ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

# Отчёт по лабораторной работе № 5

« Алгоритмы сортировки »

Выполнил работу

Карташов Игорь

Академическая группа №J3111

Принято

Вершинин Владислав

Санкт-Петербург

2024

## Введение

Цель работы: изучить алгоритмы сортировки и научиться применять их на практике.

Задачи:

1. Выбрать алгоритмы для реализации
2. Ознакомиться с их устройством
3. Изучить особенности реализации этих алгоритмов на C++
4. Реализовать алгоритмы
5. Провести анализ полученных результатов
6. Подготовить отчёт

## Теоретическая подготовка

### TreeSort

TreeSort — это алгоритм, основанный на использовании бинарного дерева поиска. Алгоритм включает два этапа: построение дерева из элементов массива и обход дерева для извлечения элементов в отсортированном порядке.

Сложность:

- Построение дерева: O(n · h), где h — высота дерева.
- Обход дерева: O(n).

Особенности:

- В худшем случае, когда дерево вырождается в список (h=n), сложность O(n**2).
- В среднем случае сложность O(n · log n).

### CountingSort

CountingSort основан на подсчете количества вхождений каждого элемента. Подходит только для сортировки целых чисел или других элементов, которые могут быть отображены в диапазон чисел.

Сложность: O(n+k), где k — диапазон значений.

Особенности:

- Линейная сложность для ограниченного диапазона.
- Не является сравнительной сортировкой.
- Потребляет дополнительную память для хранения массива частот.

### GnomeSort

GnomeSort — это упрощенная версия сортировки вставками, которая перемещается по массиву, меняя местами соседние элементы, если они стоят в неправильном порядке.

Сложность:

- Лучший случай: O(n) для почти отсортированных массивов.
- Худший случай: O(n**2).

Особенности:

- Простота реализации.
- Подходит для небольших массивов.

## Реализация

Используемые алгоритмы

В работе реализованы следующие алгоритмы сортировки:

1. TreeSort — построение бинарного дерева поиска и обход его узлов.
2. CountingSort — сортировка методом подсчета.
3. GnomeSort — сортировка методом перемещения

Реализация выполнена на языке C++ с использованием следующих библиотек:

- iostream — ввод/вывод.
- vector — работа с динамическими массивами.
- time.h — замер времени выполнения.
- cassert — проверка корректности работы через assert.
- algorithm — функции стандартной библиотеки.

