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 №3

Алгоритмы сортировки данных

Выполнили студенты группы M3О-211Б-21

Плоцкий Б.А.

Раужев Ю.М.

Проверила Дмитриева Е.А.

Москва 2022 г.

Оглавление

[Задание 3](#_Toc117510126)

[Общая блок-схема программы 4](#_Toc117510127)

[Структурные схемы алгоритмов 5](#_Toc117510128)

[Функция selection\_sort() 5](#_Toc117510129)

[Функция quick\_sort() 8](#_Toc117510130)

[Функция partition () 10](#_Toc117510131)

[Функция improved\_quick\_sort() 12](#_Toc117510132)

[Код программы 16](#_Toc117510133)

[Сопоставительный анализ 33](#_Toc117510134)

[Результаты работы программ по этапам 33](#_Toc117510135)

[Тест 1 33](#_Toc117510136)

[Тест 2 34](#_Toc117510137)

[Тест 3 36](#_Toc117510138)

[Тест 4 37](#_Toc117510139)

[Тест 5 39](#_Toc117510140)

[Тест 6 40](#_Toc117510141)

[Сравнение алгоритмов на больших размерах массивов 43](#_Toc117510142)

[Вывод 52](#_Toc117510143)

# Задание

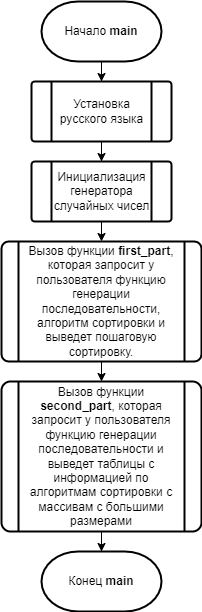
Для массива из ***n*** элементов выполнить сортировку с помощью двух указанных методов для:

1. заданной произвольным образом последовательности чисел,
2. уже отсортированных последовательностей в возрастающем и убывающем порядке (лучший и худший случаи для выполнения сортировки).

Этапы выполнения ЛР:

1. Промежуточные результаты сортировки представить по каждой итерации для массива размерностью ***n*** = 15 (необходимо выводить на печать весь массив на каждом промежуточном этапе сортировки). Сравнить число **необходимых** **сравнений и число пересылок**.
2. Выполнить сортировку массивов размерности ***n*** = 1000 ( 10000, 100000, 500000 ). Сравнить **время выполнения алгоритмов, число необходимых сравнений и число пересылок**:

# Общая блок-схема программы



# Структурные схемы алгоритмов

## Функция selection\_sort()

1. Назначение:

Сортировка последовательности методом выбора.

1. Прототип функции:

template<typename T = int>

void selection\_sort(

T\* arr, // массив

int low, // индекс, с которого начинается вывод

int high, // индекс, на котором кончается вывод

help\_data& data // количество сравнений и премещений

);

1. Обращение:

selection\_sort (arr, 0, size, data)

1. Описание параметров

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификаторы | Тип | Назначение | Входной/Выходной |
| arr | int \* | Массив | Выходной |
| low | int | Индекс первого элемента последовательности | Входной |
| high | int | Индекс элемента после последнего | Входной |
| data | help\_data& | Количество сравнений и перемещений | Выходной |

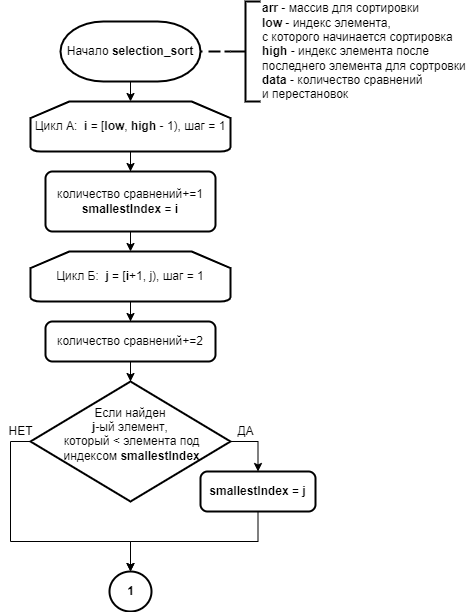


Рисунок 1. Структурная схема алгоритма функции selection\_sort().

## Функция quick\_sort()

1. Назначение:

Сортировка последовательности быстрой сортировкой.

1. Прототип функции:

template<typename T = int>

void quick\_sort(

T\* arr, // массив

int low, // индекс, с которого начинается вывод

int high, // индекс, на котром кончается вывод

help\_data& data // количество сравнений и премещений

);

1. Обращение

quick\_sort (arr, 0, size, data)

1. Описание параметров

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификаторы | Тип | Назначение | Входной/Выходной |
| arr | int \* | Массив | Выходной |
| low | int | Индекс первого элемента последовательности | Входной |
| high | int | Индекс элемента после последнего | Входной |
| data | help\_data& | Количество сравнений и перемещений | Выходной |

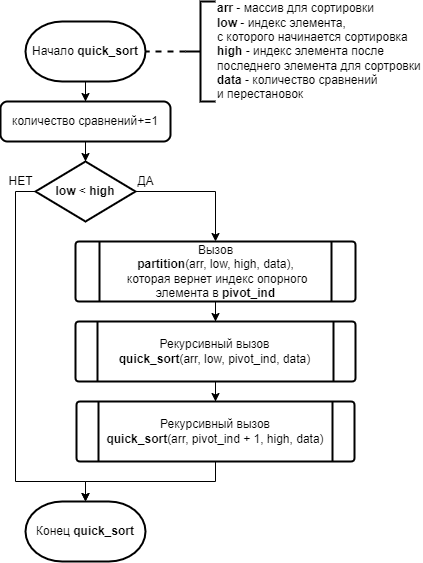


Рисунок 2. Структурная схема алгоритма функции quick\_sort().

## Функция partition ()

1. Назначение:

Вспомогательная функция для quick\_sort()..

1. Прототип функции:

template<typename T = int>

void partition(

T\* arr, // массив

int low, // индекс, с которого начинается вывод

int high, // индекс, на котром кончается вывод

help\_data& data // количество сравнений и премещений

);

1. Обращение

partition (arr, 0, size, data)

1. Описание параметров

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификаторы | Тип | Назначение | Входной/Выходной |
| arr | int \* | Массив | Выходной |
| low | int | Индекс первого элемента последовательности | Входной |
| high | int | Индекс элемента после последнего | Входной |
| data | help\_data& | Количество сравнений и перемещений | Выходной |

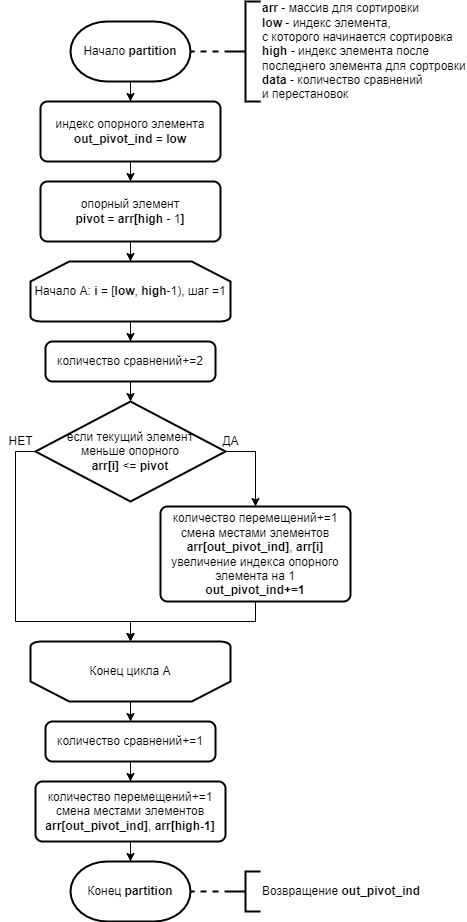


Рисунок 3. Структурная схема алгоритма функции partition().

