ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 5

«Название лабораторной работы (Методы сортировок)»

Выполнила работу

Смирнова Анна

Академическая группа №J3112

Принято

Преподаватель: Дунаев Максим Владимирович

Санкт-Петербург

2024

1. Введение

Методы сортировок

Цель: изучить методы сортировок с разной сложностью.

Задачи:

- Написать три алгоритма сортировок

- Изобразить работу алгоритмов на графиках

2. Теоретическая подготовка

- Знание языка программирования C++

- Знание алгоритмов сортировок

3. Реализация

Процесс этапа выполнения работы:

1. Постановка задачи:

Необходимо написать три алгоритма сортировок для решения задачи. Я использовала задачу, в которой нужно отсортировать массив чисел по возрастанию.

1. Анализ поставленной задачи:

Для решения поставленной задачи я использовала такие алгоритмы сортировок, как Bubble Sort, Quick Sort и Counting Sort.

1. Проектирование алгоритмов:

Bubble Sort

void bubbleSort(int arr[], int n) {

    for (int i = 0; i < n - 1; i++) {

        for (int j = 0; j < n - i - 1; j++) {

            if (arr[j] > arr[j + 1]) { // Если текущий элемент больше следующего

                // Обмен значениями

                int temp = arr[j];

                arr[j] = arr[j + 1];

                arr[j + 1] = temp;

            }

        }

    }

}

Quick Sort

// Функция для разделения массива

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

    int pivot = arr[high]; // Выбираем последний элемент в качестве опорного

    int i = low - 1; // Индекс меньшего элемента

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

        if (arr[j] <= pivot) { // Если текущий элемент меньше или равен опорному

            i++; // Увеличиваем индекс меньшего элемента

            swap(arr[i], arr[j]); // Меняем местами

        }

    }

    swap(arr[i + 1], arr[high]); // Меняем опорный элемент с элементом после последнего меньшего

    return i + 1; // Возвращаем индекс опорного элемента

}

// Функция быстрой сортировки

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

    if (low < high) {

        int pi = partition(arr, low, high); // Находим индекс опорного элемента

        // Рекурсивно сортируем элементы до и после разделения

        quickSort(arr, low, pi - 1);

        quickSort(arr, pi + 1, high);

    }

}

Counting Sort

void countingSort(vector<int>& arr) {

    // Находим максимальное значение в массиве

    int max = \*max\_element(arr.begin(), arr.end());

    // Создаем массив для подсчета частоты значений

    vector<int> count(max + 1, 0);

    // Подсчитываем частоту каждого элемента

    for (int num : arr) {

        count[num]++;

    }

    // Заполняем исходный массив отсортированными элементами

    int index = 0;

    for (int i = 0; i <= max; i++) {

        while (count[i] > 0) {

            arr[index++] = i; // Записываем значение в отсортированный массив

            count[i]--; // Уменьшаем счетчик

        }

    }

}

Для каждого алгоритма я использовала чтение из файла:

vector<int> read\_file(const string& filename) {

    vector<int> arr; // Вектор для хранения элементов массива

    ifstream inputFile(filename); // Открываем файл для чтения

    int number;

    while (inputFile >> number) { // Читаем числа из файла

        arr.push\_back(number); // Добавляем каждое число в вектор

    }

    return arr; // Возвращаем заполненный массив

}

4. Экспериментальная часть

Подсчет по памяти для 3-х алгоритмов сортировок:

1. Bubble Sort

Размер переменных: sizeof(count) + sizeof(size) = 4 + 4 = 8 байт

Размер массива: N \* sizeof(int) = N \* 4 байта

1. Quick Sort

Размер переменных: sizeof(number) + sizeof(pivot) + sizeof(i) + sizeof(j) = 4 + 4 + 4 + 4 = 16 байт

Размер массивов: N \* sizeof(arr) + N \* sizeof(array)

