**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»**

Институт компьютерных наук и технологий

Образовательная программа «Информационные системы и технологии»

Дисциплина «Теория и технология программирования»

Отчет к лабораторной работе № 6

«Методы сортировки»

Выполнил:

студент группы 3530902/20001

Сафонов А.И

Принял:

Доцент к.т.н Хлопин. С. В.

Санкт-Петербург, 2023 год

**Задание лабораторной № 6:**

Составить программу для сортировки массива данных методами: пузырьковой, отбора, вставки, Шелла и быстрой сортировки. Вывести на экран неупорядоченную (один раз) и упорядоченные (для каждого из методов) массивы данных. Составить сравнительную таблицу эффективности методов, в которой необходимо указать число сравнений и перестановок переменных в каждом методе сортировки.

Неупорядоченная матрица задается один раз случайным образом, далее она используется для каждого из методов сортировки.

Представить шахматную доску. Упорядочить белые клетки по возрастанию по строкам, черные фигуры по убыванию по столбцам

**Текст программы:**

Файл ConsoleAplication1.cpp:

#include <iostream>

#include <vector>

#include "fillrandom.h"

#include "sortMethod.h"

int main()

{

using namespace std::literals;

setlocale(LC\_ALL, "ru");

Size v\_size = GetSize();

std::vector<std::vector<int>> v(v\_size.row, std::vector<int> (v\_size.column));

FillRandom(v, v\_size);

СomparisonSortMethod(v);

}

Файл read\_input\_function.cpp:

#include "read\_input\_function.h"

#include <iostream>

#include <optional>

std::string ReadLine() {

std::string s;

std::getline(std::cin, s);

return s;

}

void CutSpace(std::string& str) {

int start\_index = 0;

int end\_index = str.length();

for (start\_index; str[start\_index] == ' '; start\_index++);

for (end\_index; str[end\_index] == ' '; end\_index--);

str = str.substr(start\_index, end\_index);

}

bool isValid(std::string& str) {

if (str[0] == '-')

return false;

else if (str[0] == '+')

str = str.substr(1, str.length());

bool have\_dots = false;

for (char ch : str) {

if (ch == '.' || ch == ',') {

if (ch == str[0]) {

return false;

}

have\_dots = true;

}

else if (have\_dots && ch != '0' || ch > '9' || ch < '0') {

return false;

}

}

return true;

}

int GetPositiveNum(std::string message, std::string error\_message) {

while (true) {

std::cout << message;

std::string str = ReadLine();

CutSpace(str);

if (!str.empty() && isValid(str) && std::stoi(str) != 0) {

return std::stoi(str);

}

std::cout << error\_message << std::endl;

}

}

Файл 2d\_size.h:

#pragma once

#include "read\_input\_function.h"

struct Size {

size\_t row;

size\_t column;

};

Size GetSize() {

using namespace std::string\_literals;

do {

Size size;

size.column = GetPositiveNum("введите n:");

size.row = GetPositiveNum("введите m:");

if (size.row \* size.column <= 100000000) {

return size;

}

std::cout << "Слишком большой массив, максимальный размер массива 100000000 элементов"s << std::endl;

} while (true);

}

Файл CopyMassiv.h:

#pragma once

#include <vector>

template<typename Elem>

std::vector<std::vector<Elem>> CopyMassiv(const std::vector<std::vector<Elem>>& v) {

std::vector<std::vector<Elem>> result;

if (v.empty()) {

return result;

}

size\_t row = v.size();

size\_t column = v[0].size();

for (size\_t i = 0; i < row; ++i) {

std::vector<Elem> row;

for (size\_t j = 0; j < column; ++j) {

row.push\_back(v[i][j]);

}

result.push\_back(row);

}

return result;

}

Файл FillRandom.h:

#pragma once

#include <vector>

#include <random>

#include "2d\_size.h"

template<typename Integer>

void FillRandom(std::vector<std::vector<Integer>>& v, Size size) {

srand(static\_cast<unsigned int> (time(NULL)));

int hlopin = 1;

for (size\_t i = 0; i < size.row; ++i) {

for (size\_t j = 0; j < size.column; ++j) {

Integer number = rand() % 3;

//Integer number = hlopin;

//hlopin = 1;

v[i][j] = number;

}

}

}

Файл log\_duration.h:

#pragma once

#include <chrono>

#include <iostream>

#define PROFILE\_CONCAT\_INTERNAL(X, Y) X##Y

#define PROFILE\_CONCAT(X, Y) PROFILE\_CONCAT\_INTERNAL(X, Y)

