ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 5

«Алгоритмы сортировки»

Выполнил работу

Смирнов Олег

Академическая группа J3110

Принято

Вершинин Владислав

Санкт-Петербург

2024

**Структура отчёта:**

1. Введение

Цель: реализовать 3 алгоритма сортировки с разной асимптотикой.

Задачи: написать pancake sort, heap sort, bucket sort.

1. Теоретическая подготовка

Знание pancake sort, heap sort (и как работает бинарная куча), bucket sort.

1. Реализация

Pancake sort: ищем максимальный элемент, переворачиваем часть массива сначала и до этого индекса, разворачиваем весь массив и отправляем максимальный элемент в конец.

Heap sort: строим полное бинарное дерево и делаем из него кучу просеиванием, свапаем root с самым правым элементом и удаляем, повторяем пока остались элементы.

Bucket sort: создаём n корзин для наших элементов. Определяем номер корзины по формуле (num - minVal) \* n / (maxVal - minVal + 1). Сортируем корзины по отдельности и объединяем.

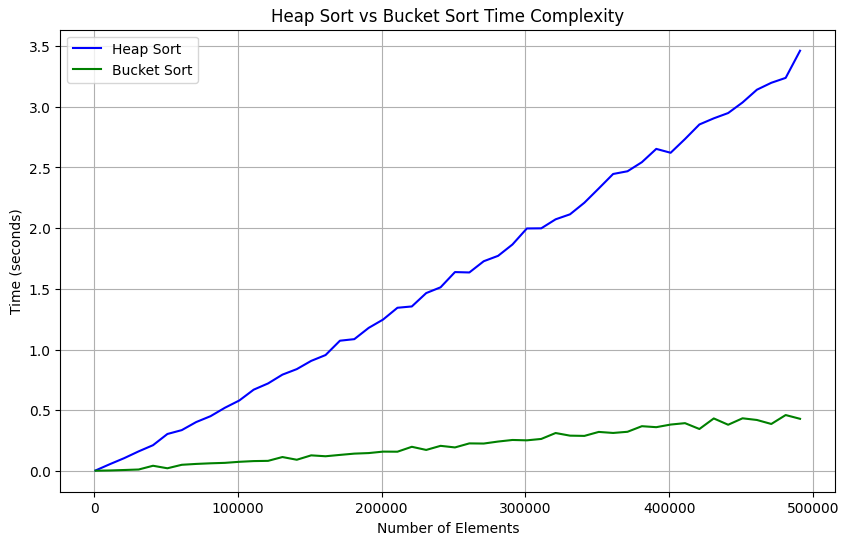
1. Экспериментальная часть

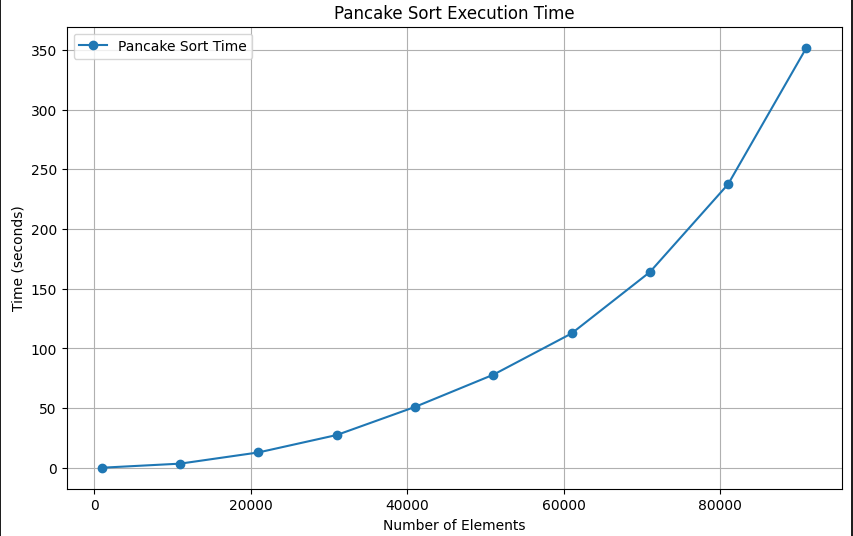
Пространственная сложность:

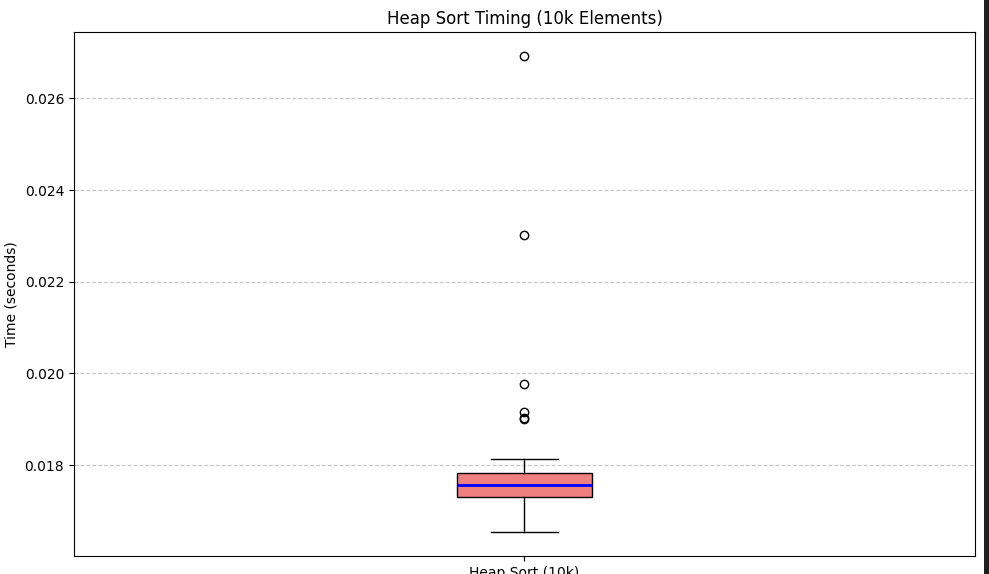
1. O(1)
2. O(1)
3. O(n) с учётом указателей и хранения длины вектора, конечно, это не совсем O(n).

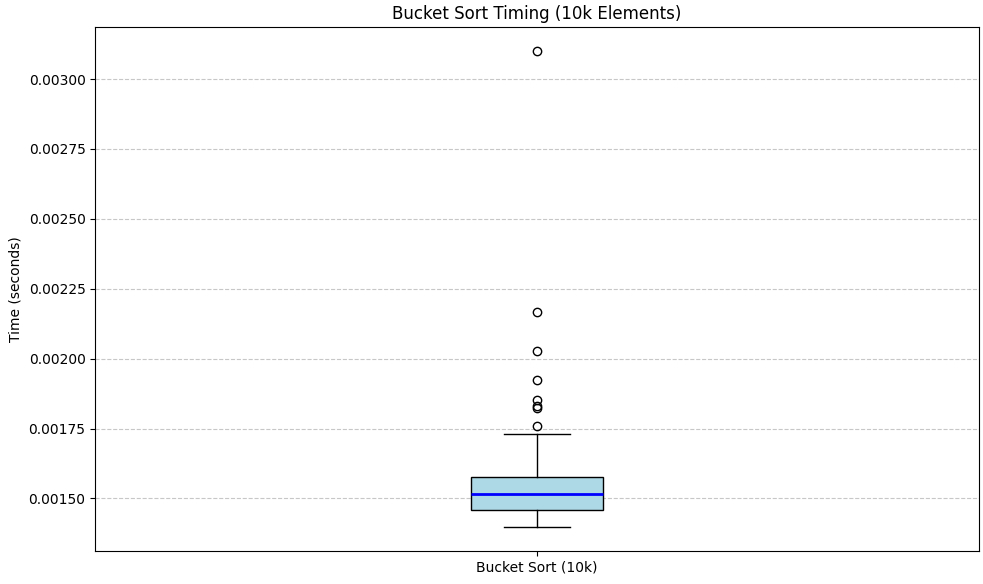
Временная сложность:

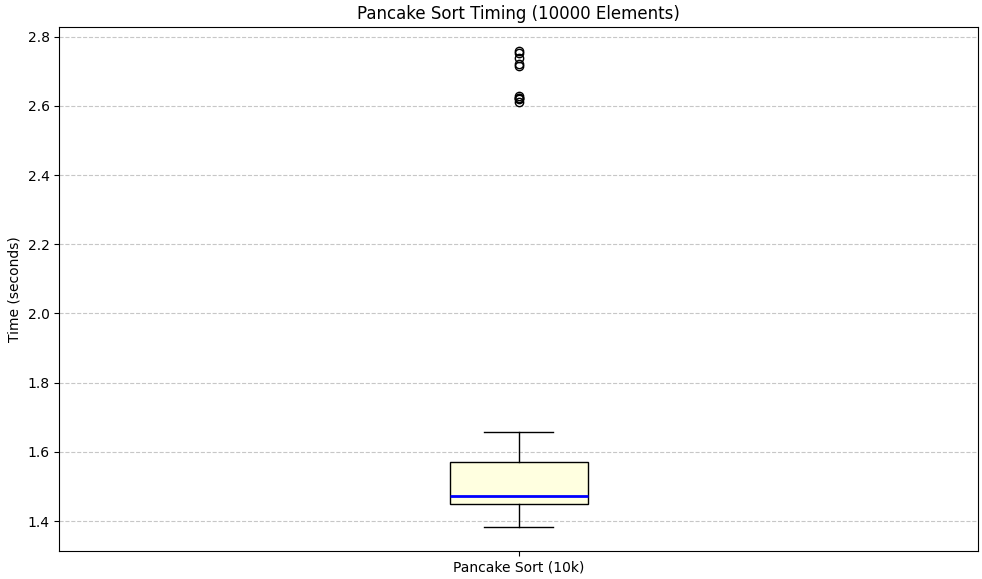
1. O(n^2)
2. O(n\*logn)
3. O(n + k\*logk), где k максимальное количество элементов в корзине

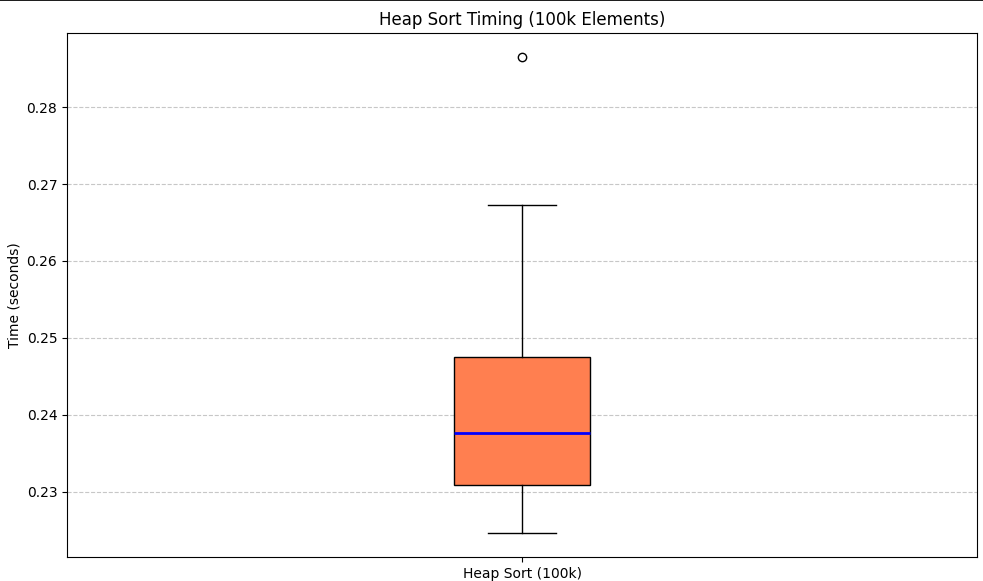


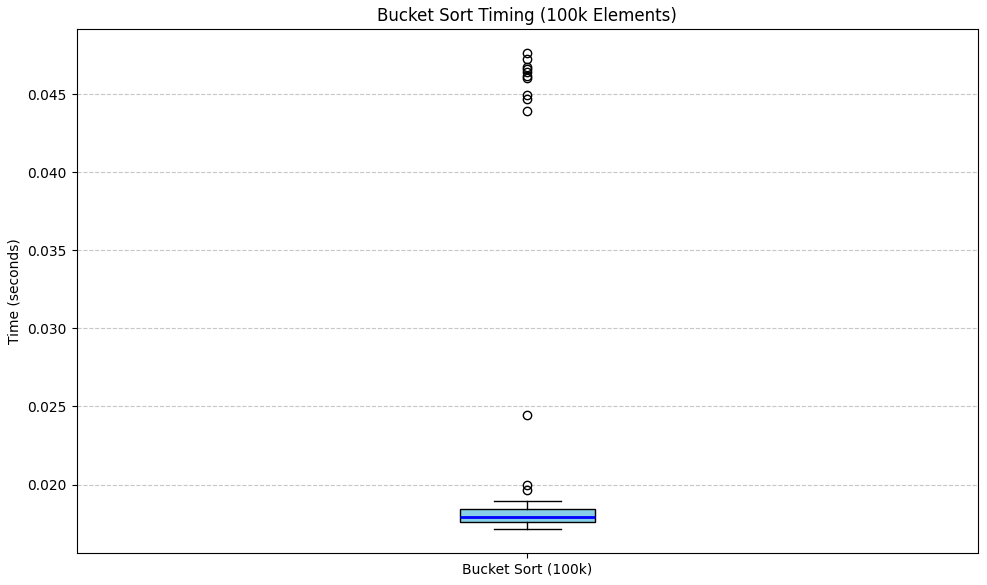












1. Заключение

В ходе лабораторной работы реализованы и протестированы три алгоритма сортировки: Pancake sort, heap sort, bucket sort. Полученные результаты подтверждают теоретические оценки их сложности.

