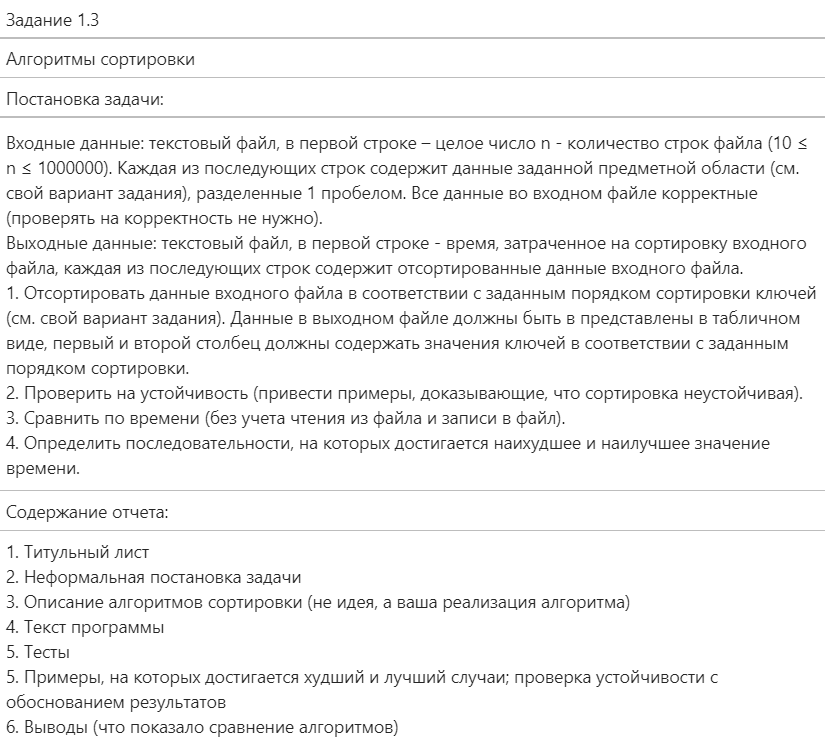
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ | | | | | | | | | | | | |
| Федеральное государственное автономное образовательное учреждение  высшего образования | | | | | | | | | | | | |
| **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. Б9121-09.03.04прогин | | | |
|  |  |  | | Шевелёв Р.В. | |
|  | | | | | | |  |  | (подпись) | |  | |
|  | | | | | | | | | | | | |
|  | | | | | | |  |  | Руководитель | | |  |
|  | | | | | | |  |  | ст. преподаватель | | | |
|  | | | |  |  | |  |  |  |  | О.А. Крестникова | |
|  | | | |  |  | |  |  | (подпись) |  | (И.О. Фамилия) | |
|  |  |  |  | | |  |  |  |  | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| г. Владивосток | | | | | | | | | | | | |
| 2022 | | | | | | | | | | | | |

**1 Неформальная постановка задачи**



Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как стол

Автоматически созданное описание

Шевелев Роман - 1 ключ - по возрастанию, 2 ключ - по убыванию, алгоритмы: Простые вставки, Быстрая (итеративная реализация (с использованием стека), разбиение Хоара, опорный - медиана из трёх)

**2 Описание типа + спецификация подпрограмм + тесты**

struct Flight\_Number {

std::string sym; //символьная часть номера полёта

int num; //номер полёта

};

struct FIO {

std::string surname; //фамилия

std::string name; //имя

std::string patronymic; //отчество

};

struct Date {

int number; //дата рождения

int month; //месяц рождения

int year; //год рождения

}

struct Passenger {

Flight\_Number flight\_number; //код рейса

FIO fio; //ФИО

Date date; //дата рождения

}

int compare\_flight\_number(Flight\_Number first, Flight\_Number second)

Сравнивает 2 записи типа Flight\_Number по значениям (sym, num) типа std::string и int соответственно. Возвращает 1, если код first окажется больше чем second, -1 в противном случае. При равенстве кодов, возвращает 0.

int compare\_FIO(FIO first, FIO second)

Сравнивает 2 записи типа FIO по значениям (surname, name, patronymic) типа std::string. Возвращает 1, если код first окажется больше чем second, -1 в противном случае. При равенстве кодов, возвращает 0.

int compare(Passenger\* first, Passenger\* second)

Проверяет эквивалентность кодов (first, second) типа Passenger. Возвращает 1, если код first окажется больше, чем second, -1 в противном случае. При равенстве кодов, возвращает 0.

Date parseDate(std::string input)

Возвращает заполненную структуру типа Date, разбивая входную строку через разделитель – ‘.’.

