**ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

**Кафедра безопасности информационных систем**

**ОТЧЁТ**

по практической работе № 4 на тему:   
**«Сортировка числовых массивов. Некоторые методы сортировки»**

по дисциплине «Алгоритмы и структуры данных»

Выполнил: студент группы ИСТ-312, Кандиков М.В.

«24» Сентября 2024 г. \_\_\_\_\_\_\_\_\_\_\_/М.В. Кандиков /

Принял: к.ф.-м.н., доцент, И.А. Моисеев

«24» Сентября 2024 г. \_\_\_\_\_\_\_\_\_\_\_/ И.А. Моисеев /

**Содержание**

**1. Титульный лист 1**

**2. Содержание 2**

**3. Основная часть 3**

3.1. Цель работы 3

3.2. Теоретическая часть 3

3.3. Практическая часть 4

3.4. Результаты выполнения программы 6

3.5. Выводы 7

**4. Приложение 8**

**Основная часть**

**Цель работы:**

Изучить и реализовать различные методы сортировки массивов, такие как сортировка выбором, сортировка вставками и сортировка обменом. Оценить их производительность по числу операций сравнения, пересылки элементов и времени выполнения.

**Теоретическая часть:**

Сортировка данных — одна из ключевых задач при работе с большими объёмами информации. Основная цель сортировки — упорядочить данные для более быстрого поиска и обработки. Наиболее популярные методы сортировки массивов включают сортировку выбором, сортировку вставками и сортировку обменом.

Описание методов сортировки:

1. **Сортировка выбором (Selection Sort):**
   * Алгоритм просматривает весь массив, находит наименьший элемент и меняет его местами с элементом на первой позиции. Этот процесс повторяется для оставшихся элементов массива.
   * Время работы: O(n²), где n — количество элементов массива.
2. **Сортировка вставками (Insertion Sort):**
   * Алгоритм проходит по массиву, вставляя каждый элемент на правильное место среди уже отсортированных элементов.
   * Время работы: O(n²) в худшем случае, O(n) — в лучшем случае (для почти отсортированных массивов).
3. **Сортировка обменом (Bubble Sort):**
   * Алгоритм многократно проходит по массиву, сравнивая соседние элементы и меняя их местами, если они находятся в неправильном порядке.
   * Время работы: O(n²).

**Практическая часть:**

Для реализации задач использовались стандартные средства языка программирования C++ и библиотека <chrono> для замера времени выполнения. Данные считывались из файла numbers.txt, содержащего 30 случайных целых чисел.

Были реализованы три метода сортировки:

* Сортировка выбором;
* Сортировка вставками;
* Сортировка обменом.

Основные фрагменты кода для реализации методов сортировки приведены ниже:

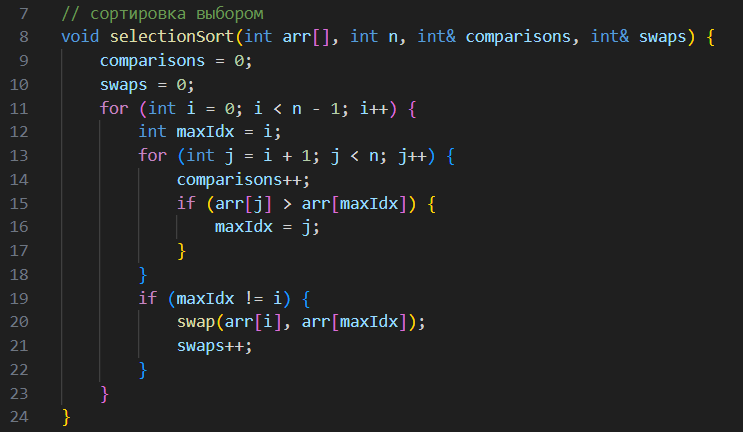


Рисунок 1. Сортировка выбором.

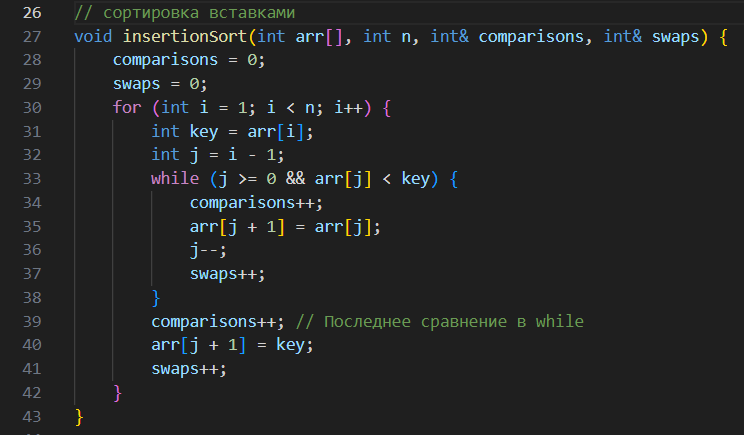


Рисунок 2. Сортировка вставками.

