Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Российский химико-технологический университет имени Д.И. Менделеева»

Факультет цифровых технологий и химического инжиниринга

Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1

ПО КУРСУ

«АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ»

Ведущий преподаватель

Ассистент Крашенинников Р. С.

СТУДЕНТ группы КС-36 Лупинос А. В.

Москва

2025

Задание

В лабораторной работе предлагается изучить способ анализа алгоритма, связанный со временем. Рассмотреть для выбранного алгоритма сортировки наилучшие, наихудшее и среднее время и соотнести его с известным для алгоритма показателем эффективности О-большое.

Допускает реализация задания на любом языке программирования, кроме лиспоподобных. Преподаватель может не знать конкретного языка реализации, поэтому вы должны быть способны объяснить алгоритм и нарисовать его без демонстрации непосредственно вашего кода.

Задание:

1. Реализовать метод сортировки перемешиванием;
2. Реализовать проведения тестирования алгоритма сериями расчетов для измерения параметров времени. За один расчет выполняется следующие операции:
    a. Генерируется массив случайных значений,
    b. Запоминается время начала расчета алгоритма сортировки,
    c. Выполняется алгоритм сортировки:
        i. Во время выполнения измерить количество повторных прохождений по массиву
        ii. Во время выполнения измерить количество выполнения операций обмена значений,
    d. Вычисляется время, затраченное на сортировку: текущее время - время начала,
    e. 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,

f. 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 О большое[1]

ii. Совмещенный график среднего, наилучшего и наихудшего времени выполнения

iii. График среднего количества обменов значений

iv. График повторных обходов массива,

g. По результатам расчетов оформляется отчет по предоставленной форме, в отчете:

i. Приводится описание алгоритма

ii. Приводится описания выполнения задачи (Описание кода и специфических элементов реализации)

iii. Приводятся выводы (Графики и их анализ).

Описание алгоритма

Сортировка перемешиванием, или Шейкерная сортировка, или двунаправленная (англ. Cocktail sort) — разновидность пузырьковой сортировки. Анализируя метод пузырьковой сортировки, можно отметить два обстоятельства.

Во-первых, если при движении по части массива перестановки не происходят, то эта часть массива уже отсортирована и, следовательно, ее можно исключить из рассмотрения.

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

Во-вторых, при движении от конца массива к началу минимальный элемент «всплывает» на первую позицию, а максимальный элемент сдвигается только на одну позицию вправо.

Эти две идеи приводят к следующим модификациям в методе пузырьковой сортировки. Границы рабочей части массива (то есть части массива, где происходит движение) устанавливаются в месте последнего обмена на каждой итерации. Массив просматривается поочередно справа налево и слева направо.

Лучший случай для сортировки перемешиванием — отсортированный массив O(n), худший — отсортированный в обратном порядке $O(n^2)$. Усредненным случаем также будет являться $O(n^2)$.

То есть, подытожив, сортировка перемешиванием является измененной версией сортировки пузырьком, в которой также руководствуются идеей постоянного обмена местами двух элементов, только в этот раз не просто с проходом по массиву от начала в сторону конца, смещая все большие элементы к концу, но еще и добавлением обратного хода, смещая малые элементы к началу.

