**Правительство Российской Федерации**

**Федеральное государственное автономное образовательное учреждение высшего профессионального образования Национальный исследовательский университет "Высшая школа экономики"**

Московский институт электроники и математики Национального исследовательского университета "Высшая школа экономики"

Департамент прикладной математики

**Отчёт Кузнецова Арсения (Группа №245)**

**По лабораторной работе №7**

**По курсу «Алгоритмизация и программирование»**

**13.05.2025**

**Москва – 2025г**

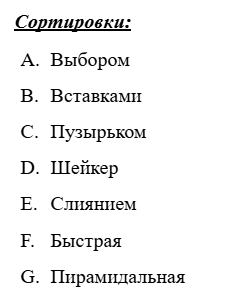
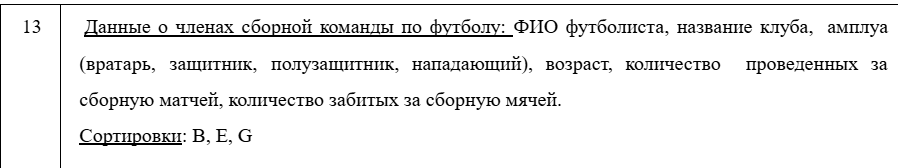
**Лабораторная работа 7**

1. Реализовать указанные в варианте алгоритмы сортировки для массива объектов в  соответствии с вариантом. Структура объекта описана в таблице. Определить функции сравнения объектов по следующему принципу: приоритет сравнения определяется порядком поля в структуре, т.е. если структурный тип Х содержит (в описании варианта) поля с именами a,b,c,d, то при сравнении двух объектов типа Х надо сравнить поля с именем a, в случае их равенства сравнить поля b и т.д.

2. Каждый алгоритм сортировки должен поддерживать сортировку в обоих направлениях: по неубыванию и по невозрастанию.

3. Выбор и запуск требуемого алгоритма и направления сортировки осуществляется через меню на этапе выполнения.

4. Провести сортировку каждым алгоритмом массивов следующих размеров: 100, 1000, 10000, 50000, 100000, 200000, 300000. Засечь (программно) время сортировки  каждым алгоритмом. По полученным точкам построить графики зависимости времени сортировки от размерности массива для каждого из алгоритмов сортировки на одной оси координат. Полученные графики включить в отчет к работе.

main.c:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include "footballer.h"

#ifdef REDDY\_GO

int main() {

    srand(time(NULL));

    int sizes[] = {10, 100, 1000, 10000, 30000};

    int num\_sizes = sizeof(sizes) / sizeof(sizes[0]);

    printf("Comparison of sorting time:\n");

    printf("%-10s | %-15s | %-15s | %-15s\n", "Size", "Insertion", "Merge", "Heap");

    printf("--------------------------------------------------------------\n");

    for (int i = 0; i < num\_sizes; i++) {

        int n = sizes[i];

        Footballer \*arr = malloc(n \* sizeof(Footballer));

        for (int j = 0; j < n; j++) generateFootballer(&arr[j]);

        if (n <= 10) {

            printf("\nGenerated (unsorted) array of %d footballers:\n", n);

            printFootballers(arr, n);

            Footballer \*copy1 = malloc(n \* sizeof(Footballer));

            memcpy(copy1, arr, n \* sizeof(Footballer));

            insertionSort(copy1, n);

            printf("\nSorted (insertionSort) array:\n");

            printFootballers(copy1, n);

            free(copy1);

            Footballer \*copy2 = malloc(n \* sizeof(Footballer));

            memcpy(copy2, arr, n \* sizeof(Footballer));

            mergeSortWrapper(copy2, n);

            printf("\nSorted (mergeSortWrapper) array:\n");

            printFootballers(copy2, n);

            free(copy2);

            Footballer \*copy3 = malloc(n \* sizeof(Footballer));

            memcpy(copy3, arr, n \* sizeof(Footballer));

            heapSort(copy3, n);

            printf("\nSorted (heapSort) array:\n");

            printFootballers(copy3, n);

            free(copy3);

