|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ | | | | | | | | | | | | |
| Федеральное государственное автономное образовательное учреждение  высшего образования | | | | | | | | | | | | |
| **Дальневосточный федеральный университет** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **Департамент программной инженерии и искусственного интеллекта** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **О Т Ч Е Т** | | | | | | | | | | | | |
| по лабораторной работе №1.3  дисциплина «Фундаментальные структуры данных и алгоритмы» | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | |  |  | Студент гр. Б9122-09.03.04прогин | | | |
|  |  |  | | А.В. Поляков | |
|  | | | | | | |  |  | (подпись) | |  | |
|  | | | | | | | | | | | | |
|  | | | | | | |  |  | Руководитель | | |  |
|  | | | | | | |  |  | ст. преподаватель | | | |
|  | | | |  |  | |  |  |  |  | О.А. Крестникова | |
|  | | | |  |  | |  |  | (подпись) |  | (И.О. Фамилия) | |
|  |  |  |  | | |  |  |  |  | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| г. Владивосток | | | | | | | | | | | | |
| 2023 | | | | | | | | | | | | |

**1 Неформальная постановка задачи**

1. Отсортировать данные входного файла в соответствии с заданным порядком сортировки ключей (код направления, ФИО). Данные в выходном файле должны быть представлены в табличном виде, первый и второй столбец должны содержать значения ключей в соответствии с заданным порядком сортировки, третий столбец - номер строки входного файла.

2. Проверить на устойчивость (привести примеры, доказывающие, что сортировка неустойчивая).

3. Сравнить алгоритмы сортировки по времени (без учета чтения из файла и записи в файл).

4. Определить, на каких значениях ключей достигается наихудшее и наилучшее значение времени сортировки.

Входные данные:

**1) текстовый файл,** каждая из строк которого содержит 2 ключа в табличном виде (первый столбец - 1 ключ – код направления, второй столбец - 2 ключ – ФИО. Все данные во входном файле корректные (проверять на корректность не нужно). Количество строк входного файла - 1000000.

**2) целое число n**- количество строк входного файла для обработки (10 ≤ n ≤ 1000000).

текстовый файл: KeysForSort.txt

Выходные данные:

2 текстовых файла, строки которых содержат отсортированные данные входного файла (первый файл - первым алгоритмом сортировки (вставками), второй файл - вторым алгоритмом сортировки (быстрая рекурсивная реализация, разбиение Ломуто, опорный – последний)), последняя строка - время, затраченное на сортировку.

текстовые файлы InsertionSort.txt, QuickSort.txt

**2 Описание алгоритмов сортировки**

Сортировка вставками:

**function** insertionSort(vector<Student>& students):

for i from i to vector.size - 1

Student key = students[i]

int j = i - 1

while j >= 0 and (students[j] < key)

students[j + 1] = students[j]

j = j - 1

end while

students[j + 1] = key;

end for

end function

Разбиение Ломуто и выбор опорного элемента

**function** partitioning(vector<Student>& students, int low, int high):

Student supporting = students[high]

i = low - 1

for j = low to j <= high - 1

if students[j] > supporting

i++

temp = students[i];

students[i] = students[j];

students[j] = temp;

end if

end for

swap(students[i + 1], students[high])

return (i + 1)

end function

Быстрая сортировка:

**function** quickSort(vector<Student>& students, int low, int high):

if low < high

sup = partitioning(students, low, high);

quickSort(students, low, sup - 1);

quickSort(students, sup + 1, high);

end if

end function

**3 Текст программы**

#include <iostream>

#include <fstream>

#include <string>

#include <cstdlib>

#include <ctime>

#include <vector>

#include <chrono>

using namespace std;

// Структура для хранения кода направления студента

struct Code {

char letter;

unsigned int numbers;

};

// Структура для хранения ФИО студента

struct FullName {

string surname;

string name;

string patronymic;

};

// Структура для хранения данных студента

struct Student {

Code code;

FullName fullName;

int lineNumber;

};

// Функция для генерации случайного кода направления

Code generateCode() {

Code code;

code.letter = 'B';

code.numbers = 1000 + rand() % 9000;

return code;

}

// Перегрузки оператора сравнения для Student

bool operator<(const Student& a, const Student& b) {

return (a.code.letter < b.code.letter) ||

(a.code.letter == b.code.letter && a.code.numbers < b.code.numbers) ||

(a.code.letter == b.code.letter && a.code.numbers == b.code.numbers && a.fullName.surname < b.fullName.surname) ||

(a.code.letter == b.code.letter && a.code.numbers == b.code.numbers && a.fullName.surname == b.fullName.surname && a.fullName.name < b.fullName.name) ||

(a.code.letter == b.code.letter && a.code.numbers == b.code.numbers && a.fullName.surname == b.fullName.surname && a.fullName.name == b.fullName.name && a.fullName.patronymic < b.fullName.patronymic);

