**Правительство Российской Федерации**

**Федеральное государственное автономное образовательное учреждение высшего профессионального образования   
"Национальный исследовательский университет   
"Высшая школа экономики"**

Московский институт электроники и математики Национального

исследовательского университета "Высшая школа экономики"

Департамент прикладной математики

**ОТЧЕТ**

**По лабораторной работе №2**

**По курсу «Алгоритмизация и программирование»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  | | --- | --- | |  | ФИО студента | | Номер группы | Дата |
| Псурцев Иван Вадимович | БПМ-192 | 03.10.2020 |
|  |
|  |
|  |

**Москва – 2020 г.**

**ЗАДАНИЕ (вариант №14)**

Общее задание

1) Реализовать на языке С++ сортировки для массива объектов в соответствии с вариантом.

2) Перегрузить операторы сравнения (>, <, >=, <=) для сравнения объектов.

3) Входные данные для сортировки массива обязательно считывать из внешних источников: текстовый файл, файл MS Excel, MS Access, данные из СУБД (любое на выбор).

4) Выбрать 7-10 наборов данных для сортировки размерности от 100 и более. Засечь (программно) время сортировки каждым алгоритмом. По полученным точкам построить графики зависимости времени сортировки от размерности массива для каждого из алгоритмов сортировки на одной оси координат. Сделать вывод о том, в каком случае, какой из методов лучше применять. Графики можно строить в любой из прикладных программ (MS Excel, Matlab, MathCad и т.д.).

5) Сделать отчет, содержащий титульный лист, код программы со спецификациями каждого метода и подробными комментариями, графики скоростей сортировок и выводы.

Массив данных с расписанием поездов, отправляющихся из пункта А в пункт В за некоторое число: Расписание содержит номер поезда,, тип поезда, (скорый, пассажирский), время отправления, время в пути (сравнение по полям – время отправления, время в пути (по убыванию))

Сортировка простыми вставками

Быстрая сортировка

**РЕШЕНИЕ**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <fstream>

#include <vector>

#include <time.h>

#include "TrainRasp.h"

const int CountData = 7; // кол-во файлов с входными данными

const int Step = 5000; // разница кол-ва элементов для сравнения

void ReadData(std::vector<TrainRasp>& table, std::string fileName = "data.txt"); // считывание из одного файла

void WriteData(std::vector<TrainRasp>& table, std::string fileName = "data.xls"); // запись таблицы поездов в файл

void WriteTime(double\* time, int size, std::string title, std::string fileName = "time.xls"); // запись таблицы времени

double SortInsert(std::vector<TrainRasp>& table); // сортировка вставками

TrainRasp GenerateRandom(); // создание рандомной строчки в таблице поездов

void GenerateInputData(); // создает данные для 7 файлов таблицы поездов

double QuickSort(std::vector<TrainRasp>& table, int left, int right); // быстрая сортировка

int main()

{

setlocale(LC\_ALL, "RUS");

//GenerateInputData();

std::vector<TrainRasp> table; // таблица поездов

std::string name = "data.txt"; // шаблон названия файла ввода

std::ofstream out("time.xls"); // открываем файл вывода времени сортировок

out << "Count of elems" << '\t'; //

for (int i = 0; i < CountData; ++i) //

{ //

out << (i + 1) \* Step << '\t'; // записываем первую строчку таблицы с кол-вом элементов

} //

out << '\n'; //

out.close(); //

double time1[CountData]; // массив, хранящий время работы функции сортировки для всех входящих данных

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

{

name.insert(4, 1, char(i + '1')); // вставка после 4 символа нужной нам цифры

ReadData(table, name); // считывание из нужного файла

time1[i] = SortInsert(table); // вызов сортировки и запись времени на сортировку вставками

name = "data.txt"; // восстанавливаем имя по умолчанию, чтобы корректно получить новое (не data21.txt, например)

table.clear(); // очищение таблицы

}

WriteTime(time1, CountData, "Time for insert sort"); // записывает время сортировки вставками в таблицу excel

for (int i = 0; i < CountData; ++i) // аналогично для быстрой сортировки

{

name.insert(4, 1, char(i + 49));

ReadData(table, name);

time1[i] = QuickSort(table, 0, (i + 1) \* 5000);

name = "data.txt";

table.clear();

}

WriteTime(time1, CountData, "Time for quick sort");

}

void GenerateInputData() // генерирует 7 файлов со случайными данными

{

srand(time(0)); // для нормальной рандомизации

std::vector<TrainRasp> table; // создание таблицы

for (int i = 0; i < 5000; ++i)

{

table.push\_back(GenerateRandom()); // в конец вектора записываем случайную строчку расписания

}

WriteData(table, "data1.txt"); // запись в 1й файл нашей случайной таблицы поездов

table.clear(); // очистка таблицы

for (int i = 0; i < 10000; ++i) // аналогично

{

table.push\_back(GenerateRandom());

