**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 № 1**

**По курсу «Алгоритмизация и программирование»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  | | --- | --- | |  | ФИО студента | | Номер группы | Дата |
| Рыбаков Владислав Евгеньевич | БПМ-204 | 10.10.2021 |
|  |
|  |
|  |

**Москва – 2021 г.**

**ОПИСАНИЕ ПРОГРАММЫ**

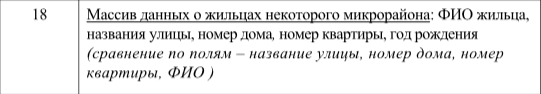
Программа сделана для сравнения двух алгоритмов сортировки: сортировка вставками и шейкер-сортировка.

На вход программа получает текстовый файл с базой данных о 10000 жителях, считывает их в динамический массив, далее идет ввод пользоваетелем количества сортируемых строк N (длины сортируемых массивов) и после этого идет оператор выбора алгоритма сортировки. Далее программа сортирует сначала первые N строк массива, потом вторые N строк и так далее, всего 6 раз (если это возможно и 6 массивов длины N не превышают 10000 элементов). Таким образом мы сортируем 6 массивов длины N с различным набором данных, используя лишь один большой массив.

На выходе программа выдает время сортировки каждого массива длины N.

**ЗАДАНИЕ (вариант №18)**

Реализовать сортировку вставками и шейкер-сортировку для массива данных о жильцах некоторого микрорайона. Перегрузить операторы сравнения для сравнения объектов. Входные данные брать из текстового или иного файла. Провести исследования по работе алгоритмов на разных наборах данных и построить графики зависимости времени работы алгоритма от количества сортируемых строк таблицы.



**ОПИСАНИЕ АЛГОРИТМОВ**

Сортировка вставками: (источник – «Алгоритмы на C++». Автор - Роберт Седжвик)

Элементы, находящиеся слева от текущего индекса, уже отсортированы в нужном порядке, но могут быть передвинуты вправо, чтобы освободить место меньшим элементам, которые могут обнаружиться позднее. Проходя по массиву слева направо, текущий элемент мы вставляем на нужную позицию в отсортированной части массива, при этом двигая вправо те элементы, которые больше него. Сортировка идет до тех пор, пока индекс не дойдет до конца массива.

Шейкер-сортировка: (источник – лекции по АиП. Автор – Сластников С.А. (в книгах «Алгоритмы на C++» Роберта Седжвика и «Алгоритмы. Справочник.» Джорджа Хайнемана шейкер-сортировки нет))

За каждый проход по массиву элементы последовательно попарно сравниваются и, если порядок в паре неверный, то производится обмен элементов. При каждом проходе алгоритма по неотсортированной части массива, очередной наибольший элемент массива ставится на свое место в конце массива рядом с предыдущим наибольшим элементом, а наименьший элемент сдвигается на 1 позицию к началу массива. В шейкер-сортировке направление сортировки поочередно меняется, чтобы элементы, которые находятся далеко от своих отсортированных позиций, смогли быстро занять свои правильные места. Алгоритм работает до тех пор, пока левая и правая границы отсортированных массивов не сомкнутся, либо пока в массиве еще есть перестановки.

**РЕШЕНИЕ**

Заголовочный файл:

#pragma once // для того, чтобы заголовочник не подключался больше 1 раза

#include <string>

using namespace std;

class Citizen

{

private: // приватные поля

string surname; // фамилия

string name; // имя

string father; // отчество

string street; // улица

int house; // номер дома

int apart; // номер квартиры

int year; // год рождения

public:

Citizen() = default; // конструктор по умолчанию

Citizen(const Citizen &a); // конструктор копирования

Citizen(const string surname, const string name, const string father, const string street, const int house, const int apart, const int year); // конструктор со всеми аргументами

void set(const string surname, const string name, const string father, const string street, const int house, const int apart, const int year); // сеттер для сразу всех полей класса

string get\_surname() const; // геттер для фамилии

string get\_name() const; // геттер для имени

string get\_father() const; // геттер для отчества

string get\_street() const; // геттер для улицы

int get\_house() const; // геттер для номера дома

int get\_apart() const; // геттер для номера квартиры

int get\_year() const; // геттер для года рождения

Citizen& operator=(const Citizen &a); // перегрузка оператора присваивания

friend bool operator>(const Citizen& a, const Citizen& b); // перегрузка операторов сравнения

friend bool operator==(const Citizen& a, const Citizen& b);

friend bool operator>=(const Citizen& a, const Citizen& b);

friend bool operator<(const Citizen& a, const Citizen& b);

friend bool operator<=(const Citizen& a, const Citizen& b);

