ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 5

«Алгоритмы cортировки»

Выполнил работу

Ковров Евгений

Академическая группа №J3110

Принято

Должность, звание Вершинин Владислав

Санкт-Петербург

2024

1. Введение

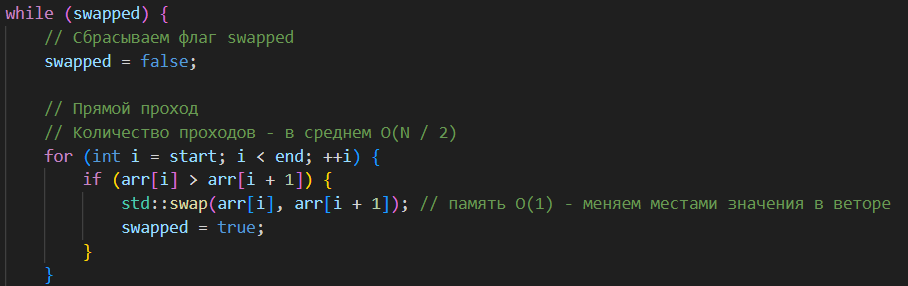
Цель работы: создать программные решения, реализующие 3 различных алгоритма сортировки.

Задачи:

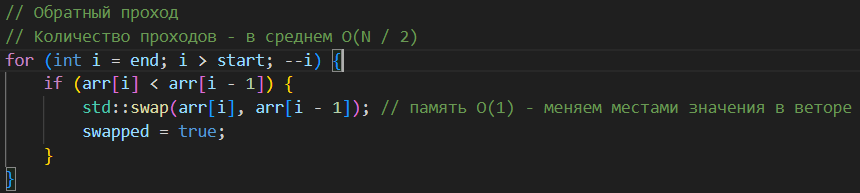
1. Найти алгоритмы сортировки, удовлетворяющие заданным условиям
2. Реализовать алгоритмы на языке C++
3. Протестировать код
4. Подготовить отчёт

Реализация

В качестве алгоритма с временной сложностью до О(N ^ 2) и пространственной сложностью О(1) была выбрана cocktail shake sort. Она аналогична bubble sort, но на каждой итерации делает по 2 прохода – слева направо и справа налево.

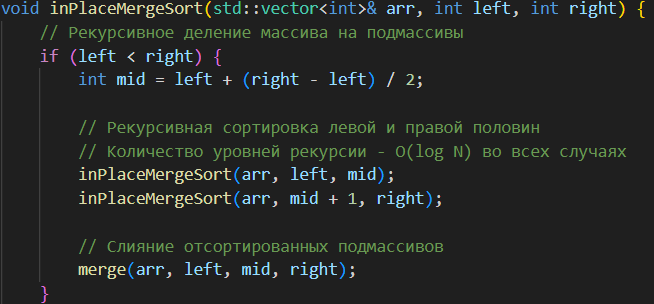


Изображение №1 – Прямой проход

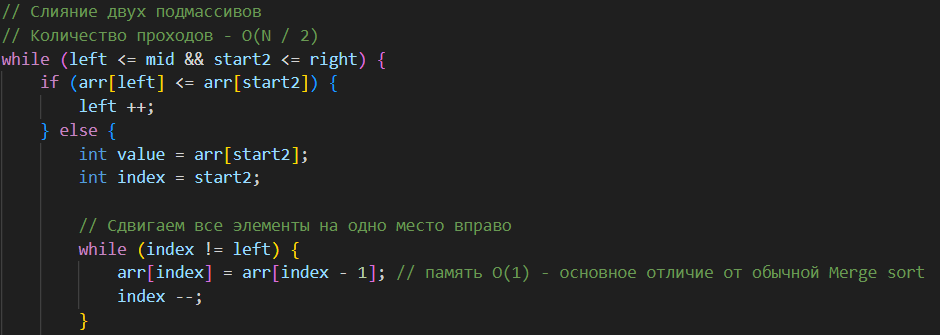


Изображение №2 – Прямой проход

В качестве алгоритма с временной сложностью до О(N \* log N) и пространственной сложностью О(1) была выбрана in-place merge sort. Она является модернизированной версией merge sort, в отличие от которой не использует дополнительную память для хранения временных подмассивов, а производит слияние на месте.