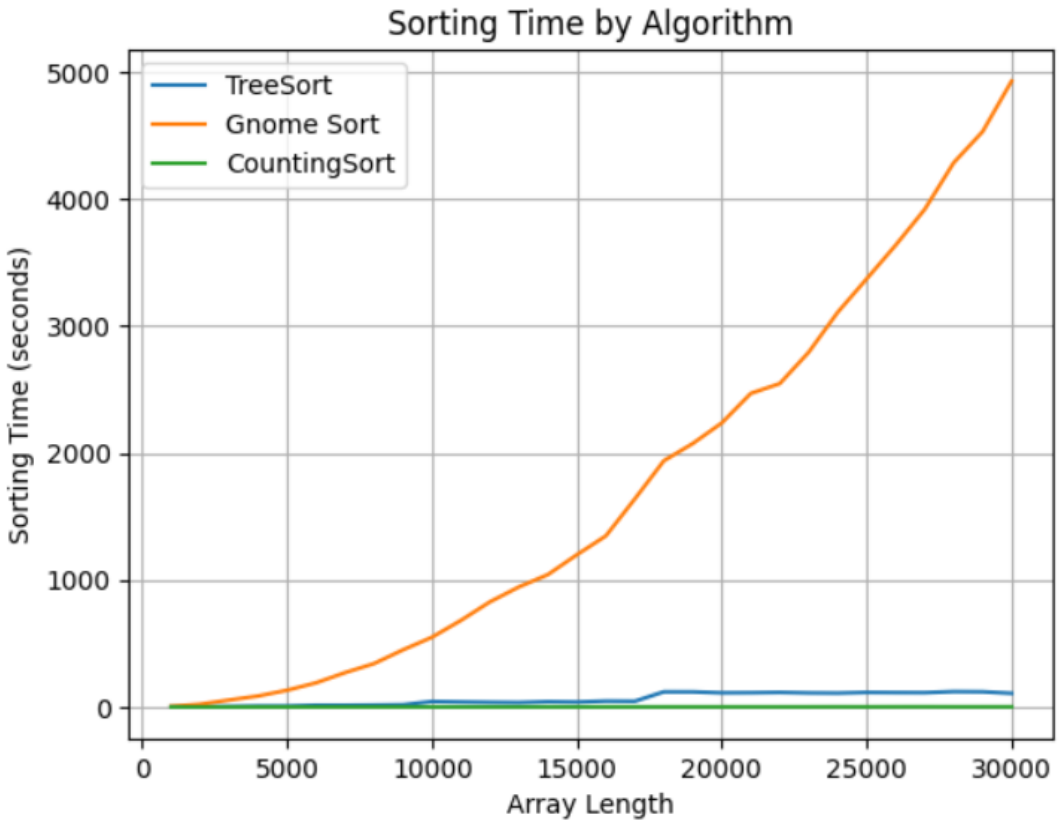
Тестирование

Для каждого алгоритма написаны два набора тестов:

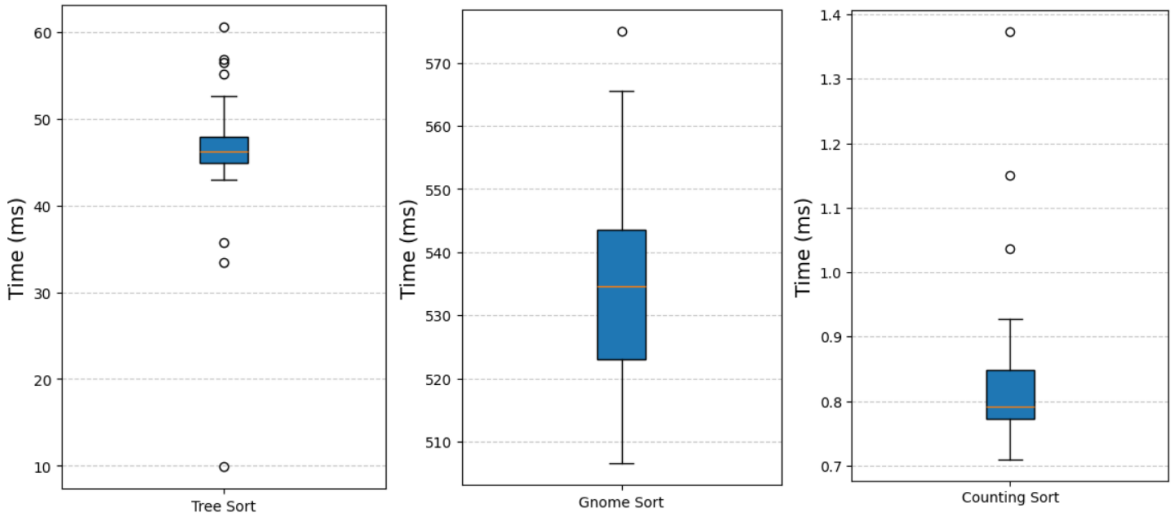
1. Тесты корректности: Проверка, что алгоритм возвращает отсортированный массив.
2. Тесты на производительность: Замер времени выполнения на массивах разной длины.