};

// прототипы функций

void createMass(Citizen\* a); // функция заполнения массива данными из файла

void insert\_sort(Citizen\* a, const int start, const int n); // сортировка вставками

void sheker\_sort(Citizen\* mass, const int start, const int n); // шейкер-сортировка

Основной файл с функцией main:

#include <iostream> //стандартная библиотека

#include <string> // библиотека для строк

#include <fstream> // библиотека для работы с файлами

#include <ctime> // библиотека для работы с временем

#include "citizen.h" // подключение заголовочного файла с объявлением класса и прототипами функций

using namespace std; // использование пространства имен

#define TICK(X) clock\_t X = clock() // в переменную Х кладем начальное время

#define TOCK(X) printf("Time: %g sec.\n", (double)(clock() - (X)) / CLOCKS\_PER\_SEC) // выводим разницу между настоящим и засеченным временем в переменной Х (g - кратчайшее представление чисел с плавающей точкой)

int main()

{

Citizen\* one = new Citizen[10000]; // выделяем память для массива на 10000 элементов

createMass(one); // функция заполнения массива данными из файла

int n, ch; // переменные для количества сортируемых элементов и для выбора алгоритма сортировки

cout << "Enter number of lines: ";

cin >> n;

cout << "Choose sort method:\n(1) Insert sort\n(2) Sheker sort\nYour choice: "; // вывод меню

cin >> ch;

switch (ch) {

case 1: // случай выбора сортировки вставками

cout << "\nTime of 6 different datasets:\n";

for (int i = 0; i < 6; i++) { // сделаем сортировку для 6 разных наборов данных по n элементов в каждом

if (i \* n + n < 10000) { // проверяем, что мы не выйдем за рамки массива

cout << i + 1 << " dataset "; // порядковый номер датасета

TICK(Time); // засекаем время

insert\_sort(one, i \* n, n); // запускаем сортировку. i\*n - начальный элемент сортируемого массива, то есть мы сортируем первые n элементов, потом следующие n элементов и так далее

//таким образом мы делаем сортировку разных наборов данных, используя одну большую таблицу

TOCK(Time); // выводим затраченное время на экран

}

}

break;

case 2: // случай выбора шейкер-сортировки

cout << "\nTime of 6 different datasets:\n";

for (int i = 0; i < 6; i++) {

if (i \* n + n < 10000) { // проверяем, что мы не выйдем за рамки массива

cout << i << " dataset ";

TICK(Time); // запускаем время

sheker\_sort(one, i \* n, n); // запускаем сортировку. i\*n - начальный элемент сортируемого массива, то есть мы сортируем первые n элементов, потом следующие n элементов и так далее

//таким образом мы делаем сортировку разных наборов данных, используя одну большую таблицу

TOCK(Time); // выводим время

}

}

break;

}

delete [] one; // обязательно очищаем память после выделения памяти

}

Файл с реализацией функций:

#include "citizen.h"

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

Citizen::Citizen(const string surname, const string name, const string father, const string street, const int house, const int apart, const int year) { // реализация конструктора со всеми аргументами

this->surname = surname; // присваиваем полям объекта соответствующие полученные значения

this->name = name;

this->father = father;

this->street = street;

this->house = house;

this->apart = apart;

this->year = year;

}

void Citizen::set(const string surname, const string name, const string father, const string street, const int house, const int apart, const int year) { // реализация сеттера

this->surname = surname; // присваиваем полям объекта соответствующие полученные значения

this->name = name;

this->father = father;

this->street = street;

this->house = house;

this->apart = apart;

this->year = year;

}

Citizen::Citizen(const Citizen& a) { // реализация конструктора копирования

this->surname = a.get\_surname(); // полям созданного объекта присваиваем значения полей полученного на вход объекта с помощью геттеров

this->name = a.get\_name();

this->father = a.get\_father();

this->street = a.get\_street();

this->house = a.get\_house();

this->apart = a.get\_apart();

this->year = a.get\_year();

