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 М3О-209Б-22

Кожевников А. В., Томчук Д. М.

Проверила доцент, к.т.н., Дмитриева Е.А.

Москва 2023 г.

Оглавление

[Структурные схемы алгоритмов 4](#_Toc149512729)

[Функция main 4](#_Toc149512730)

[Функция Ordered\_Array\_Search 5](#_Toc149512731)

[Функция betterLinearSearch 6](#_Toc149512732)

[Функция sentinelLinearSearch 7](#_Toc149512733)

[Функция Binary\_Search 8](#_Toc149512734)

[Код программы 10](#_Toc149512735)

[Тестирование программы 14](#_Toc149512736)

[Тестирование корректных режимов 14](#_Toc149512737)

[Сопоставительный анализ 17](#_Toc149512738)

[Вывод 22](#_Toc149512739)

**Задание**

Для массива из ***n*** элементов выполнить сортировку с помощью двух указанных методов (по вариантам) 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, 50000, 100000 ). Сравнить **время выполнения алгоритмов, число необходимых сравнений и число пересылок**.

**Варианты заданий**

|  |  |
| --- | --- |
| № вар. | метод |
| 1 | выбором, быстрая |
| 2 | выбором, слиянием |
| 3 | выбором, Шелла |
| 4 | вставкой, быстрая |
| 5 | вставкой, слиянием |
| 6 | вставкой, Шелла |
| 7 | шейкерная, быстрая |
| 8 | шейкерная, слиянием |
| 9 | шейкерная, Шелла |

**Отчет по лабораторной работе должен содержать:**

* задание;
* структурные схемы главной функции, алгоритмов сортировки варианта работы;
* тексты программ;
* результаты работы программ по этапам и результаты сопоставительного анализа – в виде таблиц;
* выводы по работе, выводы о целесообразности применения методов для различных наборов данных.

# Структурные схемы алгоритмов

## Функция main

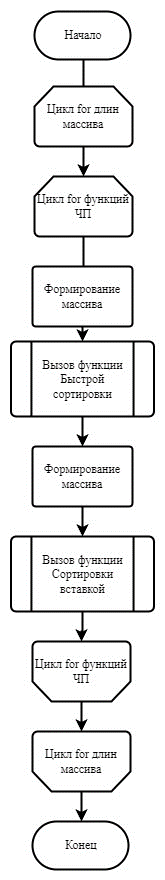


Рисунок 1. Структурная схема алгоритма функции main.

## Функция quickSort

1. Назначение:

Сортировка методом разбиения массива относительно ключевого элемента и рекурсивного вызова quickSort для подмассивов, полученных в результате разбиения

2. Прототип функции:

void quickSort(int\* Array, int begin, int end, unsigned \_\_int64\* answer, int len)

3. Обращение к функции:

quickSort(main\_array, 0, lenMain - 1, answer, lenMain)

4. Блок-схема:

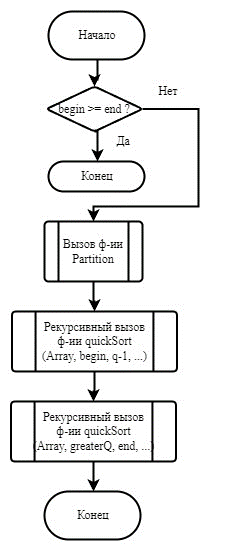


Рисунок 2. Структурная схема алгоритма функции quickSort

5. Описание параметров:

|  |  |  |  |
| --- | --- | --- | --- |
| Имя параметра | Тип параметра | Назначение | Вход или выход |
| Array | int\* | Сортируемый массив | Вход |
| begin | Int | Начальный индекс массива (подмассива) | Вход |
| end | Int | Конечный индекс массива (подмассива) | Вход |
| answer | unsigned \_\_int64\* | Массив ответов | Вход |
| len | int | Размер всего массива | Вход |

## Функция Partition

1. Назначение:

Функция расставляет элементы, меньшие опорного, слева от опорного, а элементы, большие опорного, справа от опорного, и находит индекс опорного элемента, в результате.