Выбросы pancake sort: массив отсортирован практически в обратном порядке.

Heap sort: выбросов мало, алгоритм стабильно работает на разных данных. Выбросы появляются при большом количестве просеиваний. Массив практически в обратном порядке и возможно ещё какие-то чуть более хитрые случаи.

Bucket sort: выбросов крайне много, это из-за неравномерного распределения величины чисел.

1. Приложения

Pancake sort

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

void flip(vector<int>& arr, int k) {

    reverse(arr.begin(), arr.begin() + k);

}

int findMaxIndex(const vector<int>& arr, int n) {

    int maxIndex = 0;

    for (int i = 1; i < n; ++i) {

        if (arr[i] > arr[maxIndex]) {

            maxIndex = i;

        }

    }

    return maxIndex;

}

// Реализация Pancake Sort

void pancakeSort(vector<int>& arr) {

    int n = arr.size();

    for (int currSize = n; currSize > 1; --currSize) {

        // Найти индекс максимального элемента в текущем подмассиве

        int maxIndex = findMaxIndex(arr, currSize);

        // Если максимальный элемент не на месте, перемещаем его

        if (maxIndex != currSize - 1) {

            // Переворачиваем, чтобы переместить максимальный элемент в начало

            if (maxIndex != 0) {

                flip(arr, maxIndex + 1);

            }

            // Переворачиваем, чтобы переместить максимальный элемент в конец подмассива

            flip(arr, currSize);

        }

    }

}

int main() {

    int n;

    cin >> n;

    vector<int> arr(n);

    for (int i = 0; i < n; i++){

        cin >> arr[i];

    }

    pancakeSort(arr);

    for (int i = 0; i < n; i++){

        cout << arr[i] << ' ';

    }

    cout << '\n';

    return 0;

}

#include <gtest/gtest.h>

// Лучший случай: массив уже отсортирован

TEST(PancakeSortTest, BestCase) {

    vector<int> input = {1, 2, 3, 4, 5};

    vector<int> expected = {1, 2, 3, 4, 5};

    pancakeSort(input);

    EXPECT\_EQ(input, expected);

}

// Средний случай: массив с произвольным порядком элементов

TEST(PancakeSortTest, AverageCase) {

    vector<int> input = {5, 1, 4, 2, 3};

    vector<int> expected = {1, 2, 3, 4, 5};

    pancakeSort(input);

    EXPECT\_EQ(input, expected);

}

// Худший случай: массив отсортирован в обратном порядке

TEST(PancakeSortTest, WorstCase) {

    vector<int> input = {5, 4, 3, 2, 1};

    vector<int> expected = {1, 2, 3, 4, 5};

    pancakeSort(input);

    EXPECT\_EQ(input, expected);

}

// Дополнительный случай: массив с повторяющимися элементами

TEST(PancakeSortTest, Duplicates) {

    vector<int> input = {4, 2, 2, 5, 1, 3, 3};

    vector<int> expected = {1, 2, 2, 3, 3, 4, 5};

    pancakeSort(input);

    EXPECT\_EQ(input, expected);

}

// Пустой массив

TEST(PancakeSortTest, EmptyArray) {

    vector<int> input = {};

    vector<int> expected = {};

    pancakeSort(input);

    EXPECT\_EQ(input, expected);

}

// Массив из одного элемента

TEST(PancakeSortTest, SingleElement) {

    vector<int> input = {42};

    vector<int> expected = {42};

    pancakeSort(input);

    EXPECT\_EQ(input, expected);

}

Heap sort

#include <iostream>

#include <vector>

using namespace std;

void heapify(vector<int>& arr, int n, int i) {

    int largest = i;

    int left = 2 \* i + 1;

    int right = 2 \* i + 2;

    if (left < n && arr[left] > arr[largest])

        largest = left;

    if (right < n && arr[right] > arr[largest])

        largest = right;

    if (largest != i) {

        swap(arr[i], arr[largest]);

        heapify(arr, n, largest);

    }

}

void heapSort(vector<int>& arr) {

    int n = arr.size();

    for (int i = n / 2 - 1; i >= 0; i--)

        heapify(arr, n, i);

    for (int i = n - 1; i >= 0; i--) {

        swap(arr[0], arr[i]);

        heapify(arr, i, 0);