1. Counting Sort

Размер переменных: sizeof(number) + sizeof(max) + sizeof(index) = 4 + 4 + 4 = 12 байт

Размер массивов: N \* sizeof(max) + N \* sizeof(arr) + N \* sizeof(arr)

Подсчет асимптотики для 3-х алгоритмов сортировок:

1. Bubble Sort

Чтение данных из файла: O(n)

Сортировка: O(n²)

1. Quick Sort

Чтение данных из файла: O(n)

Сортировка средний и лучший случай: O (n \* log n)

Сортировка худший случай: O(n²)

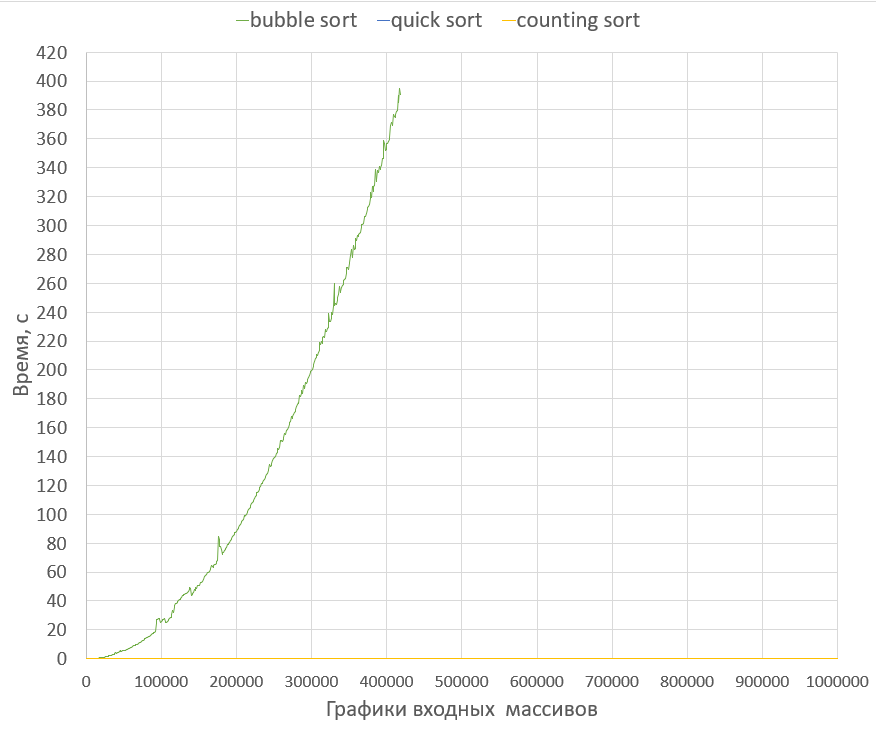
1. Counting Sort

Чтение данных из файла: O(n)