2. Прототип функции:

void Partition(int\* Array, int begin, int end, int& Q, int& greaterQ, unsigned \_\_int64\* answer, int len)

3. Обращение к функции:

Partition(Array, begin, end, qIndex, greaterIndex, answer, len);

4. Блок-схема:

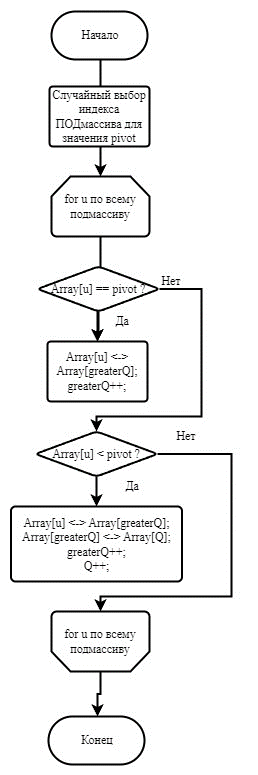


Рисунок 3. Структурная схема алгоритма функции Partition

5. Описание параметров:

|  |  |  |  |
| --- | --- | --- | --- |
| Имя параметра | Тип параметра | Назначение | Вход или выход |
| Array | int\* | Сортируемый массив | Вход |
| begin | Int | Начальный индекс массива (подмассива) | Вход |
| end | Int | Конечный индекс массива (подмассива) | Вход |
| Q | Int | Индекс опорного элемента | Вход |
| greaterQ | int | Индекс ближайшего числа, большего опорного элемента | Вход |
| answer | unsigned \_\_int64\* | Массив ответов | Вход |

## Функция Insertion\_Sort

1. Назначение:

Функция сортировки методом вставки

2. Прототип функции:

void Insertion\_Sort(int\* Array, int len, unsigned \_\_int64\* answer)

3. Обращение к функции:

Insertion\_Sort(main\_array, lenMain, answer)

4. Блок-схема:

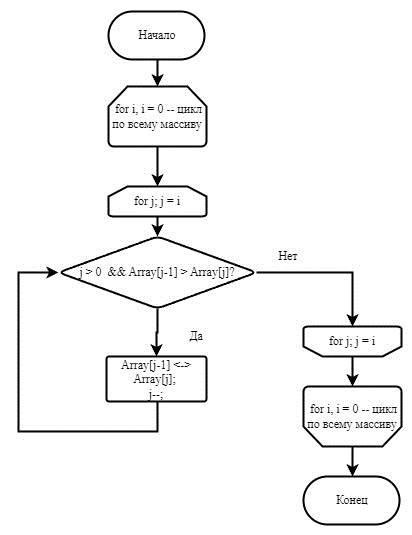


Рисунок 4. Структурная схема алгоритма функции Insertion\_Sort

5. Описание параметров:

|  |  |  |  |
| --- | --- | --- | --- |
| Имя параметра | Тип параметра | Назначение | Вход или выход |
| len | int | Длина массива | Вход |
| array | Int\* | Массив данных | Вход |
| answer | unsigned \_\_int64\* | Массив ответов | Вход |

# Код программы

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Программирование \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*Project type :Win64 Console Application \*

\*Project name :lab1\_search.sln \*

\*File name :lab1\_search.cpp \*

\*Language :CPP, MSVS 2022 \*

\*Programmers :Кожевников Артем Вадимович, Томчук Дмитрий Максимович, М3О-209Б-22 \*

\*Modified By : \*

\*Created :18.10.2023 \*

\*Last revision:29.10.2023 \*

\*Comment : \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <chrono>

#include <iostream>

#include <fstream>

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* П Р О Т О Т И П Ы Ф У Н К Ц И Й \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int betterLinearSearch(int\* array, int len, int target, int\*answer); // Последовательный поиск, ВОЗВРАЩАЕТ ИСКОМЫЙ ИНДЕКС И КОЛИЧЕСТВО СРАВНЕНИЙ

int sentinelLinearSearch(int\* array, int len, int target, int\* answer); // Ускоренный последовательный поиск, ВОЗВРАЩАЕТ ИСКОМЫЙ ИНДЕКС И КОЛИЧЕСТВО СРАВНЕНИЙ

int Ordered\_Array\_Search(int\* array, int len, int target, int\* answer); //последовательный поиск в упорядоченном массиве, ВОЗВРАЩАЕТ ИСКОМЫЙ ИНДЕКС И КОЛИЧЕСТВО СРАВНЕНИЙ=0

int Binary\_Search(int\* array, int len, int target, int\* answer); //бинарный поиск, ВОЗВРАЩАЕТ ИСКОМЫЙ ИНДЕКС И КОЛИЧЕСТВО СРАВНЕНИЙ=0

void generate(int\* array, int len); // Генерация нового массива

void linearAscendINT(int\* Array, int len); //Генерация возрастающей последовательности

void noteRes(const char\* filename, int lenArray, int time, int is\_compare); // Запись в файл