        }

        double t1 = measureSortingTime(insertionSort, arr, n);

        double t2 = measureSortingTime(mergeSortWrapper, arr, n);

        double t3 = measureSortingTime(heapSort, arr, n);

        printf("\n");

        printf("%-10d | %-15.6f | %-15.6f | %-15.6f\n", n, t1, t2, t3);

        free(arr);

    }

    return 0;

}

// Ветку #else оставил без изменений — она работает корректно

#else

int main() {

    srand(time(NULL));

    int sizes[] = {100, 1000, 10000, 50000};

    int num\_sizes = sizeof(sizes) / sizeof(sizes[0]);

    printf("Choose sorting algorithm:\n");

    printf("B - Insertion Sort\nE - Merge Sort\nG - Heap Sort\n> ");

    char algo;

    scanf(" %c", &algo);

    printf("Sorting order (1 for ascending, -1 for descending): ");

    scanf("%d", &sort\_order);

    void (\*sortFunction)(Footballer[], int) = NULL;

    switch(algo) {

        case 'B': sortFunction = insertionSort; break;

        case 'E': sortFunction = mergeSortWrapper; break;

        case 'G': sortFunction = heapSort; break;

        default: printf("Invalid choice!\n"); return 1;

    }

    FILE \*csv\_file = fopen("results/sort\_results.csv", "w");

    if (!csv\_file) {

        printf("Error creating file!\n");

        return 1;

    }

    if (ftell(csv\_file) == 0) {

        fputs("\xEF\xBB\xBF", csv\_file);

        fprintf(csv\_file, "Algorithm;Order;Size;Time (sec)\n");

    }

    printf("\n%-10s | %-10s\n", "Size", "Time");

    printf("--------------------\n");

    for (int i = 0; i < num\_sizes; i++) {

        int n = sizes[i];

        Footballer \*arr = malloc(n \* sizeof(Footballer));

        for (int j = 0; j < n; j++) {

            generateFootballer(&arr[j]);

        }

        double time = measureSortingTime(sortFunction, arr, n);

        printf("%-10d | %-10.3f\n", n, time);

        fprintf(csv\_file, "%c;%d;%d;%.3f\n", algo, sort\_order, n, time);

        free(arr);

    }

    fclose(csv\_file);

    printf("\nResults saved to sort\_results.csv\n");

    return 0;

}

#endif

footballer.h:

#ifndef STRUCT\_H

#define STRUCT\_H

#define REDDY\_GO

typedef struct {

    char fullName[33];     // ФИО футболиста

    char clubName[20];     // Название клуба

    char position[20];     // Амплуа

    int age;               // Возраст

    int matchesPlayed;     // Количество проведенных матчей

    int goalsScored;       // Количество забитых мячей

} Footballer;

extern int sort\_order;

extern const char\* fullNames[];

extern const char\* clubNames[];

extern const char\* positions[];

// Прототипы функций

void generateFootballer(Footballer \*f);

void printFootballers(Footballer arr[], int n);

int compareFootballers(const Footballer \*a, const Footballer \*b);

void insertionSort(Footballer arr[], int n);

void merge(Footballer arr[], int l, int m, int r);

void mergeSort(Footballer arr[], int l, int r);

void mergeSortWrapper(Footballer arr[], int n);

void heapify(Footballer arr[], int n, int i);

void heapSort(Footballer arr[], int n);

double measureSortingTime(void (\*sortFunc)(Footballer[], int), Footballer arr[], int n);

void printResults(const char \*name, double time);