Описание выполнения задачи

Для выполнения алгоритма сортировки перемешиванием была реализована программа на языке программирования C++:

```
#include <iostream>
#include <random>
#include <vector>
#include <chrono>
#include <limits>

using namespace std;

struct SortStats
{
   long long swap_count;
   int full_passes;
};

SortStats cocktailSort(vector<double> &arr, int size)
{
   int left_border = 0;
```

```
    int right_border = size - 1;
    bool flag;

    long long swap_count = 0;
    int full_passes = 0;

    while (left_border <= right_border)
    {
        flag = false;
        ++full_passes;
        for (int i = right_border; i > left_border; --i)
        {
            if (arr[i - 1] > arr[i])
            {
                swap(arr[i - 1], arr[i]);
                flag = true;
                ++swap_count;
            }
        }
        ++left_border;

        for (int i = left_border; i < right_border; ++i)
        {
            if (arr[i] > arr[i + 1])
            {
                swap(arr[i], arr[i + 1]);
                flag = true;
                ++swap_count;
            }
        }
        --right_border;

        if (!flag) break;
    }

    return {swap_count, full_passes};
}

int main() {
    int sizes[8] = {1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000};
    vector<vector<double>> sec_times(8);
    vector<vector<long long>> swap_counts(8);
    vector<vector<int>> full_passes(8);

    vector<double> best_time(8, numeric_limits<double>::max());
    vector<double> worst_time(8, numeric_limits<double>::lowest());
    vector<double> avg_time(8, 0);
    vector<double> avg_swaps(8, 0);
    vector<double> avg_passes(8, 0);

    random_device rd;
    mt19937 gen(rd());
    uniform_real_distribution<> distribution(-1, 1);
```

```
for (int s = 0; s < 8; ++s)
{
   for (int k = 0; k < 20; ++k)
   {
      int M = sizes[s];
      vector<double> arr(M);
      for (auto& element: arr)
      {
         element = distribution(gen);
      }

      chrono::high_resolution_clock::time_point start = chrono::high_resolution_clock::now();
      SortStats stats = cocktailSort(arr, M);
      chrono::high_resolution_clock::time_point end = chrono::high_resolution_clock::now();

      chrono::duration<double> sec_diff = end - start;
      double time_s = sec_diff.count();
      sec_times[s].push_back(sec_diff.count());
      swap_counts[s].push_back(stats.swap_count);
      full_passes[s].push_back(stats.full_passes);

      best_time[s] = min(best_time[s], time_s);
      worst_time[s] = max(worst_time[s], time_s);
      avg_time[s] += time_s;
      avg_swaps[s] += (double)stats.swap_count;
      avg_passes[s] += stats.full_passes;
      cout << "END OF " << k + 1 << " TRY." << endl;
   }
   avg_time[s] /= 20.0;
   avg_swaps[s] /= 20.0;
   avg_passes[s] /= 20.0;
   cout << "==END OF " << sizes[s] << " SIZE OF ARRAY==" << endl;
}

cout << "\n=== Sorting Times (Seconds) ===" << endl;
for (int s = 0; s < 8; ++s)
{
   cout << "Size " << sizes[s] << ": ";
   for (double time : sec_times[s])
   {
      cout << time << " s, ";
   }
   cout << endl;
}

cout << "\n=== Swap Counts ===" << endl;
for (int s = 0; s < 8; ++s)
{
   cout << "Size " << sizes[s] << ": ";
   for (auto swaps : swap_counts[s])
   {
      cout << swaps << ", ";
```

```
        }
        cout << endl;
    }

    cout << "\n=== Full Passes ===" << endl;
    for (int s = 0; s < 8; ++s)
    {
        cout << "Size " << sizes[s] << ": ";
        for (int passes : full_passes[s])
        {
            cout << passes << ", ";
        }
        cout << endl;
    }

    cout << "\n=== Best Times (Seconds) ===" << endl;
    for (int s = 0; s < 8; ++s)
    {
        cout << sizes[s] << "," << best_time[s] << endl;
    }

    cout << "\n=== Worst Times (Seconds) ===" << endl;
    for (int s = 0; s < 8; ++s)
    {
        cout << sizes[s] << "," << worst_time[s] << endl;
    }

    cout << "\n=== Average Times (Seconds) ===" << endl;
    for (int s = 0; s < 8; ++s)
    {
        cout << sizes[s] << "," << avg_time[s] << endl;
    }

    cout << "\n=== Average Swap Counts ===" << endl;
    for (int s = 0; s < 8; ++s)
    {
        cout << sizes[s] << "," << avg_swaps[s] << endl;
    }

    cout << "\n=== Average Full Passes ===" << endl;
    for (int s = 0; s < 8; ++s)
    {
        cout << sizes[s] << "," << avg_passes[s] << endl;
    }

    return 0;
}
```

Данная программа реализует алгоритм шейкерной сортировки (cocktail sort) и тестирует его производительность на массивах различных размеров. Алгоритм

многократно выполняется для каждого размера массива, измеряются время работы, количество перестановок элементов и количество полных проходов по массиву.