#define UNIQUE\_VAR\_NAME\_PROFILE PROFILE\_CONCAT(profileGuard, \_\_LINE\_\_)

#define LOG\_DURATION(x) LogDuration UNIQUE\_VAR\_NAME\_PROFILE(x)

class LogDuration {

public:

// заменим имя типа std::chrono::steady\_clock

// с помощью using для удобства

using Clock = std::chrono::steady\_clock;

LogDuration(const std::string& id)

: id\_(id) {

}

~LogDuration() {

using namespace std::chrono;

using namespace std::literals;

const auto end\_time = Clock::now();

const auto dur = end\_time - start\_time\_;

std::cerr << id\_ << ": "s << duration\_cast<microseconds>(dur).count() << " mcs"s << std::endl;

}

private:

const std::string id\_;

const Clock::time\_point start\_time\_ = Clock::now();

};

Файл read\_input\_function.h:

#pragma once

#include <string>

#include <optional>

std::string ReadLine();

std::optional<int> GetPositiveNumInner();

void CatSpace(std::string& str);

int GetPositiveNum(std::string message = "Введите число", std::string error\_message = "Вы ввели не корректное значение, попробуйте еще раз");

файл SortInfo.h

#pragma once

struct SortInfo {

SortInfo() = default;

SortInfo(size\_t n, size\_t m) : comparisons(n), permutations(m)

{

}

size\_t comparisons = 0;

size\_t permutations = 0;

};

SortInfo operator+ (SortInfo left, SortInfo right) {

return SortInfo{ left.comparisons + right.comparisons, left.permutations + right.permutations };

}

Файл SortMethod.h

#pragma once

#include <vector>

#include <unordered\_map>

#include "CopyMassiv.h"

#include "write\_output\_function.h"

#include "SortInfo.h"

#include "log\_duration.h"

template<typename Elem>

std::pair<Elem\*\*, size\_t> GetWhiteRow(std::vector<Elem>& v, bool FirstWhite) {

int\*\* result = new int\* [v.size() / 2 + 1];

int res\_current = 0;

for (int j = FirstWhite? 0 : 1; j < v.size(); j+=2) {

result[res\_current++] = &v[j];

}

return { result, res\_current};

}

template<typename Elem>

std::pair<Elem\*\*, size\_t> GetBlackColumnReverse(std::vector<std::vector<Elem>>& v, size\_t column) {

Elem\*\* result = new int\* [v.size() / 2 + 1];

size\_t res\_current = 0;

size\_t start = (v.size() - column % 2) % 2? v.size() -1: v.size() >= 2 ? v.size() -2 : 0;

for (int i = start; i>=0; i-= 2) {

result[res\_current++] = &v[i][column];

}

return { result, res\_current };

}

template<typename Elem>

SortInfo BableSort(Elem\*\*\* v, size\_t size) {

SortInfo info;

if (size <= 1) {

return info;

}

size\_t i, j;

for (i = 0; i < size - 1; i++) {

int current\_swap = 0;

for (j = 0; j < size - i - 1; j++) {

if (\*v[0][j] > \*v[0][j + 1]) {

std::swap(\*v[0][j], \*v[0][j + 1]);

++current\_swap;

}

++info.comparisons;

}

if (current\_swap == 0) {

break;

}

info.permutations = info.permutations + current\_swap;

}

return info;

}

template<typename Elem>

SortInfo SelectionSort(Elem\*\*\* v, size\_t size) {

SortInfo info;

if (size <= 1) {

return info;

}

size\_t i, j, min\_idx;

for (i = 0; i < size - 1; i++)

{

min\_idx = i;

for (j = i + 1; j < size; j++)

{

if (\*v[0][j] < \*v[0][min\_idx]) {

min\_idx = j;

}

++info.comparisons;

}

if (min\_idx != i) {

std::swap(\*v[0][i], \*v[0][min\_idx]);

++info.permutations;

}

}

return info;

}

template<typename Elem>

SortInfo InsertionSort(Elem\*\*\* v, size\_t size) {

SortInfo info;

if (size <= 1) {

return info;

}

for (size\_t i = 1; i < size; i++) {

size\_t j = i;

while (j > 0) {

++info.comparisons;

if (\*v[0][j] < \*v[0][j-1]) {

std::swap(\*v[0][j - 1], \*v[0][j]);

++info.permutations;

}

else {

break;

}

j--;

}

}

return info;