#endif // STRUCT\_H

footballer.c:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include "footballer.h"

int sort\_order = 1; // 1 для возрастающего порядка, -1 для убывающего

const char\* fullNames[] = {"Mukhin Jaromir", "Prytkov Roman", "Slastnikov Sergey", "Bober Stanislav", "Shchur Lev", "Kochetkov Yuri", "Volkova Tatiana"};

const char\* clubNames[] = {"MIEM", "FCS", "MSU", "VSU", "MIPT", "MISIS", "ITMO"};

const char\* positions[] = {"forward", "midfielder", "defender", "goalkeeper"};

void generateFootballer(Footballer \*f) {

    strcpy(f->fullName, fullNames[rand() % 7]);

    strcpy(f->clubName, clubNames[rand() % 7]);

    strcpy(f->position, positions[rand() % 4]);

    f->age = 18 + rand() % 23;

    f->matchesPlayed = rand() % 100;

    f->goalsScored = rand() % 50;

}

void printFootballers(Footballer arr[], int n) {

    for (int i = 0; i < n; i++) {

        printf("%-20s | %-20s | %-10s | Age: %2d | Matches: %3d | Goals: %2d\n",

               arr[i].fullName, arr[i].clubName, arr[i].position,

               arr[i].age, arr[i].matchesPlayed, arr[i].goalsScored);

    }

}

int compareFootballers(const Footballer \*a, const Footballer \*b) {

    int result = strcmp(a->fullName, b->fullName);

    if (result != 0) return result \* sort\_order;

    result = strcmp(a->clubName, b->clubName);

    if (result != 0) return result \* sort\_order;

    result = strcmp(a->position, b->position);

    if (result != 0) return result \* sort\_order;

    result = a->age - b->age;

    if (result != 0) return result \* sort\_order;

    result = a->matchesPlayed - b->matchesPlayed;

    if (result != 0) return result \* sort\_order;

    result = a->goalsScored - b->goalsScored;

    return result \* sort\_order;

}

void insertionSort(Footballer arr[], int n) {

    for (int i = 1; i < n; i++) {

        Footballer key = arr[i];

        int j = i - 1;

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

            arr[j + 1] = arr[j];

            j--;

        }

        arr[j + 1] = key;

    }

}

void merge(Footballer arr[], int l, int m, int r) {

    int n1 = m - l + 1;

    int n2 = r - m;

    Footballer \*L = malloc(n1 \* sizeof(Footballer));

    Footballer \*R = malloc(n2 \* sizeof(Footballer));

    for (int i = 0; i < n1; i++) L[i] = arr[l + i];

    for (int j = 0; j < n2; j++) R[j] = arr[m + 1 + j];

    int i = 0, j = 0, k = l;

    while (i < n1 && j < n2) {

        if (compareFootballers(&L[i], &R[j]) <= 0) {

            arr[k++] = L[i++];

        } else {

            arr[k++] = R[j++];

        }

    }

    while (i < n1) arr[k++] = L[i++];

    while (j < n2) arr[k++] = R[j++];

    free(L);

    free(R);

}

void mergeSort(Footballer arr[], int l, int r) {

    if (l < r) {

        int m = l + (r - l) / 2;

        mergeSort(arr, l, m);

        mergeSort(arr, m + 1, r);

        merge(arr, l, m, r);

    }

}

void mergeSortWrapper(Footballer arr[], int n) {

    mergeSort(arr, 0, n - 1);

}

void heapify(Footballer arr[], int n, int i) {

    int largest = i;

    int left = 2 \* i + 1;

    int right = 2 \* i + 2;

    if (left < n && compareFootballers(&arr[left], &arr[largest]) > 0)

        largest = left;

    if (right < n && compareFootballers(&arr[right], &arr[largest]) > 0)

        largest = right;

    if (largest != i) {

        Footballer temp = arr[i];

        arr[i] = arr[largest];

        arr[largest] = temp;

        heapify(arr, n, largest);

    }

}

void heapSort(Footballer arr[], int n) {

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

        heapify(arr, n, i);

    for (int i = n - 1; i > 0; i--) {

        Footballer temp = arr[0];

        arr[0] = arr[i];

        arr[i] = temp;

        heapify(arr, i, 0);

    }

}

double measureSortingTime(void (\*sortFunc)(Footballer[], int), Footballer arr[], int n) {

