    // количество сравнений  
);  
   
// алгоритм Sentinel Linear Search  
template<typename T>  
T SLS(  
    T\* arr,         // исходный массив****int****size,           // размер массива  
    T target,           // цель поиска****int****& num\_of\_comp    // количество сравнений  
);  
   
// функция, которая сортирует массив и  
// добавляет фиктивный элемент в конец  
template<typename T>  
T support\_T(  
    T\*& arr,            // исходный массив****int****& size,          // размер массива****int****target\_pos      // позиция цели  
);  
   
// алгоритм T  
template<typename Tt>  
Tt T(  
    Tt\* arr,            // исходный массив****int****size,           // размер массива  
    Tt target,          // цель поиска****int****& num\_of\_comp    // количество сравнений  
);  
   
// функция, которая сортирует массив и  
// убирает с конца фиктивный элемент  
template<typename T>  
T support\_B(  
    T\*& arr,            // исходный массив****int****& size,          // размер массива****int****target\_pos      // позиция цели  
);  
   
// алгоритм бинарного поиска  
template<typename T>  
T B(  
    T\* arr,         // исходный массив****int****size,           // размер массива  
    T target,           // цель поиска****int****& num\_of\_comp    // количество сравнений  
);  
   
// стандартная вспомогательная функция  
template<typename T>  
T support\_default(T\*& arr,****int****&,****int****target\_pos);  
   
// функция измерения времени выполнения функций  
template<typename T>****int****measure\_time(  
    T\*& arr,                // исходный массив****int****size,               // размер массива  
    T& target,          // элемент поиска****int****& num\_of\_comp,       // количество сравнений  
    NS& time,               // время работы функции  
    T(\*search\_func)(        // функция поиска  
        T\* arr,         // исходный массив****int****size,           // размер массива  
        T target,           // цель поиска****int****& num\_of\_comp    // количество сравнений  
        )  
);  
   
// вызов всех функций поиска и вывод результатов в таблицу****void****gen\_arr\_draw\_table(****int****size,   //размер массива****int****target  // элемент поиска  
);  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*                      К О Н С Т А Н Т Ы                        \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// массив с вспомогательными функциями  
template<typename T>  
T(\*sup\_funcs[])(T\*& arr,****int****&,****int****target\_pos) =  
{  
    support\_default,  
    support\_default,  
    support\_T,  
    support\_B  
};  
   
// массив с функциями поиска  
template<typename Tt>  
Tt(\*search\_funcs[])(Tt\*,****int****, Tt,****int****&) =  
{  
    BLS,  
    SLS,  
    T,  
    B  
};  
   
// названия алгоритмов****const******char****\* algorithms[] =  
{  
    "BLS",  
    "SLS",  
    "T",  
    "B"  
};  
   
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*                Г Л А В Н А Я   Ф У Н К Ц И Я                  \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/****int****main()  
{  
    setlocale(LC\_ALL, "ru");  
   
    srand(time(NULL));  
   
    // запрашиваем у пользователя расположение элемента****int****choose = input\_and\_check(1, 3, "Введите расположение элемента,который надо найти:\n\  
\t1.В начале\n\t2.В середине\n\t3.В конце\n", "Строго целые числа 1,2 или 3\n");  
   
    // позиция элемента поиска****int****target\_pos = 0;  
   
    // 10 раз просчитываем работу поиска****for****(****int****i = MIN\_ARR\_SIZE; i <= MAX\_ARR\_SIZE;  
        i += (MAX\_ARR\_SIZE - MIN\_ARR\_SIZE) / NUMB\_OF\_TABLES  
        )  
    {****switch****(choose)  
        {****case****1:  
            target\_pos = 0;****break****;****case****2:  
            target\_pos = i / 2;****break****;****case****3:  
            target\_pos = i - 1;****break****;  
        }  
        gen\_arr\_draw\_table(i, target\_pos);  
    }****return****0;  
}  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*             Р Е А Л И З А Ц И Я   Ф У Н К Ц И Й               \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
//#include <string>  
// упорядочная функция по возрастанию  
template<typename T>****void****f1(T\* arr,****int****size,****int****max,****int****step)  
{  
    //k, b - коэффициенты прямой  
    T k = max / (step \* size);  
    T b = MIN\_VALUE;****int****x = 0.0; // координата x  
   
