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-211Б-22

Коноплев К.А

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

Ивашенцев И.В.

Москва 2023 г.

Оглавление

[Задание 3](#_Toc151300276)

[Структурные схемы алгоритмов 4](#_Toc151300277)

[Функция Ordered\_Array\_Search 4](#_Toc151300278)

[Функция Binary\_Search 5](#_Toc151300279)

[Функция main 6](#_Toc151300280)

[Код программы 7](#_Toc151300281)

[Графики зависимости времени от размерности и расположения ключа 11](#_Toc151300282)

[Ключ расположен в начале 11](#_Toc151300283)

[Ключ расположен в середине 13](#_Toc151300284)

[Ключ расположен в конце 15](#_Toc151300285)

[Вывод 17](#_Toc151300286)

# 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**.

|  |  |
| --- | --- |
| № вар. | метод |
| 1 | выбором, быстрая |

Вариант задания:

# Структурные схемы алгоритмов

## Функция selectionSort

Сортировка массива выбором в порядке неубывания.

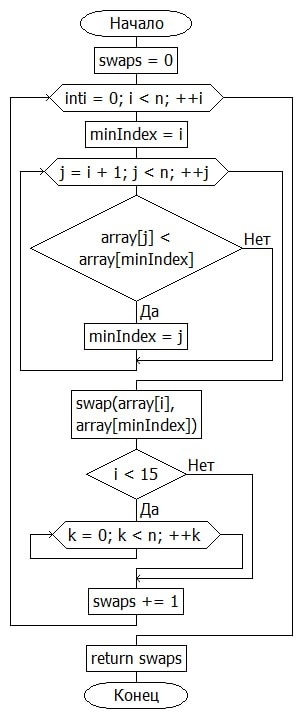
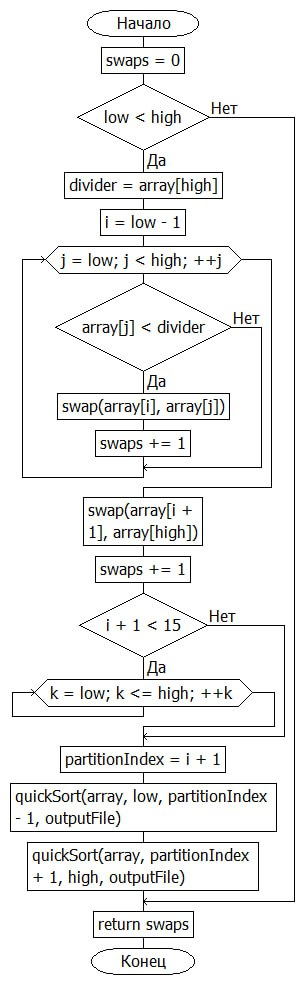


Рис. 1. Структурная схема алгоритма функции selectionSort

## Функция quickSort

Быстрая сортировка массива по неубыванию.



Риc. 2. Структурная схема алгоритма функции quickSort.

## Функция main

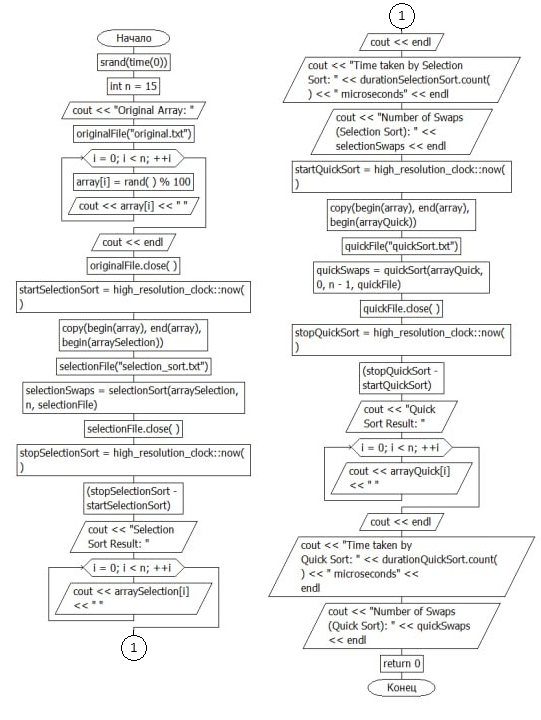


Рис. 3. Структурная схема алгоритма функции main.