    Footballer \*arr\_copy = malloc(n \* sizeof(Footballer));

    memcpy(arr\_copy, arr, n \* sizeof(Footballer));

    clock\_t start = clock();

    sortFunc(arr\_copy, n);

    clock\_t end = clock();

    free(arr\_copy);

    return (double)(end - start) / CLOCKS\_PER\_SEC;

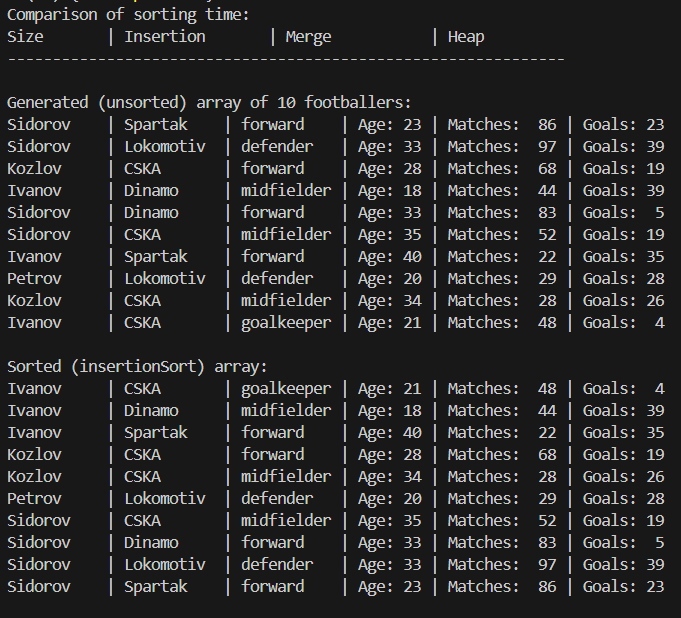
}

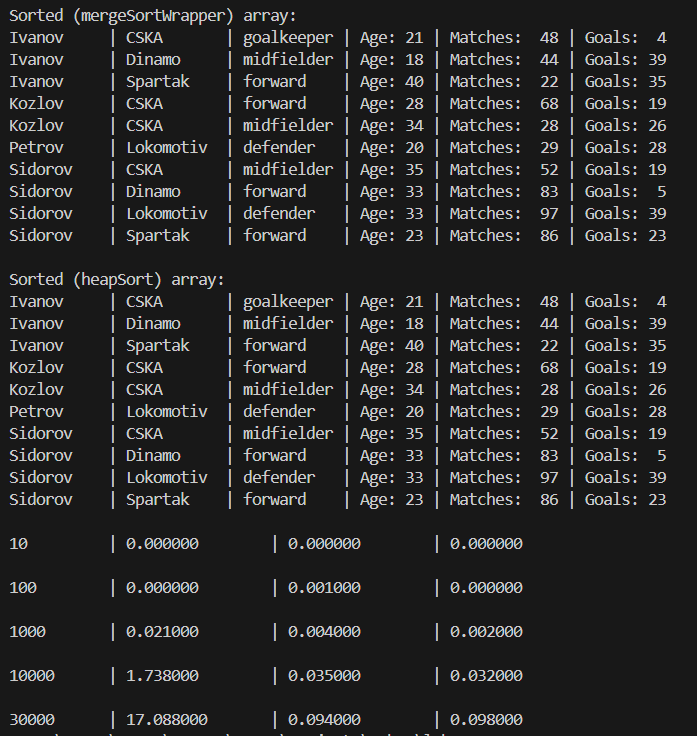
void printResults(const char \*fullName, double time) {

    printf("%-33s | %.6f sec\n", fullName, time);

