**НЕГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ ЧАСТНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«МОСКОВСКИЙ ФИНАНСОВО-ПРОМЫШЛЕННЫЙ УНИВЕРСИТЕТ «СИНЕРГИЯ»**

|  |  |  |
| --- | --- | --- |
| **Факультет/Институт** |  | Информационных технологий |
|  |  | (наименование факультета/ Института) |
| **Направление/специальность** |  | Информационные системы и программирование |
| **подготовки:** |  | (код и наименование направления /специальности подготовки) |
| **Форма обучения:** |  | Очная |
|  |  | (очная, очно-заочная, заочная) |
| **Департамент Цифровой экономики** | | |

**Лабораторный практикум № *2***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **по дисциплине** |  | Разработка программных модулей | | |
|  | | |  | (наименование дисциплины) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Обучающийся** |  | ***Рубцов Тимур Евгеньевич*** |  |  |
|  |  | (ФИО) |  | (подпись) |
| **Группа** |  | ***203 ПРОГ*** |  |
|  |  |  |  |



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Преподаватель** |  | **Сибирев Иван Валерьевич** |  |  |

### Лабораторная работа №2. Алгоритмы поиска и сортировки в массивах Ответы на контрольные вопросы:

1. **Устойчив ли интерполяционный поиск на неравномерном объеме информации?**

Интерполяционный поиск **не устойчив** на неравномерном объеме информации. Этот алгоритм предполагает, что данные распределены достаточно равномерно, так как он рассчитывает позицию искомого элемента, используя линейную интерполяцию между минимальным и максимальным значением в массиве. При неравномерном распределении данных, где большая часть значений сосредоточена в одной области, интерполяционный поиск может сильно неэффективен, так как его "прыжки" могут быть слишком большими или слишком маленькими, приводя к большому числу лишних вычислений.

1. **Какая сложность у быстрой сортировки? За счет чего она достигается?**

**Сложность быстрой сортировки**:

* + **Средняя и лучшая** сложность: **O(n log n)**.
  + **Худшая** сложность: **O(n²)**, если всегда выбирается наихудший опорный элемент (например, первый или последний элемент в отсортированном массиве).

Быстрая сортировка достигает своей эффективности благодаря **разделению массива** на две части (разделение и завоевание). На каждом этапе она выбирает опорный элемент (например, медиану, первый или последний элемент), который делит массив на две части: элементы, меньшие или равные опорному элементу, и элементы, большие опорного. Эти две части затем сортируются рекурсивно. В среднем, благодаря удачному выбору опорного элемента и равномерному делению массива, она достигает логарифмической сложности по количеству уровней рекурсии и линейной сложности на каждом уровне для обработки элементов, что и приводит к **O(n log n)**.

2.2. Индивидуальные задания

**Пример программы:**

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

#include <algorithm>

using namespace std;

// Структура для хранения данных

struct Person {

int number; // Номер

string fullName; // ФИО

string address; // Домашний адрес

string registrationDate; // Дата учета

};

// Функция для ввода данных

void inputData(vector<Person>& persons) {

Person p;

cout << "Введите номер: ";

cin >> p.number;

cin.ignore(); // Очищаем буфер после ввода числа

cout << "Введите ФИО: ";

getline(cin, p.fullName);

cout << "Введите домашний адрес: ";

getline(cin, p.address);

cout << "Введите дату учета (в формате гггг-мм-дд): ";

getline(cin, p.registrationDate);

persons.push\_back(p);

}

// Функция для отображения списка

void displayData(const vector<Person>& persons) {

cout << "Номер | ФИО | Адрес | Дата учета" << endl;

for (const auto& person : persons) {

cout << person.number << " | " << person.fullName << " | " << person.address << " | " << person.registrationDate << endl;

}

}

// Линейный поиск по дате учета

int linearSearch(const vector<Person>& persons, const string& searchDate) {

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

if (persons[i].registrationDate == searchDate) {

return i; // Возвращаем индекс найденного элемента

}

}

return -1; // Если элемент не найден

}

// Метод сортировки прямым выбором

void selectionSort(vector<Person>& persons) {

for (size\_t i = 0; i < persons.size() - 1; ++i) {

size\_t minIdx = i;

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

if (persons[j].registrationDate < persons[minIdx].registrationDate) {

minIdx = j;

}

}

swap(persons[i], persons[minIdx]);

}

}

// Быстрая сортировка (QuickSort)

int partition(vector<Person>& persons, int low, int high) {

string pivot = persons[high].registrationDate;

int i = (low - 1);

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

if (persons[j].registrationDate <= pivot) {

++i;

swap(persons[i], persons[j]);

}

}

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

return (i + 1);

}

void quickSort(vector<Person>& persons, int low, int high) {

if (low < high) {

int pi = partition(persons, low, high);

quickSort(persons, low, pi - 1);

quickSort(persons, pi + 1, high);

}

}

// Функция для удаления повторяющихся записей по ФИО и адресу

void removeDuplicates(vector<Person>& persons) {

vector<Person> uniquePersons;

for (const auto& person : persons) {

bool exists = false;

for (const auto& uniquePerson : uniquePersons) {

if (uniquePerson.fullName == person.fullName && uniquePerson.address == person.address) {

exists = true;

break;

}

}

if (!exists) {

uniquePersons.push\_back(person);

}

}

persons = uniquePersons;

}

int main() {

vector<Person> persons;

int choice;

// Ввод данных

while (true) {

cout << "1. Добавить запись" << endl;

cout << "2. Показать все записи" << endl;

cout << "3. Линейный поиск по дате учета" << endl;

cout << "4. Сортировка методом прямого выбора" << endl;

cout << "5. Быстрая сортировка" << endl;

cout << "6. Удалить повторяющиеся записи" << endl;

cout << "7. Выйти" << endl;

cout << "Введите номер действия: ";

cin >> choice;

cin.ignore(); // Очищаем буфер после ввода числа

if (choice == 1) {

inputData(persons);

} else if (choice == 2) {

displayData(persons);

} else if (choice == 3) {

string searchDate;

cout << "Введите дату для поиска: ";

getline(cin, searchDate);

int index = linearSearch(persons, searchDate);

if (index != -1) {

cout << "Найдено: " << persons[index].fullName << " | " << persons[index].address << " | " << persons[index].registrationDate << endl;

} else {

cout << "Запись не найдена." << endl;

}

} else if (choice == 4) {

selectionSort(persons);

cout << "Записи отсортированы методом прямого выбора." << endl;

} else if (choice == 5) {

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

cout << "Записи отсортированы методом QuickSort." << endl;

} else if (choice == 6) {

removeDuplicates(persons);

cout << "Повторяющиеся записи удалены." << endl;

} else if (choice == 7) {

break;

} else {

cout << "Некорректный выбор." << endl;

}

}

return 0;

}