## Функция improved\_quick\_sort()

1. Назначение

Сортировка последовательности быстрым алгоритмом

1. Прототип функции

template<typename T = int>

void improved\_quick\_sort (

T\* arr, // массив

int low, // индекс, с которого начинается вывод

int high, // индекс, на котром кончается вывод

help\_data& data // количество сравнений и премещений

);

1. Обращение

improved\_quick\_sort (arr, 0, size, data)

1. Описание параметров

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификаторы | Тип | Назначение | Входной/Выходной |
| arr | int\* | Массив | Выходной |
| low | int | Индекс первого элемента последовательности | Входной |
| high | int | Индекс элемента после последнего | Входной |
| data | help\_data& | Количество сравнений и перемещений | Выходной |

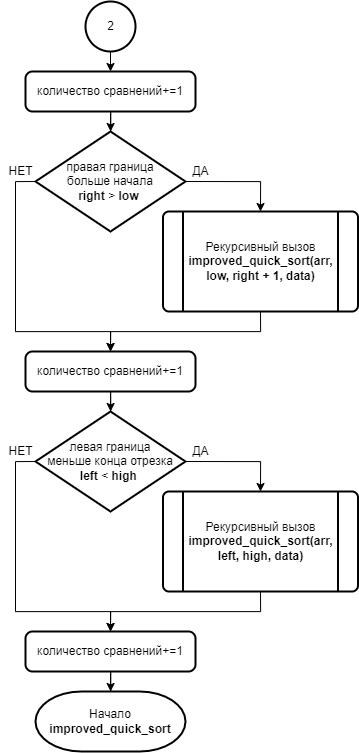
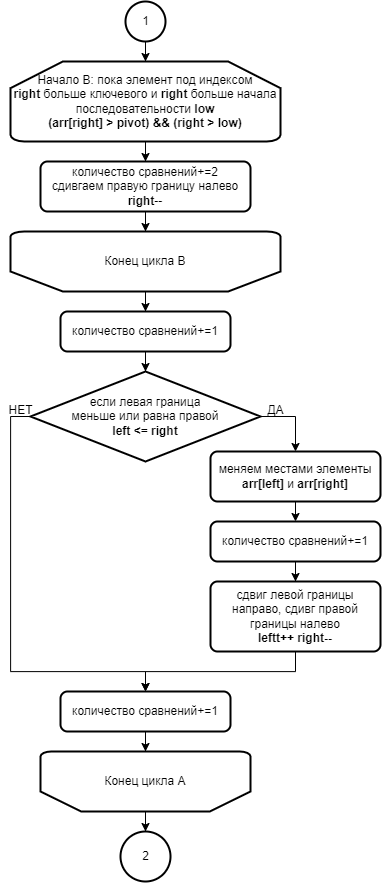
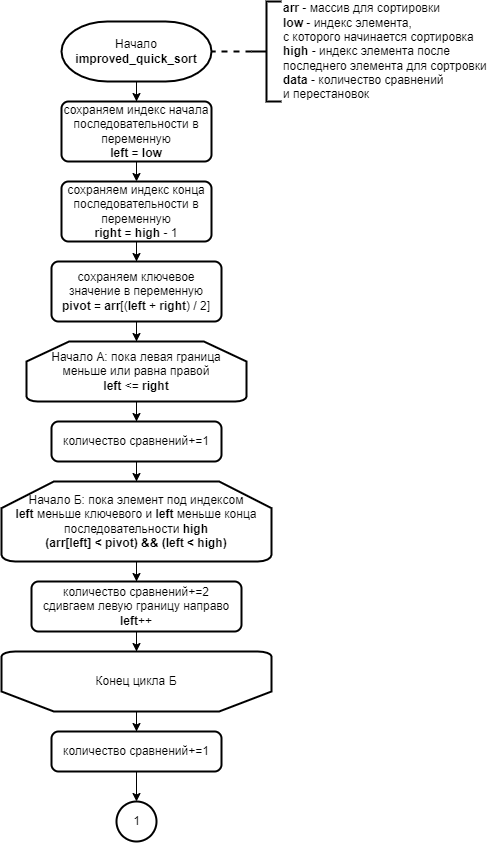


Рисунок 3. Структурная схема алгоритма функции improved\_quick\_sort()

# Код программы

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\  
\*                    кафедра № 304 2 курс 3 семестр информатика         \*  
\*-----------------------------------------------------------------------\*  
\*   Project type : solution                                             \*  
\*   Project name : matrix                                               \*  
\*   File name    : matrix.cpp                                           \*  
\*   Language     : c/c++                                                \*  
\*   Programmers  : Плоцкий Б.А. Раужев Ю. М.                            \*  
\*   Created      :  19/10/22                                            \*  
\*   Last revision:  24/10/22                                            \*  
\*   Comment(s)   : Вариант № 1                                          \*  
\*                                                                       \*  
\*   Для массива из n элементов выполнить сортировку с помощью двух      \*  
\*   указанных методов для:                                              \*  
\*       1.  заданной произвольным образом последовательности чисел,     \*  
\*       2.  уже отсортированных последовательностей в возрастающем      \*  
\*       и убывающем порядке (лучший и худший случаи для                 \*  
\*       выполнения сортировки).                                         \*  
\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
#include <iostream>  
#include <iomanip>  
#include <chrono>****using******namespace****std;  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*           ДЛЯ КОНСОЛИ         \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// убрать комментарий, если нужна  
// пошаговая печать массивов  
// РАСКОММЕНТИРОВАТЬ ДЛЯ ПЕРВОГО ЭТАПА  
#define EVERY\_STEP\_PRINT  
   
// количество таблиц в консоли  
#define NUMB\_OF\_TABLES 5  
   
// заполнение len элементов элементом symb  
#define OUT\_W(symb, len) fixed << setfill(symb) << setw(len)  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*           ДЛЯ МАССИВА         \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
#ifdef EVERY\_STEP\_PRINT  
// ЕСЛИ НУЖНА ПЕЧАТЬ КАЖДОГО ШАГА  
// минимальное и максимальное значение  
// для чисел в массиве  
#define MIN\_VALUE 10  
#define MAX\_VALUE 20  
   
// минимальный и максимальный  
// размер массива  
#define MIN\_ARR\_SIZE 10  
#define MAX\_ARR\_SIZE 15  
   