}

Citizen& Citizen::operator=(const Citizen& a) { // реализация перегрузки оператора присваивания

this->surname = a.get\_surname(); // полям созданного объекта присваиваем значения полей полученного на вход объекта с помощью геттеров

this->name = a.get\_name();

this->father = a.get\_father();

this->street = a.get\_street();

this->house = a.get\_house();

this->apart = a.get\_apart();

this->year = a.get\_year();

return \*this; //возвращаем ссылку на текущий объект

}

bool operator>(const Citizen& a, const Citizen& b) { // реализация перегрузки оператора > (по условию приоритета сравнения полей из варианта лабораторной работы)

if (a.get\_street() > b.get\_street()) { // сравниваем улицы двух объектов в лексикографическом порядке

return true; // если левая больше, то возвращаем true

}

else if ((a.get\_street() == b.get\_street()) and (a.get\_house() > b.get\_house())) { // если улицы одинаковые, сравниваем номер домов

return true; // если левая больше, то возвращаем true

}

else if ((a.get\_street() == b.get\_street()) and (a.get\_house() == b.get\_house()) and (a.get\_apart() > b.get\_apart())) { // если номера домов равные сравниваем номера квартир

return true; // если левая больше, то возвращаем true

}

else if ((a.get\_street() == b.get\_street()) and (a.get\_house() == b.get\_house()) and (a.get\_apart() == b.get\_apart()) and (a.get\_surname() > b.get\_surname())) { // если номера квартир равны, то сравниваем фамилии

return true; // если левая больше, то возвращаем true

}

else if ((a.get\_street() == b.get\_street()) and (a.get\_house() == b.get\_house()) and (a.get\_apart() == b.get\_apart()) and (a.get\_surname() == b.get\_surname()) and (a.get\_name() > b.get\_name())) { // если фамилии равны, то сравниваем имена

return true; // если левая больше, то возвращаем true

}

else if ((a.get\_street() == b.get\_street()) and (a.get\_house() == b.get\_house()) and (a.get\_apart() == b.get\_apart()) and (a.get\_surname() == b.get\_surname()) and (a.get\_name() == b.get\_name()) and (a.get\_father() > b.get\_father())) { // если имена равны то сравниваем отчества

return true; // если левая больше, то возвращаем true

}

return false; // иначе возвращаем false

}

bool operator==(const Citizen& a, const Citizen& b) { // перегружаем оператор сравнения ==

if ((a.get\_street() == b.get\_street()) and (a.get\_house() == b.get\_house()) and (a.get\_apart() == b.get\_apart()) and (a.get\_surname() == b.get\_surname()) and (a.get\_name() == b.get\_name()) and (a.get\_father() == b.get\_father())) return true; // если все поля объектов равны, то возвращаем true

return false; // иначе - flase

}

bool operator>=(const Citizen& a, const Citizen& b) { // перегружаем оператор >=

if (a > b || a == b) return true; // если левый объект больше или равен, то возвращаем true

return false; // иначе возвращаем false

}

bool operator<(const Citizen& a, const Citizen& b) { // перегружаем оператор <

if (a >= b) return false; // если левый объект больше или равен, то возвращаем false

return true; // иначе возвращаем true

}

bool operator<=(const Citizen& a, const Citizen& b) { // перегружаем оператор <=

if (a > b) return false; // если левый объект больше, то возвращаем false

return true; // иначе возвращаем true

}

string Citizen::get\_surname() const { return surname; } // геттеры, которые возвращают значение нужного поля объекта

string Citizen::get\_name() const { return name; }

string Citizen::get\_father() const { return father; }

string Citizen::get\_street() const { return street; }

int Citizen::get\_house() const { return house; }

int Citizen::get\_apart() const { return apart; }

int Citizen::get\_year() const { return year; }

void createMass(Citizen\* a) { // реализация записи информации из файла в массив

string path = "C:\\333\\dataset.txt"; // путь к файлу с базой данных

ifstream fin; // файловая переменная

fin.open(path); // открываем файл

if (!fin.is\_open()) { // если файл не открылся, то выдаем ошибку

cout << "Error" << endl;

}

else { // если открылся, то делаем печать данных в массив

int i = 0; // счетчик

string surname, name, father, street, house, apart, year; // так как файл текстовый, то все считываем из него в строковые переменные, названные соответственно с полями класса

while (!fin.eof()) { // пока не конец файла считываем из него строки

getline(fin, street, '\t'); // строки разделены табуляцией, которую указываем как разделитель

getline(fin, house, '\t'); // последовательно считываем все слова из одной строки

getline(fin, apart, '\t'); // одна строка - одна запись в БД

getline(fin, surname, '\t');

getline(fin, name, '\t');

getline(fin, father, '\t');

getline(fin, year); // после последней строчки идет символ окончания строки

if(i < 10000) a[i].set(surname, name, father, street, stoi(house), stoi(apart), stoi(year)); // проверяем что мы не вышли за пределы массива и записываем в i-ую переменную считанные значения