# Код программы

#include <iostream>

#include <ctime>

#include <cstdlib>

#include <random>

#include <fstream>

#include <chrono>

using namespace std;

using namespace chrono;

const int n = 15;

// Функция для обмена двух элементов массива

void swap(int& a, int& b) {

int temp = a;

a = b;

b = temp;

}

// Сортировка выбором

int selectionSort(int array[], int n, int& comp, ofstream& outputFile) {

int swaps = 0;

for (int i = 0; i < n; ++i) {

int minIndex = i;

for (int j = i + 1; j < n; ++j) {

comp++;

if (array[j] < array[minIndex]) {

minIndex = j;

}

}

// Обмен текущего элемента с минимальным

swap(array[i], array[minIndex]);

// Вывод массива после каждой итерации в файл

if (n <= 15) {

outputFile << "Selection Sort Iteration " << i + 1 << ": ";

for (int k = 0; k < n; ++k) {

outputFile << array[k] << " ";

}

outputFile << endl;

}

swaps += 1;

}

return swaps;

}

// Быстрая сортировка

int quickSort(int array[], int low, int high, int &comp, ofstream& outputFile) {

int swaps = 0;

comp++;

if (low < high) {

int divider = array[high];

int i = low - 1;

// Разделение массива на две части: элементы, меньшие divider, и элементы, большие divider

for (int j = low; j < high; ++j) {

comp++;

if (array[j] < divider) {

++i;

swap(array[i], array[j]);

swaps += 1;

}

}

// Обмен опорного элемента с элементом, стоящим следующим после меньших элементов

swap(array[i + 1], array[high]);

swaps += 1;

// Вывод массива после каждой итерации в файл

if (n <= 15) {

outputFile << "Quick Sort Iteration: ";

for (int k = low; k <= high; ++k) {

outputFile << array[k] << " ";

}

outputFile << endl;

}

int partitionIndex = i + 1;

// Рекурсивно сортируем обе части массива

quickSort(array, low, partitionIndex - 1, comp, outputFile);

quickSort(array, partitionIndex + 1, high, comp, outputFile);

}

return swaps;

}

int main() {

srand(time(0));

int array[n];

// Генерация и вывод исходного массива

cout << "Original Array: ";

ofstream originalFile("original.txt");

originalFile << "Original Array: ";

for (int i = 0; i < n; ++i) {

array[i] = rand() % 100; // Заполняем массив случайными числами от 0 до 99

cout << array[i] << " ";

originalFile << array[i] << " ";

}

cout << endl;

originalFile.close();

// Измерение времени для сортировки выбором

int compSelection = 0;

auto startSelectionSort = high\_resolution\_clock::now();

int arraySelection[n];

copy(begin(array), end(array), begin(arraySelection));

ofstream selectionFile("selection\_sort.txt");

int selectionSwaps = selectionSort(arraySelection, n, compSelection, selectionFile);

auto stopSelectionSort = high\_resolution\_clock::now();

auto durationSelectionSort = duration\_cast<microseconds>(stopSelectionSort - startSelectionSort);

// Вывод отсортированного массива и времени выполнения сортировки выбором

selectionFile << "Selection Sort Result: ";

for (int i = 0; i < n; ++i) {

selectionFile << arraySelection[i] << " ";

}

selectionFile << endl;

selectionFile << "Time taken by Selection Sort: " << durationSelectionSort.count() << " microseconds" << endl;

selectionFile << "Number of Swaps (Selection Sort): " << selectionSwaps << endl;

selectionFile << "Number of Comparisons (Selection Sort): " << compSelection << endl;

selectionFile.close();