## Экспериментальная часть

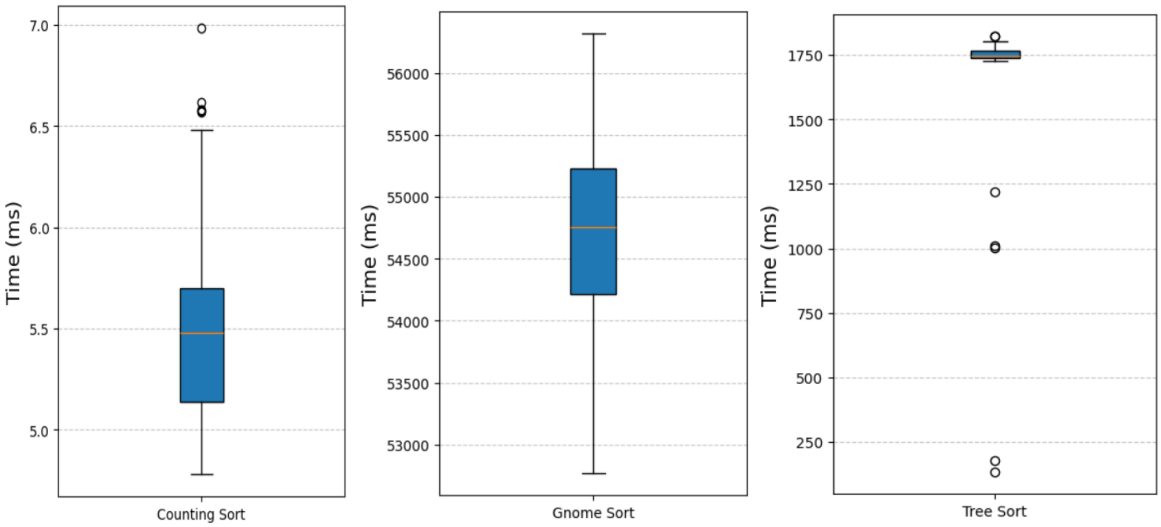
Затраченное время для каждой сортировки в зависимости от размера массива:




Распределения затраченного времени для размера 10**4:




Распределения затраченного времени для размера 10**5:


Time (ms)
7.0
6.5
6.0
5.5
5.0
Counting Sort
Time (ms)
56000
55500
55000
54500
54000
53500
53000
Gnome Sort
Time (ms)
1750
1500
1250
1000
750
500
250
Tree Sort

## Заключение

В ходе лабораторной работы реализованы и протестированы три алгоритма сортировки: TreeSort, CountingSort и GnomeSort. Полученные результаты подтверждают теоретические оценки их сложности.

Выбросы и их причины:

- TreeSort:

  Выбросы наблюдаются на сильно упорядоченных или полностью отсортированных данных, так как дерево становится не сбалансированным, что увеличивает время обработки.
- CountingSort:

  Для массивов с большим диапазоном значений время работы увеличивается из-за затрат на инициализацию и обработку массивов частот. Это особенно заметно при наличии редких, но больших значений.
- GnomeSort:

  Замедление фиксируется на случайных данных большой длины, так как алгоритм выполняет множество итераций

## Приложения

## Gnome Sort

```cpp
#include <iostream>
#include <cassert>
#include <time.h>
#include <cmath>
#include <cstdlib>
#include <algorithm>
#include <vector>

using namespace std;

// Гномья сортировка
void GnomeSort(vector<int>& arr) {
    int n = arr.size(); // O(1), 4 байта
    int index = 0; // O(1), 4 байта

    // Дополнительной памяти нет, так как массив изменяется на месте.

    // Проходимся по массиву
    while (index < n) { // В худшем случае: O(n^2)
        //Если порядок двух соседних элементов правильный идем дальше
        if (index==0 || arr[index] >= arr[index - 1]) {
            index++; // O(1)
        }
        else {
            //меняем соседние элементы местами и отходим назад по индексу
            swap(arr[index], arr[index-1]); // O(1)
            index--; // O(1)
        }
    }
}


//Вывод массива
void printArray(const std::vector<int>& arr) {
    for (const int& num : arr) { // O(n)
        cout << num << " ";
    }
}


// Тесты на правильность ответов (без проверки на скорость)
void correct_tests() {
    vector<int> excepted = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

    vector<int> input_1 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

```
    GnomeSort(input_1);
    assert(input_1 == excepted);

    vector<int> input_2 = {2, 7, 8, 5, 3, 4, 6, 9, 0, 1};
    GnomeSort(input_2);
    assert(input_2 == excepted);

    vector<int> input_3 = {6, 1, 7, 9, 4, 8, 3, 2, 0, 5};
    GnomeSort(input_3);
    assert(input_3 == excepted);

    vector<int> input_4 = {2, 4, 1, 3, 5, 6, 8, 7, 9, 0};
    GnomeSort(input_4);
    assert(input_4 == excepted);
    cout << "Correctness tests - complete" << std::endl;
}