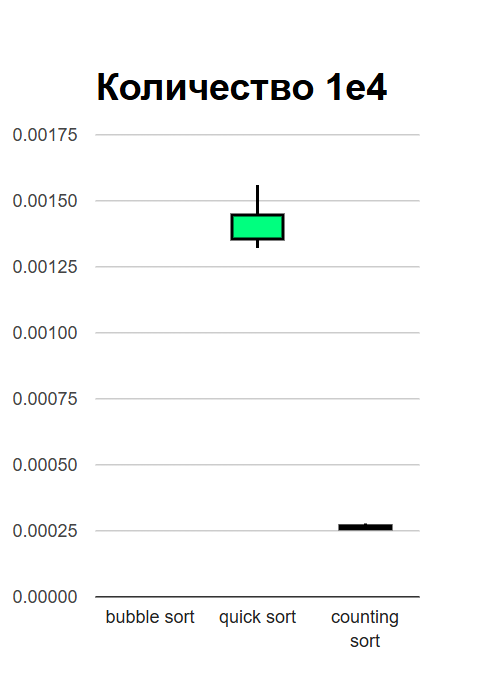
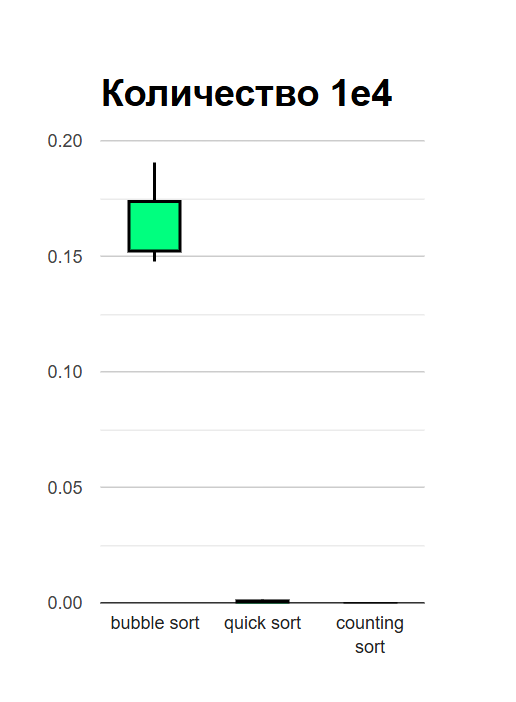
Поиск max: O(k)

Подсчет частоты: O(n)

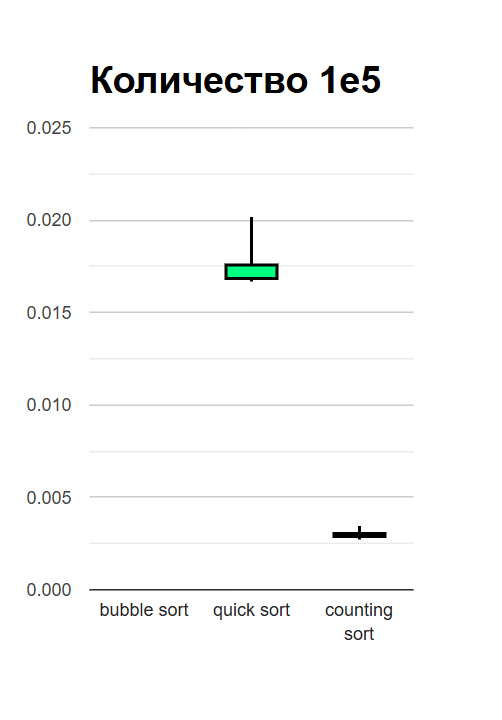
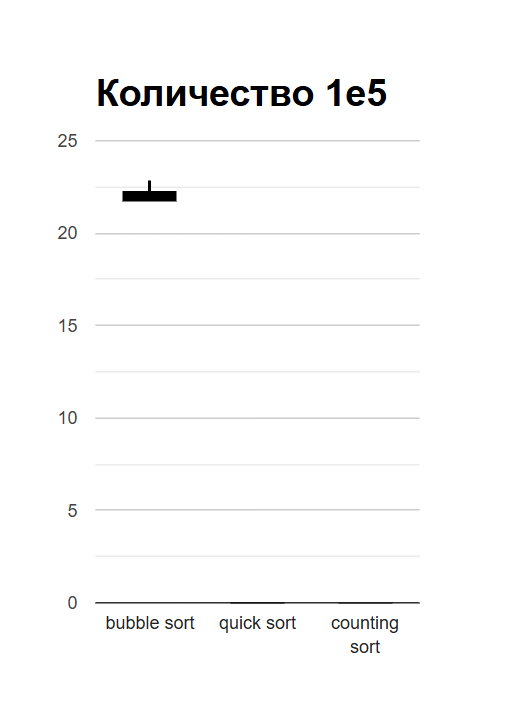
Общее: O(n + k)



Линейный график работы алгоритмов для массивов от 1000 до 1e6 с шагом в 1000



Box plot графики для времени работы алгоритмов с числом элементов 1e4. Количество запусков не меньше 50 для одного графика.