#else  
// ЕСЛИ НЕ НУЖНА ПЕЧАТЬ КАЖДОГО ШАГА  
// минимальное и максимальное значение  
// для чисел в массиве  
#define MIN\_VALUE 10  
#define MAX\_VALUE 20000  
   
// минимальный и максимальный  
// размер массива  
#define MIN\_ARR\_SIZE 1000  
#define MAX\_ARR\_SIZE 500000  
#endif  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*   ДЛЯ ПОСЛЕДОВАТЕЛЬНОСТЕЙ     \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// шаг чисел при генерации последовательности  
#define STEP 1  
   
// максимальное значение при генерации чисел  
// для упорядоченных последовательностей  
#define MAX\_GENERATE MAX\_VALUE  
   
// необходима для хранения значений  
// сравнений и перемещений****struct****help\_data  
{  
    unsigned****long******long****num\_of\_comp; // число сравнений  
    unsigned****long******long****num\_of\_swap; // число перемещений  
};  
   
// тип данных TTIME  
typedef chrono::milliseconds TTIME;  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*              П Р О Т О Т И П Ы   Ф У Н К Ц И Й                \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*         В С П О М О Г А Т Е Л Ь Н Ы Е  Ф У Н К Ц И Й          \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// возвращение случайного значения из  
// интервала (min, max)  
template<typename T =****int****>  
T rand\_num(T min, T max);  
   
// копирование массива  
template<typename T =****int****>****void****copy\_arr(  
    T\* source,  // источник данных  
    T\* target,  // место копирования данных****int****low,    // индекс начального элемента****int****high    // индекс элемента после последнего  
);  
   
// печать массива в поток  
template<typename T =****int****>****void****print\_arr(  
    T\* arr,                 // массив****int****low = 0,            // индекс, с которого начинается вывод****int****high = 1,           // индекс, на котром кончается вывод  
    ostream& stream = cout  // поток вывода  
);  
   
// ввод и проверка значений  
template<typename T =****int****>  
T input\_and\_check(T \_min, T \_max,****const******char****\* welcome\_str,****const******char****\* err\_str);  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*   Г Е Н Е Р А Ц И Я   П О С Л Е Д О В А Т Е Л Ь Н О С Т Е Й   \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// упорядочная функция по возрастанию  
template<typename T =****int****>****void****f1(T\* arr,****int****low,****int****high);  
   
// упорядочная функция по убыванию  
template<typename T =****int****>****void****f2(T\* arr,****int****low,****int****high);  
   
// рандомизация значений массива  
template<typename T =****int****>****void****randomize\_array(T\* arr,****int****low,****int****high);  
   
// чтение массива  
template<typename T =****int****>****void****read\_arr(T\* arr,****int****low,****int****high);  
   
//функция измерения времени работы другой функции  
template<typename T>  
TTIME measure\_time(  
    T\* arr,             // исходный массив****int****size,           // размер массива  
    help\_data& data,    //количество сравнений и премещений****void****(\*sort\_funcs)(  
        T\* arr,         // массив****int****low,        // индекс, с которого начинается вывод****int****high,       // индекс, на котром кончается вывод  
        help\_data& data // количество сравнений и премещений  
        )  
);  
   
// создание массива с помощю определенной функции  
template<typename T>****void****generate\_arr(  
    T\*& arr,            // массив****int****size,           // размер массива****int****ind\_of\_gen\_func // индекс функции генерации массива  
);  
   
// сортирует, выводит массив  
template<typename T>****void****draw\_table(  
    T\* arr,         // массив****int****size,           // размер массива****int****ind\_of\_gen\_func // индекс функции генерации массива  
);  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*               C О Р Т И Р О В К А   В Ы Б О Р О М             \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
template<typename T =****int****>****void****selection\_sort(  
    T\* arr,         // массив****int****low,        // индекс, с которого начинается вывод****int****high,       // индекс, на котором кончается вывод  
    help\_data& data // количество сравнений и премещений  
);  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*               Б Ы С Т Р А Я   С О Р Т И Р О В К А             \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// функция перераспределяющая элементы  
template<typename T =****int****>****int****partition(  
    T\* arr,         // массив****int****low,        // индекс, с которого начинается вывод****int****high,       // индекс, на котром кончается вывод  
    help\_data& data // количество сравнений и премещений  
);  
   
// быстрая сортировка  
// не работает с массивом,  
// размер которого >= 4000  
template<typename T =****int****>****void****quick\_sort(  
    T\* arr,         // массив****int****low,        // индекс, с которого начинается вывод****int****high,       // индекс, на котром кончается вывод  
    help\_data& data // количество сравнений и премещений  
);  
   
// быстрая сортировка, которая  
// работает с большими размерами массивов  
template<typename T =****int****>****void****improved\_quick\_sort(  
    T\* arr,         // массив****int****low,        // индекс, с которого начинается вывод****int****high,       // индекс, на котром кончается вывод  
    help\_data& data // количество сравнений и премещений  
);  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*               О С Н О В Н Ы Е   Ф У Н К Ц И И                 \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
   
// функция позволяет увидеть пошаговую работу  
// алгоритма сортировки и выводит число сравнений  
// и перестановок****void****first\_part();  
   
// функция генерирует массивы разными размерами  
// и сортирует их двумя алгоритмами сортировки****void****second\_part();  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*                      К О Н С Т А Н Т Ы                        \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// массив функций генерации последовательности  
template<typename T =****int****>****void****(\*gen\_funcs[])(T\* arr,****int****low,****int****high) =  
{  
    f1,  
    f2,  
    randomize\_array,  
    read\_arr  
};  
   
// названия функций генерации****const******char****gen\_f\_names[4][12] = {  
    "возрастание",  
    "убывание",  
    "случайная",  
    "чтение"  
};  
   
// массив функций сортировки  
template<typename T =****int****>****void****(\*sort\_funcs[])(T\* arr,****int****low,****int****high, help\_data& data) =  
{  
    selection\_sort,  
    //quick\_sort  
    improved\_quick\_sort  
};  
   
// названия функций сортировки****const******char****sort\_f\_names[2][8] = {  
    "выбором",  
    "быстрая"  
};  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*                Г Л А В Н А Я   Ф У Н К Ц И Я                  \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/****int****main()  
{  
    // установка русского языка в консоли  
    setlocale(LC\_ALL, "ru");  
   
    // инициализация генератора случайных чисел  
    srand(time(NULL));  
   
    // выполнение первого этапа  
    first\_part();  
   
    // выполнение второго этапа  
    second\_part();****return****0;  
}  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*              Р Е А Л И З А Ц И Я   Ф У Н К Ц И Й              \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*         В С П О М О Г А Т Е Л Ь Н Ы Е  Ф У Н К Ц И Й          \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// возвращение случайного значения из  
// интервала (min, max)  
template<typename T =****int****>  
T rand\_num(T min, T max)  
{****return****rand() % (max - min) + min;  
}  
   
// копирование массива  
template<typename T =****int****>****void****copy\_arr(  
    T\* source,  // источник данных  
    T\* target,  // место копирования данных****int****low,    // индекс начального элемента****int****high    // индекс элемента после последнего  
)  
{****for****(****int****i = low; i < high; i++)  
    {  
        target[i] = source[i];  
    }  
}  
   