// Измерение времени для быстрой сортировки

int compQuick = 0;

auto startQuickSort = high\_resolution\_clock::now();

int arrayQuick[n];

copy(begin(array), end(array), begin(arrayQuick));

ofstream quickFile("quickSort.txt");

int quickSwaps = quickSort(arrayQuick, 0, n - 1, compQuick, quickFile);

auto stopQuickSort = high\_resolution\_clock::now();

auto durationQuickSort = duration\_cast<microseconds>(stopQuickSort - startQuickSort);

// Вывод отсортированного массива и времени выполнения быстрой сортировки

quickFile << "Quick Sort Result: ";

for (int i = 0; i < n; ++i) {

quickFile << arrayQuick[i] << " ";

}

quickFile << endl;

quickFile << "Time taken by Quick Sort: " << durationQuickSort.count() << " microseconds" << endl;

quickFile << "Number of Swaps (Quick Sort): " << quickSwaps << endl;

quickFile << "Number of Comparisons (Quick Sort): " << compQuick << endl;

quickFile.close();

return 0;

}

# Графики зависимости времени от размерности и расположения ключа

## Результаты работы программы с массивом из 15 элементов:

Рис. 4. Изначальный массив

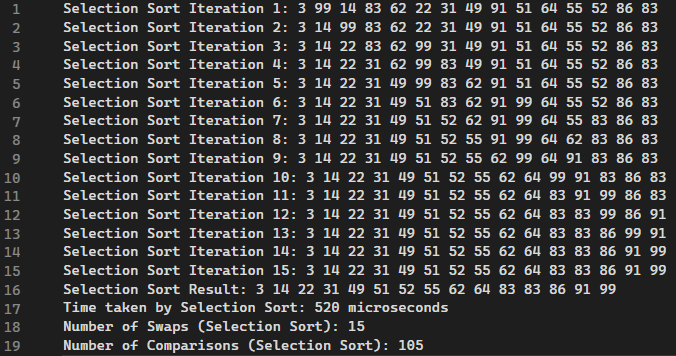


Рис. 5. Результат работы функции selectionSort

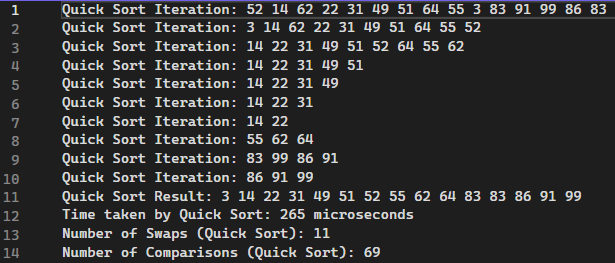


Рис. 6. Результат работы функции quickSort

## Результаты сопоставительного анализа:

Таблица 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Кол-во элементов,  n | Время выполнения, мс | | Число сравнений | | Число пересылок | |
|  | Вставками | Быстрая | Вставками | Быстрая | Вставками | Быстрая |
| 15 | 520 | 265 | 105 | 69 | 15 | 11 |
| 1000 | 997 | 135 | 499500 | 14062 | 1000 | 822 |
| 10000 | 53534 | 1246 | 4995000 | 608796 | 10000 | 7592 |
| 50000 | 1852834 | 14229 | 12499755000 | 13047783 | 50000 | 20106 |
| 100000 |  |  |  |  |  |  |

# Вывод

Программа работает корректно на основе проделанных тестов. Были освоены принципы работы алгоритмов сортировки массивов, а именно сортировка вставками (selectionSort), быстрая сортировка(quickSort). Также были изучены функции, позволяющие производить оценку длительности времени выполнения алгоритмов с учетом вызова функций, а именно функция high\_resolution\_clock() из библиотеки chrono. В ходе проведения лабораторной работы было выяснено:

1. Быстрая сортировка работает в несколько десятков раз быстрее чем сортировка вставками
2. Быстрая сортировка совершает меньше операций сравнений
3. Быстрая сортировка совершает меньшее число пересылок

Общий вывод: быстрая сортировка ЛУЧШЕ!!!!!