    // заполнение массива****for****(****int****i = 0; i < size; i++, x += step)  
    {  
        arr[i] = k \* x + b;  
    }  
}  
   
// изменение размера динамического массива  
template<typename T>****void****resize\_arr(T\*& arr,****int****& size,****int****inc)  
{  
   
    // увеличение длины массива  
    size += inc;  
   
    // создание нового массива  
    T\* new\_arr =****new****T[size];****if****(size - 1 != 0)  
    {  
        // копирование данных в новый массив из старого****for****(****int****i = 0; i < size - 1; i++)  
        {  
            new\_arr[i] = arr[i];  
        }// end for i  
   
        // удаление старого массива  
        delete[] arr;  
    }  
   
    // присваивание старому массиву адресс памяти нового  
    arr = new\_arr;  
}  
   
// возвращение случайного значения из  
// интервала (min, max)  
template<typename T>  
T rand\_num(T min, T max)  
{****return****rand() % (max - min) + min;  
}  
   
// рандомизация значений массива  
template<typename T>****void****randomize\_array(T\* arr,****int****size)  
{****for****(****int****i = 0; i < size; i++)  
    {  
        arr[i] = rand\_num(MIN\_VALUE, MAX\_VALUE);  
    }  
}  
   
// печать массива в поток  
template<typename T>****void****print\_arr(T\* arr,****int****size, ostream& stream)  
{  
    // вывод элементов массива****for****(****int****i = 0; i < size; i++)  
    {  
        stream << arr[i] << ' ';  
    }  
    cout << '\n';  
}  
   
// ввод и проверка значений  
template<typename T>  
T input\_and\_check(T \_min, T \_max,****const******char****\* welcome\_str,****const******char****\* err\_str)  
{  
    // размер массива  
    T num;  
   
    // вывод сообщения  
    cout << welcome\_str << "\n";  
    cin >> num;  
   
    // если было введено не то****if****(num > \_max || num < \_min) {  
        // если была введена не цифра****if****(cin.fail())  
        {  
            cin.clear();  
            cin.ignore(INT\_MAX, '\n');  
        }  
   
        // отчистка консоли  
        system("cls");  
        cout << err\_str << "\n";  
   
        // рекурсивное обращение  
        num = input\_and\_check(\_min, \_max, welcome\_str, err\_str);  
    }****return****num;  
}  
   
// алгоритм Better Linear Search  
template<typename T>  
T BLS(  
    T\* arr,         // исходный массив****int****size,           // размер массива  
    T target,           // цель поиска****int****& num\_of\_comp    // количество сравнений  
)  
{****for****(****int****i = 0; i < size; i++)  
    {  
        num\_of\_comp++;****if****(arr[i] == target)  
        {****return****i;  
        }  
    }  
    num\_of\_comp++;****return****-1;  
}  
   
// алгоритм Sentinel Linear Search  
template<typename T>  
T SLS(  
    T\* arr,         // исходный массив****int****size,           // размер массива  
    T target,           // цель поиска****int****& num\_of\_comp    // количество сравнений  
)  
{  
    // последний элемент массива****int****last = arr[size - 1];  
   
    // элемент, который нужно найти  
    // располагаем в последеней ячейке массива  
    arr[size - 1] = target;  
   
    // счетчик элементов массива****int****i = 0;****while****(arr[i] != target)  
    {  
        num\_of\_comp++;  
        i++;  
    }  
    num\_of\_comp++;  
   
    // Возвращаем последний элемент в массив  
    arr[size - 1] = last;  
   
    // если элемент был найден где-то в массиве  
    // или на его конце, то возвращаем индекс  
    num\_of\_comp++;****if****((i < size - 1) || (arr[size - 1] == target))  
    {****return****i;  
    }****return****-1;  
}  
   
// функция, которая сортирует массив и  
// добавляет фиктивный элемент в конец  
template<typename T>  
T support\_T(  
    T\*& arr,            // исходный массив****int****& size,          // размер массива****int****target\_pos      // позиция цели  
)  
{  
    // формирование неубывающей последовательности  
    f1(arr, size);  
   