// печать массива в поток  
template<typename T =****int****>****void****print\_arr(  
    T\* arr,         // массив****int****low,        // индекс, с которого начинается вывод****int****high,       // индекс, на котром кончается вывод  
    ostream& stream // поток вывода  
)  
{  
    // вывод элементов массива****for****(****int****i = low; i < high; i++)  
    {  
        stream << arr[i] << ' ';  
    }  
    stream << '\n';  
}  
   
// ввод и проверка значений  
template<typename T =****int****>  
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;  
}  
   
//функция измерения времени работы другой функции  
template<typename T>  
TTIME measure\_time(  
    T\* arr,****int****size,  
    help\_data& data,****void****(\*sort\_funcs)(  
        T\* arr,****int****low,****int****high,  
        help\_data& data  
        )  
)  
{  
    // начало отсчета времени  
    auto begin = chrono::steady\_clock::now();  
   
    // вызов функции сортировки  
    sort\_funcs(arr, 0, size, data);  
   
    // конец отсчета времени  
    auto end = chrono::steady\_clock::now();****return****chrono::duration\_cast<TTIME>(end - begin);  
}  
   
// создание массива с помощю определенной функции  
template<typename T>****void****generate\_arr(T\*& arr,****int****size,****int****ind\_of\_gen\_func)  
{  
    // удаление массива****if****(arr != nullptr)  
        delete[] arr;  
   
    // выделение памяти  
    arr =****new****T[size];  
   
    // генерация последовательности  
    gen\_funcs<T>[ind\_of\_gen\_func](arr, 0, size);  
}  
   
// сортирует, выводит массив  
template<typename T>****void****draw\_table(T\* arr,****int****size,****int****ind\_of\_gen\_func)  
{  
    // количество сравнений и перестановок  
    help\_data data = { 0,0 };  
   
    // отрисовка шапки таблицы  
    cout << OUT\_W('\_', 84) << '\n';  
    cout << "|\_ф.\_генерации\_|\_ф.\_сортировки\_|\_размер\_|\_время\_(мс)\_|\_\_сравнения\_\_|\_перестановки\_|\n";  
   
    // сортировка и вывод таблицы****for****(****int****ind\_of\_sort\_func = 0; ind\_of\_sort\_func < 2; ind\_of\_sort\_func++)  
    {  
        // создание копии массива  
        T\* arr\_copy =****new****T[size];  
        copy\_arr(arr, arr\_copy, 0, size);  
   
        // измерение времени работы функции  
        TTIME elapsed\_time = measure\_time(arr\_copy, size, data, sort\_funcs<T>[ind\_of\_sort\_func]);  
   
        // вывод строки таблицы  
        cout << "| " << OUT\_W(' ', 12) << gen\_f\_names[ind\_of\_gen\_func]  
            << " | " << OUT\_W(' ', 13) << sort\_f\_names[ind\_of\_sort\_func]  
            << " | " << OUT\_W(' ', 6) << size  
            << " | " << OUT\_W(' ', 10) << elapsed\_time.count()  
            << " | " << OUT\_W(' ', 11) << data.num\_of\_comp  
            << " | " << OUT\_W(' ', 12) << data.num\_of\_swap  
            << " |\n";  
        data = { 0,0 };  
   
        delete[] arr\_copy;  
    }  
    cout << OUT\_W('-', 84) << '\n';  
}  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*   Г Е Н Е Р А Ц И Я   П О С Л Е Д О В А Т Е Л Ь Н О С Т Е Й   \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// упорядочная функция по возрастанию  
template<typename T =****int****>****void****f1(T\* arr,****int****low,****int****high)  
{  
    //k, b - коэффициенты прямой  
    T k = MAX\_GENERATE / (STEP \* (high - low + 1));  
    T b = 0;****double****x = 0.0; // координата x  
   
    // заполнение массива****for****(****int****i = low; i < high; i++, x += STEP)  
    {  
        arr[i] = k \* x + b;  
    }  
}  
   
// упорядочная функция по убыванию  
template<typename T =****int****>****void****f2(T\* arr,****int****low,****int****high)  
{  
    //k, b - коэффициенты прямой  
    T k = MAX\_GENERATE / (STEP \* (high - low + 1));  
    T b = 0;****double****x = high - low - 1; // координата x  
   
    // заполнение массива****for****(****int****i = low; i < high; i++, x -= STEP)  
    {  
        arr[i] = x \* k + b;  
    }  
}  
   
// рандомизация значений массива  
template<typename T =****int****>****void****randomize\_array(T\* arr,****int****low,****int****high)  
{****for****(****int****i = low; i < high; i++)  
    {  
        arr[i] = rand\_num(MIN\_VALUE, MAX\_VALUE);  
    }  
}  
   
// чтение массива  
template<typename T =****int****>****void****read\_arr(T\* arr,****int****low,****int****high)  
{****for****(****int****i = low; i < high; i++)  
    {  
        cin >> arr[i];  
    }  
}  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*               C О Р Т И Р О В К А   В Ы Б О Р О М             \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
template<typename T =****int****>****void****selection\_sort(T\* arr,****int****low,****int****high, help\_data& data)  
{  
#ifdef EVERY\_STEP\_PRINT  
    cout << "\tдо    изменений: ";  
    print\_arr(arr, low, high);  
#endif // EVERY\_STEP\_PRINT  
   
    // проходимся по всем элементам массива  
    // кроме последнего, он уже будет в  
    // нужном порядке, когда мы дойдем до него****for****(****int****i = low; i < high - 1; i++)  
    {  
        // индекс наименьшего элемента****int****smallestIndex = i;  
   
        // ищем еще более меньший элемент,  
        // чем под индексом smallestIndex****for****(****int****j = i + 1; j < high; j++)  
        {  
            // если был найден элемент,  
            // который меньше элемента под  
            // индексом smallestIndex, то  
            // обновляем smallestIndex  
            data.num\_of\_comp++;****if****(arr[j] < arr[smallestIndex])  
            {  
                smallestIndex = j;  
            }  
        }  
   
        // меняем местами текущий элемент и наименьший  
        data.num\_of\_swap++;  
        swap(arr[i], arr[smallestIndex]);  
   
        // если нужна пошаговая печать  
#ifdef EVERY\_STEP\_PRINT  
        cout << "\t" << setfill(' ') << setw(3) << i << " с "  
            << setfill(' ') << setw(2) << smallestIndex << " индекс: ";  
        print\_arr(arr, low, high);  
#endif // EVERY\_STEP\_PRINT  
    }  
}  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*               Б Ы С Т Р А Я   С О Р Т И Р О В К А             \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// функция перераспределяющая элементы  
template<typename T =****int****>****int****partition(T\* arr,****int****low,****int****high, help\_data& data)  
{  
#ifdef EVERY\_STEP\_PRINT  
    cout << "\nлевый: " << low << " правый: " << high << " partition: \n";  
    cout << "\tдо    изменений: ";  
    print\_arr(arr, low, high);  
#endif // EVERY\_STEP\_PRINT  
   
    // индекс, в которм в конце алгоритма  
    // окажется опорный элемент****int****out\_pivot\_ind = low;  
   