Структура кода

- Подключаемые библиотеки:
  - <iostream> — используется для вывода результатов работы программы.
  - <random> — необходима для генерации случайных чисел.
  - <vector> — используется для хранения массивов чисел и результатов тестирования.
  - <chrono> — применяется для измерения времени выполнения сортировки.
  - <limits> — используется для задания начальных значений минимального и максимального времени.
- Структура SortStats:
  - swap_count — хранит количество перестановок элементов в ходе сортировки.
  - full_passes — хранит количество полных проходов по массиву.
- Функция cocktailSort:
  - Реализует алгоритм шейкерной сортировки (вариацию пузырьковой сортировки с двухсторонним проходом).
  - Использует два индекса (left_border и right_border), ограничивающих зону, в которой выполняются перестановки.
  - Проход сначала слева направо, затем справа налево.
  - Если за проход не произошло перестановок, алгоритм завершает работу.
  - Возвращает структуру SortStats с информацией о количестве перестановок и полных проходов.
- Функция main:

    - Тестирует алгоритм на массивах 8 различных размеров: от 1000 до 128000 элементов.
    - Для каждого размера выполняется 20 запусков, фиксируются время работы, количество перестановок и количество полных проходов.
    - Используется генератор случайных чисел (std::mt19937 и std::uniform_real_distribution) для заполнения массивов случайными значениями от -1 до 1.
    - Время сортировки измеряется с помощью std::chrono::high_resolution_clock.
- По завершении тестов выводятся:
    - Все измеренные времена работы,
    - Количество перестановок,
    - Количество полных проходов,
    - Лучшее, худшее и среднее время выполнения,
    - Среднее количество перестановок и проходов.

Специфический элемент реализации - двусторонний проход в cocktailSort. В отличие от классической пузырьковой сортировки, движение происходит в обе стороны, что ускоряет сортировку. Сначала элементы "всплывают" в конец массива, затем "опускаются" в начало.

Выводы

В ходе выполнения лабораторной работы была реализована сортировка перемешиванием (шейкерная) 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 наглядного представления были построены четыре диаграммы на основе полученных значений в ходе одного тестового запуска программы (табл. 1):

1. Совмещенный график наихудшего времени выполнения сортировки и сложности алгоритма указанной в нотации О большое (методом подбора было

найдено значение константы, при которой линия функции лежит не ниже линии полученного времени: 1.38E-8) (рис. 1);

Size of Array	Average Swaps	Average Passes	Best Time	Average Time	Worst Time	O = 1.38E-8 * x^2
1000	251 741	255,7	0,0063338	0,00802309	0,0096153	0,013771
2000	1 001 920	508,8	0,024717	0,0315762	0,044403	0,05508
4000	4 008 850	1 008,8	0,107111	0,134631	0,220318	0,22033
8000	16 024 000	2 018,5	0,332252	0,478214	0,597188	0,88134
16000	63 972 500	4 015,4	2,26652	2,4972	3,2649	3,52536
32000	255 918 000	8 016,3	5,02189	5,97947	7,07789	14,10144
64000	1 022 340 000	15 986,5	21,319	25,6045	28,7167	56,40574
128000	4 097 920 000	32 062,4	95,0342	102,243	109,587	225,62297

Таблица 1 - Полученные результаты за один запуск программы

2. График среднего, наилучшего и наихудшего времени выполнения (рис. 2);
3. График среднего количества обменов значений (рис. 3);
4. График повторных обходов массива (рис. 4).