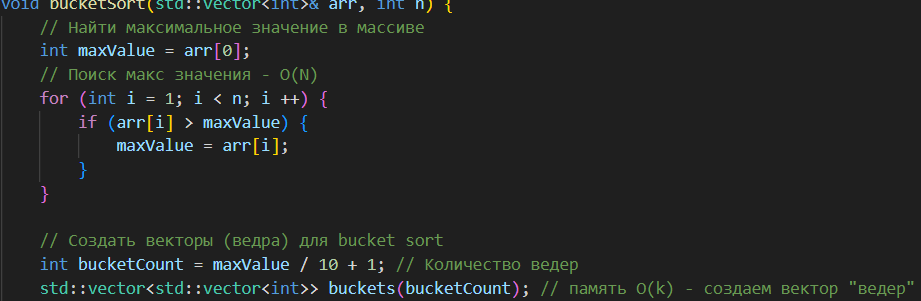


Изображение №3 – Рекурсивная функция сортировки



Изображение №4 – Функция слияния подмассивов

В качестве алгоритма с временной сложностью до О(N \* k) и пространственной сложностью О(N \* k) была выбрана bucket sort. Она делит массив на k упорядоченных «вёдер», затем сортирует каждое из них и в заключение соединяет в отсортированный массив.



Изображение №5 – Поиск максимального значения и вычисления количества «вёдер» на его основе



Изображение №6 – Деление на «вёдра», сортировка и слияние в отсортированный массив

1. Экспериментальная часть

Для проведения экспериментов была создана программа, псевдослучайно генерирующая массив заданного размера.

На первом этапе тестирования было замерено время сортировки псевдослучайных массивов размером от 1000 до 10^6 элементов всеми тремя алгоритмами. На основе полученных данных был построен график.

Изображение №7 – График зависимости времени сортировки от размера массива

Затем было измерено время сортировки всеми алгоритмами 50 различных псевдослучайных массивов размеров 10^4 и 10^5 элементов. На основе собранных данных были построены графики типа box plot.

Изображение №8 – График для 10^4 Изображение №9 – График для 10^5

1. Заключение

В ходе выполнения работы мною были реализованы 3 алгоритма сортировки. Для каждого из них были соблюдены требования по асимптотике и памяти. Были собраны экспериментальные данные, на основе которых построены графики. В целом, они достаточно точно описывают теоретическую асимптотику. Некоторые выбросы могут быть вызваны «хорошими» и «плохими» случаями в исходном массиве.

Cocktail shake sort лучше всего подходит для случаев экстремальных ограничений по памяти и небольших массивов. In-place merge sort немного более требовательна к памяти за счёт большего количества вспомогательных переменных, однако показывает себя сильно лучше на более больших массивах. Наконец, bucket sort – самая быстрая, но весьма требовательная к памяти.

Приложения

ПРИЛОЖЕНИЕ А

Листинг кода файла cocktail\_shake\_sort.cpp

#include <iostream>

#include <vector>

#include <fstream>

#include <sstream>

#include <chrono>

void read\_csv(const std::string& filename, std::vector<int>& intArray) { // Функция чтения чисел из csv в массив

    std::ifstream file(filename);

    if (!file.is\_open()) {

        std::cerr << "Error opening file: " << filename << std::endl;

        return;

    }

    std::string line;

    while (std::getline(file, line)) {

        std::stringstream ss(line);

        std::string value;

        while (std::getline(ss, value, ',')) { // Split by comma

            try {

                int number = std::stoi(value); // Convert to int

                intArray.push\_back(number); // Add to the vector

            } catch (const std::invalid\_argument& e) {

                std::cerr << "Invalid number: " << value << std::endl;

            } catch (const std::out\_of\_range& e) {

                std::cerr << "Number out of range: " << value << std::endl;

            }

        }

    }

    file.close();

}

void cocktailShakerSort(std::vector<int>& arr) {

    bool swapped = true;

    int start = 0;

    int end = arr.size() - 1;

    // Количество проходов:

    // Лучший случай - О(1)

    // Худший и средий случай - О(N)

    while (swapped) {

        // Сбрасываем флаг swapped

        swapped = false;

        // Прямой проход

        // Количество проходов - в среднем О(N / 2)

        for (int i = start; i < end; ++i) {

            if (arr[i] > arr[i + 1]) {

                std::swap(arr[i], arr[i + 1]); // память O(1) - меняем местами значения в веторе

                swapped = true;

            }

        }

        // Если ничего не было обменяно, массив уже отсортирован

        if (!swapped) {

            break;