    // опорный элемент  
    T pivot = arr[high - 1];****for****(****int****i = low; i < high - 1; i++)  
    {  
   
        // если текущий элемент меньше опорного  
        data.num\_of\_comp++;****if****(arr[i] <= pivot)  
        {  
            // меняем местами текущий элемент  
            // и элемент под индексом out\_ind  
            data.num\_of\_swap++;  
            swap(arr[out\_pivot\_ind++], arr[i]);  
   
            // если нужна пошаговая печать  
#ifdef EVERY\_STEP\_PRINT  
            cout << "\t" << setfill(' ') << setw(3) << i - low << " с "  
                << setfill(' ') << setw(2) <<  
                (out\_pivot\_ind == 0 ? out\_pivot\_ind : out\_pivot\_ind - 1) - low  
                << " индекс: ";  
            print\_arr(arr, low, high);  
#endif // EVERY\_STEP\_PRINT  
        }  
    }  
   
    // ставим опорный элемент в позицию out\_pivot\_ind  
    data.num\_of\_swap++;  
    swap(arr[out\_pivot\_ind], arr[high - 1]);  
   
    // пошаговая печать  
#ifdef EVERY\_STEP\_PRINT  
    cout << "\t" << setfill(' ') << setw(3) << high - 1 << " с "  
        << setfill(' ') << setw(2) <<  
        (out\_pivot\_ind == 0 ? out\_pivot\_ind : out\_pivot\_ind)  
        << " индекс: ";  
    print\_arr(arr, low, high);  
   
    cout << "\tпосле  изменен.: ";  
    print\_arr(arr, low, high);  
#endif // EVERY\_STEP\_PRINT****return****out\_pivot\_ind;  
}  
   
// быстрая сортировка  
// не работает с массивом,  
// размер которого >= 4000  
template<typename T =****int****>****void****quick\_sort(T\* arr,****int****low,****int****high, help\_data& data)  
{  
#ifdef EVERY\_STEP\_PRINT  
    cout << "\nлевый: " << low << " правый: " << high << "\n";  
    cout << "\tдо    изменений: ";  
    print\_arr(arr, low, high);  
#endif // EVERY\_STEP\_PRINT  
    // если минимальный индекс меньше максимального****if****(low < high)  
    {  
        // переставляем элементы относительно  
        // определенного индекса, возвращаем  
        // этот индекс****int****pivot\_ind = partition(arr, low, high, data);  
   
        // переставляем левую часть массива  
        // относительно индекса ключевого элемента  
        quick\_sort(arr, low, pivot\_ind, data);  
   
        // переставляем правую часть массива  
        // относительно индекса ключевого элемента  
        quick\_sort(arr, pivot\_ind + 1, high, data);  
   
        // если нужна пошаговая печать  
#ifdef EVERY\_STEP\_PRINT  
        cout << "\nлевый: " << low << " правый: " << high << " quick sort: ";  
        print\_arr(arr, low, high);  
#endif // EVERY\_STEP\_PRINT  
    }  
}  
   
// быстрая сортировка, которая  
// работает с большими размерами массивов  
template<typename T =****int****>****void****improved\_quick\_sort(T\* arr,****int****low,****int****high, help\_data& data)  
{  
#ifdef EVERY\_STEP\_PRINT  
    cout << "\nлевый: " << low << " правый: " << high << "\n";  
    cout << "\tдо    изменений: ";  
    print\_arr(arr, low, high);  
#endif // EVERY\_STEP\_PRINT  
   
    // индекс числа слева от ключевого****int****left = low;  
    // индекс числа справа от ключевого****int****right = high - 1;  
    // ключевое число  
    T key = arr[(left + right) / 2];  
   
    // переставляем элементы в массиве****while****(left <= right)  
    {  
        // двигаем индекс левого числа вправо  
        // если число под левым индексом  
        // меньше ключевого элемента****while****(arr[left] < key)  
        {  
            data.num\_of\_comp++;  
            left++;  
        }  
        data.num\_of\_comp++;  
   
        // двигаем индекс правого числа влево  
        // если число под правым индексом  
        // больше ключевого элемента****while****(arr[right] > key)  
        {  
            data.num\_of\_comp++;  
            right--;  
        }  
        data.num\_of\_comp++;  
   
        // если индекс левого числа <=  
        // индексу правого, то меняем  
        // значения под этими индексами  
        // и сдвигаем иба индекса к центру  
        // отрезка (low, high)****if****(left <= right)  
        {  
            swap(arr[left], arr[right]);  
            data.num\_of\_swap++;  
#ifdef EVERY\_STEP\_PRINT  
            cout << "\t" << setfill(' ') << setw(3) << left - low << " с "  
                << setfill(' ') << setw(2) << right - low << " индекс: ";  
            print\_arr(arr, low, high);  
#endif // EVERY\_STEP\_PRINT  
            left++;  
            right--;  
        }  
        data.num\_of\_comp++;  
    }  
   
    // пошаговая печать  
#ifdef EVERY\_STEP\_PRINT  
    cout << "\tпосле  изменен.: ";  
    print\_arr(arr, low, high);  
#endif // EVERY\_STEP\_PRINT  
   
    // если правый индекс не дошел до  
    // начала отрезка (low, high)****if****(right > low)  
    {  
        improved\_quick\_sort(arr, low, right + 1, data);  
    }  
   
    // если левый индекс не дошел до  
    // конца отрезка (low, high)****if****(left < high)  
    {  
        improved\_quick\_sort(arr, left, high, data);  
    }  
}  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*               О С Н О В Н Ы Е   Ф У Н К Ц И И                 \*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
// функция позволяет увидеть пошаговую работу  
// алгоритма сортировки и выводит число сравнений  
// и перестановок****void****first\_part()  
{  
    cout << OUT\_W('=', 27) << '\n';  
    cout << "||\tПЕРВЫЙ ЭТАП" << "\t||\n";  
    cout << OUT\_W('=', 27) << '\n';  
   
    // считывание типа последовательности****int****type\_of\_gen = input\_and\_check(1, 3,  
        "Выберете способ генерации последовательности:\n\  
    1. по возрастанию\n\t2. по убыванию\n\t3. случайная\n",  
        "Вводимое значение должно быть 1,2 или 3\n"  
    );****int****\* arr = nullptr;     // массив для сортировки****int****\* arr\_deaf = nullptr;    // массив с изначальной полседовательностью****int****size = 15;  // размер массивов  
   
    // создание массива с полседовательностью  
    generate\_arr(arr, size, type\_of\_gen - 1);  
   
    arr\_deaf =****new******int****[size];   // выделение памяти  
   
    // копирование массива. необходимо для вывода  
    copy\_arr(arr, arr\_deaf, 0, size);  
   
    // считывание типа алгоритма сортировки****int****type\_of\_alg = input\_and\_check(1, 2,  
        "Выберете алгоритм сортировки:\n\  
    1. выбором\n\t2. быстрая\n",  
        "Вводимое значение должно быть 1 или 2\n"  
    );  
   
    // количество сравнений и перестановок  
    help\_data data = { 0,0 };  
   