}

template<typename Elem>

SortInfo ShellSort(Elem\*\*\* v, size\_t size) {

SortInfo info;

size\_t temp, inc;

for (inc = size / 2; inc > 0; inc /= 2) {

for (size\_t i = inc; i < size; i++)

{

for (int j = i - inc; j >= 0; j -= inc) {

++info.comparisons;

if (\*v[0][j] > \*v[0][j+inc]) {

++info.permutations;

std::swap(\*v[0][j], \*v[0][j + inc]);

}

else {

break;

}

}

}

}

return info;

}

template<typename Elem>

SortInfo QuickSortInner(Elem\*\*\* v, int low, int high) {

SortInfo info;

int i = low;

int j = high;

int pivot\_index = (i + j) / 2;

int pivot = \*v[0][pivot\_index];

while (i <= j)

{

while (\*v[0][i] < pivot) {

i++;

++info.comparisons;

}

if (pivot\_index != i) {

++info.comparisons;

}

while (\*v[0][j] > pivot) {

j--;

++info.comparisons;

}

if (pivot\_index != j) {

++info.comparisons;

}

if (i <= j)

{

if (\*v[0][i] != \*v[0][j]) {

++info.permutations;

std::swap(\*v[0][i], \*v[0][j]);

}

i++;

j--;

}

}

if (j > low)

info = info + QuickSortInner(v, low, j);

if (i < high)

info = info + QuickSortInner(v, i, high);

return info;

}

template<typename Elem>

SortInfo QuickSort(Elem\*\*\* v, size\_t size) {

return size <= 1 ? SortInfo{0, 0} : QuickSortInner(v, 0, size-1);

}

template<typename Elem>

SortInfo SortedChess(std::vector<std::vector<Elem>>& v, SortInfo SortFunction(Elem\*\*\* v, size\_t size)) {

SortInfo info;

int row = v.size();

int column = v[0].size();

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

auto [white\_row, size] = GetWhiteRow(v[i], i % 2 == 1);

info = info + SortFunction(&white\_row, size);

}

for (int j = 0; j < column; j++) {

auto [black\_column, size] = GetBlackColumnReverse(v, j);

info = info + SortFunction(&black\_column, size);

}

return info;

}

template<typename Elem>

void SortedPrint(const std::string& name , const std::vector<std::vector<Elem>>& massiv) {

using namespace std::literals;

std::cout << name << ":"s << std::endl << massiv;

}

template<typename Elem>

void СomparisonSortMethod(std::vector<std::vector<Elem>> v) {

using namespace std::literals;

SortedPrint("original array"s, v);

std::unordered\_map<std::string, SortInfo> info;

auto for\_sort\_massiv = CopyMassiv(v);

{

LOG\_DURATION("BubbleSortTime: "s);

info["Bubble sort"s] = SortedChess(for\_sort\_massiv, BableSort);

}

SortedPrint("Bubble sort"s, for\_sort\_massiv);

for\_sort\_massiv = CopyMassiv(v);

{

LOG\_DURATION("Selection sort: ");

info["Selection sort"] = SortedChess(for\_sort\_massiv, SelectionSort);

}

SortedPrint("Selection sort"s, for\_sort\_massiv);

for\_sort\_massiv = CopyMassiv(v);

{

LOG\_DURATION("Insertion sort: ");

info["Insertion sort"] = SortedChess(for\_sort\_massiv, InsertionSort);

}

SortedPrint("Insertion sort"s, for\_sort\_massiv);

for\_sort\_massiv = CopyMassiv(v);

{

LOG\_DURATION("Shell sort: ");

info["Shell sort"] = SortedChess(for\_sort\_massiv, ShellSort);

}

SortedPrint("Shell sort"s, for\_sort\_massiv);

for\_sort\_massiv = CopyMassiv(v);

{

LOG\_DURATION("Quick sort: ");

info["Quick sort"] = SortedChess(for\_sort\_massiv, QuickSort);

}

SortedPrint("Quick sort"s, for\_sort\_massiv);

PrintComparisentTable(info);

}

Файл write\_output\_function.h

#pragma once

#include <iostream>

#include <vector>

#include <sstream>

#include <iomanip>

#include <stdio.h>

#include <unordered\_map>

#include "SortInfo.h"

#include <locale.h>

#include "log\_duration.h"

#define MIN\_BORDER 2

#define MIN\_ELEM\_WIDTH 3

#define MAX\_SIZE\_OUTPUT\_MASSIV 20

class PrintContainer {

public:

PrintContainer(size\_t n)

:elem\_width\_((MIN\_ELEM\_WIDTH > n ? MIN\_ELEM\_WIDTH : n) + MIN\_BORDER)