    }

}

int main() {

    int n;

    cin >> n;

    vector<int> arr(n);

    for (int i = 0; i < n; i++){

        cin >> arr[i];

    }

    heapSort(arr);

    for (int i = 0; i < n; i++){

        cout << arr[i] << ' ';

    }

    cout << '\n';

    return 0;

}

#include <gtest/gtest.h>

// Лучший случай: массив уже отсортирован

TEST(HeapSortTest, BestCase) {

    vector<int> input = {1, 2, 3, 4, 5};

    vector<int> expected = {1, 2, 3, 4, 5};

    heapSort(input);

    EXPECT\_EQ(input, expected);

}

// Средний случай: массив с произвольным порядком элементов

TEST(HeapSortTest, AverageCase) {

    vector<int> input = {5, 1, 4, 2, 3};

    vector<int> expected = {1, 2, 3, 4, 5};

    heapSort(input);

    EXPECT\_EQ(input, expected);

}

// Худший случай: массив отсортирован в обратном порядке

TEST(HeapSortTest, WorstCase) {

    vector<int> input = {5, 4, 3, 2, 1};

    vector<int> expected = {1, 2, 3, 4, 5};

    heapSort(input);

    EXPECT\_EQ(input, expected);

}

// Дополнительный случай: массив с повторяющимися элементами

TEST(HeapSortTest, Duplicates) {

    vector<int> input = {4, 2, 2, 5, 1, 3, 3};

    vector<int> expected = {1, 2, 2, 3, 3, 4, 5};

    heapSort(input);

    EXPECT\_EQ(input, expected);

}

// Пустой массив

TEST(HeapSortTest, EmptyArray) {

    vector<int> input = {};

    vector<int> expected = {};

    heapSort(input);

    EXPECT\_EQ(input, expected);

}

// Массив из одного элемента

TEST(HeapSortTest, SingleElement) {

    vector<int> input = {42};

    vector<int> expected = {42};

    heapSort(input);

    EXPECT\_EQ(input, expected);

}

Bucket sort

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

void bucketSort(vector<float>& arr) {

    int n = arr.size();

    float minVal = \*min\_element(arr.begin(), arr.end());

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

    // Создать ведра и распределить элементы

    vector<vector<float>> buckets(n);

    for (float num : arr) {

        int bucketIndex = (num - minVal) \* n / (maxVal - minVal + 1); // Индексация ведра

        buckets[bucketIndex].push\_back(num);

    }

    // Сортировать и объединять ведра

    arr.clear();

    for (auto& bucket : buckets) {

        sort(bucket.begin(), bucket.end());

        arr.insert(arr.end(), bucket.begin(), bucket.end());

    }

}

int main() {

    int n;

    cin >> n;

    vector<float> arr(n);

    for (int i = 0; i < n; i ++){

        cin >> arr[i];

    }

    bucketSort(arr);

    for (int i = 0; i < n; i ++){

        cout << arr[i] << ' ';

    }

    cout << '\n';

    return 0;

}

#include <gtest/gtest.h>

// Лучший и средний случай: равномерное распределение

TEST(BucketSortTest, BestCase) {

    vector<float> input = {0.1, 0.2, 0.3, 0.4, 0.5};

    vector<float> expected = {0.1, 0.2, 0.3, 0.4, 0.5};

    bucketSort(input);

    EXPECT\_EQ(input, expected);

}

// Худший случай: почти все в одной корзине

TEST(BucketSortTest, WorstCase) {

    vector<float> input = {0.1, 0.11, 0.12, 0.13, 0.8};

    vector<float> expected = {0.1, 0.11, 0.12, 0.13, 0.8};

    bucketSort(input);

    EXPECT\_EQ(input, expected);

}

// Массив с одинаковыми элементами

TEST(BucketSortTest, Duplicates) {

    vector<float> input = {0.3, 0.3, 0.3, 0.3, 0.3};

    vector<float> expected = {0.3, 0.3, 0.3, 0.3, 0.3};

    bucketSort(input);

    EXPECT\_EQ(input, expected);

}

// Пустой массив

TEST(BucketSortTest, EmptyArray) {

    vector<float> input = {};

    vector<float> expected = {};

    bucketSort(input);

    EXPECT\_EQ(input, expected);

}

// Массив с одним элементом

TEST(BucketSortTest, SingleElement) {

    vector<float> input = {0.42};

    vector<float> expected = {0.42};

    bucketSort(input);

    EXPECT\_EQ(input, expected);

}