ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 5

«Алгоритмы сортировки»

Выполнил работу

Макаров Эльдар

Академическая группа J3114

Принято

Должность, звание Фамилия Имя преподавателя

Санкт-Петербург

2024

1. Введение

Цель работы: изучение и реализация алгоритмов для сортировки данных

Задачи:

Найти 3 алгоритма, подходящих под условия.

Реализовать алгоритм на языке C++.

Оценить временную и пространственную сложность.

Проверить алгоритмы для лучших, средних и худших случаев

1. Теоретическая подготовка

Алгоритмы сортировки — это алгоритмы для упорядочения элементов в списке. Они берут последовательность из n элементов и переставляют элементы таким образом, чтобы получившаяся последовательность удовлетворяла определённому условию.

1. Реализация

Этап 1: Анализ задачи

В первую очередь я начал искать подходящие для этой задачи алгоритмы. Я перебрал множество вариантов, но остановился на следующих трех:

Brick sort – это относительно простой алгоритм сортировки. Он сравнивает все пары соседних элементов с индексами odd/even и, если пара находится в неправильном порядке (первое больше второго), элементы переключаются.

Heap sort — алгоритм сортировки, использующий структуру данных двоичная куча. Он организует элементы массива в бинарную кучу, а затем сортирует эту кучу, перемещая наибольший элемент в массиве.

Flashsort — это алгоритм сортировки, который показывает линейную вычислительную сложность O(n) для равномерно распределённых наборов данных. Он эффективно реализует сортировку с помощью «карманов». Для этого каждый из n входных элементов распределяется в одно из m «карманов», затем все элементы перестраиваются в соответствии с «карманами» и сортируются по отдельности.

Этап 2: Разработка алгоритма

В этой лабораторной работе мы используем уже существующие алгоритмы, так что как таковой разработки не было, я нашел алгоритм в интернете и использовал его.

Этап 3: Написание кода

Код:  
#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

#include <algorithm>

#define HEAP\_SORT // Активируйте одну из директив: BRICK\_SORT, HEAP\_SORT, FLASH\_SORT

using namespace std;

// Функция создания случайного списка

vector<int> generateRandomList(int size) {

    vector<int> arr(size);

    srand(time(0)); // Инициализация генератора случайных чисел

    for (int i = 0; i < size; i++) {

        arr[i] = rand();

    }

    return arr;

}

// Функция для вывода первых и последних элементов списка

void printArray(const vector<int>& arr, int limit = 10) {

    int n = arr.size();

    for (int i = 0; i < limit && i < n; i++) {

        cout << arr[i] << " ";

    }

    if (n > limit) {

        cout << "... ";

        for (int i = n - limit; i < n; i++) {

            cout << arr[i] << " ";

        }

    }

    cout << endl;

}

// Brick Sort

#ifdef BRICK\_SORT

void brickSort(vector<int>& arr) {

    int n = arr.size();

    bool sorted = false;

    while (!sorted) {

        sorted = true;

        // Чётные индексы

        for (int i = 0; i < n - 1; i += 2) {

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

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

                sorted = false;

            }

        }

        // Нечётные индексы

        for (int i = 1; i < n - 1; i += 2) {

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

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

                sorted = false;

            }

        }

    }

}

#endif

// Heap Sort

#ifdef HEAP\_SORT

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);

    }

}

#endif

// Flash Sort

#ifdef FLASH\_SORT

void flashSort(vector<int>& arr) {

    int n = arr.size();

    if (n <= 1) return;

    int minVal = arr[0], maxIdx = 0;

    for (int i = 1; i < n; i++) {

        if (arr[i] < minVal) minVal = arr[i];

        if (arr[i] > arr[maxIdx]) maxIdx = i;

    }

    if (arr[maxIdx] == minVal) return;

    int m = int(0.1 \* n);

    vector<int> L(m, 0);

    double c = double(m - 1) / (arr[maxIdx] - minVal);

    for (int i = 0; i < n; i++) {

        int k = int(c \* (arr[i] - minVal));

        L[k]++;

    }

    for (int i = 1; i < m; i++) {

        L[i] += L[i - 1];