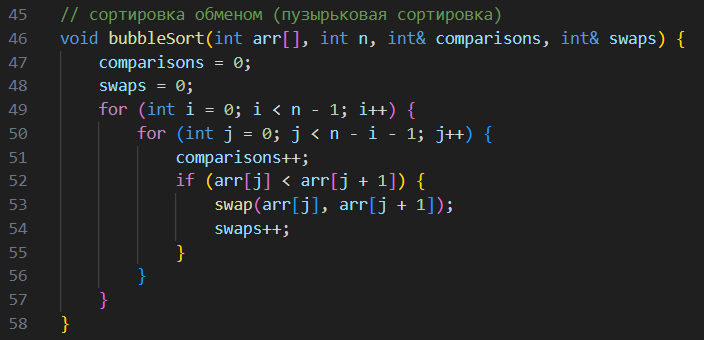


Рисунок 3. Сортировка обменом.

**Результаты выполнения программы:**

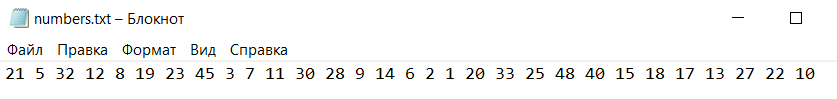


Рисунок 4. Файл numbers.txt, в котором находится массив с числами, который нужно отсортировать.

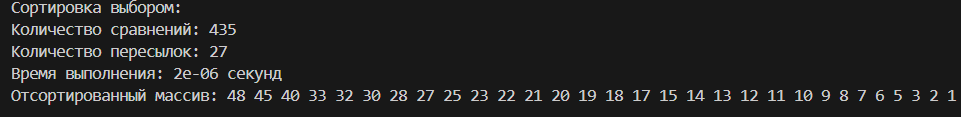
Для массива чисел из файла numbers.txt, получились следующие результаты: 

Рисунок 4. Результат для сортировки выбором.

1. **Сортировка выбором (Selection Sort):**

* **Количество сравнений**: 435 — максимальное среди всех алгоритмов, что ожидаемо, так как алгоритм сравнивает каждый элемент с каждым другим.
* **Количество пересылок**: 27 — минимальное, так как пересылка происходит только при необходимости изменения положения минимального элемента.
* **Время выполнения**: 2.9e-06 секунд — быстрее, чем сортировка обменом, но медленнее сортировки вставками.

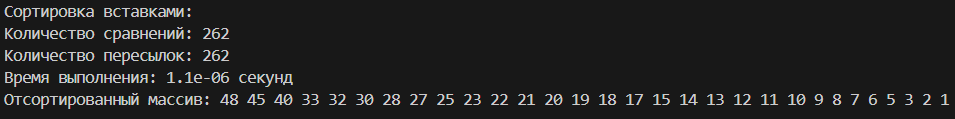


Рисунок 5. Результат для сортировки вставками.

1. **Сортировка вставками (Insertion Sort):**

* **Количество сравнений**: 262 — меньше, чем у других алгоритмов, так как сравнение происходит только до того момента, пока элемент не вставится в нужное место.
* **Количество пересылок**: 262 — равное количеству сравнений, так как каждый элемент перемещается в процессе вставки.
* **Время выполнения**: 1.2e-06 секунд — самое быстрое, так как в среднем этот алгоритм хорошо работает на почти отсортированных данных.

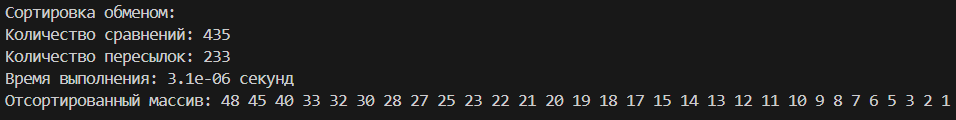


Рисунок 6. Результат для сортировки обменом.

1. **Сортировка обменом (Bubble Sort):**

* **Количество сравнений**: 435 — сравнимо с сортировкой выбором, так как каждый элемент также сравнивается со всеми соседями.
* **Количество пересылок**: 233 — существенно больше, чем у сортировки выбором, так как пересылки происходят на каждом шаге при каждом обмене элементов.
* **Время выполнения**: 3.5e-06 секунд — самое медленное, что также соответствует ожидаемому поведению для алгоритмов обмена.

**Выводы:**

По результатам выполненной практической работы можно сделать следующие выводы: сортировка вставками показала наилучшую производительность среди трёх алгоритмов как по количеству сравнений, так и по времени выполнения. Этот алгоритм хорошо подходит для массивов небольшого размера или частично отсортированных данных.