}

bool operator>(const Student& a, const Student& b) {

return (a.code.letter > b.code.letter) ||

(a.code.letter == b.code.letter && a.code.numbers > b.code.numbers) ||

(a.code.letter == b.code.letter && a.code.numbers == b.code.numbers && a.fullName.surname > b.fullName.surname) ||

(a.code.letter == b.code.letter && a.code.numbers == b.code.numbers && a.fullName.surname == b.fullName.surname && a.fullName.name > b.fullName.name) ||

(a.code.letter == b.code.letter && a.code.numbers == b.code.numbers && a.fullName.surname == b.fullName.surname && a.fullName.name == b.fullName.name && a.fullName.patronymic > b.fullName.patronymic);

}

// Перегрузка оператора вывода для Code

ostream& operator<<(ostream& os, const Code& code) {

os << code.letter << code.numbers;

return os;

}

// Перегрузка оператора вывода для Student

ostream& operator<<(ostream& os, const Student& student) {

os << student.code.letter << student.code.numbers << " " << student.fullName.surname << " " << student.fullName.name << " " << student.fullName.patronymic;

return os;

}

// Генерация рандомного имени студента

FullName generateFullName() {

string names[] = {"Artem", "Ekaterina", "Maxim", "Alexander", "Dmitry", "Nikita", "Egor", "Mikhail", "Roman", "Vladimir", "Pavel", "Daniel", "Anton", "Alexey", "Sergei"};

string surnames[] = {"Polyakov", "Krushinina", "Johnson", "Brown", "Taylor", "Wilson", "Harris", "Clark", "White", "Anderson", "Allen", "Davis", "Martin", "Hall", "Lee"};

string patronymics[] = {"Victorovich", "Vladislavovna", "Ivanovich", "Petrovich", "Sidorovich", "Mikhailovich", "Fedorovich", "Egorovich", "Romanovich", "Vladimirovich", "Pavlovich", "Daniilovich", "Antonovich", "Alexandrovich", "Sergeevich"};

FullName fullName;

fullName.surname = surnames[rand() % 15];

fullName.name = names[rand() % 15];

fullName.patronymic = patronymics[rand() % 15];

return fullName;

}

// Генератор ключей для входного файла

void generateKyesFile() {

ofstream inFile("C:/Users/User/Desktop/FEFU/Second\_course/FDSA/Sorting/KeysForSort.txt");

for (int i = 0; i < 1000; ++i) {

Student student;

student.code = generateCode();

student.fullName = generateFullName();

inFile << student.code << " " << student.fullName.surname << " "

<< student.fullName.name << " " << student.fullName.patronymic << endl;

}

inFile.close();

cout << "Ключи сгенерированы\n" << endl;

}

// Функция для чтения данных студентов из файла

vector<Student> readKeyFromFile() {

ifstream inFile("C:/Users/User/Desktop/FEFU/Second\_course/FDSA/Sorting/KeysForSort.txt");

vector<Student> students;

int lineNumber = 1;

while (!inFile.eof()) {

Student student;

inFile >> student.code.letter >> student.code.numbers;

inFile >> student.fullName.surname >> student.fullName.name >> student.fullName.patronymic;

if (!inFile.fail()) {

student.lineNumber = lineNumber++;

students.push\_back(student);

}

}

inFile.close();

return students;

}

void printStudentVector(vector<Student> students) {

for (const auto& student : students) {

cout << student.code << " " << student.fullName.surname << " "

<< student.fullName.name << " " << student.fullName.patronymic << endl;

}

}

// Функция сортировки вставками

void insertionSort(vector<Student>& students) {

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

Student key = students[i];

int j = i - 1;

while (j >= 0 && students[j] < key) {

students[j + 1] = students[j];

j = j - 1;

}

students[j + 1] = key;

}

}

void printInsertionSortedVectorToFile(vector<Student> students, chrono::duration<double> insertionSortTime, ofstream& outFile) {

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

outFile << students[i] << " " << students[i].lineNumber << endl;

}

outFile << insertionSortTime.count() << " секунд" << endl;

cout << "Отсортированные вставками элементы выведены в файл\n" << endl;

}

int partitioning(vector<Student>& students, int low, int high) {

Student supporting = students[high]; // Опорный элемент

int i = low - 1;

for (int j = low; j <= high - 1; j++) {

// Если текущий элемент меньше или равен опорному

if (students[j] > supporting) {

i++; // увеличиваем индекс меньшего элемента

auto temp = students[i];

students[i] = students[j];

students[j] = temp;

}

}

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

return (i + 1);

}

void quickSort(vector<Student>& students, int low, int high) {

if (low < high) {

int sup = partitioning(students, low, high);

quickSort(students, low, sup - 1);

quickSort(students, sup + 1, high);

}

}

void printQuickSortedVectorToFile(vector<Student> students, ofstream& outFile) {

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

outFile << students[i] << " " << students[i].lineNumber << endl;