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Г Л О Б А Л Ь Н Ы Е П Е Р Е М Е Н Н Ы Е \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// К О Н С Т А Н Т Ы

const int functions\_total = 4;

const int shift = 19000; // Увеличение размерности массива

const char\* files[functions\_total\*3] = { "BLSbest.txt", "BLSnormal.txt", "BLSworst.txt", "SLSbest.txt", "SLSnormal.txt", "SLSworst.txt",

"OASbest.txt", "OASnormal.txt", "OASworst.txt", "BSbest.txt", "BSnormal.txt", "BSworst.txt" };

// П Е Р Е М Е Н Н Ы Е

int(\*func\_array[functions\_total])(int\*, int, int, int\*) = { betterLinearSearch, sentinelLinearSearch, Ordered\_Array\_Search, Binary\_Search }; // МАССИВ ФУНКЦИЙ

int answer[2] = { 0, 0 }; // Результат работы алгоритма: индекс и количество сравнений

int key = 0; // № ключа

int step = 0; // шаг для перехода к первому, среднему и последнему элементам внутри цикла

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Г Л А В Н А Я П Р О Г Р А М М А \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main()

{

setlocale(LC\_ALL, "Russian");

for (int funcN = 0; funcN < functions\_total; funcN++) // Поиск с помощью BLS, SLS, OAS, BS

{

cout << "Функция №" << funcN << endl;

for (int curr\_len = 10000; curr\_len <= 200000; curr\_len += shift) // Прогоняем поиск по 10-ти массивам с размерностями от 10000 до 200000

{

int\* array\_main = new int[curr\_len]; // Инициализация массива с заданной длиной

if (funcN < functions\_total / 2) generate(array\_main, curr\_len); // Случайный массив, если поиск BLS || SLS

else linearAscendINT(array\_main, curr\_len); // Упорядоченный массив, если поиск OAS || BS

step = curr\_len / 2 - 1; // Шаг для выбора разных ключей

for (int keyN = 0; keyN < 3; keyN++)

{

answer[0], answer[1] = 0; // Инициализируем ответ заново

auto begin = std::chrono::steady\_clock::now();

func\_array[funcN](array\_main, curr\_len, array\_main[key], answer); // Вы

auto end = std::chrono::steady\_clock::now();

auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::microseconds>(end - begin); // ЕСЛИ ЧТО НУЖНО ПЕРЕВЕСТИ В СЕКУНДЫ

cout << "Длина массива: " << curr\_len << "\t№ Ключа: " << key << "\tСравнений: " << answer[1] << "\tВремя работы: " << elapsed\_ms.count() << endl;

noteRes(files[(functions\_total - 1) \* funcN + keyN], curr\_len, elapsed\_ms.count(), answer[1]); // Запись в файл

key += step; // Переход на следующий ключ

} // for keyN

key = 0;

delete[] array\_main;

} // for curr\_len

} // for funcN

} // main

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Р Е А Л И З А Ц И Я Ф У Н К Ц И Й \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void generate(int\* array, int len)

{

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

{

array[i] = rand();

} // for i

} // generate

int betterLinearSearch(int\* array, int len, int target, int\* answer)

{

int counter = 0; // Количество сравнений в if

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

{

counter++;

if (array[i] == target) {

answer[0] = i; answer[1] = i + counter;

return 1; // Элемент найден

} // if

} // for i

return 0; // Элемент не найден

} // betterLinearSearch

int sentinelLinearSearch(int\* array, int len, int target, int\* answer)

{

int last = array[len - 1];

array[len - 1] = target;

int i = 0;

while (array[i] != target)

{

i++;

}

array[len - 1] = last;

if (array[i] == target) // Возвращаем количество сравнений

{

answer[0] = i; answer[1] = i + 1;

return 1;

}

return 0;

} // betterLinearSearch

int Ordered\_Array\_Search(int\* array, int len, int target, int\* answer) // Имеет ту же сложность, что и лекционный вариант: O(n) - в худшем случае

{

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

{

if (array[i] == target)

{

answer[0] = i;

return 1;

}

}

return 0;

} // Ordered\_Array\_Search

int Binary\_Search(int\* array, int len, int target, int\* answer) {

int min = 0;

int max = len;

while (max > min) {

int m = (max + min) / 2;

if (array[m] < target) {

min = m + 1;

}

if (array[m] > target) {

max = m - 1;

