Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Российский химико-технологический университет имени Д.И. Менделеева»
Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1

Выполнила студентка группы .....КС-33..... Георгиевская Анастасия Игоревна
Ссылка на репозиторий: .....
https://github.com/MUCTR-IKT-CPP/GeorgievskayaAA_33_alg/tree/main/lab%201

Приняли: .....Пысин Максим Дмитриевич
.....Лобанов Алексей Владимирович

Дата сдачи: 19.02.2025

Оглавление

Описание задачи.

Задание:

- Реализовать метод сортировки соответствующий вашему варианту:
  i. Сортировка пузырьком
- Реализовать проведения тестирования алгоритма сериями расчетов для измерения параметров времени.
  За один расчет выполняется следующие операции:
  i. Генерируется массив случайных значений
  ii. Запоминается время начала расчета алгоритма сортировки
  iii. Выполняется алгоритм сортировки
    - Во время выполнения измерить количество повторных прохождений по массиву.
    - Во время выполнения измерить количество выполнения операций обмена значений.

  iv. Вычисляется время затраченное на сортировку: текущее время - время начала
  v. Сохраняется время для одной попытки После этого расчет повторяется до окончания серии.

  i. Алгоритм вычисляется 8 сериями по 20 раз за серию.
  ii. Алгоритм в каждой серии вычисляется для массива размером M. (1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000)
  iii. Массив заполняется значения числами с плавающей запятой в интервале от -1 до 1.
  iv. Для серии запоминаются все времена которые были замерены
- По полученным данным времени построить графики зависимости времени от числа элементов в массиве:
  i. Совмещенный график наихудшего времени выполнения сортировки и сложности алгоритма указанной в нотации O большое.

Для построения графика вычисляется O большое для каждого размера массива. При этом при вычислении функции O(c * g(N)) подбирается такая константа c, что бы при значении > 1000 график O(N) был выше графика наихудшего случая, но второй график на его фоне не превращался в прямую линию

  ii. Совмещенный график среднего, наилучшего и наихудшего времени исполнения.
  iii. График среднего количества обмена значений.
  iv. График повторных обходов массива.
- По результатом расчетов оформляется отчет по предоставленной форме, в отчете:
  ii. Приводится описание алгоритма.

iii. Приводится описания выполнения задачи (Описание кода и специфических элементов реализации)
iv. Приводятся выводы (Графики и их анализ)

Описание сортировки пузырьком.

Сортировка пузырьком — это простой алгоритм сортировки, который работает по принципу многократного прохода по массиву и «пузырьковому» перемещению элементов в нужном порядке. Процесс сортировки состоит из серии сравнений соседних элементов и их обмена местами, если они расположены в неправильном порядке. После каждого прохода наибольший элемент «всплывает» в конец массива (или начало, в зависимости от реализации), как пузырьки воздуха, поднимающиеся на поверхность воды.

Этапы работы:

1. Сравнение элементов: Алгоритм сравнивает соседние элементы массива и, если они идут в неправильном порядке, меняет их местами.
2. Проход по массиву: После одного прохода, самый большой элемент будет в конце массива (в случае сортировки по возрастанию).
3. Повторение шагов: Процесс повторяется для оставшейся части массива (каждый раз игнорируя уже отсортированные элементы).
4. Завершение работы: Алгоритм завершает работу, когда за проход по массиву не было сделано ни одного обмена элементов (массив отсортирован).

Оценка сложности алгоритма

Время выполнения:

1. Лучший случай — O(n): когда массив уже отсортирован, алгоритм выполнит один проход по массиву, не совершив ни одного обмена.
2. Средний и худший случаи — $O(n^2)$: каждый элемент массива нужно сравнить с каждым другим, что ведет к квадратичной сложности. Даже в случае частично отсортированных массивов алгоритм будет выполнять все возможные сравнения.

Преимущества алгоритма

1. Простота реализации: Алгоритм очень прост в реализации, что делает его хорошим выбором для обучения и на практике.
2. Подходит для почти отсортированных массивов: В случае, если массив уже почти отсортирован, алгоритм может сработать эффективно благодаря раннему прекращению (если за проход не было обменов).

Недостатки алгоритма

1. Низкая эффективность на больших данных: Пузырьковая сортировка является неэффективной на больших массивах, поскольку ее сложность — $O(n^2)$. Это делает ее непригодной для реальных задач с большими объемами данных.
2. Много лишних сравнений: Даже если массив частично отсортирован, алгоритм будет делать множество ненужных сравнений.