    // генерация последовательности в соответствии с вводом  
    sort\_funcs<****int****>[type\_of\_alg - 1](arr, 0, size, data);  
   
    // вывод результатов  
    cout << "\nНачальный массив: ";  
    print\_arr(arr\_deaf, 0, size);  
   
    cout << "Конечный  массив: ";  
    print\_arr(arr, 0, size);  
   
    cout << "Число  сравнений: " << data.num\_of\_comp << '\n';  
    cout << "Число  пересылок: " << data.num\_of\_swap << "\n\n\n";  
   
    // очистка памяти  
    delete[] arr;  
    delete[] arr\_deaf;  
}  
   
// функция генерирует массивы разными размерами  
// и сортирует их двумя алгоритмами сортировки****void****second\_part()  
{  
    cout << OUT\_W('=', 27) << '\n';  
    cout << "||\tВТОРОЙ ЭТАП" << "\t||\n";  
    cout << OUT\_W('=', 27) << '\n';  
   
    // считывание генерирующей функции****int****ind\_of\_gen\_func = input\_and\_check(1, 4,  
        "Введите номер генерирующей функции:\n\  
    1. по возрастанию\n\t2. по убыванию\n\t3. случайная\n",  
        "Значения должны быть 1, 2 или 3\n"  
    );  
    // индексы должны начинаться с 0  
    ind\_of\_gen\_func--;  
   
    // массив для последовательности  
    unsigned****int****\* arr = nullptr;  
   
    // выполняем генерацию массива и  
    // выполняем сортировки****for****(****int****size = MIN\_ARR\_SIZE; size <= MAX\_ARR\_SIZE;  
        size += (MAX\_ARR\_SIZE - MIN\_ARR\_SIZE) / (NUMB\_OF\_TABLES - 1)  
        )  
    {  
        // генерация массива  
        generate\_arr(arr, size, ind\_of\_gen\_func);  
   
        // печать таблицы  
        draw\_table(arr, size, ind\_of\_gen\_func);  
    }  
}  
   
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End Of main.cpp File \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

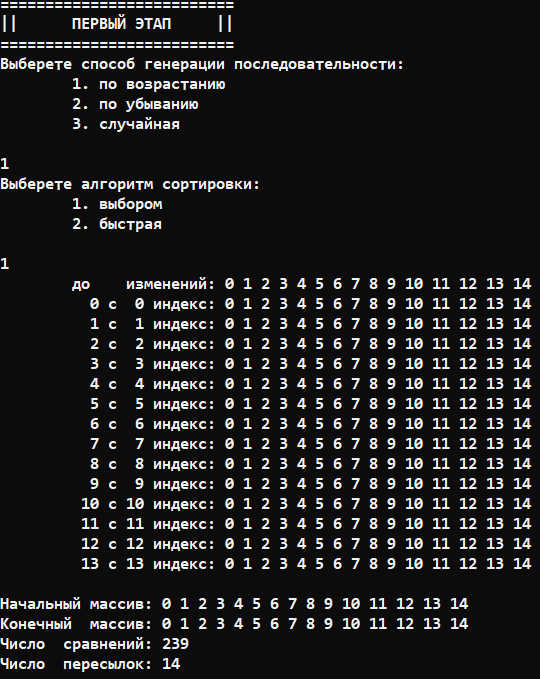
# Сопоставительный анализ

## Результаты работы программ по этапам

### Тест 1

Последовательность генерируется по возрастанию.

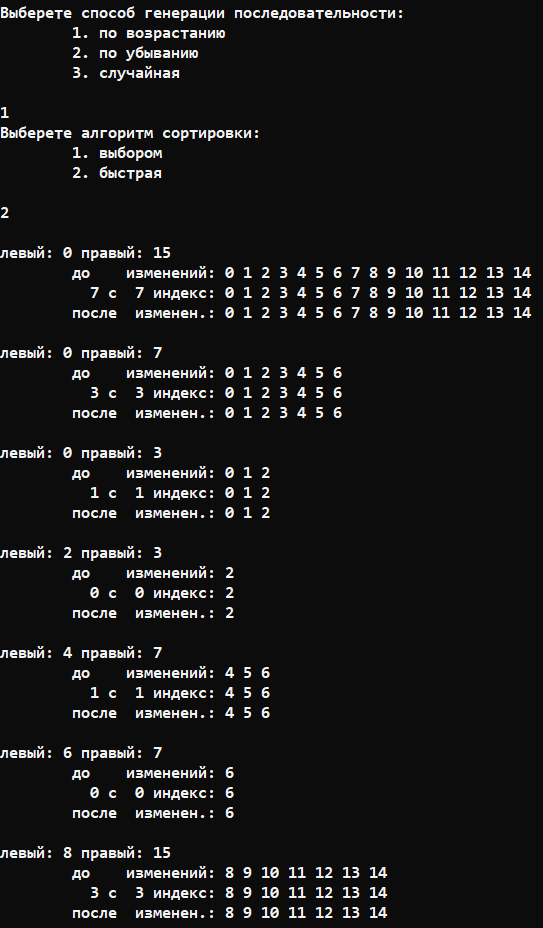
Сортировка выбором.

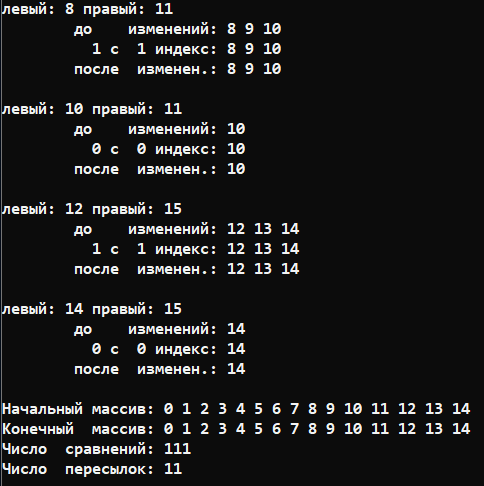


### Тест 2

Последовательность генерируется по возрастанию.

Сортировка быстрая.

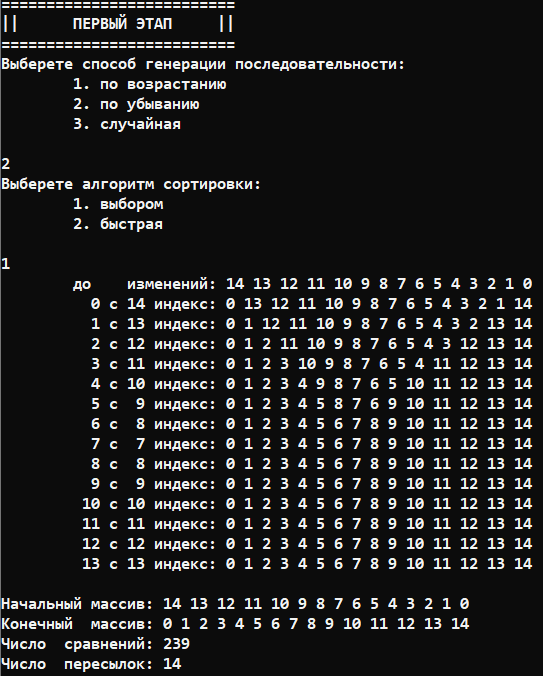




### Тест 3

Последовательность генерируется по убыванию.