}

cout << "Отсортированные быстрой рекурсивной реализацией элементы выведены в файл\n" << endl;

}

int main() {

srand(time(nullptr));

vector<Student> students;

//generateKyesFile();

students = readKeyFromFile();

ofstream insertionFile("C:/Users/User/Desktop/FEFU/Second\_course/FDSA/Sorting/InsertionSort.txt");

// Сортировка вставками

auto start = chrono::high\_resolution\_clock::now();

vector<Student> forInsertionSort = students;

insertionSort(forInsertionSort);

auto end = chrono::high\_resolution\_clock::now();

chrono::duration<double> insertionSortTime = end - start;

cout << "Время сортировки вставками: " << insertionSortTime.count() << " секунд" << endl;

printInsertionSortedVectorToFile(forInsertionSort, insertionSortTime, insertionFile);

insertionFile.close();

ofstream quickFile("C:/Users/User/Desktop/FEFU/Second\_course/FDSA/Sorting/QuickSort.txt");

// Быстрая рекурсивная реализация

start = chrono::high\_resolution\_clock::now();

vector<Student> forQuickSort = students;

quickSort(forQuickSort, 0, forQuickSort.size() - 1);

end = chrono::high\_resolution\_clock::now();

chrono::duration<double> quickSortTime = end - start;

cout << "Время быстрой сортировки: " << quickSortTime.count() << " секунд" << endl;

printQuickSortedVectorToFile(forQuickSort, quickFile);

quickFile << quickSortTime.count() << " секунд" << endl;

quickFile.close();

return 0;

}

**4. Тесты**

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Быстрая сортировка**  **Quick sort** | **Сортировка вставками**  **Insertion sort** |
| Сортировка на случайном наборе  из 30000 ключей  (средний случай) | Вход: KeysForSort.txt  Выход: QuickSort.txt  Среднее время: 0.041 секунд | Вход: KeysForSort.txt  Выход: InsertionSort.txt  Среднее время: 11.36 секунд |
| Сортировка упорядоченного по убыванию набора из 30000 ключей  (худший случай для быстрой и для вставок) | Вход: KeysForSort.txt  Выход: QuickSort.txt  Среднее время: 24.17 секунды | Вход: KeysForSort.txt  Выход: InsertionSort.txt  Среднее время: 0.013 секунд |
| Сортировка упорядоченного по возрастанию набора из 30000 ключей  (худший случай для быстрой, лучший случая для вставок) | Вход: KeysForSort.txt  Выход: QuickSort.txt  Среднее время: 23.41 секунды | Вход: KeysForSort.txt  Выход: InsertionSort.txt  Среднее время: 47.01 секунд |
| Сортировка частично упорядоченного набора из 30000 ключей с элементами, при которых некоторые опорные элементы разбивают подмассивы пополам  (стремящийся к лучшему случай для быстрой) | Вход: KeysForSort.txt  Выход: QuickSort.txt  Среднее время: 6.73 секунд | Вход: KeysForSort.txt  Выход: InsertionSort.txt  Среднее время: 0.013 секунд |
| Проверка на устойчивость на небольшом наборе входных данных | Вход: KeysForSort.txt | Вход: KeysForSort.txt |
|  | Выход: QuickSort.txt | Выход: InsertionSort.txt |

**6. Выводы**

Быстрая сортировка не эффективна, если значения уже упорядочены (либо по возрастанию, либо по убыванию). Если элементы в порядке убывания и сортировка вставками тоже в порядке убывания, то на таком наборе сортировка вставками будет максимально эффективна (время у быстрой n^2, у вставок n)

Быстрая сортировка эффективна на случайных наборах значений, то есть, когда большая вероятность того, что опорные элементы будут разбивать подмассивы как минимум 1 к 9 и как максимум пополам. Сортировка вставками проявляет меньшую эффективность на таких наборах (время у быстрой n\*log(n), у вставок n^2).

Быстрая сортировка эффективна на наборах, когда опорные элементы разбивают подмассивы пополам. Сортировка вставками на таких наборах не так эффективна (время у быстрой n\*log(n), у вставок n^2).

Проверка на устойчивость:

Неустойчивость быстрой сортировки показана в последнем тесте из пункта 4: в файле пять одинаковых ключей «B8551 Polyakov Sergei Romanovich» в строках входного файла 3, 4, 6, 8 и 10. После сортировки они располагаются в порядке 10, 3, 4, 8, 6 – элементы менялись местами, что говорит о том, что эта сортировка неустойчивая.

Устойчивость быстрой сортировки показана в этом же тесте: в файле пять одинаковых ключей «B8551 Polyakov Sergei Romanovich» в строках входного файла 3, 4, 6, 8 и 10. После сортировки они располагаются в порядке 3, 4, 6, 8, 10 – элементы не менялись местами, что говорит о том, что эта сортировка устойчивая.