}

if (array[m] == target)

{

answer[0] = m;

return 1;

}

}

if (array[min] == target) {

answer[0] = min;

return 1;

}

else { return 0; }

} // Binary\_Search

void linearAscendINT(int\* Array, int len)

{

int inf = 0;

int sup = 32767;

double coeff = (abs(sup - inf) \* 1.0) / len; // Угловой коэффициент

int Const = inf; // На сколько поднять/опустить

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

{

Array[i] = coeff \* i + Const; // Отбрасываем дробную часть

} // for i

} // void linearAscendINT

void noteRes(const char\* filename, int lenArray, int time, int is\_compare)

{

ofstream fout(filename, ios::app);

fout << lenArray << " " << time;

if (is\_compare) fout << " " << is\_compare << "\n"; // Если BLS/SLS, то значение, возвращаемое функцией, и есть количество сравнений

else fout << "\n";

fout.close();} // void noteRes

# Тестирование программы

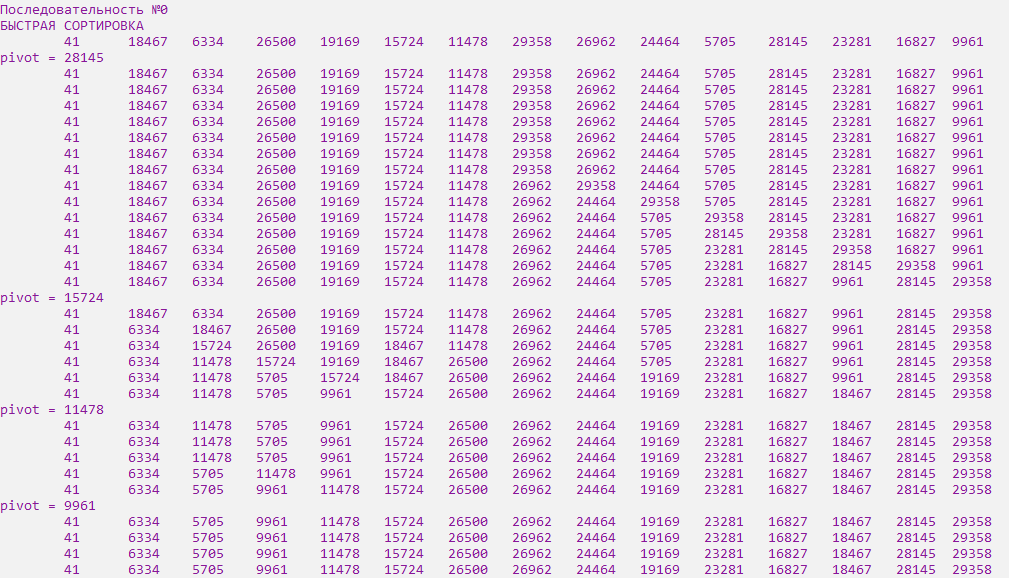
## Программа не нуждается в проверке на некорректные данные, поскольку все исходные данные заранее заданы в программе.