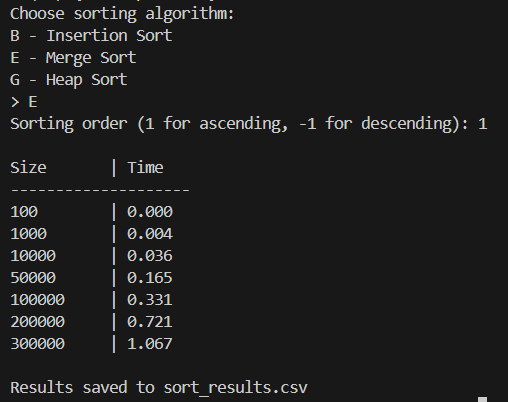
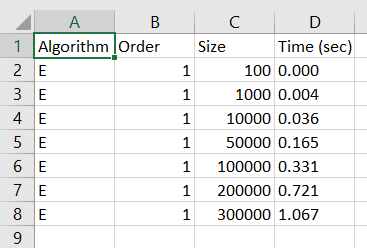
}

***TESTS:***

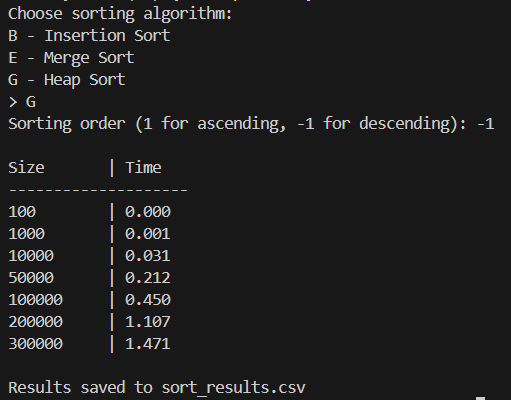
1. Подключаем #define REDDY\_GO

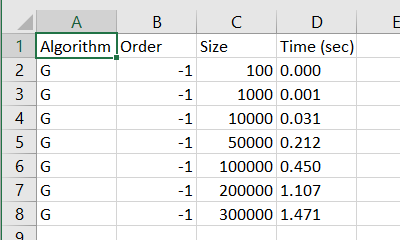


1. Если не REDDY\_GO ( Слиянием, в порядке увеличения)

1. (Пирамидальная , в порядке убывания)





Также файл подключен к коду на питон, для обновления графиков на google colab:

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from time import sleep

from IPython.display import clear\_output

import io

from google.colab import files

# Загрузка файла (если нужно)

uploaded = files.upload()

filename = next(iter(uploaded))  # Получаем имя загруженного файла

# Или используем файл из текущей директории

# filename = 'sort\_results.csv'

def load\_data(file\_path):

    try:

        # Читаем файл с учетом возможного разделителя

        df = pd.read\_csv(io.StringIO(uploaded[file\_path].decode('utf-8')), sep=';|,', engine='python')

        print("\nДанные успешно загружены:")

        print(df.head())

        return df

    except Exception as e:

        print(f"Ошибка при загрузке файла: {e}")

        return None

def create\_plots(df):

    if df is None or df.empty:

        print("Нет данных для построения графиков")

        return

    # Очищаем вывод перед созданием новых графиков

    clear\_output(wait=True)

    # 1. Линейный график

    plt.figure(figsize=(12, 6))

    plt.title('Сравнение алгоритмов сортировки (линейная шкала)', fontsize=14)

    sns.lineplot(data=df, x='Size', y='Time (sec)', hue='Algorithm', marker='o')

    plt.xlabel('Размер массива')

    plt.ylabel('Время (секунды)')

    plt.grid(True)

    plt.legend(title='Алгоритм')

    plt.tight\_layout()

    plt.show()

    # 2. Логарифмический график

    plt.figure(figsize=(12, 6))

    plt.title('Сравнение алгоритмов сортировки (логарифмическая шкала)', fontsize=14)

    plt.xscale('log')

    plt.yscale('log')

    sns.lineplot(data=df, x='Size', y='Time (sec)', hue='Algorithm', marker='o')

    plt.xlabel('Размер массива (лог)')

    plt.ylabel('Время (лог)')

    plt.grid(True, which='both', linestyle='--')

    plt.legend(title='Алгоритм')

    plt.tight\_layout()

    plt.show()

def main():

    print("Начинаем мониторинг файла...")

    last\_content = None

    try:

        while True:

            # В Colab нет watchdog, поэтому используем простую проверку

            current\_content = uploaded[filename]

            if current\_content != last\_content:

                print("\nОбнаружены изменения в файле. Обновляю графики...")

                df = load\_data(filename)

                create\_plots(df)

                last\_content = current\_content

            sleep(2)  # Проверяем каждые 2 секунды

    except KeyboardInterrupt:

        print("\nМониторинг остановлен")

if \_\_name\_\_ == "\_\_main\_\_":

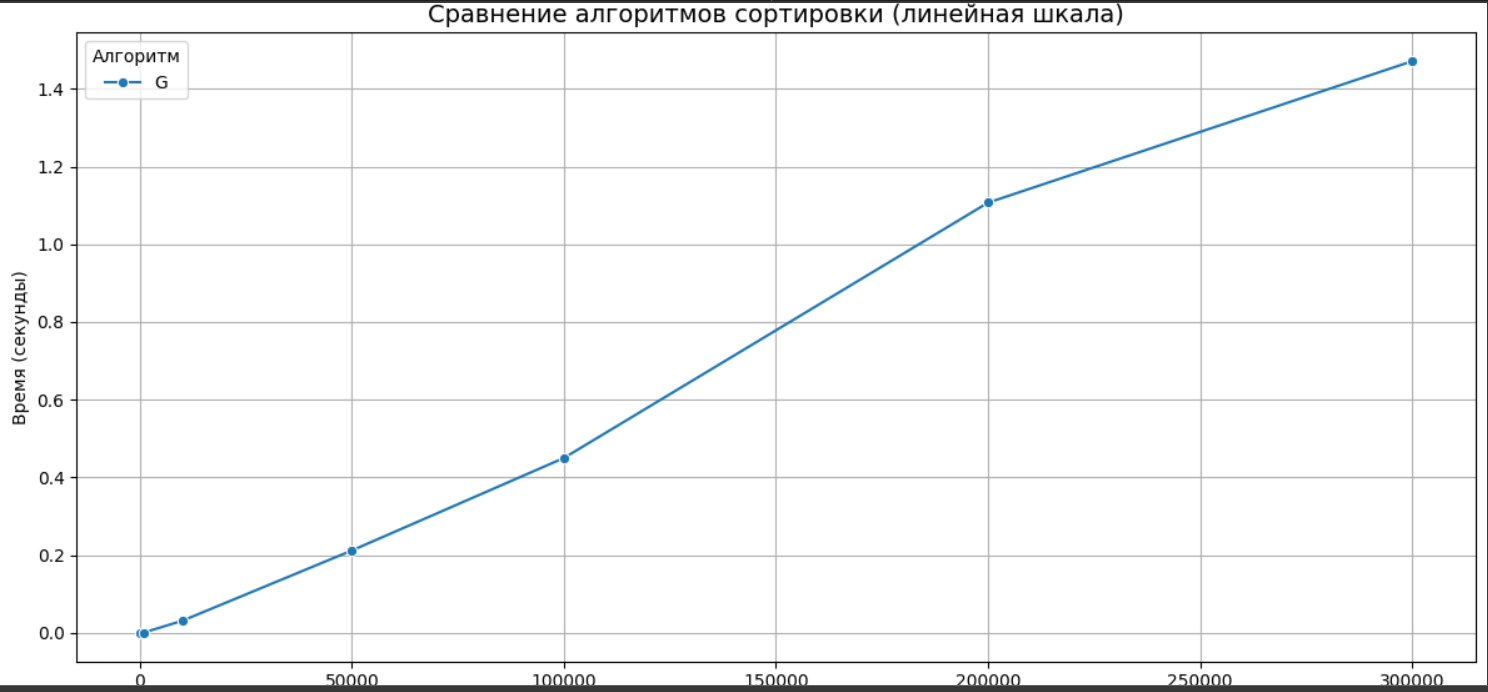
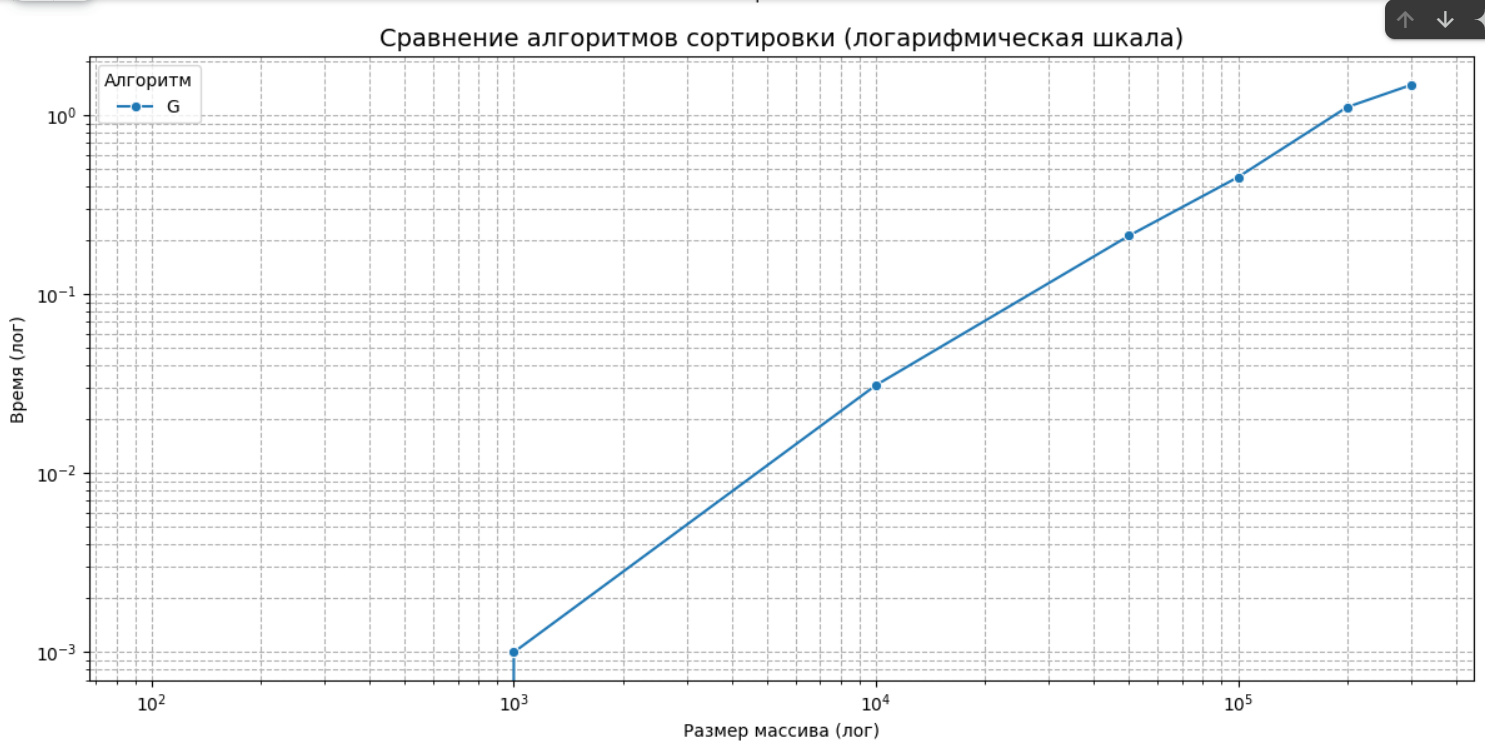
    # Первоначальная загрузка данных и построение графиков