    // добавление фиктивного значения в конец массива  
    // если еще нет фиктивного элемента****if****(arr[size - 1] != INT\_MAX)  
    {  
        resize\_arr(arr, size);  
    }****return****arr[target\_pos];  
}  
   
// алгоритм T  
template<typename Tt>  
Tt T(  
    Tt\* arr,            // исходный массив****int****size,           // размер массива  
    Tt target,          // цель поиска****int****& num\_of\_comp    // количество сравнений  
)  
{  
    // добавление в конец массва фиктивной записи  
    arr[size - 1] = INT\_MAX;****int****i = 0;****while****(target > arr[i])  
    {  
        num\_of\_comp++;  
        i++;  
    }  
    num\_of\_comp++;  
   
    num\_of\_comp++;****if****(target == arr[i])  
    {****return****i;  
    }****return****-1;  
}  
   
// функция, которая сортирует массив и  
// убирает с конца фиктивный элемент  
template<typename T>  
T support\_B(  
    T\*& arr,            // исходный массив****int****& size,          // размер массива****int****target\_pos      // позиция цели  
)  
{  
    // убирание последнего элемента из массива,  
    // если он является фиктивным****if****(arr[size - 1] == INT\_MAX)  
    {  
        resize\_arr(arr, size, -1);  
    }  
   
    // формирование неубывающей последовательности  
    f1(arr, size);  
   
    //print\_arr(arr, size);****return****arr[target\_pos];  
}  
   
// алгоритм бинарного поиска  
template<typename T>  
T B(  
    T\* arr,         // исходный массив****int****size,           // размер массива  
    T target,           // цель поиска****int****& num\_of\_comp    // количество сравнений  
)  
{  
    // левая граница****int****left = 0;  
    // правая граница****int****right = size - 1;  
    // индекс среднего элемента****int****mid;****while****(left <= right)  
    {  
        num\_of\_comp++;  
   
        mid = (left + right) / 2;  
        num\_of\_comp++;****if****(arr[mid] == target)  
        {****return****mid;  
        }****else*** *{  
            num\_of\_comp++;****if****(arr[mid] > target)  
            {  
                right = mid - 1;  
            }****else*** *{  
                left = mid + 1;  
            }  
        }  
    }  
    num\_of\_comp++;****return****-1;  
}  
   
// стандартная вспомогательная функция  
template<typename T>  
T support\_default(T\*& arr,****int****&,****int****target\_pos)  
{  
    // необзодимо для отметки элемента  
    // который надо найти****if****(arr[target\_pos] > 0)  
    {  
        arr[target\_pos] \*= -1;  
    }****return****arr[target\_pos];  
}  
   
// функция измерения времени выполнения функций  
template<typename T>****int****measure\_time(  
    T\*& arr,                // исходный массив****int****size,               // размер массива  
    T& target,          // элемент поиска****int****& num\_of\_comp,       // количество сравнений  
    NS& time,               // время работы функции  
    T(\*search\_func)(        // функция поиска  
        T\* arr,         // исходный массив****int****size,           // размер массива  
        T target,           // цель поиска****int****& num\_of\_comp    // количество сравнений  
        )  
)  
{  
    // обнуление количества сравнений  
    num\_of\_comp = 0;  
   
    // начало отсчета времени  
    auto begin = chrono::steady\_clock::now();  
   
    // вызов функции поиска****int****res = search\_func(arr, size, target, num\_of\_comp);  
   
    // конец отсчета времени  
    auto end = chrono::steady\_clock::now();  
   
    // если необходимое число было  
    // помечено для SLS И BLS алгоритмов  
    // то убираем пометку****if****(target < 0)  
    {  
        target \*= -1;  
    }  
   
    // вывод в консоль времени работы программы  
    time = chrono::duration\_cast<NS>(end - begin);****return****res;  
}  
   