        }

        // Уменьшаем конец для следующей итерации

        --end;

        // Сбрасываем флаг swapped для обратного прохода

        swapped = false;

        // Обратный проход

        // Количество проходов - в среднем О(N / 2)

        for (int i = end; i > start; --i) {

            if (arr[i] < arr[i - 1]) {

                std::swap(arr[i], arr[i - 1]); // память O(1) - меняем местами значения в веторе

                swapped = true;

            }

        }

        // Увеличиваем начало для следующей итерации

        ++start;

    }

    // Итого:

    // Временная сложность:

    // Лучший случай - О(N), средний и худший - О(N ^ 2)

    // Память:

    // Во всех случаях - О(1), т.к. только меняем местами 2 значения

}

// Функция для вывода массива

void printArray(const std::vector<int>& arr) {

    for (int num : arr) {

        std::cout << num << " ";

    }

    std::cout << std::endl;

}

int main() {

    // for (int i = 1000; i <= 1000000; i \*= 10) {

    //     std::vector<int> arr;

    //     std::string tmp\_file = std::to\_string(i) + ".csv";

    //     read\_csv(tmp\_file, arr);

    //     auto start = std::chrono::high\_resolution\_clock::now(); // Фиксируем время старта

    //     cocktailShakerSort(arr);

    //     auto end = std::chrono::high\_resolution\_clock::now(); // Фиксируем время окончания

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

    //     std::cout << "Execution time for " << i << " : " << duration.count() << " seconds" << std::endl; // Выводим время работы

    // }

    //  for (int i = 1000; i <= 1000000; i \*= 10) {

    //     for (int j = 1; j <= 9; j ++) {

    //         std::vector<int> arr;

    //         std::string tmp\_file = std::to\_string(i \* j) + ".csv";

    //         read\_csv(tmp\_file, arr);

    //         auto start = std::chrono::high\_resolution\_clock::now(); // Фиксируем время старта

    //         cocktailShakerSort(arr);

    //         auto end = std::chrono::high\_resolution\_clock::now(); // Фиксируем время окончания

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

    //         std::cout << i \* j << " " << duration.count() << std::endl; // Выводим время работы

    //     }

    // }

    for (int i = 1; i <= 50; i ++) {

        std::vector<int> arr;

        std::string tmp\_file = "100000\_" + std::to\_string(i) + ".csv";

        read\_csv(tmp\_file, arr);

        int n = arr.size();

        auto start = std::chrono::high\_resolution\_clock::now(); // Фиксируем время старта

        cocktailShakerSort(arr);

        auto end = std::chrono::high\_resolution\_clock::now(); // Фиксируем время окончания

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

        std::cout << duration.count() << std::endl; // Выводим время работы

    }

    return 0;

}

ПРИЛОЖЕНИЕ B

Листинг кода файла in-place\_merge\_sort.cpp

#include <iostream>

#include <vector>

#include <fstream>

#include <sstream>

#include <chrono>

void read\_csv(const std::string& filename, std::vector<int>& intArray) { // Функция чтения чисел из csv в массив

    std::ifstream file(filename);

    if (!file.is\_open()) {

        std::cerr << "Error opening file: " << filename << std::endl;

        return;

    }

    std::string line;

    while (std::getline(file, line)) {

        std::stringstream ss(line);

        std::string value;

        while (std::getline(ss, value, ',')) { // Split by comma

            try {

                int number = std::stoi(value); // Convert to int

                intArray.push\_back(number); // Add to the vector

            } catch (const std::invalid\_argument& e) {

                std::cerr << "Invalid number: " << value << std::endl;

            } catch (const std::out\_of\_range& e) {

                std::cerr << "Number out of range: " << value << std::endl;

            }

        }

    }

    file.close();

}

void merge(std::vector<int>& arr, int left, int mid, int right) {

    int start2 = mid + 1;

    // Если первый элемент меньше или равен последнему элементу второго подмассива, ничего не делаем

    if (arr[mid] <= arr[start2]) {

        return;

    }

    // Слияние двух подмассивов

    // Количество проходов - О(N / 2)

    while (left <= mid && start2 <= right) {

        if (arr[left] <= arr[start2]) {

            left ++;

        } else {

            int value = arr[start2];

            int index = start2;

            // Сдвигаем все элементы на одно место вправо

            while (index != left) {

                arr[index] = arr[index - 1]; // память O(1) - основное отличие от обычной Merge sort

                index --;