Выполнение задачи.

Для выполнения задачи лабораторной был использован язык С.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <stdint.h>  // Для uint64_t

void generation(double arr[], int n) {
   int i = 0;
    for(i = 0; i < n; i++) {
        arr[i] = (2.0 * rand() / RAND_MAX) - 1.0;
    }
}

void bubble(double arr[], int n, unsigned long long* pass_count, unsigned long
long* swap_count) {
    *pass_count = 0;
    *swap_count = 0;
    double temp;

    int i, j = 0;
    for(i = 0; i < n - 1; i++) {
        (*pass_count)++;
        for (j = 0; j < n - 1 - i; j++) {
            if (arr[j] > arr[j + 1]) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                (*swap_count)++;
            }
        }
    }
}

void measure(int m, int s) {
```

```
    FILE *f = fopen("times.txt", "a");
    if (!f) {
        printf("Error opening file\n");
        return;
    }

    double *arr = (double *)malloc(m * sizeof(double));
    if (arr == NULL) {
        fprintf(stderr, "Memory allocation failed\n");
        return;
    }

    double ttime = 0.0;
    clock_t start_time, end_time;

    // Сначала выводим размер массива в файл
    fprintf(f, "\n--- MASSIVE %d ---\n", m);
    printf("\n%d\n", m);
    fprintf(f, "TRY TIME PASS SWAP\n");

   int i = 0;
    for (i = 0; i < s; i++) {
        unsigned long long pass_count = 0, swap_count = 0;  // Обнуляем счетчики на каждой итерации
        generation(arr, m);  // Генерация случайных данных
        start_time = clock();

        bubble(arr, m, &pass_count, &swap_count);     // Сортировка с отслеживанием количества проходов и обменов

        end_time = clock();

        ttime = ((double)(end_time - start_time)) / CLOCKS_PER_SEC;
        fprintf(f, "%2d %.10f %llu %llu\n", i + 1, ttime, pass_count, swap_count);  // Выводим результаты в файл
        printf("%d ", i + 1);  // Выводим номер текущего прогона на экран
```

```
    }

    fclose(f);
    free(arr);  // Освобождаем память
}


int main() {
    srand(time(NULL)); // Инициализация генератора случайных чисел

    int sizes[] = {1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000};
    int num_sizes = sizeof(sizes) / sizeof(sizes[0]), series = 20;

    FILE *f = fopen("times.txt", "a");
    if (!f) {
        printf("Error opening file\n");
        return 1;
    }

   int i = 0;
    for(i = 0; i < num_sizes; i++) {
        measure(sizes[i], series);
    }