}

WriteData(table, "data2.txt");

table.clear();

for (int i = 0; i < 15000; ++i)

{

table.push\_back(GenerateRandom());

}

WriteData(table, "data3.txt");

table.clear();

for (int i = 0; i < 20000; ++i)

{

table.push\_back(GenerateRandom());

}

WriteData(table, "data4.txt");

table.clear();

for (int i = 0; i < 25000; ++i)

{

table.push\_back(GenerateRandom());

}

WriteData(table, "data5.txt");

table.clear();

for (int i = 0; i < 30000; ++i)

{

table.push\_back(GenerateRandom());

}

WriteData(table, "data6.txt");

table.clear();

for (int i = 0; i < 35000; ++i)

{

table.push\_back(GenerateRandom());

}

WriteData(table, "data7.txt");

table.clear();

}

TrainRasp GenerateRandom() // создание случайной строчки в таблице поездов

{

TrainRasp t;

t.num = rand() % 9999 + 1; // 4х значные числа

t.type = rand() % 2 + 1; // 1 или 2

t.timeForTravel = rand() % 1000 + 10; // от 10 до 1009

t.timeDeparture = rand() % 1440; // от 0.00 до 23.59

return t;

}

void ReadData(std::vector<TrainRasp>& table, std::string fileName) // считывание таблицы поездов

{

std::stream in(fileName); // открываем файловый поток

TrainRasp temp;

if (!in) // если не удалось открыть

{

std::cout << "Can't open a file";

return;

}

while (!in.eof()) // пока не дошли до конца файла

{

in >> temp; // считывание строчки(структуры)

table.push\_back(temp); // запись в таблцу

}

}

double SortInsert(std::vector<TrainRasp>& table) // сортировка вставками

{

clock\_t start = clock();

//int count = 0;

for (int i = 1; i < table.size(); i++)

{

for (int j = i; j > 0 && table[j] < table[j - 1]; j--)

{

TrainRasp tmp = table[j - 1];

table[j - 1] = table[j];

table[j] = tmp;

//count++;

}

}

clock\_t end = clock();

//std::cout << count << ' ';

return (double)(end - start) / CLOCKS\_PER\_SEC;

}

void WriteData(std::vector<TrainRasp>& table, std::string fileName) // запись таблицы поездов в файл

{

std::ofstream out(fileName); //открытие файла для вывода

for (int i = 0; i < table.size(); ++i) //записывает поочередно строчки таблицы в файл

{

out << table[i];

}

out.close();

}

void WriteTime(double\* time, int size, std::string title, std::string fileName) // записывает время сортировок таблиц в таблицу excel

{

std::ofstream out(fileName, std::ios::app); // открывает таблицу excel в режиме дополнения (дописывает в файл)

out << title << '\t'; // записывает заголовок строчки

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

{

out << time[i] << '\t'; // поочередно записывает время в строчку

}

out << '\n';

out.close(); // закрывает поток вывода

}

double QuickSort(std::vector<TrainRasp>& table, int left, int right) // быстрая сортировка

{

clock\_t start = clock(); // запись времени начала

int i = left;

int j = right;

TrainRasp pivot;

pivot = table[(left + right) / 2];

while (i <= j)

{

while ((table[i] < pivot) == true && i < right)

{

i++;

}

while ((pivot < table[j]) == true && left < j)

{

j--;

}

if (i <= j)

{

std::swap(table[i], table[j]);

i++;

j--;

}

}

if (left < j)

{

QuickSort(table, left, j);

}

if (i < right)

{

QuickSort(table, i, right);

}

clock\_t end = clock(); // запись времени конца работы

return (double)(end - start) / CLOCKS\_PER\_SEC;

}

#include "TrainRasp.h"

#include <fstream>

#include <string>

std::istream& operator>>(std::istream& in, TrainRasp& t)

{

in >> t.num >> t.type >> t.timeForTravel >> t.timeDeparture; // при вводе последовательно забиваем вначале номер, потом тип, потом...

return in;

}

std::ostream& operator<<(std::ostream& out, TrainRasp& t) // перегрузка оператора вывода

{

out << t.num << '\t' << t.type << '\t' << t.timeForTravel << '\t' << t.timeDeparture << '\n'; // \t разделяет столбцы в excel, \n - строки

return out;

}

bool operator<(TrainRasp& t1, TrainRasp& t2) // перегрузка оператора сравнения

{

return t1.timeDeparture < t2.timeDeparture; // сравнение по времени отправления

}

bool operator>(TrainRasp& t1, TrainRasp& t2)

{

return t1.timeDeparture > t2.timeDeparture;

}

bool operator>=(TrainRasp& t1, TrainRasp& t2)

{

return t1.timeDeparture >= t2.timeDeparture;

}

bool operator<=(TrainRasp& t1, TrainRasp& t2)