Сортировка выбором выполняет минимальное количество пересылок, однако требует наибольшего количества сравнений, что делает её менее эффективной по времени выполнения по сравнению с сортировкой вставками.

Сортировка обменом оказалась самой медленной по времени выполнения и требует большого количества пересылок, что делает её менее предпочтительной для сортировки массивов, особенно больших.

Таким образом, для сортировки небольших массивов, особенно частично отсортированных, наилучшим выбором является сортировка вставками.

Для небольших массивов с произвольным распределением данных **сортировка вставками** является наилучшим выбором по времени и количеству операций. **Сортировка выбором** эффективна в ситуациях, где важно минимизировать количество пересылок. **Сортировка обменом**, хотя и проста, показывает худшую производительность и не рекомендуется для использования в реальных задачах на больших массивах.

**Приложение**

Листинг программы:

#include <iostream>

#include <fstream>

#include <chrono> // для точного измерения времени

using namespace std;

// сортировка выбором

void selectionSort(int arr[], int n, int& comparisons, int& swaps) {

    comparisons = 0;

    swaps = 0;

    for (int i = 0; i < n - 1; i++) {

        int maxIdx = i;

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

            comparisons++;

            if (arr[j] > arr[maxIdx]) {

                maxIdx = j;

            }

        }

        if (maxIdx != i) {

            swap(arr[i], arr[maxIdx]);

            swaps++;

        }

    }

}

// сортировка вставками

void insertionSort(int arr[], int n, int& comparisons, int& swaps) {

    comparisons = 0;

    swaps = 0;

    for (int i = 1; i < n; i++) {

        int key = arr[i];

        int j = i - 1;

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

            comparisons++;

            arr[j + 1] = arr[j];

            j--;

            swaps++;

        }

        comparisons++; // Последнее сравнение в while

        arr[j + 1] = key;

        swaps++;

    }

}

// сортировка обменом (пузырьковая сортировка)

void bubbleSort(int arr[], int n, int& comparisons, int& swaps) {

    comparisons = 0;

    swaps = 0;

    for (int i = 0; i < n - 1; i++) {

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

            comparisons++;

            if (arr[j] < arr[j + 1]) {

                swap(arr[j], arr[j + 1]);

                swaps++;

            }

        }

    }

}

// вывод массива

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

    for (int i = 0; i < n; i++) {

        cout << arr[i] << " ";

    }

    cout << endl;

}

// чтение массива из файла

void readArrayFromFile(const string& filename, int arr[], int& size) {

    ifstream file(filename);

    if (!file) {

        cout << "Не удалось открыть файл " << filename << endl;

        exit(1);

    }

    size = 0;

    while (file >> arr[size]) {

        size++;

    }

    file.close();

}

int main() {

    const int N = 30; // Количество элементов массива

    int arr[N];

    int size;

    // считываем массив из файла

    readArrayFromFile("numbers.txt", arr, size);

    // копии массива для разных сортировок

    int arrSelection[N], arrInsertion[N], arrBubble[N];

    copy(arr, arr + size, arrSelection);

    copy(arr, arr + size, arrInsertion);

    copy(arr, arr + size, arrBubble);

    // сортировка выбором

    int comparisons = 0, swaps = 0;

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

    selectionSort(arrSelection, size, comparisons, swaps);

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

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

    cout << "Сортировка выбором:\n";

    cout << "Количество сравнений: " << comparisons << endl;

    cout << "Количество пересылок: " << swaps << endl;

    cout << "Время выполнения: " << durationSelection.count() << " секунд" << endl;

    cout << "Отсортированный массив: ";

    printArray(arrSelection, size);

    cout << endl;

    // сортировка вставками

    comparisons = 0, swaps = 0;

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

    insertionSort(arrInsertion, size, comparisons, swaps);

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

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

    cout << "Сортировка вставками:\n";

    cout << "Количество сравнений: " << comparisons << endl;

    cout << "Количество пересылок: " << swaps << endl;

    cout << "Время выполнения: " << durationInsertion.count() << " секунд" << endl;

    cout << "Отсортированный массив: ";

    printArray(arrInsertion, size);

    cout << endl;

    // сортировка обменом

    comparisons = 0, swaps = 0;

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

    bubbleSort(arrBubble, size, comparisons, swaps);

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

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

    cout << "Сортировка обменом:\n";

    cout << "Количество сравнений: " << comparisons << endl;

    cout << "Количество пересылок: " << swaps << endl;

    cout << "Время выполнения: " << durationBubble.count() << " секунд" << endl;

    cout << "Отсортированный массив: ";

    printArray(arrBubble, size);

    cout << endl;

    return 0;

}