Box plot графики для времени работы алгоритмов с числом элементов 1e5. Количество запусков не меньше 50 для одного графика.

Анализ графика и таблицы:

По полученным значениям в результате изменения размера входного массива видно, что время выполнения работы алгоритма Bubble Sort значительно превышает время работы двух других алгоритмов, поэтому получилось реализовать его график только количества элементов чуть больше 400 тыс. На box plot графиках видно, как варьируется время выполнения алгоритмов при неизменном входном массиве: Counting Sort остаётся почти неизменна, в отличии от других сортировок.

5. Заключение

В ходе выполнения работы мною был реализованы 3 алгоритма сортировок. На графиках наглядно видно, как работают сортировки для большого количества входных данных, что позволяет понять – в каких случаях стоит использовать каждый алгоритм.

Выводы: Асимптотика позволяет понять, как алгоритм будет вести себя при больших объемах данных. Алгоритм со сложностью O(n) будет более эффективен, чем O(n²) при больших n.

На выбросы влияет производительность компьютера, скорость чтения данных из памяти, кэширование, значения данных, которые сильно отличаются от других, структура данных.

Выбор алгоритма сортировки зависит от конкретной задачи и характеристик входных данных. Для небольших массивов можно использовать Bubble Sort, для больших — Quick Sort, а Counting Sort.

ПРИЛОЖЕНИЕ А

Листинг кода файла Bubble\_Sort.cpp

#include <iostream>

#include <fstream>

using namespace std;

// Функция для сортировки массива методом пузырька

void bubbleSort(int arr[], int n) {

    for (int i = 0; i < n - 1; i++) {

        for (int j = 0; j < n - i - 1; j++) {

            if (arr[j] > arr[j + 1]) { // Если текущий элемент больше следующего

                // Обмен значениями

                int temp = arr[j];

                arr[j] = arr[j + 1];

                arr[j + 1] = temp;

            }

        }

    }

}

int main() {

    const int size = 100000; // Максимальный размер массива

    int array[size]; // Статический массив

    int count = 0; // Счетчик элементов

    ifstream inputFile("datasets/dataset\_100000.txt"); // Открываем файл для чтения

    // Чтение чисел из файла

    while (count < size && inputFile >> array[count]) { // Считываем числа в массив

        count++;

    }

    bubbleSort(array, count); // Сортируем массив методом пузырька

    cout << "Отсортированный массив: ";

    for (int i = 0; i < count; i++) {

        cout << array[i] << " ";

    }

    cout << endl;

    return 0;

}

Листинг кода файла Quick\_Sort.cpp

#include <iostream>

#include <fstream>

#include <vector>

using namespace std;

vector<int> read\_file(const string& filename) {

    vector<int> arr; // Вектор для хранения элементов массива

    ifstream inputFile(filename); // Открываем файл для чтения

    int number;

    while (inputFile >> number) { // Читаем числа из файла

        arr.push\_back(number); // Добавляем каждое число в вектор

    }

    return arr; // Возвращаем заполненный массив

}

// Функция для разделения массива

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

    int pivot = arr[high]; // Выбираем последний элемент в качестве опорного

    int i = low - 1; // Индекс меньшего элемента

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

        if (arr[j] <= pivot) { // Если текущий элемент меньше или равен опорному

            i++; // Увеличиваем индекс меньшего элемента

            swap(arr[i], arr[j]); // Меняем местами

        }

    }

    swap(arr[i + 1], arr[high]); // Меняем опорный элемент с элементом после последнего меньшего

    return i + 1; // Возвращаем индекс опорного элемента

}

// Функция быстрой сортировки

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

    if (low < high) {

        int pi = partition(arr, low, high); // Находим индекс опорного элемента

        // Рекурсивно сортируем элементы до и после разделения

        quickSort(arr, low, pi - 1);

        quickSort(arr, pi + 1, high);