// функция std::stoi(string) переводит строку в целое число (не как раскодировку, а как написанное число, например строка "45" станет целым числом 45)

i++; // увеличиваем счетчик

}

}

fin.close(); // закрываем файл

}

void insert\_sort(Citizen\* a, const int start, const int n){ // реализация алгоритма сортировки вставками

// здесь start - порядковый номер элемента, с которого начинаем сортировку, а n - длина сортируемого массива

int i, j; // счетчики

Citizen value; // переменная класса, в которой временно будет храниться значение заменяемого элемента

for (i = start + 1; i < start + n; i++) // начиная со 2 элемента и до конца проходим по массиву из n элементов

{

value = a[i]; // помещаем i-тое значение во временную переменную

for (j = i - 1; j >= start && a[j] > value; j--) // далее проходимся по уже отсортированной части массива влево и находим подходящее для i-того элемента место

{

a[j + 1] = a[j]; // сдвигаем на 1 вправо все элементы отсортированной части массива, пока они больше нашего i-того элемента

}

a[j + 1] = value; // когда подходящее место найдено, то записываем туда значение нашего i-того элемента

}

}

void sheker\_sort(Citizen\* mass, const int start, const int n){ // реализация алгоритма шейкер-сортировки

int left = start, right = start + n - 1; // левая и правая границы сортируемой области массива

int flag = 1; // флаг наличия перемещений

// выполняем цикл пока левая и правая граница не сравняются и пока в массиве имеются перемещения

while ((left < right) && flag > 0) // пока левая граница меньше правой и пока элементы стоят неотсортированные

{

flag = 0; // ставим флаг равным 0

for (int i = left; i < right; i++) // двигаемся слева направо

{

if (mass[i] > mass[i + 1]) // если следующий элемент меньше текущего,

{ // меняем их местами

Citizen t = mass[i];

mass[i] = mass[i + 1];

mass[i + 1] = t;

flag = 1; // делаем флаг равным 1, тем самым показывая, что изменения были

}

}

right--; // сдвигаем правую границу на один элемент влево

for (int i = right; i > left; i--) // двигаемся справа налево

{

if (mass[i - 1] > mass[i]) // если предыдущий элемент больше текущего,

{ // меняем их местами

Citizen t = mass[i];

mass[i] = mass[i - 1];

mass[i - 1] = t;

flag = 1; // делаем флаг равным 1, тем самым показывая, что изменения были

}

}

left++; // сдвигаем левую границу на один элемент вправо

}

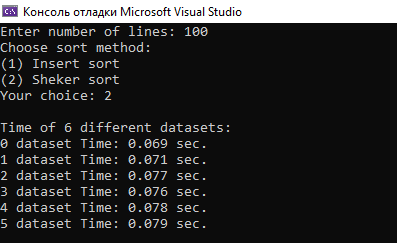
}

**ТЕСТЫ**

**Тест № 1**

*Результаты теста 1*

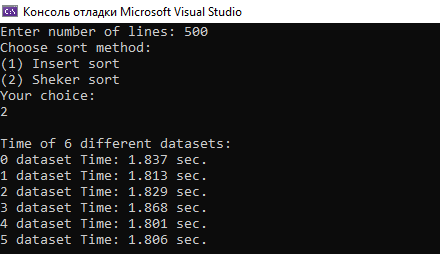
**Вводим число 100. Выбираем шейкер-сортировку. Программа сортирует 6 массивов по 100 элементов методом шейкер сортировки.**

****

**Тест № 2**

*Результаты теста 2*

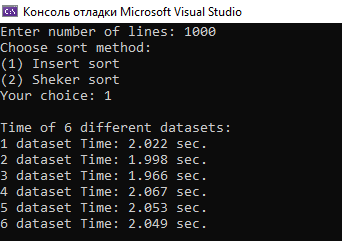
**Вводим число 500. Выбираем шейкер-сортировку. Программа сортирует 6 массивов по 500 элементов методом шейкер сортировки.**