{

return t1.timeDeparture <= t2.timeDeparture;

}

#pragma once

#include <fstream>

struct TrainRasp // строчка таблицы поездов

{

int num; // number

int type; //1 - fast, 2 - passanger, 0 - not valid

int timeForTravel; //in minutes

int timeDeparture; //in minutes

};

// перегрузка операторов

std::istream& operator>>(std::istream& in, TrainRasp& t);

std::ostream& operator<<(std::ostream& out, TrainRasp& t);

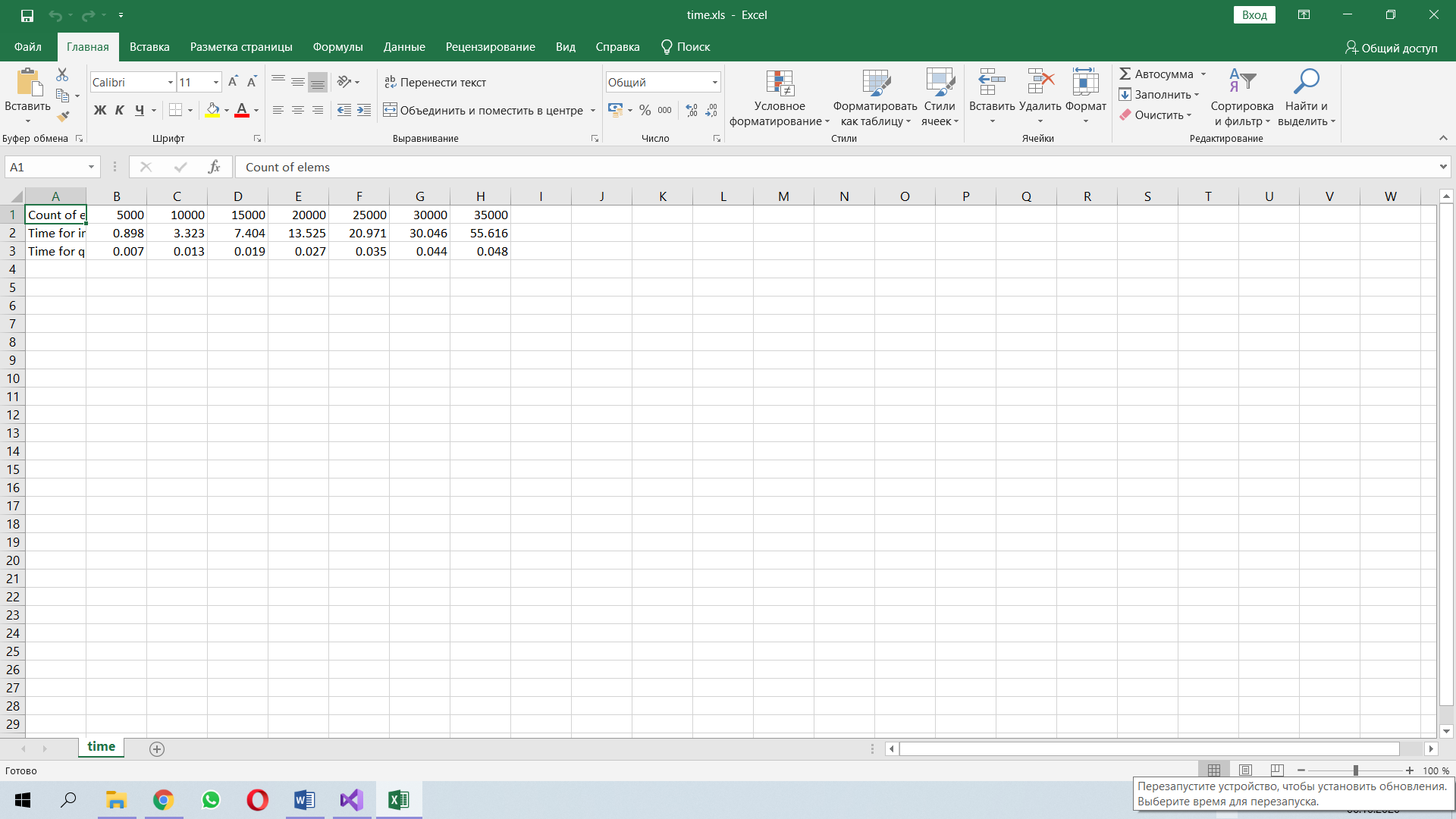
bool operator<(TrainRasp& t1, TrainRasp& t2);

bool operator>(TrainRasp& t1, TrainRasp& t2);

bool operator>=(TrainRasp& t1, TrainRasp& t2);

bool operator<=(TrainRasp& t1, TrainRasp& t2);

**ТЕСТЫ**



**Вывод:** быстрая сортировка работает намного быстрее чем сортировка простыми вставками.