    df = load\_data(filename)

    create\_plots(df)

    # Запускаем мониторинг изменений

    main()

  import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

# Создаем DataFrame с предоставленными данными

data = {

    'Size': [10, 100, 1000, 10000, 30000],

    'Insertion': [0.001, 0.0001, 0.019, 1.844, 17.198],  # Заменил 0.000 на 0.0001

    'Merge': [0.0001, 0.0001, 0.003, 0.035, 0.096],      # Заменил 0.000 на 0.0001

    'Heap': [0.0001, 0.001, 0.003, 0.029, 0.095]         # Заменил 0.000 на 0.0001

}

df = pd.DataFrame(data)

# Преобразуем данные в "длинный" формат для seaborn

df\_long = df.melt(id\_vars='Size', var\_name='Algorithm', value\_name='Time (sec)')

# Устанавливаем современный стиль графиков

plt.style.use('seaborn-v0\_8')

sns.set\_style("whitegrid")

sns.set\_palette("husl")

plt.rcParams['figure.facecolor'] = 'white'

plt.rcParams['axes.facecolor'] = 'white'

# 1. Линейный график в линейном масштабе

plt.figure(figsize=(12, 6), dpi=100)

plt.title('Сравнение времени выполнения алгоритмов сортировки', fontsize=14, pad=20)

ax = sns.lineplot(data=df\_long, x='Size', y='Time (sec)', hue='Algorithm',

                 marker='o', markersize=8, linewidth=2.5)

plt.xlabel('Размер массива', fontsize=12)

plt.ylabel('Время (секунды)', fontsize=12)

plt.xticks(fontsize=10)

plt.yticks(fontsize=10)

plt.ylim(bottom=0)  # Начинаем ось Y с нуля

plt.grid(True, linestyle='--', alpha=0.6)

plt.legend(title='Алгоритм', title\_fontsize=12, fontsize=11, framealpha=1)

plt.tight\_layout()

plt.show()

# 2. Улучшенный логарифмический график

plt.figure(figsize=(12, 6), dpi=100)

plt.title('Сравнение времени выполнения (логарифмическая шкала)', fontsize=14, pad=20)

# Создаем свою сетку значений для осей

x\_ticks = [10, 100, 1000, 10000, 30000]

y\_ticks = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100]

plt.xscale('log')

plt.yscale('log')

ax = sns.lineplot(data=df\_long, x='Size', y='Time (sec)', hue='Algorithm',

                 marker='o', markersize=8, linewidth=2.5)

# Устанавливаем свои метки на осях

plt.xticks(x\_ticks, labels=[str(x) for x in x\_ticks], fontsize=10)

plt.yticks(y\_ticks, labels=[f"{y:.4f}" if y < 0.001 else f"{y:.3f}" for y in y\_ticks], fontsize=10)

plt.xlabel('Размер массива', fontsize=12)

plt.ylabel('Время (секунды)', fontsize=12)

plt.grid(True, which='both', linestyle='--', alpha=0.6)

# Устанавливаем границы осей

plt.xlim(left=8, right=40000)

plt.ylim(bottom=0.00005, top=100)  # Расширяем границы для лучшего отображения

plt.legend(title='Алгоритм', title\_fontsize=12, fontsize=11, framealpha=1)

plt.tight\_layout()

plt.show()

# 3. Столбчатая диаграмма для наглядного сравнения

plt.figure(figsize=(14, 7), dpi=100)

plt.title('Время выполнения алгоритмов по размерам массивов', fontsize=14, pad=20)

df\_long['Size\_str'] = df\_long['Size'].astype(str)

ax = sns.barplot(data=df\_long, x='Size\_str', y='Time (sec)', hue='Algorithm',

                edgecolor='black', linewidth=0.5)

plt.xlabel('Размер массива', fontsize=12)

plt.ylabel('Время (секунды)', fontsize=12)

plt.xticks(fontsize=10)

plt.yticks(fontsize=10)

plt.ylim(bottom=0)

plt.grid(True, axis='y', linestyle='--', alpha=0.6)

# Добавляем значения времени на столбцы

for p in ax.patches:

    height = p.get\_height()

    if height > 0:

        ax.text(p.get\_x() + p.get\_width()/2., height + 0.05,

                f'{height:.3f}',

                ha="center", fontsize=9)

plt.legend(title='Алгоритм', title\_fontsize=12, fontsize=11, framealpha=1)

plt.tight\_layout()

plt.show()