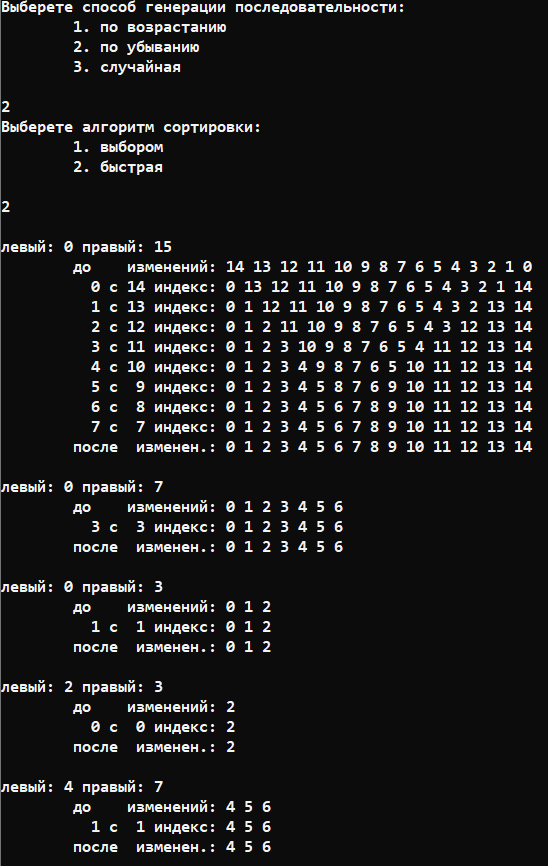
Сортировка выбором.

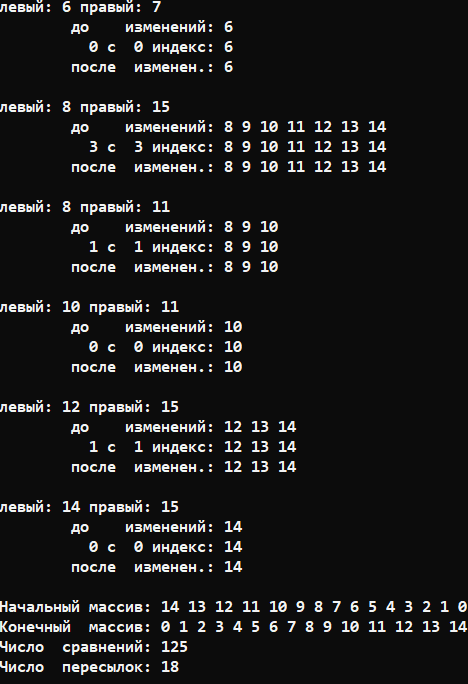


### Тест 4

Последовательность генерируется по убыванию.

Сортировка быстрая.

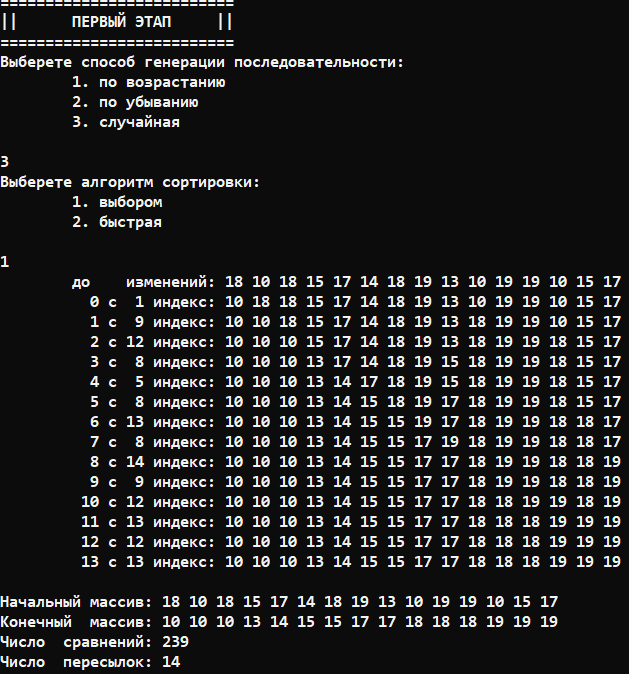




### Тест 5

Последовательность генерируется случайно.

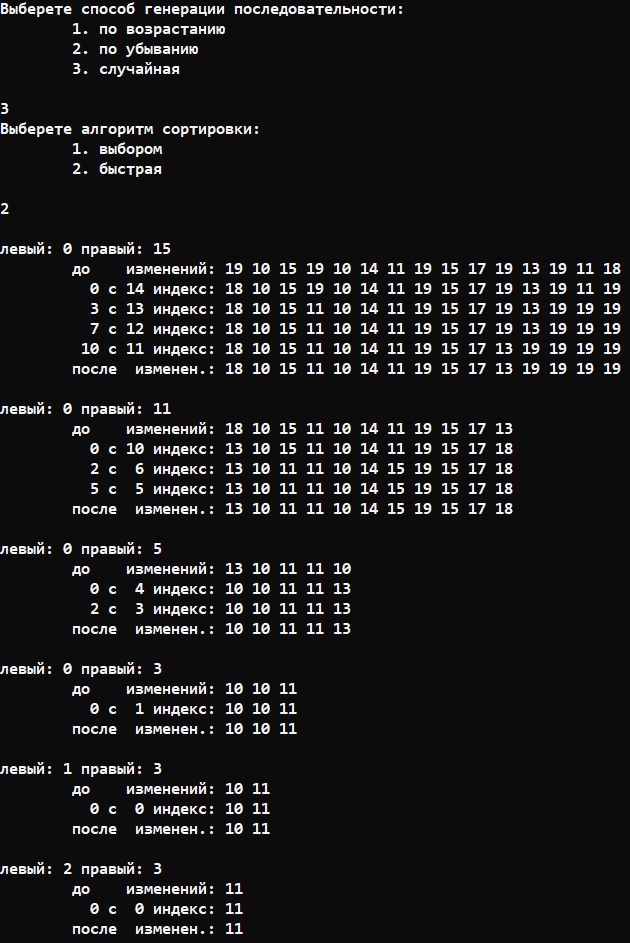
Сортировка выбором.