// Тесты на время (без проверки на правильность ответов)
void time_tests() {
    std::vector<int> test_sizes = { 1000, 2000, 3000, 4000, 5000, 6000, 7000,
8000, 9000, 10000, };

    for (const int sz : test_sizes) {
        vector<int> random_vec(sz, 0);
        srand(time(0));
        generate(random_vec.begin(), random_vec.end(), rand);

        clock_t start = clock();
        GnomeSort(random_vec);

        cout << "На сортировку массива длины " << sz << " ушло " <<
double(clock() - start)/CLOCKS_PER_SEC*100 << " мс" << endl;
    }
}


int main() {
    correct_tests();
    time_tests();
    vector<int> arr = {3, 9, 1, 4, 2, 8, 5, 7, 6};
    GnomeSort(arr);
    printArray(arr);

    return 0;
}
```

## Counting Sort

```cpp
#include <vector>
#include <iostream>
#include <cassert>
#include <time.h>
#include <stack>
#include <cmath>
#include <cstdlib>
#include <algorithm>

using namespace std;


//COртировка подсчетом
void CountingSort(std::vector<int>& arr) {
    if (arr.empty()) return;

    int min_val = *min_element(arr.begin(), arr.end()); // O(n), 4 байта
    int max_val = *max_element(arr.begin(), arr.end()); // O(n), 4 байта

    //Создаём массив частот
    int range = max_val - min_val + 1; // O(1), 4 байта
    vector<int> count(range, 0); // O(range), 4 * range байт

    // Считаем количество вхождений для каждого числа
    for (int num : arr) { // O(n)
        count[num - min_val]++;
    }

    // Перезаписываем массив с учётом частот
    int index = 0; // O(1), 4 байта
    for (int i = 0; i < range; i++) { // O(range + n)
        while (count[i]-- > 0) {
            arr[index++] = i + min_val;
        }
    }
}


// Вывод массива
void printArray(const std::vector<int>& arr) {
    for (const int& num : arr) { // O(n)
        cout << num << " ";
    }
}
```

```
// Тесты на правильность ответов (без проверки на скорость)
void correct_tests() {
    vector<int> excepted = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

    vector<int> input_1 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    CountingSort(input_1);
    assert(input_1 == excepted);

    vector<int> input_2 = {2, 7, 8, 5, 3, 4, 6, 9, 0, 1};
    CountingSort(input_2);
    assert(input_2 == excepted);

    vector<int> input_3 = {6, 1, 7, 9, 4, 8, 3, 2, 0, 5};
    CountingSort(input_3);
    assert(input_3 == excepted);

    vector<int> input_4 = {2, 4, 1, 3, 5, 6, 8, 7, 9, 0};
    CountingSort(input_4);
    assert(input_4 == excepted);
    cout << "Correctness tests - complete" << std::endl;
}


// Тесты на время (без проверки на правильность ответов)
void time_tests() {
    std::vector<int> test_sizes = { 1000, 2000, 3000, 4000, 5000, 6000, 7000,
8000, 9000, 10000,};

    for (const int sz : test_sizes) {
        vector<int> random_vec(sz, 0);
        srand(time(0));
        generate(random_vec.begin(), random_vec.end(), rand);

        clock_t start = clock();
        CountingSort(random_vec);

        cout << "На сортировку массива длины " << sz << " ушло " <<
double(clock() - start)/CLOCKS_PER_SEC*100 << " мс" << endl;
    }
}


int main() {
    correct_tests();
    time_tests();
```

```
    vector<int> arr = {3, 9, 1, 4, 2, 8, 5, 7, 6};
    CountingSort(arr);
    printArray(arr);

    return 0;
}
```

## Tree Sort

```
#include <vector>
#include <iostream>
#include <cassert>
#include <time.h>
#include <stack>
#include <cmath>
#include <cstdlib>
#include <algorithm>

using namespace std;


struct Node {
    int value; // 4 байта
    Node* left; // 8 байта
    Node* right; // 8 байта