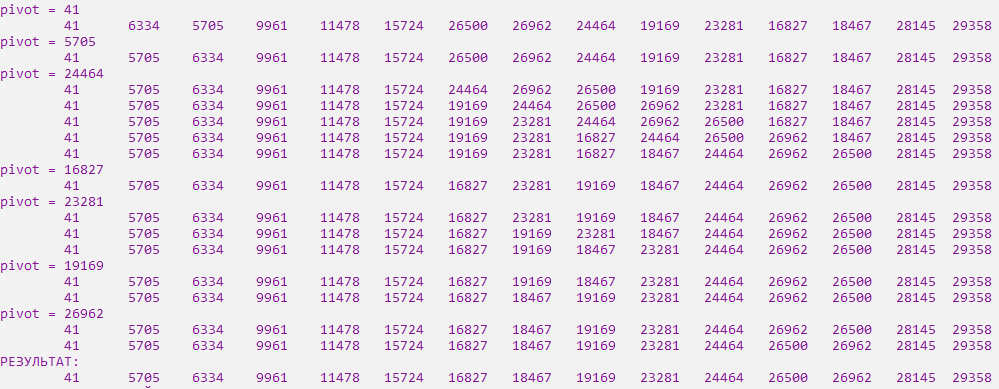
## Тестирование корректных режимов

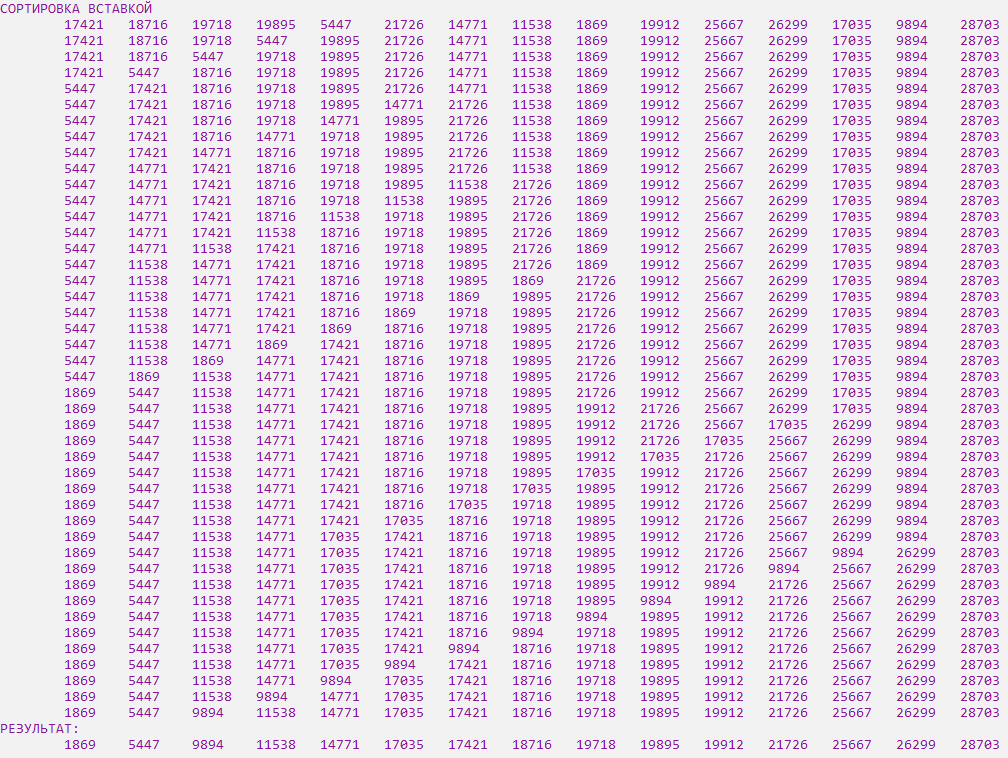
Тест 1

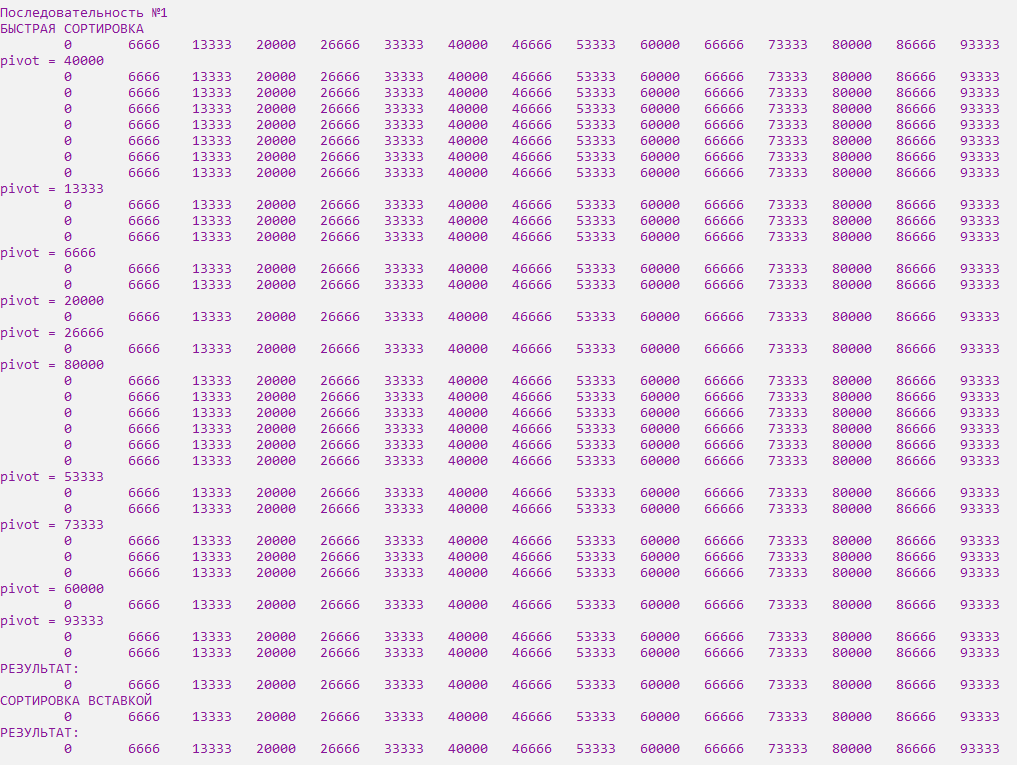
Цель: Проверить корректность двух алгоритмов сортировки при длинах массивов = 15 в трех случаях: худшем, лучшем и случайном; вывести промежуточные вычисления в виде целого массива на разных итерациях сортировки.