            }

            arr[left] = value;

            // Обновляем индексы

            left ++;

            mid ++;

            start2 ++;

        }

    }

}

void inPlaceMergeSort(std::vector<int>& arr, int left, int right) {

    // Рекурсивное деление массива на подмассивы

    if (left < right) {

        int mid = left + (right - left) / 2;

        // Рекурсивная сортировка левой и правой половин

        // Количество уровней рекурсии - О(log N) во всех случаях

        inPlaceMergeSort(arr, left, mid);

        inPlaceMergeSort(arr, mid + 1, right);

        // Слияние отсортированных подмассивов

        merge(arr, left, mid, right);

    }

    // Итого:

    // Временная сложность:

    // Лучший случай - О(log N), средний и худший - О(N \* log N)

    // Память:

    // Во всех случаях - О(1), т.к. только сдвигаем все значения в массиве по очереди

}

int main() {

    // for (int i = 1000; i <= 1000000; i \*= 10) {

    //     std::vector<int> arr;

    //     std::string tmp\_file = std::to\_string(i) + ".csv";

    //     read\_csv(tmp\_file, arr);

    //     int n = arr.size();

    //     auto start = std::chrono::high\_resolution\_clock::now(); // Фиксируем время старта

    //     inPlaceMergeSort(arr, 0, n - 1);

    //     auto end = std::chrono::high\_resolution\_clock::now(); // Фиксируем время окончания

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

    //     std::cout << "Execution time for " << i << " : " << duration.count() << " seconds" << std::endl; // Выводим время работы

    // }

//    for (int i = 1000; i <= 1000000; i \*= 10) {

//         for (int j = 1; j <= 9; j ++) {

//             std::vector<int> arr;

//             std::string tmp\_file = std::to\_string(i \* j) + ".csv";

//             read\_csv(tmp\_file, arr);

//             int n = arr.size();

//             auto start = std::chrono::high\_resolution\_clock::now(); // Фиксируем время старта

//             inPlaceMergeSort(arr, 0, n - 1);

//             auto end = std::chrono::high\_resolution\_clock::now(); // Фиксируем время окончания

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

//             std::cout << i \* j << " " << duration.count() << std::endl; // Выводим время работы

//         }

//     }

    for (int i = 1; i <= 50; i ++) {

        std::vector<int> arr;

        std::string tmp\_file = "100000\_" + std::to\_string(i) + ".csv";

        read\_csv(tmp\_file, arr);

        int n = arr.size();

        auto start = std::chrono::high\_resolution\_clock::now(); // Фиксируем время старта

        inPlaceMergeSort(arr, 0, n - 1);

        auto end = std::chrono::high\_resolution\_clock::now(); // Фиксируем время окончания

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

        std::cout << duration.count() << std::endl; // Выводим время работы

    }

    return 0;

}

ПРИЛОЖЕНИЕ C

Листинг кода файла bucket\_sort.cpp

#include <iostream>

#include <vector>

#include <algorithm> // Для std::sort

#include <fstream>

#include <sstream>

#include <chrono>

void read\_csv(const std::string& filename, std::vector<int>& intArray) { // Функция чтения чисел из csv в массив

 std::ifstream file(filename);

    if (!file.is\_open()) {

        std::cerr << "Error opening file: " << filename << std::endl;

        return;

    }

    std::string line;

    while (std::getline(file, line)) {

        std::stringstream ss(line);

        std::string value;

        while (std::getline(ss, value, ',')) { // Split by comma

            try {

                int number = std::stoi(value); // Convert to int

                intArray.push\_back(number); // Add to the vector

            } catch (const std::invalid\_argument& e) {

                std::cerr << "Invalid number: " << value << std::endl;

            } catch (const std::out\_of\_range& e) {

                std::cerr << "Number out of range: " << value << std::endl;

            }

        }

    }

    file.close();

}

void bucketSort(std::vector<int>& arr, int n) {

    // Найти максимальное значение в массиве

    int maxValue = arr[0];

    // Поиск макс значения - O(N)

    for (int i = 1; i < n; i ++) {

        if (arr[i] > maxValue) {

            maxValue = arr[i];

        }

    }

    // Создать векторы (ведра) для bucket sort

    int bucketCount = maxValue / 10 + 1; // Количество ведер

    std::vector<std::vector<int>> buckets(bucketCount); // память O(k) - создаем вектор "ведер"