{

}

template<typename Elem>

std::ostream& Print2d(std::ostream& out, const std::vector<std::vector<Elem>>& container) {

using namespace std::literals;

for (auto line : container) {

Print(out, line);

out << std::endl;

}

return out;

}

template<typename Container>

std::ostream& Print(std::ostream& out, const Container& container) {

using namespace std::literals;

for (const auto& elem : container) {

PrintElem(out, elem);

}

return out;

}

template<typename Elem>

std::ostream& PrintElem(std::ostream& out, const Elem& elem) {

out << std::setw(elem\_width\_) << elem;

return out;

}

private:

int elem\_width\_ = MIN\_ELEM\_WIDTH + MIN\_BORDER;

};

template<typename Elem>

std::ostream& operator<< (std::ostream& out, const std::vector<std::vector<Elem>>& container) {

if (container.size() > MAX\_SIZE\_OUTPUT\_MASSIV || !container.empty() && container[0].size() > MAX\_SIZE\_OUTPUT\_MASSIV) {

using namespace std::literals;

out << " massiv..."s << std::endl;

return out;

}

long max\_elem = 0;

for (size\_t i = 0; i < container.size(); i++) {

long max\_in\_line = \*std::max\_element(container[i].begin(), container[i].end());

if (max\_in\_line > max\_elem) {

max\_elem = max\_in\_line;

}

}

PrintContainer print(std::to\_string(max\_elem).length());

print.Print2d(out, container);

return out;

}

void PrintComparisentTable(const std::unordered\_map<std::string, SortInfo>& info) {

using namespace std;

cout << endl << "Comparison table : " << endl;

cout << " \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"s << endl;

cout << " | "s << setw(14 ) << " sortType "s << " | " << std::setw(14) << "iteration quantity"s << " | " << setw(15) << "swap quantity"s << " |"s << endl;

cout << " |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|" << endl;

for (const auto& [name, sort\_info] : info) {

cout << " | " << setw(14) << name << " | " << setw(18) << sort\_info.comparisons

<< " | " << setw(15) << sort\_info.permutations << " |" << endl;

}

cout << " |" << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_| ";

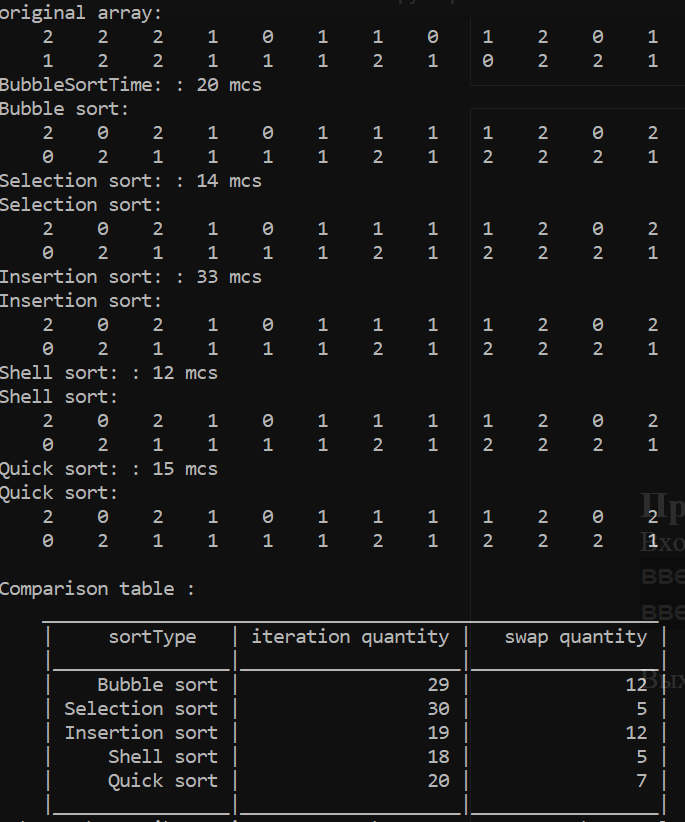
}

**Пример работы:**

Входные данные:



Выходные данные:



**Выводы:** в ходе выполнения лабораторной работы я познакомился с 5 видами сортировки и их реализациями. Полностью изучив их, мы обрели понимание, как работают сортировки, а так же, благодаря полученным знаниям, можно оптимизировать будущие проекты, подбирая нужный алгоритм сортировки, ведь в разных ситуациях, один алгоритм работает быстрее другого, и наоборот.