Полученный результат:

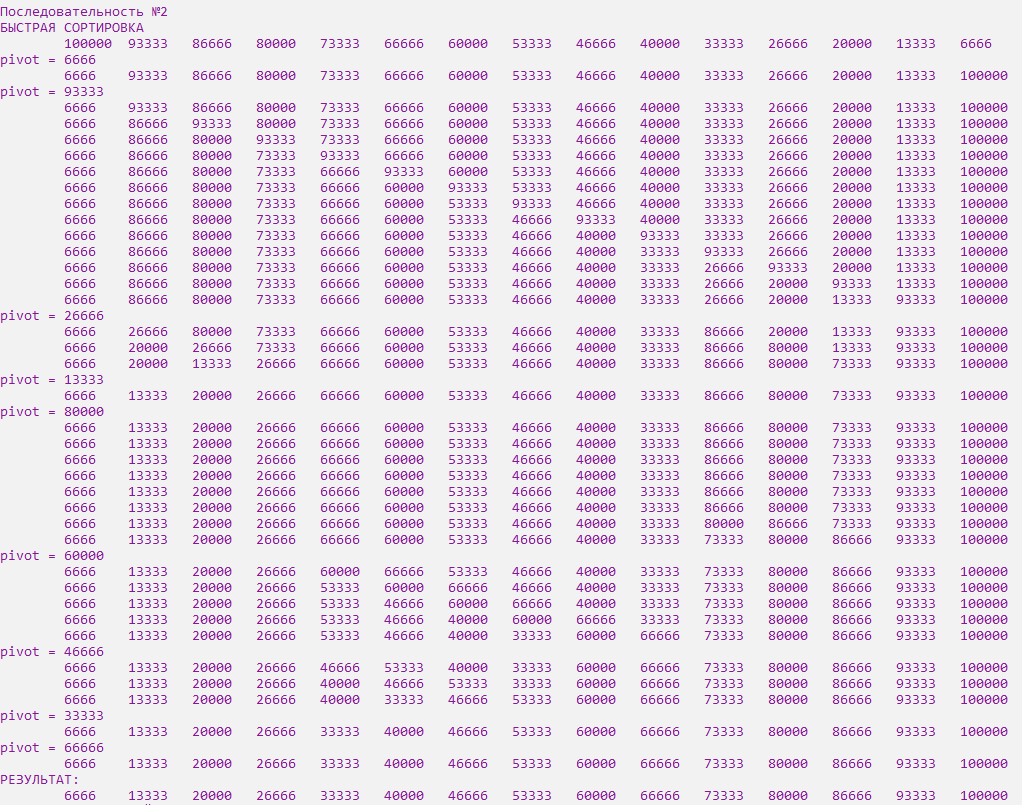


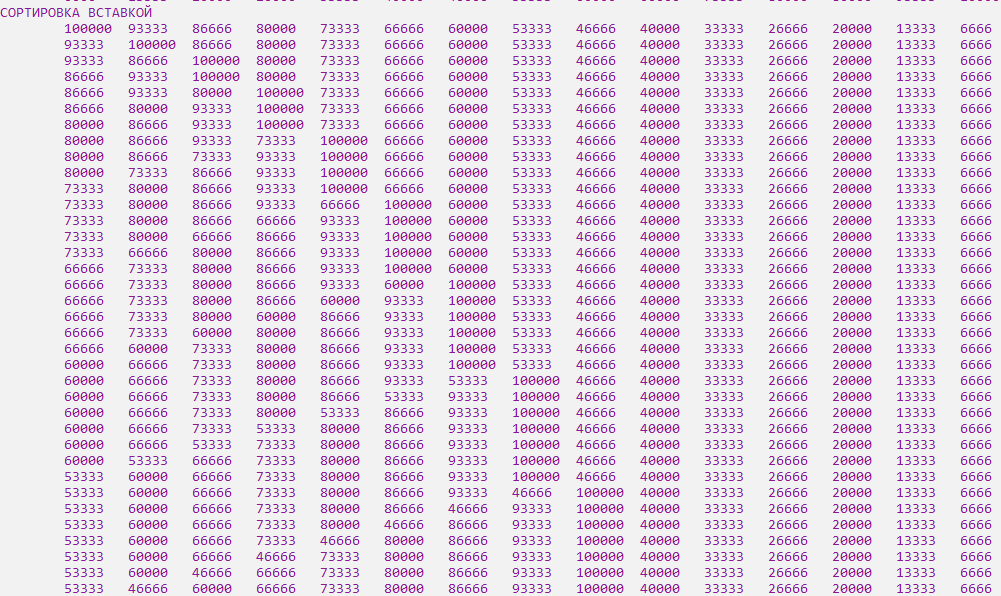