    // Конструктор для создания нового узла.
    Node(int val): value(val), left(nullptr), right(nullptr) {}
};


// Функция для вставки элемента в дерево
Node* insertIterative(Node* root, int value) {
    if (root == nullptr) {
        return new Node(value); // O(1), 20 байт на узел
    }

    Node* current = root; // 8 байт
    while (true) { // Цикл обходит дерево до листа
        if (value < current->value){
            if (current->left == nullptr){
                current->left = new Node(value);  // O(1), 20 байт
                break;
            } else {
```

```
                current = current->left; // O(1)
            }
        } else {
            if (current->right == nullptr) {
                current->right = new Node(value);  // O(1), 20 байт
                break;
            } else {
                current = current->right; // O(1)
            }
        }
    }
    return root;
    // O(h)
    // Память: 20 байт на каждый новый узел.
}


// Функция для обхода дерева (сортировка дерева)
vector<int> Sorting(Node* root) {
    vector<int> result; //O(n)
    stack<Node*> st; // O(h)  h — высота дерева
    Node* current = root; //8 байт

    while (!st.empty() || current != nullptr) { // пока стек не пуст или есть
узлы для обработки
        if (current != nullptr) {
            st.push(current);
            current = current->left; // Переход к левому поддереву
        } else {
            current = st.top();
            st.pop();
            result.push_back(current->value); // Обработка узла
            current = current->right; // Переход к правому поддереву
        }
    }

    return result; // Каждый узел обрабатывается ровно один раз - O(n)
}


// Основная функция TreeSort
vector<int> TreeSort(vector<int>& arr) {
    if (arr.empty()) {
        return {}; // O(1)
    }

    Node* root = nullptr; // 8 байт
```

```
    for (int value: arr) { // O(n * h)  // Обход всех элементов массива.
        root = insertIterative(root, value);  // Вставляем элементы в дерево
(O(h) для каждого элемента).
    }

    return Sorting(root); // O(n) // Сортируем дерево и возвращаем результат
}



// Вывод массива
void printArray(const std::vector<int>& arr) {
    for (const int& num : arr) { // O(n)
        cout << num << " ";
    }
}



// Тесты на правильность ответов (без проверки на скорость)
void correct_tests() {
    vector<int> excepted = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

    vector<int> input_1 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    input_1 = TreeSort(input_1);
    assert(input_1 == excepted);

    vector<int> input_2 = {2, 7, 8, 5, 3, 4, 6, 9, 0, 1};
    input_2 = TreeSort(input_2);
    assert(input_2 == excepted);

    vector<int> input_3 = {6, 1, 7, 9, 4, 8, 3, 2, 0, 5};
    input_3 = TreeSort(input_3);
    assert(input_3 == excepted);

    vector<int> input_4 = {2, 4, 1, 3, 5, 6, 8, 7, 9, 0};
    input_4 = TreeSort(input_4);
    assert(input_4 == excepted);
    cout << "Correctness tests - complete" << std::endl;
}



// Тесты на время (без проверки на правильность ответов)
void time_tests() {
    std::vector<int> test_sizes = { 1000, 2000, 3000, 4000, 5000, 6000, 7000,
8000, 9000, 10000, 100000};

    for (const int sz : test_sizes) {
```

```
        vector<int> random_vec(sz, 0);
        srand(time(0));
        generate(random_vec.begin(), random_vec.end(), rand);

        clock_t start = clock();
        TreeSort(random_vec);

        cout << "На сортировку массива длины " << sz << " ушло " <<
double(clock() - start)/CLOCKS_PER_SEC*100 << " мс" << endl;
    }
}


int main() {
    correct_tests();
    time_tests();
    vector<int> arr = {3, 9, 1, 4, 2, 8, 5, 7, 6};
    vector<int> sortedArr = TreeSort(arr);
    printArray(sortedArr);

    return 0;
}
```