Flight\_Number parseNumber(std::string input)

Возвращает заполненную структуру типа Flight\_Number, разбивая входную строку на 2 символа и 4 цифры.

std::vector<Passenger\*> readData(std::string path)

Считывает из файла с именем **path** типа string записи типа **Passenger** и помещает их в массив **data** типа **vector**.

void generateFile(int n, std::string path)

Генерирует файл с именем path типа string, в которых содержатся записи Passenger(2буквы4цифры Фамилия Имя Отчество Дата.Месяц.Год)

void writeData(std::vector<Passenger\*>data, std::string path, long long microseconds)

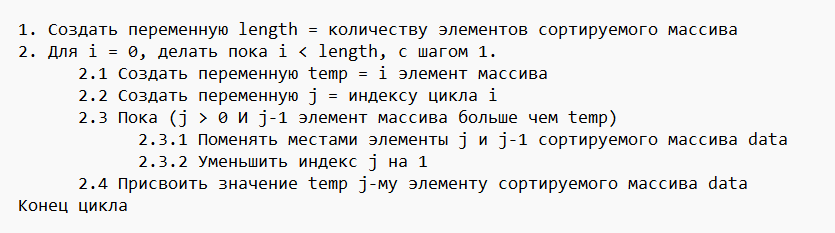
Записывает отсортированные записи data типа Passenger. В первой строке файла сохраняет время microseconds типа long long, за которое произошла сортировка.

void StraightInsert(std::vector<Passenger\*>& data)

Сортирует массив data типа vector с записями типа Passenger в первую очередь по возрастанию по номеру рейса, во вторую – по убыванию ФИО пассажира. Сортировка выполняется алгоритмом “Простые вставки”.

Результат сортировки записывается при помощи функции writeData.

Алгоритм на естественном языке:



std::pair<int,int> partition(std::vector<Passenger\*>& data, int start, int end)

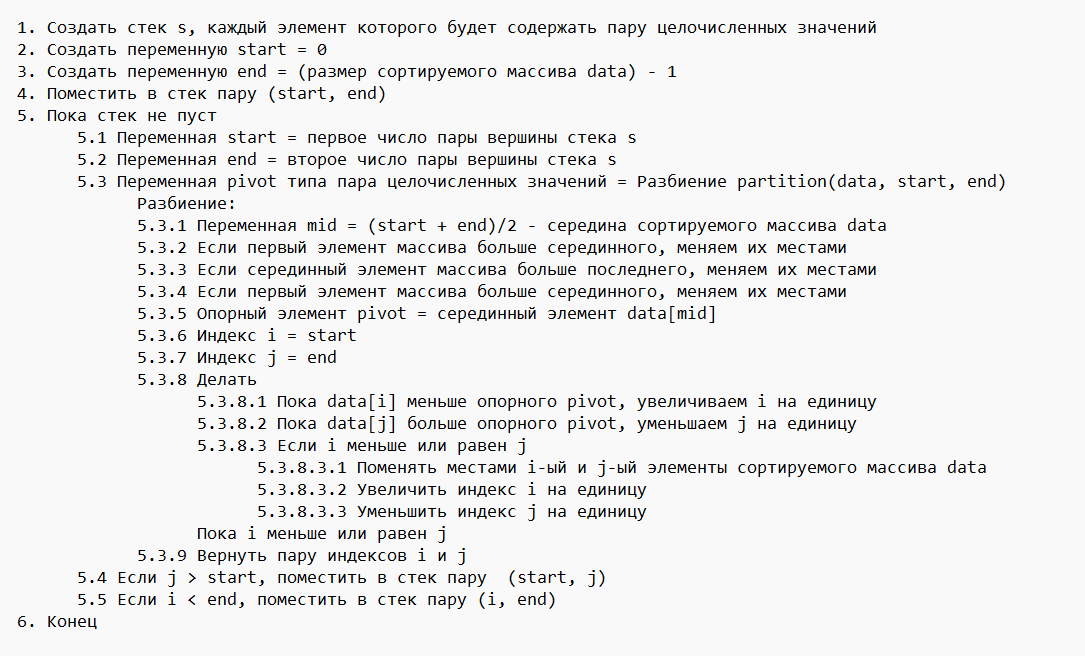
Выполняет сортировку массива с помощью разбиения Хоара (два индекса i и j, идущие навстречу друг другу с разных концов массива). Является воспомогательной функцией для основной iterativeQuicksort. Возвращает два индекса i, j типа int с помощью которых будет происходить следующие итерации сортировки.

void iterativeQuicksort(std::vector<Passenger\*>& data)

Сортирует массив data типа vector с записями типа Passenger в первую очередь по возрастанию по номеру рейса, во вторую – по убыванию ФИО пассажира. Сортировка выполняется алгоритмом “Быстрая сортировка с разбиением Хоара, итеративная реализация с использованием стека, опорный элемент – средний из трёх”.