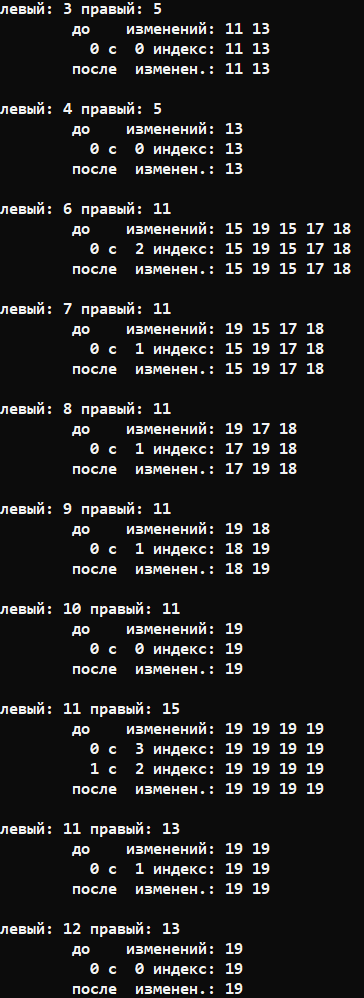


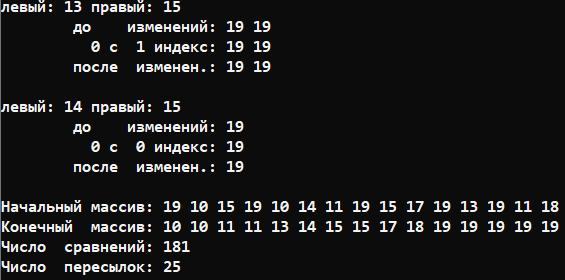
### Тест 6

Последовательность генерируется случайно.

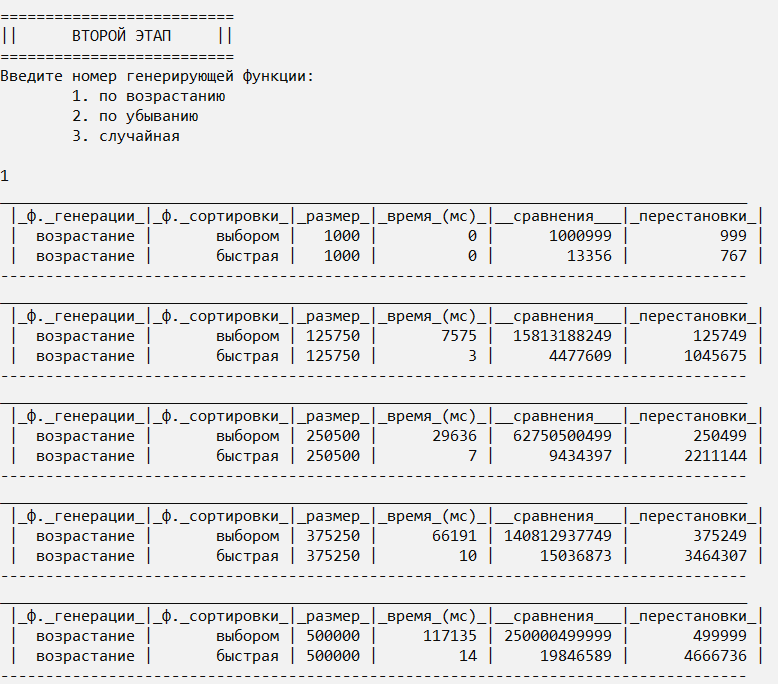
Сортировка быстрая.





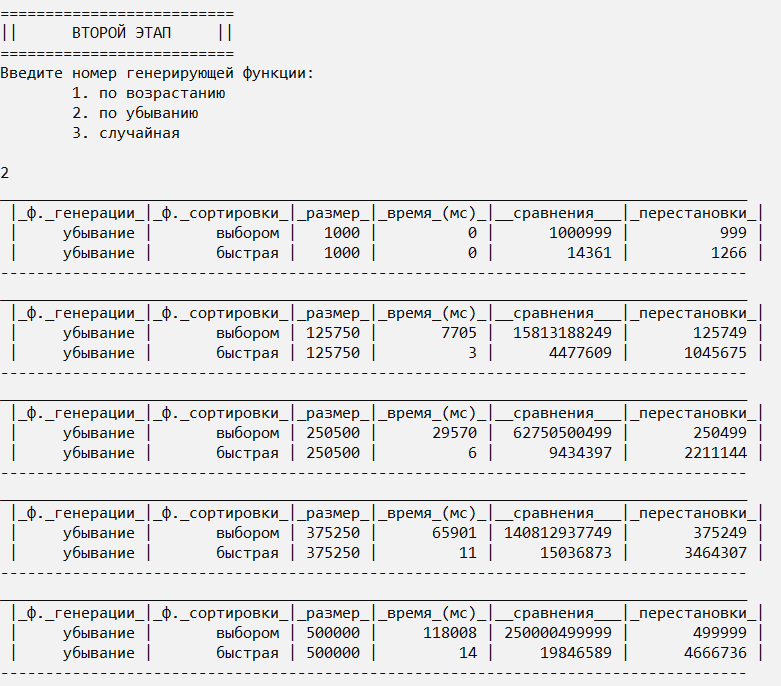


## Сравнение алгоритмов на больших размерах массивов

Последовательность генерируется по возрастанию. 

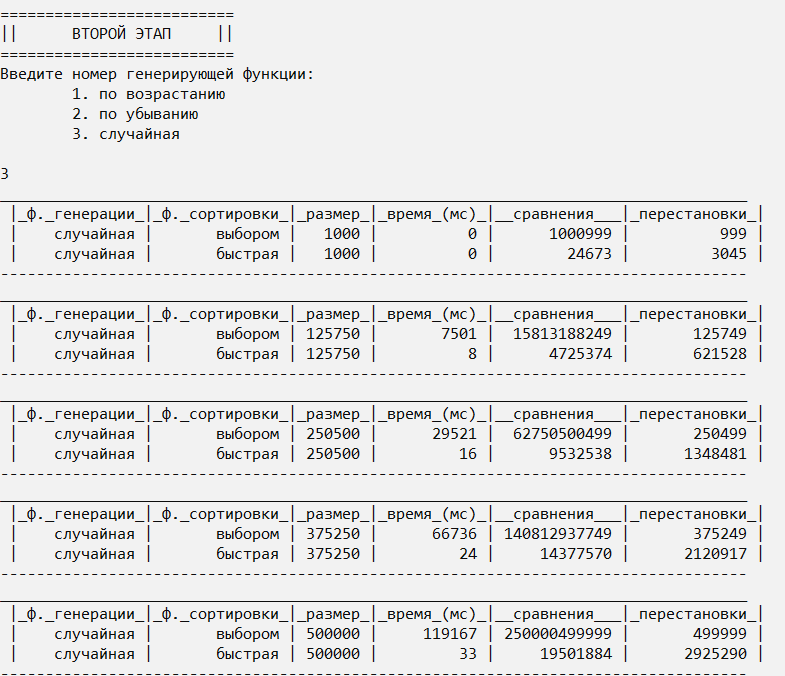
Графики:

Последовательность генерируется по убыванию.



Графики:

Последовательность генерируется случайно.



Графики:

# Вывод

1. Для всех последовательностей (по убыванию/возрастанию, случайно) быстрая сортировка (Q\_S) затрачивает наименьшее количество времени, чем сортировка выбором (S\_S).
2. Для случайной последовательности быстрая сортировка (Q\_S) затрачивает большее количество времени, чем для последовательностей по убыванию/возрастанию.+
3. Количество сравнений, выполненных сортировкой выбором (S\_S), гораздо больше, чем количество сравнений у быстрой сортировки (Q\_S).
4. Количество перемещений, выполненных сортировкой выбором (S\_S), гораздо меньше, чем количество перемещений у быстрой сортировки (Q\_S).