    fclose(f);
    return 0;
}
```

Данная программа реализует алгоритм сортировки пузырьком и тестирует его производительность на массивах различных размеров. Для каждого массива измеряется время сортировки, количество перестановок элементов и обходов массива.

Структура кода:

- Подключаемые библиотеки
    - <stdio.h> — стандартная библиотека для ввода-вывода, включает функции для работы с файлами (fopen, fclose, fprintf), а также функции вывода в консоль (printf).

    - <stdlib.h> — библиотека для работы с динамической памятью, генерацией случайных чисел и другими утилитами (например, malloc, rand, free).
    - <time.h> — библиотека для работы с временем. Здесь она используется для отслеживания времени выполнения программы с помощью функции clock().
    - <stdint.h> — библиотека для работы с типами данных фиксированной ширины. В данном коде она подключена, но не используется явно (поскольку тип unsigned long long используется напрямую).
- Функция generation
    - Вход: пустой массив заданной длины, количество элементов
    - Назначение: Генерация случайных чисел в диапазоне от -1 до 1 для массива заданной длины.
    - Как работает: Функция использует стандартную функцию rand(), чтобы получить случайное число в диапазоне от 0 до RAND_MAX, а затем с помощью математической операции масштабирует его в диапазон от -1 до 1.
    - Результат: заполненный числами от -1 до 1 массив
- Функция bubble
    - Вход: массив чисел, количество элементов, количество обходов иперестановок
    - Назначение: Сортировка массива методом пузырька.
    - Как работает: Метод пузырька проходит по массиву, сравнивает соседние элементы и меняет их местами, если они идут в неправильном порядке. Подсчитываются два показателя: обходы и обмены.