    }

    swap(arr[maxIdx], arr[0]);

    int count = 0, i = 0, k = m - 1;

    while (count < n) {

        while (i > L[k] - 1) {

            i++;

            k = int(c \* (arr[i] - minVal));

        }

        int evicted = arr[i];

        while (i != L[k]) {

            k = int(c \* (evicted - minVal));

            swap(evicted, arr[--L[k]]);

            count++;

        }

    }

    for (int i = 1; i < m; i++) {

        for (int j = L[i - 1]; j < L[i]; j++) {

            int key = arr[j];

            int l = j - 1;

            while (l >= L[i - 1] && arr[l] > key) {

                arr[l + 1] = arr[l];

                l--;

            }

            arr[l + 1] = key;

        }

    }

}

#endif

int main() {

    const int SIZE = 10000000;

    vector<int> arr = generateRandomList(SIZE);

    cout << "Array size: " << SIZE << endl;

    cout << "Initial array (first and last 10 elements):" << endl;

    printArray(arr);

    clock\_t start = clock();

#ifdef BRICK\_SORT

    brickSort(arr);

#elif defined(HEAP\_SORT)

    heapSort(arr);

#elif defined(FLASH\_SORT)

    flashSort(arr);

#endif

    clock\_t end = clock();

    cout << "Sorted array (first and last 10 elements):" << endl;

    printArray(arr);

    cout << "Time taken: " << double(end - start) / CLOCKS\_PER\_SEC << " seconds" << endl;

    return 0;

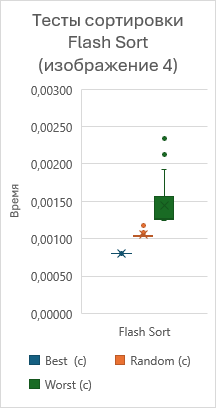
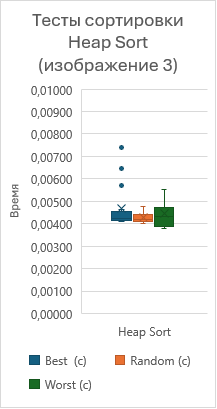
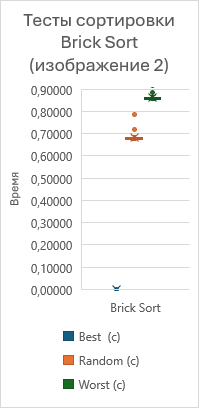
}

1. Экспериментальная часть

Асимптотика Brick Sort: O(n^2) в худшем и среднем случаях.

Асимптотика Heap Sort: O(n /log n) во всех случаях.

Асимптотика Flash Sort: O(n) в среднем случае, O(n^2) в худшем случае.



На графике 1 видно, что Brick Sort крайне неэффективен по сравнению с Heap Sort, из-за чего мне пришлось сильно ограничить Brick Sort на графике, чтобы остальные виды сортировок были видны.

На графиках (изображение 2, 3, 4) видно, что Brick Sort существенно медленнее остальных алгоритмов, при этом, Flash Sort лучше Heap Sort, но не так сильно.

1. Заключение

В ходе выполнения лабораторной работы мною были проанализированы и реализованы алгоритмы сортировки массивов. Алгоритм Brick Sort показывает медленную работу даже на массивах среднего размера. Это связано с квадратичной временной сложностью и большим числом сравнений и обменов. На практике данный алгоритм неэффективен для больших массивов и может быть использован только для малых объемов данных. Алгоритм Heap Sort демонстрирует стабильные результаты, близкие к теоретическим ожиданиям. На больших массивах время выполнения растет логарифмически, что соответствует заявленной сложности. Однако в ряде случаев были замечены незначительные выбросы времени выполнения, вызванные затратами на перераспределение памяти при работе с большими объемами данных. Алгоритм Flash Sort демонстрирует крайне высокую скорость на массивах со случайным распределением элементов, что подтверждает его среднюю сложность O(n). Однако на сильно упорядоченных данных время выполнения может значительно увеличиться, из-за чего возникают выбросы.

В качестве дальнейших исследований можно предложить оптимизацию алгоритма с точки зрения уменьшения затрат использования памяти и ускорения его работы.