Результат сортировки записывается при помощи функции writeData.

Алгоритм на естественном языке:



**3 Сравнительная таблица \ тесты**

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Сортировка 1к элементов Лучший случай:  Уже отсортированная последовательность | Файл с записями “1000Best” | Отсортированный файл  Файл с записями output.txt  Время сортировки вставками:  ~0 сек.  Время быстрой итеративной сортировки:  ~0 сек. |
| Сортировка 1к элементов Средний случай:  Случайная последовательность | Файл с записями “1000Middle” | Отсортированный файл  Файл с записями output.txt  Время сортировки вставками:  ~0 сек.  Время быстрой итеративной сортировки:  ~0 сек. |
| Сортировка 1к элементов Худший случай  Последовательность, отсортированная в обратном порядке | Файл с записями “1000Worst” | Отсортированный файл  Файл с записями output.txt  Время сортировки вставками:  ~1 сек.  Время быстрой итеративной сортировки:  ~0 сек. |
| Сортировка 10к элементов Лучший случай:  Уже отсортированная последовательность | Файл с записями “10000Best” | Отсортированный файл  Файл с записями output.txt  Время сортировки вставками:  ~0 сек.  Время быстрой итеративной сортировки:  ~0 сек. |
| Сортировка 10к элементов Средний случай:  Случайная последовательность | Файл с записями “10000Middle” | Отсортированный файл  Файл с записями output.txt  Время сортировки вставками:  ~47 сек.  Время быстрой итеративной сортировки:  ~1 сек. |
| Сортировка 10к элементов Худший случай  Последовательность, отсортированная в обратном порядке | Файл с записями “10000Worst” | Отсортированный файл  Файл с записями output.txt  Время сортировки вставками:  ~97 сек.  Время быстрой итеративной сортировки:  ~1 сек. |
| Сортировка 40к элементов Лучший случай:  Уже отсортированная последовательность | Файл с записями “40000Best” | Отсортированный файл  Файл с записями output.txt  Время сортировки вставками:  ~0 сек.  Время быстрой итеративной сортировки:  ~3 сек. |
| Сортировка 40к элементов Средний случай:  Случайная последовательность | Файл с записями “40000Middle” | Отсортированный файл  Файл с записями output.txt  Время сортировки вставками:  ~12 минут 48 сек.  Время быстрой итеративной сортировки:  ~4 сек. |
| Сортировка 40к элементов Худший случай  Последовательность, отсортированная в обратном порядке | Файл с записями “40000Worst” | Отсортированный файл  Файл с записями output.txt  Время сортировки вставками:  ~26 минут 3 сек.  Время быстрой итеративной сортировки:  ~3 сек. |

**4 Выводы**

Анализ временных результатов и их сравнение показали, что итеративная реализация быстрой сортировки быстрее справляется со средним и худшим случаем, чем сортировка простыми вставками. Однако, сортировка вставками оказалась более эффективна на множествах, которые попадают под “Лучший случай”. То есть на множествах с малым кол-вом перестановок сортировка вставками справится лучше. На множествах средним и наихудшим кол-вом перестановок выигрывает быстрая сортировка с разбиением Хоара, причём сортировка вставками оказывается тотально неэффективной.  
 Это подтверждается таблицей сложности алгоритмов сортировок:



**5 Текст программы**

#include <iostream>

#include <vector>

#include <fstream>

#include <string>

#include <chrono>

#include <stack>

struct Flight\_Number {

std::string sym;

int num;

};

struct FIO {

std::string surname;

std::string name;

std::string patronymic;

};

struct Date {

int number;

int month;

int year;

};

struct Passenger {

Flight\_Number flight\_number;

FIO fio;

Date date;

};

int compare\_flight\_number(Flight\_Number first, Flight\_Number second) {

if (first.sym > second.sym) return 1;

if (first.sym < second.sym) return -1;

if (first.num > second.num) return 1;

if (first.num < second.num) return -1;

return 0;

}

int compare\_FIO(FIO first, FIO second) {

if (first.surname > second.surname) return 1;

if (first.surname < second.surname) return -1;

if (first.name > second.name) return 1;

if (first.name < second.name) return -1;

if (first.patronymic > second.patronymic) return 1;

if (first.patronymic < second.patronymic) return -1;

return 0;