    - Результат: отсортированный массив, количество обменов и обходов.
- Функция measure
    - Вход: количество элементов, количество попыток в серии.
    - Назначение: Выполняет замеры времени работы сортировки для массивов различных размеров.
    - Как работает: Получаем массив случайных чисел. Для каждого прогона (всего s прогонов) измеряется время выполнения сортировки методом пузырька с использованием clock().Результаты измерений, включая время, количество проходов и перестановок, записываются в файл times.txt.
    - Результат: Файл times.txt с необходимыми данными.
- Функция main
    - Назначение: Инициализирует программу, настраивает параметры и вызывает функцию измерения для различных размеров массивов.
    - Как работает: Массив sizes[] содержит размеры массивов для тестирования. Для каждого размера массива вызывается функция measure, которая выполняет сортировку и измеряет время. Результаты всех тестов записываются в файл times.txt.

- По завершении программы выводится файл times.txt, в котором присутствуют:
    - Измеренное время
    - Количество перестановок
    - Количество полных проходов

Заключение.

В ходе выполнения лабораторной работы была реализована сортировка пузырьком на языке C и приведено ее тестирование на массивах различного размера. Для анализа эффективности заданной сортировки были измерены некоторые параметры времени и количество обменов и обходов (подр. в разделе “Задание”).

Для более наглядного представления на основе полученных данных построены 4 диаграммы:

1. Совмещенный график наихудшего времени выполнения сортировки и сложности алгоритма указанной в нотации O большое (методом подбора было 9 найдено значение константы, при которой линия функции лежит не ниже линии полученного времени: 5,26E-09)(рис. 1);