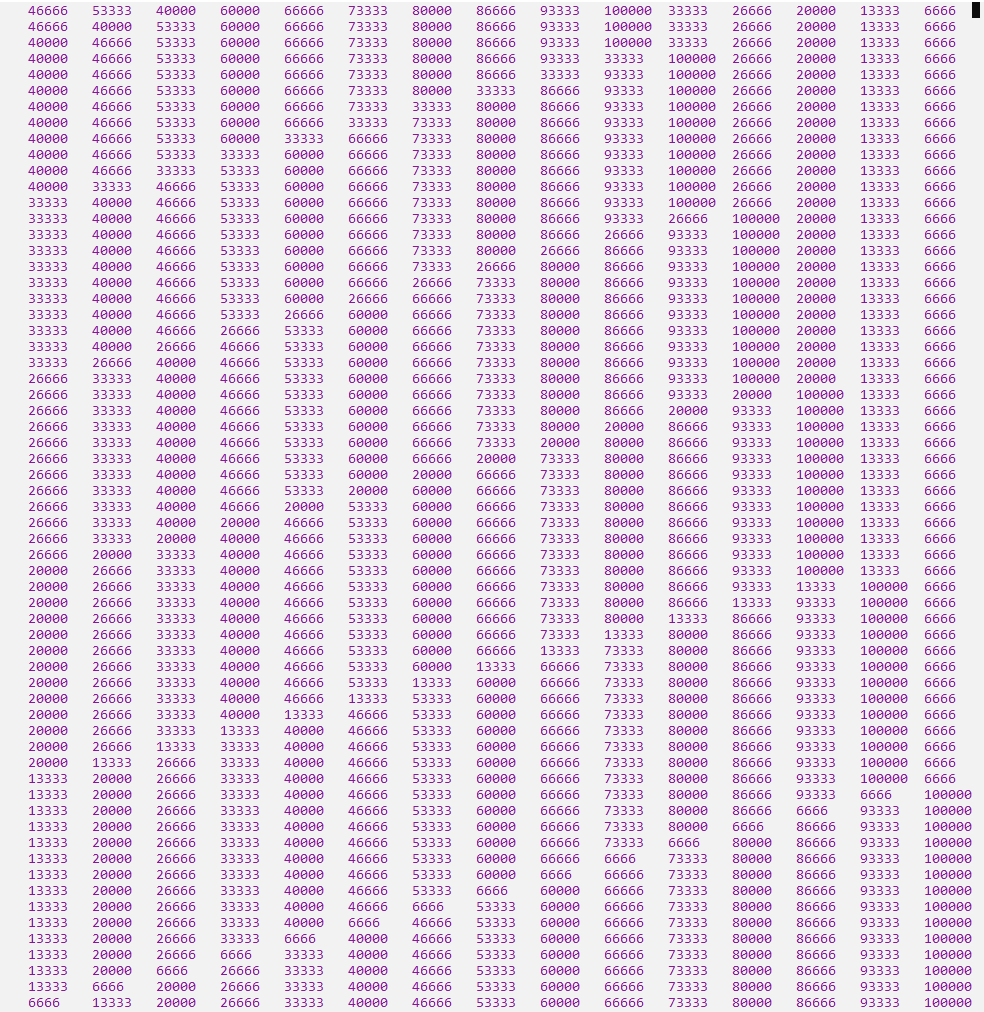


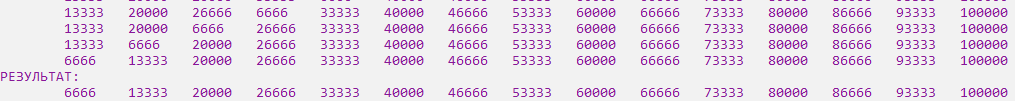












Вывод по тесту: программа работает корректно.