}

int compare(Passenger\* first, Passenger\* second) {

if (compare\_flight\_number(first->flight\_number, second->flight\_number) == 1) return 1;

if (compare\_flight\_number(first->flight\_number, second->flight\_number) == -1) return -1;

if (compare\_FIO(first->fio, second->fio) == 1) return -1;

if (compare\_FIO(first->fio, second->fio) == -1) return 1;

return 0;

}

Date parseDate(std::string input) {

Date date;

int dots\_cnt = 0;

for (int i = 0; input.length(); i++) {

if (input[i] == '.') {

dots\_cnt++;

continue;

}

else {

if (dots\_cnt == 0) {

date.number = stoi(input.substr(i, 2));

i++;

}

else if (dots\_cnt == 1) {

date.month = stoi(input.substr(i, 2));

i++;

}

else {

date.year = stoi(input.substr(i, 4));

break;

}

}

}

return date;

};

Flight\_Number parseNumber(std::string input) {

Flight\_Number number;

number.sym = input.substr(0, 2);

number.num = stoi(input.substr(2, 4));

return number;

}

std::vector<Passenger\*> readData(std::string path) {

setlocale(LC\_ALL, "Russian");

std::ifstream fin;

std::vector<Passenger\*>data;

fin.open(path);

int n; fin >> n;

std::string s;

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

{

Passenger\* person = new Passenger;

fin >> s;

person->flight\_number = parseNumber(s);

fin >> person->fio.surname >> person->fio.name >> person->fio.patronymic;

fin >> s;

person->date = parseDate(s);

data.push\_back(person);

}

fin.close();

return data;

}

void generateFile(int n, std::string path) {

std::ofstream fout;

fout.open(path);

fout << n << '\n';

std::string surnames[126] = { "Антонов", "Зиновьев", "Куропаткин", "Матанализов", "Алгоритмов", "Сортировков", "Иванова", "Кулибина", "Литерная", "Блоха", "Бабай", "Базар", "Бибик", "Гадючка", "Дохлик", "Жуйвода", "Непейвода", "Моська", "Голобев", "Дракула", "Кощеев", "Сова", "Слива", "Интралигатор", "Кисель", "Зажерило",

"Алиферов", "Андреев", "Анненков", "Барбаянов", "Башинский", "Березницкий", "Бессонов", "Бобровский", "Богатьев", "Божок", "Бондаренко", "Бухалихин", "Ван", "Ваяй", "Виноградов", "Виссарионов", "Власов", "Власов", "Гандзюк", "Герус", "Гоголева", "Горбарёв", "Давыдов", "Данишимаев", "Ди", "Егорова", "Еньшин", "Ерёменко",

"Ермак", "Ерошенко", "Зеленин", "Иванов", "Карпачёв", "Касьяненко", "Киянов", "Козачко", "Колпакова", "Коряушкин", "Косенко", "Косилов", "Костюченко", "Кочетков", "Кропочев", "Кузьмин", "Кутузов", "Лагунова", "Лазарев", "Лебединский", "Литовченко", "Лукин", "Макогон", "Мареев", "Мартын", "Матушкин", "Машкин", "Михайлов", "Моисеев",

"Мулява", "Мызников", "Налимов", "Нечепуренко", "Олейникова", "Осипенко", "Охлопков", "Павлов", "Пак", "Пасько", "Перепечаев", "Подолинский", "Половников", "Попов", "Проценко", "Пьявка", "Раздобаров", "Распутный", "Селиверстов", "Сивцева", "Спорышев", "Старцев", "Столетний", "Сутугин", "Тарасов", "Тахавов", "Терёхин",

"Тимофеев","Трегубова", "Туровский", "Тюкова", "Уваров", "Хегай", "Хмелевский", "Храмцова", "Царегородцев", "Цыплаков", "Чечёткин", "Шевелёв", "Шешеня", "Шибанов", "Шмидт", "Якуба" };

std::string names[65] = { "Агнес", "Адольф", "Адриан", "Борис", "Валентин", "Василий", "Виталий", "Георгий", "Григорий", "Даниил", "Денис", "Дмитрий", "Евгений", "Егор", "Иван", "Игорь", "Константин", "Леонид", "Максим", "Матвей", "Никита", "Николай", "Олег", "Павел", "Пётр", "Роман", "Руслан", "Сергей", "Степан", "Тимур", "Фёдор", "Юрий",