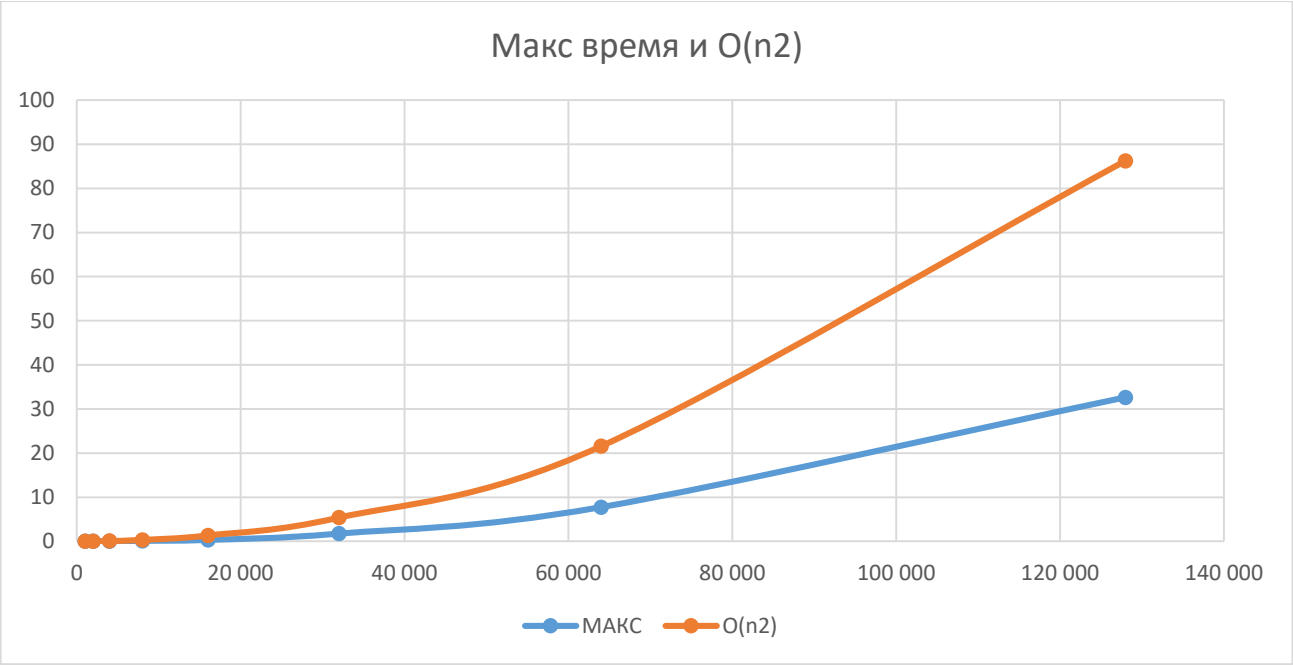

рис. 1. Сравнение худшего времени и сложности алгоритма

2. График среднего, наилучшего и наихудшего времени выполнения (рис. 2.1). Для более наглядного представления различий двух графиков при малом количестве элементов был построен дополнительный график с логарифмической шкалой (рис. 2.2);

МИН	0,001776	0,006775	0,026812	0,108770	0,313000	1,712000	7,471000	31,317000
СР	0,001892	0,007002	0,027451	0,114033	0,324200	1,738250	7,584800	31,847650
МАКС	0,002377	0,008342	0,029713	0,125428	0,334000	1,764000	7,754000	32,638000
$O(n^2)$	0,005263	0,021053	0,084211	0,336842	1,347368	5,389474	21,557895	86,231579

Таблица 1. Значения для построения графиков

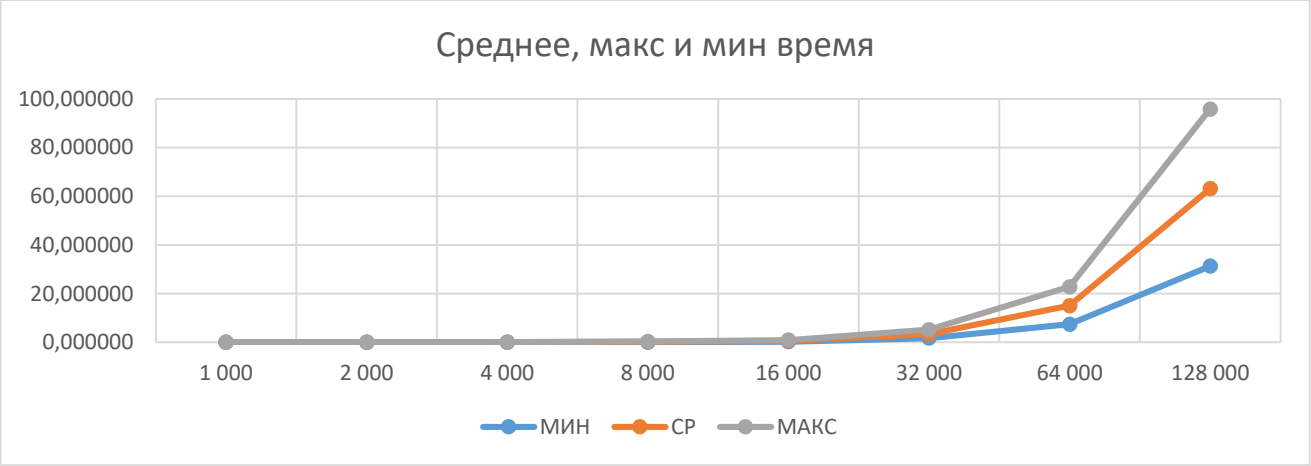

рис. 2.1. Среднее, максимальное и минимальное время.

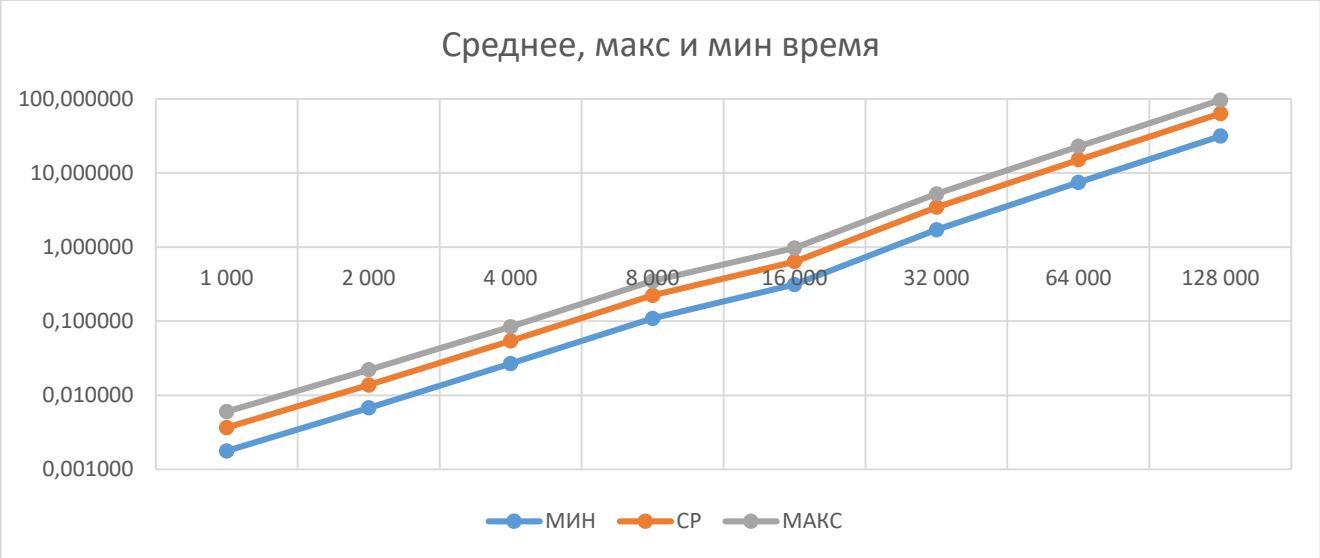

рис. 2.2. Среднее, максимальное и минимальное время. Логарифмическая шкала.

3. График среднего количества обменов значений (рис. 3);

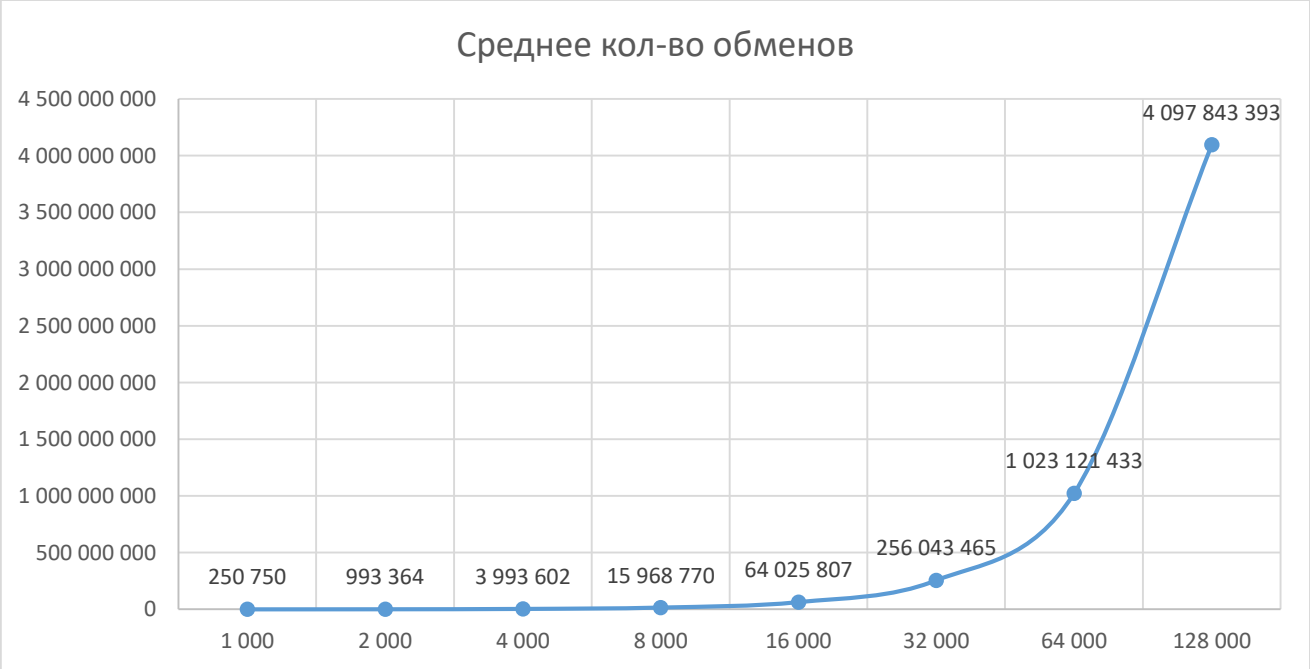

рис. 3. Среднее количество обменов

4. График повторных обходов массива (рис. 4).

рис. 4. Среднее количество повторных обходов

На основании построенных графиков можно подтвердить теоретическую оценку сложности алгоритма $O(n^2)$. Диаграммы показывают, что время выполнения растет квадратично с увеличением размера массива. Среднее количество обменов значений и полных проходов массива указывают на то, что в данном запуске не встретилось уже отсортированных массивов, которые могли бы ускорить работу алгоритма. Также было подтверждено, что на больших объемах данных данная сортировка неэффективна.