// вызов всех функций поиска и вывод результатов в таблицу****void****gen\_arr\_draw\_table(****int****size,****int****target\_pos)  
{****long****\* arr =****new******long****[size]; // массив****int****num\_of\_comp = 0;        // количество сравнений  
    NS elapsed\_time;            // время выполнения функции****long****target;                    // цель поиска  
   
    // рандомизация массива  
    randomize\_array(arr, size);  
   
    // вывод таблицы  
    cout << OUT\_W('\_', 83) << "\n";  
    cout << "|\_Размер\_массива:\_\_\_|\_" << OUT\_W('\_', 58) << size << "\_|\n";  
    cout << "| Алгоритм |  Ключ  | Индекс ключа | Количество сравнений | Время выполнения(нс) |\n";  
   
    // вызов всех функций поиска****for****(****int****i = 0; i < 4; i++)  
    {  
        // поиск элемента на позиции target\_pos  
        // + подготовка массивов для работы алгоритмов поиска  
        target = sup\_funcs<****long****>[i](arr, size, target\_pos);  
   
        // получение индекса элемента в массиве****int****index = measure\_time<****long****>(arr, size, target, num\_of\_comp,  
            elapsed\_time, search\_funcs<****long****>[i]);  
   
        // вывод строки таблицы  
        cout << "| " << OUT\_W(' ', 8) << algorithms[i]  
            << " | " << OUT\_W(' ', 6) << target  
            << " | " << OUT\_W(' ', 12) << index  
            << " | " << OUT\_W(' ', 20) << num\_of\_comp  
            << " | " << OUT\_W(' ', 20) << elapsed\_time.count() << " |\n";  
   
        // печать массива, если это необходимо****if****(NEED\_PRINT)  
        {  
            cout << "|МАССИВ ДЛЯ " << OUT\_W(' ', 8) << algorithms[i] << "|";  
            print\_arr(arr, size);  
        }  
    }  
    // конец таблицы  
    cout << OUT\_W('-', 83) << "\n";  
   
    // удаление массива  
    delete[] arr;  
   
}  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End Of LW2.cpp File \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

# Тестирование

## Элемент в середине массива

### Результат программы

|  |  |  |  |
| --- | --- | --- | --- |
| Размер массива | Функция | Время выполнения | Кол-во сравнений |
| 10000 | BLS | 6300 | 5001 |
| SLS | 5000 | 5002 |
| T | 5000 | 5002 |
| B | 300 | 38 |
| 29000 | BLS | 19300 | 14501 |
| SLS | 14600 | 14502 |
| T | 17500 | 14502 |
| B | 200 | 41 |
| 48000 | BLS | 30500 | 24001 |
| SLS | 26500 | 24002 |
| T | 23400 | 24002 |
| B | 200 | 44 |
| 67000 | BLS | 41100 | 33501 |
| SLS | 33000 | 33502 |
| T | 32600 | 33502 |
| B | 200 | 47 |
| 86000 | BLS | 52600 | 43001 |
| SLS | 42400 | 43002 |
| T | 42500 | 43002 |
| B | 300 | 47 |
| 105000 | BLS | 63900 | 52501 |
| SLS | 56700 | 52502 |
| T | 51000 | 52502 |
| B | 400 | 47 |
| 124000 | BLS | 76600 | 62001 |
| SLS | 60600 | 62002 |
| T | 86100 | 62002 |
| B | 700 | 47 |
| 143000 | BLS | 88800 | 71501 |
| SLS | 70700 | 71502 |
| T | 70700 | 71502 |
| B | 200 | 50 |
| 162000 | BLS | 99900 | 81001 |
| SLS | 84600 | 81002 |
| T | 78600 | 81002 |
| B | 200 | 50 |
| 181000 | BLS | 112100 | 90501 |
| SLS | 89300 | 90502 |
| T | 117700 | 90502 |
| B | 300 | 50 |
| 200000 | BLS | 126000 | 100001 |
| SLS | 108200 | 100002 |
| T | 156100 | 100002 |
| B | 200 | 50 |

График

## Элемент в начале массива.

### Результат программы

График

## Элемент в конце массива.

### Результат программы

График

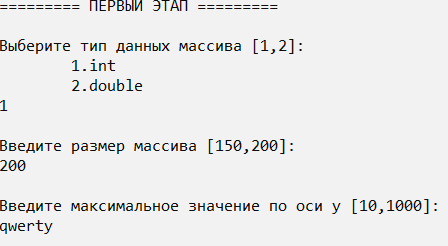
# Тестирование программы

## Тестирование некорректных режимов

### Тест 1

**Цель:** Проверить работу программы при введении некорректных данных

**Исходные данные:** 1 200 qwerty



**Ожидаемый Результат:** Вывод сообщения об ошибке:

Должно быть введено значение в интервале [10,1000].