    }

}

int main() {

    vector<int> array = read\_file("datasets/dataset\_1000.txt"); // Пример массива для сортировки

    int n = array.size();

    quickSort(array, 0, n - 1); // Сортируем массив

    cout << "Отсортированный массив: ";

    for (int num : array) {

        cout << num << " ";

    }

    cout << endl;

    return 0;

}

Листинг кода файла Counting\_Sort.cpp

#include <iostream>

#include <fstream>

#include <vector>

#include <algorithm> // Для max\_element

using namespace std;

vector<int> read\_file(const string& filename) {

    vector<int> arr; // Вектор для хранения элементов массива

    ifstream inputFile(filename); // Открываем файл для чтения

    int number;

    while (inputFile >> number) { // Читаем числа из файла

        arr.push\_back(number); // Добавляем каждое число в вектор

    }

    return arr; // Возвращаем заполненный массив

}

void countingSort(vector<int>& arr) {

    // Находим максимальное значение в массиве

    int max = \*max\_element(arr.begin(), arr.end());

    // Создаем массив для подсчета частоты значений

    vector<int> count(max + 1, 0);

    // Подсчитываем частоту каждого элемента

    for (int num : arr) {

        count[num]++;

    }

    // Заполняем исходный массив отсортированными элементами

    int index = 0;

    for (int i = 0; i <= max; i++) {

        while (count[i] > 0) {

            arr[index++] = i; // Записываем значение в отсортированный массив

            count[i]--; // Уменьшаем счетчик

        }

    }

}

int main() {

    vector<int> arr = read\_file("datasets/dataset\_1000.txt");

    countingSort(arr); // Сортируем массив

    cout << "Отсортированный массив: ";

    for (int num : arr) {

        cout << num << " ";

    }

    cout << endl;

    return 0;

}

ПРИЛОЖЕНИЕ Б

Время выполнения Bubble Sort для количества элементов с 1000 до 418000. (частично разделены)  




Также для эффективного получения результатов я написала алгоритм, который записывал результаты выполнения сортировок в txt файл, для дальнейшего построения графиков.

Пример на Counting Sort:

int main() {

    ofstream outputFile("quick\_sort\_results.txt"); // Открываем файл для записи результатов

    for (int size = 1000; size <= 1000000; size += 1000) { // Изменяем размер массива от 1000 до 1,000,000 с шагом 1000

        string filename = "datasets/dataset\_" + to\_string(size) + ".txt"; // Формируем имя файла

        vector<int> array = read\_file(filename); // Пример массива для сортировки

        int n = array.size();

        auto start = chrono::high\_resolution\_clock::now(); // Запоминаем время начала сортировки

        quickSort(array, 0, n - 1); // Сортируем массив

        // Конец замера времени

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

        chrono::duration<double> duration = end - start; // Вычисляем продолжительность

        // Записываем результат в файл

        outputFile << size << duration.count() << endl;

    }

    outputFile.close(); // Закрываем файл после записи

    cout << "Сортировка завершена." << endl;

    return 0;

}

Также с помощью Chat GPT был написан алгоритм для генерации датасетов для различных числовых значений:

import random

def generate\_datasets(base\_filename, num\_datasets, step):

    for i in range(num\_datasets):

        count = (i + 1) \* step  # Количество чисел в текущем датасете

        filename = f"{base\_filename}\_{count}.txt"  # Имя файла с количеством чисел

        # Генерация случайных чисел от 1 до count

        numbers = [str(random.randint(1, count)) for \_ in range(count)]

        with open(filename, 'w') as f:

            f.write('\t'.join(numbers))  # Запись чисел с разделителем табуляция

        print(f"Сгенерирован датасет: {filename}")

# Параметры генерации

num\_datasets = 1000    # Количество датасетов

step = 1000            # Шаг (количество элементов в каждом датасете)

# Генерация датасетов

generate\_datasets('dataset', num\_datasets, step)