# Сопоставительный анализ

Наилучший случай

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Длина | **BLS** | | **SLS** | | **OAS** | **BS** |
| Сравнений | Время | Сравнений | Время | Время | Время |
| 10000 | 0 | 0 | 0 | 0 | 0 | 1 |
| 29000 | 0 | 0 | 0 | 0 | 0 | 1 |
| 48000 | 0 | 0 | 0 | 0 | 0 | 1 |
| 67000 | 0 | 0 | 0 | 0 | 0 | 1 |
| 86000 | 0 | 4 | 0 | 0 | 0 | 0 |
| 105000 | 0 | 1 | 0 | 0 | 1 | 5 |
| 124000 | 0 | 2 | 0 | 0 | 1 | 0 |
| 143000 | 0 | 0 | 0 | 1 | 1 | 0 |
| 162000 | 0 | 4 | 0 | 0 | 0 | 0 |
| 181000 | 0 | 0 | 0 | 0 | 1 | 0 |
| 200000 | 0 | 0 | 0 | 0 | 1 | 0 |

Средний случай

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Длина | **BLS** | | **SLS** | | **OAS** | **BS** |
| Сравнений | Время | Сравнений | Время | Время | Время |
| 10000 | 4999 | 15 | 4999 | 14 | 15 | 1 |
| 29000 | 14499 | 42 | 14499 | 57 | 53 | 1 |
| 48000 | 23999 | 70 | 23014 | 68 | 91 | 1 |
| 67000 | 33499 | 185 | 21869 | 59 | 104 | 0 |
| 86000 | 31689 | 74 | 15349 | 60 | 166 | 0 |
| 105000 | 2973 | 7 | 52499 | 145 | 215 | 0 |
| 124000 | 21097 | 52 | 17898 | 50 | 168 | 0 |
| 143000 | 62777 | 358 | 4901 | 14 | 234 | 0 |
| 162000 | 13170 | 41 | 7631 | 22 | 228 | 0 |
| 181000 | 54285 | 176 | 24367 | 68 | 465 | 1 |
| 200000 | 41741 | 103 | 269 | 1 | 420 | 0 |

Худший случай

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Длина | **BLS** | | **SLS** | | **OAS** | **BS** |
| Сравнений | Время | Сравнений | Время | Время | Время |
| 10000 | 9998 | 29 | 14 | 0 | 34 | 0 |
| 29000 | 28998 | 68 | 28998 | 83 | 77 | 1 |
| 48000 | 47998 | 141 | 36988 | 85 | 170 | 1 |
| 67000 | 8338 | 19 | 1678 | 6 | 229 | 1 |
| 86000 | 16491 | 39 | 8457 | 58 | 712 | 3 |
| 105000 | 16202 | 44 | 14701 | 89 | 366 | 1 |
| 124000 | 21866 | 53 | 79359 | 234 | 507 | 1 |
| 143000 | 539 | 3 | 56463 | 159 | 440 | 2 |
| 162000 | 70671 | 812 | 3588 | 11 | 709 | 2 |
| 181000 | 5363 | 18 | 85617 | 243 | 666 | 1 |
| 200000 | 87785 | 219 | 44479 | 98 | 775 | 1 |

# Вывод

Вывод по результатам сопоставительного анализа:

Анализ времени работы всех алгоритмов

* В наилучшем случае расположения искомого элемента алгоритмы BLS, SLS и OAS работают одинаково быстро.
* В среднем и наихудшем случаях быстрее всех работает алгоритм BS, в то же время медленнее всех в таких случаях работают BLS и OAS (работают одинаково медленно).
* Таким образом, в среднем и наихудшем случаях BS – самый эффективный алгоритм. В наилучшем случае BLS, SLS, OAS – одинаково эффективны

Анализ количества сравнений в алгоритмах BLS и SLS

* В наилучшем случае BLS и SLS имеют одинаковое количество сравнений
* В среднем и наилучшем случаях SLS имеет меньше сравнений, чем BLS
* Таким образом, с точки зрения количества операций, а. следовательно, и быстроты работы, алгоритм SLS эффективнее BLS.