**Полученный результат:**



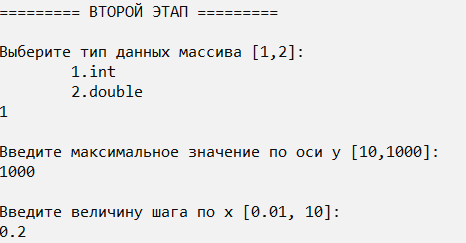
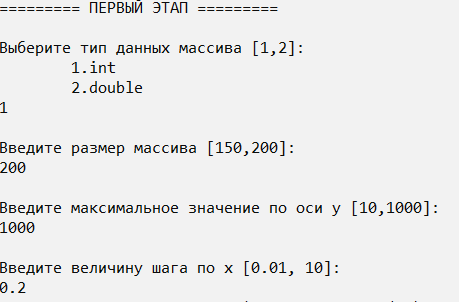
**Вывод по тесту:** Ожидаемый результат совпал с полученным результатом. Тест ошибок не обнаружил.

## Тестирование корректных режимов

### Тест 1

**Цель:** Проверить работу программы при выполнении всех условий задачи.

**Исходные данные:** 1 200 1000 0.2



**Ожидаемый Результат:**

Время выполнения 1 функции = 3000 (нс)

Время выполнения 2 функции = 600 (нс)

Время выполнения 3 функции = 8600 (нс)

Время выполнения 4 функции = 15600 (нс)

Время выполнения 5 функции = 14400 (нс)

Время выполнения 6 функции = 13400 (нс)

Длина последовательности |f1 мкС |f2 мкС |f3 мкС |f4 мкС |f5 мкС |f6 мкС |

500000| 1084 | 1139 | 20470 | 6346 | 13380 | 20500 |

1000000| 2164 | 2264 | 42607 | 12734 | 26839 | 41261 |

1500000| 3223 | 3563 | 62023 | 18923 | 41698 | 63468 |

2000000| 4603 | 4773 | 81696 | 25520 | 54821 | 82164 |

2500000| 5439 | 5696 |106791 | 31774 | 68604 |104894 |

3000000| 6526 | 7123 |128728 | 38289 | 87759 |128972 |

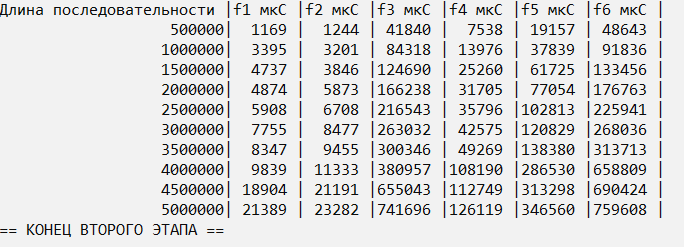
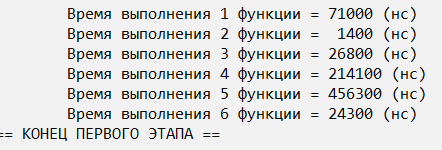
3500000| 7951 | 7924 |148033 | 44747 | 96866 |146140 |

4000000| 8813 | 9271 |163997 | 49858 |108219 |167511 |

4500000| 9957 | 10903 |186861 | 57289 |121730 |185694 |

5000000| 10926 | 11407 |205685 | 63566 |134516 |208095 |

**Полученный результат:**



**Вывод по тесту:** Ожидаемый результат совпал с полученным результатом. Тест ошибок не обнаружил.

# Вывод

Работа программы завершена на основании:

1) Полученные результаты совпали с ожидаемыми;

2) Считаем набор тестов полным.