    // Разделить массив на ведра

    // Проходим по массиву - О(N)

    for (int i = 0; i < n; i ++) {

        int bucketIndex = arr[i] / 10; // Индекс ведра

        buckets[bucketIndex].push\_back(arr[i]); // память О(N \* k)

    }

    // Отсортировать каждое ведро и собрать результат

    int index = 0;

    // Проход по всем вёдрам - О(k)

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

        // Сложность std::sort

        // Время - (N / k) \* log (N / k) во всех случаях

        // Память - log (N / k) во всех случаях

        std::sort(buckets[i].begin(), buckets[i].end()); // Сортируем ведро

        // Проход по всем элементам ведра - О(N / k)

        for (int j = 0; j < buckets[i].size(); j ++) {

            arr[index++] = buckets[i][j]; // Собрать отсортированные элементы

        }

    }

    // Итого:

    // Временная сложность:

    // Лучший случай - О(N), средний и худший - О(N \* k)

    // Память:

    // Во всех случаях - О(N \* k)

}

int main() {

    // for (int i = 1000; i <= 1000000; i \*= 10) {

    //     std::vector<int> arr;

    //     std::string tmp\_file = std::to\_string(i) + ".csv";

    //     read\_csv(tmp\_file, arr);

    //     int n = arr.size();

    //     auto start = std::chrono::high\_resolution\_clock::now(); // Фиксируем время старта

    //     bucketSort(arr, n);

    //     auto end = std::chrono::high\_resolution\_clock::now(); // Фиксируем время окончания

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

    //     std::cout << "Execution time for " << i << " : " << duration.count() << " seconds" << std::endl; // Выводим время работы

    // }

    // for (int i = 1000; i <= 1000000; i \*= 10) {

    //     for (int j = 1; j <= 9; j ++) {

    //         std::vector<int> arr;

    //         std::string tmp\_file = std::to\_string(i \* j) + ".csv";

    //         read\_csv(tmp\_file, arr);

    //         int n = arr.size();

    //         auto start = std::chrono::high\_resolution\_clock::now(); // Фиксируем время старта

    //         bucketSort(arr, n);

    //         auto end = std::chrono::high\_resolution\_clock::now(); // Фиксируем время окончания

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

    //         std::cout << i \* j << " " << duration.count() << std::endl; // Выводим время работы

    //     }

    // }

    for (int i = 1; i <= 50; i ++) {

        std::vector<int> arr;

        std::string tmp\_file = "100000\_" + std::to\_string(i) + ".csv";

        read\_csv(tmp\_file, arr);

        int n = arr.size();

        auto start = std::chrono::high\_resolution\_clock::now(); // Фиксируем время старта

        bucketSort(arr, n);

        auto end = std::chrono::high\_resolution\_clock::now(); // Фиксируем время окончания

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

        std::cout << duration.count() << std::endl; // Выводим время работы

    }

    return 0;

}

ПРИЛОЖЕНИЕ D

Листинг кода файла array\_generator.cpp

#include <iostream>

#include <vector>

#include <algorithm>

#include <fstream>

#include <random>

int main() {

    int n = 100000;

    // Запрашиваем количество натуральных чисел

    // std::cout << "Enter array len: ";

    // std::cin >> n;

    // Генерируем массив из натуральных чисел

    std::vector<int> numbers(n);

    for (int i = 0; i < n; ++i) {

        numbers[i] = i + 1;  // Заполняем числами от 1 до n

    }

    for (int i = 1; i <= 50; i ++) {

        // Перемешиваем массив с использованием random\_device и mt19937

        std::random\_device rd;

        std::mt19937 g(rd());

        std::shuffle(numbers.begin(), numbers.end(), g);

        // Сохраняем массив в CSV файл

        std::string filename = std::to\_string(n) + "\_" + std::to\_string(i) + ".csv";

        // std::cout << "Enter file name: ";

        // std::cin >> filename;

        std::ofstream outFile(filename);

        if (!outFile) {

            std::cerr << "Ошибка при открытии файла для записи!" << std::endl;

            return 1;

        }

        // Записываем числа в CSV файл

        for (size\_t i = 0; i < numbers.size(); ++i) {

            outFile << numbers[i];

            if (i < numbers.size() - 1) {

                outFile << ",";  // Добавляем запятую между числами

            }

        }

        outFile.close();  // Закрываем файл

        std::cout << "Array has been saved to " << filename << std::endl;

    }

    return 0;

}