"Ярослав", "Алиса", "Александра", "Алёна", "Алла", "Анастасия", "Анжелика", "Валентина", "Вероника", "Виктория", "Галина", "Дарья", "Диана", "Елизавета","Жанна", "Инна", "Ирина", "Карина", "Кристина", "Ксения", "Маргарита", "Мария", "Милана", "Наталья", "Ника", "Оксана", "Ольга", "Руслана", "Светлана", "София", "Софья", "Тамара", "Жанна" };

std::string patronymic[36] = { "Александрович", "Алексеевич", "Анатольевич", "Аркадьевич", "Богданович", "Борисович", "Валерьевич", "Викторович", "Владимирович", "Вольфович", "Геннадиевич", "Григорьевич", "Денисович", "Евгеньевич", "Ефимович", "Игнатьевич", "Иосифович", "Кириллович", "Львович", "Матвеевич", "Николаевич", "Павлович", "Петрович",

"Робертович", "Романович","Саныч","Северинович","Станиславович","Тарасович","Тимофеевич","Федорович","Филиппович","Эдуардович","Юрьевич","Яковлевич","Ярославович" };

srand((unsigned)time(NULL));

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

{

std::string mon, date;

int month = rand() % 12 + 1;

if (month < 10) mon = '0' + std::to\_string(month);

else mon = std::to\_string(month);

int dat = rand() % 31 + 1;

if (dat < 10) date = '0' + std::to\_string(dat);

else date = std::to\_string(dat);

fout << char('A' + rand() % ('Z' - 'A')) << char('A' + rand() % ('Z' - 'A')) << std::to\_string(rand() % 9000 + 1000) << ' '\

<< surnames[rand() % 126] << ' ' << names[rand() % 65] << ' ' << patronymic[rand() % 36] << ' ' << date\

<< '.' << mon << '.' << std::to\_string(rand() % 82 + 1940) << '\n';

}

fout.close();

}

void writeData(std::vector<Passenger\*>data, std::string path, long long microseconds) {

std::ofstream fout;

fout.open(path);

fout << microseconds << '\n';

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

{

fout << data[i]->flight\_number.sym << data[i]->flight\_number.num << ' ';

fout << data[i]->fio.surname << ' ' << data[i]->fio.name << ' ' << data[i]->fio.patronymic << ' ';

std::string mon;

if (data[i]->date.month < 10) mon = '0' + std::to\_string(data[i]->date.month);

else mon = std::to\_string(data[i]->date.month);

std::string dat;

if (data[i]->date.number < 10) dat = '0' + std::to\_string(data[i]->date.number);

else dat = std::to\_string(data[i]->date.number);

fout << dat << '.' << mon << '.' << data[i]->date.year << '\n';

}

fout.close();

}

void StraightInsert(std::vector<Passenger\*>& data)

{

size\_t length = data.size();

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

{

Passenger\* temp = data[i];

size\_t j = i;

while (j > 0 && compare(data[j - 1], temp) == 1)

{

data[j] = data[j - 1];

j--;

}

data[j] = temp;

}

}

std::pair<int,int> partition(std::vector<Passenger\*>& data, int start, int end)

{

Passenger\* pivot = new Passenger;

Passenger\* temp = new Passenger;

int mid = (start + end) / 2;

if (compare(data[start], data[mid]) == 1) { temp = data[start]; data[start] = data[mid]; data[mid] = temp; }

if (compare(data[mid], data[end]) == 1) { temp = data[mid]; data[mid] = data[end]; data[end] = temp; }

if (compare(data[start], data[mid]) == 1) { temp = data[start]; data[start] = data[mid]; data[mid] = temp; }

pivot = data[mid];

int i = start;

int j = end;

do {

while (compare(data[i], pivot) == -1) i++;

while (compare(data[j], pivot) == 1) j--;

if (i <= j) {

temp = data[i];

data[i] = data[j];

data[j] = temp;

i++;

j--;

}

} while (i <= j);

return std::pair<int,int>(i,j);

}

void iterativeQuicksort(std::vector<Passenger\*>& data)

{

std::stack<std::pair<int, int>> s;

int start = 0;

int end = (int)data.size() - 1;

s.push(std::make\_pair(start, end));

while (!s.empty())

{

start = s.top().first, end = s.top().second;

s.pop();

std::pair <int, int> pivot = partition(data, start, end);

if (pivot.second > start) { s.push(std::make\_pair(start, pivot.second)); }

if (end > pivot.first) { s.push(std::make\_pair(pivot.first, end)); }

}

}

int main()

{

int n = 1000;

generateFile(n, "input.txt");

std::vector<Passenger\*>data = readData("input.txt");

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

//StraightInsert(data);

//iterativeQuicksort(data);

long long microseconds = std::chrono::duration\_cast<std::chrono::microseconds>(std::chrono::high\_resolution\_clock::now() - start).count();

writeData(data, "output.txt", microseconds);

}