****

**Тест № 3**

*Результаты теста …*

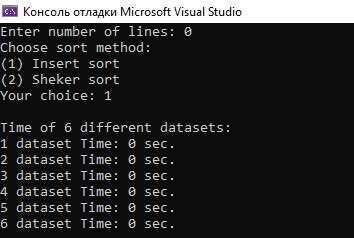
**Вводим число 1000. Выбираем сортировку вставками. Программа сортирует 6 массивов по 1000 элементов методом сортировки вставками.**

****

**Тест № 4**

*Результаты теста 4*

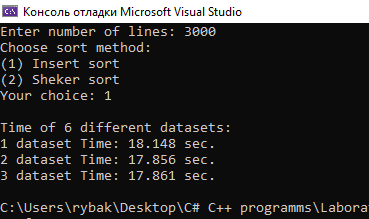
**Вводим число 0. Выбираем сортировку вставками. Программа сортирует 6 массивов по 0 элементов методом сортировки вставками, то есть по сути не делает ничего.**

****

**Тест № 5**

*Результаты теста 5*

**Вводим число 3000. Выбираем сортировку вставками. Программа сортирует 3 массивов по 3000 элементов методом сортировки вставками, так как на четвертый массив размера 3000 элементов уже не хватает базы данных.**

****

**ДАННЫЕ**

Данные сгенерированы случайным образом в таблице Excel, а потом сохранены в виде текстового файла, где одна строчка – одна запись из БД. Всего таких записей 10000.



**ИССЛЕДОВАНИЕ**

Таблица для сортировки вставками: (верхняя строчка – кол-во строк)

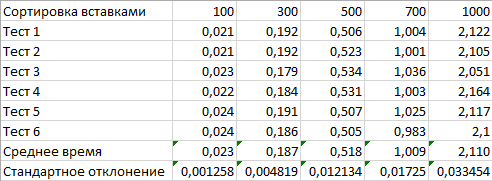


Таблица для шейкер сортировки: (верхняя строчка – кол-во строк)

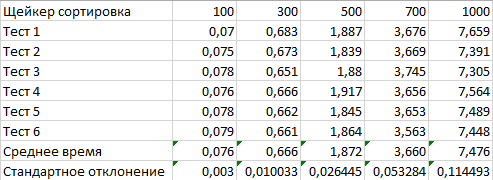
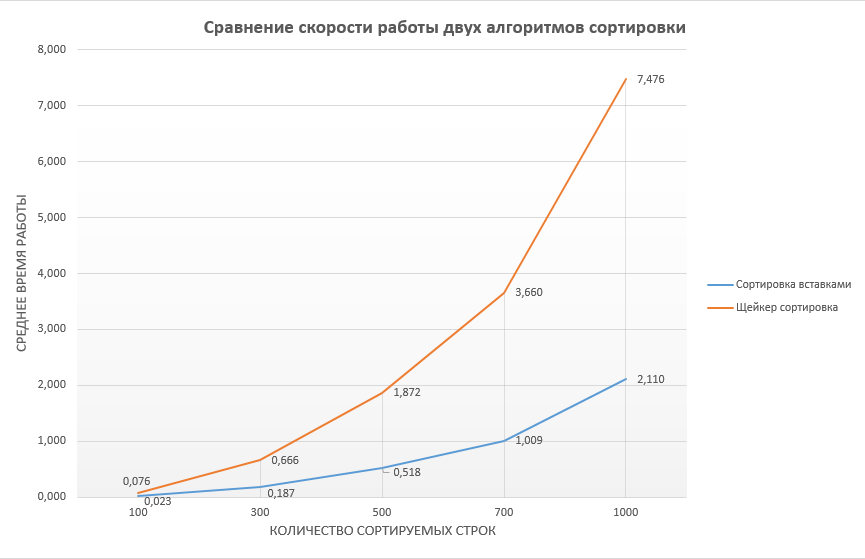


График:



**ВЫВОД**

Экспериментальным путем было доказано, что алгоритм сортировки вставками работает эффективнее, чем алгоритм шейкер-сортировки.

Поскольку у обоих логарифмов сложность это О(n^2), то у обоих алгоритмов графиком зависимости времени работы от количества строк является парабола, а значит разница между временем их работы различается только на константу. При различных наборах данных алгоритм сортировки вставками в среднем работает в 3,5 раза быстрее, чем алгоритм шейкер сортировки (это вычисляется из полученных нами данных).

В целом, оба алгоритма оказываются неэффективными и могут быть применимы только к небольшому количеству данных. Их преимущество – это простота понимания.