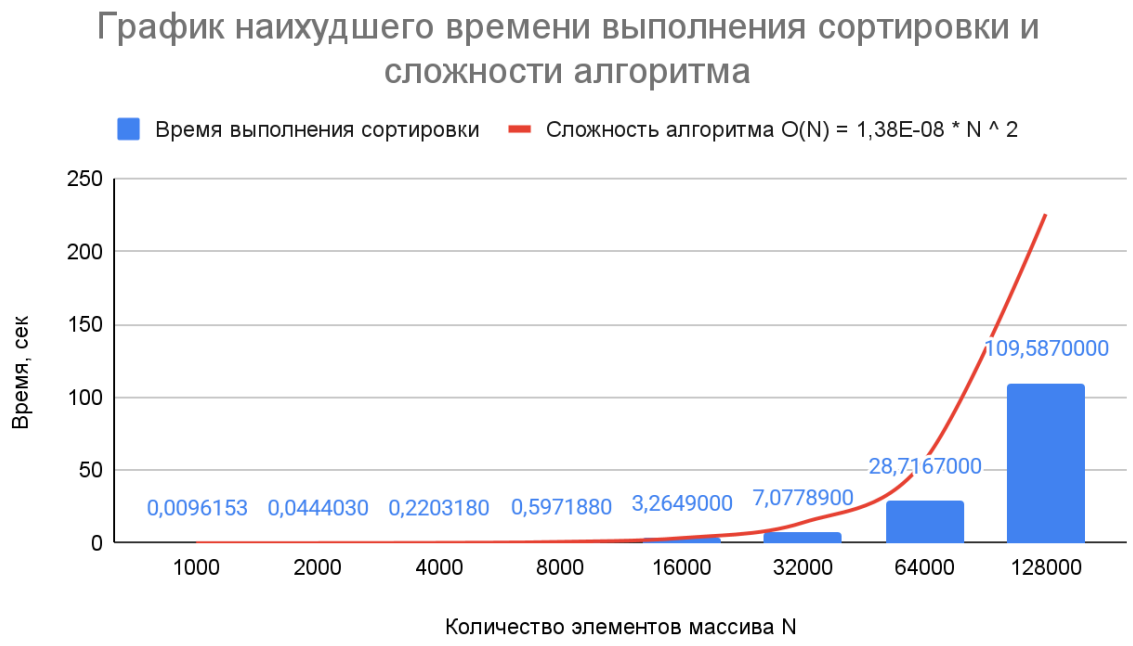

Рисунок 1 - График наихудшего времени выполнения сортировки и сложности алгоритма (п. 1)

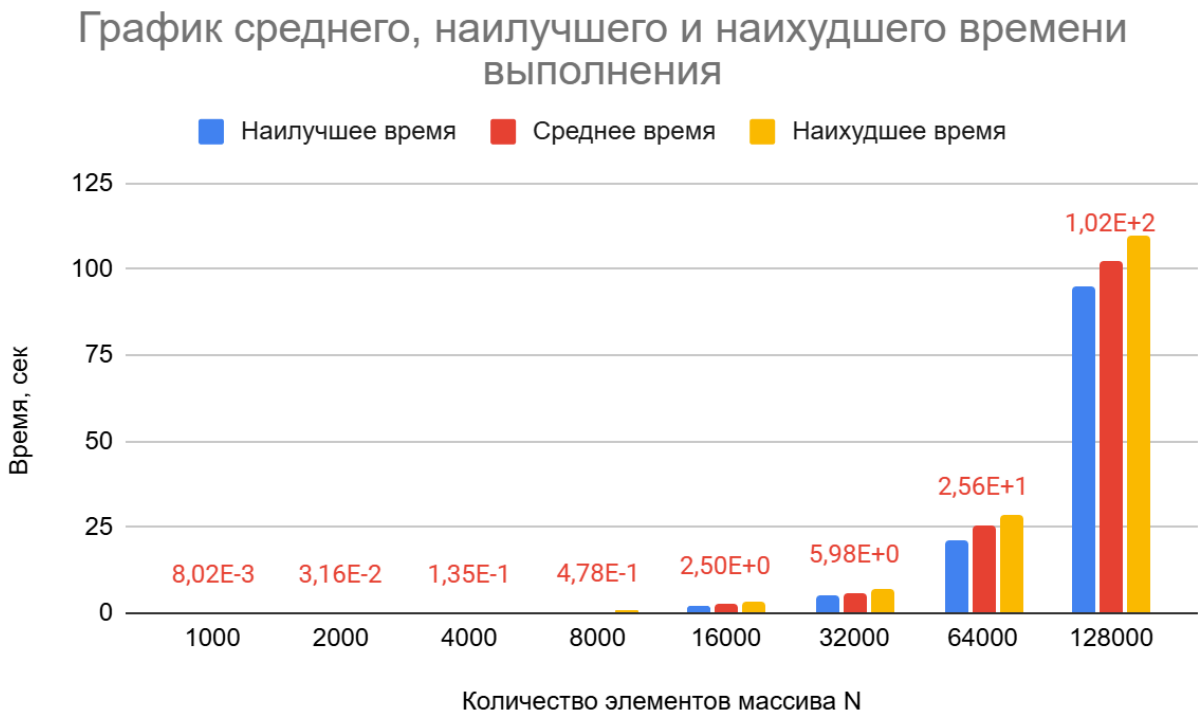

Рисунок 2 - График среднего, наилучшего и наихудшего времени выполнения (п. 2)

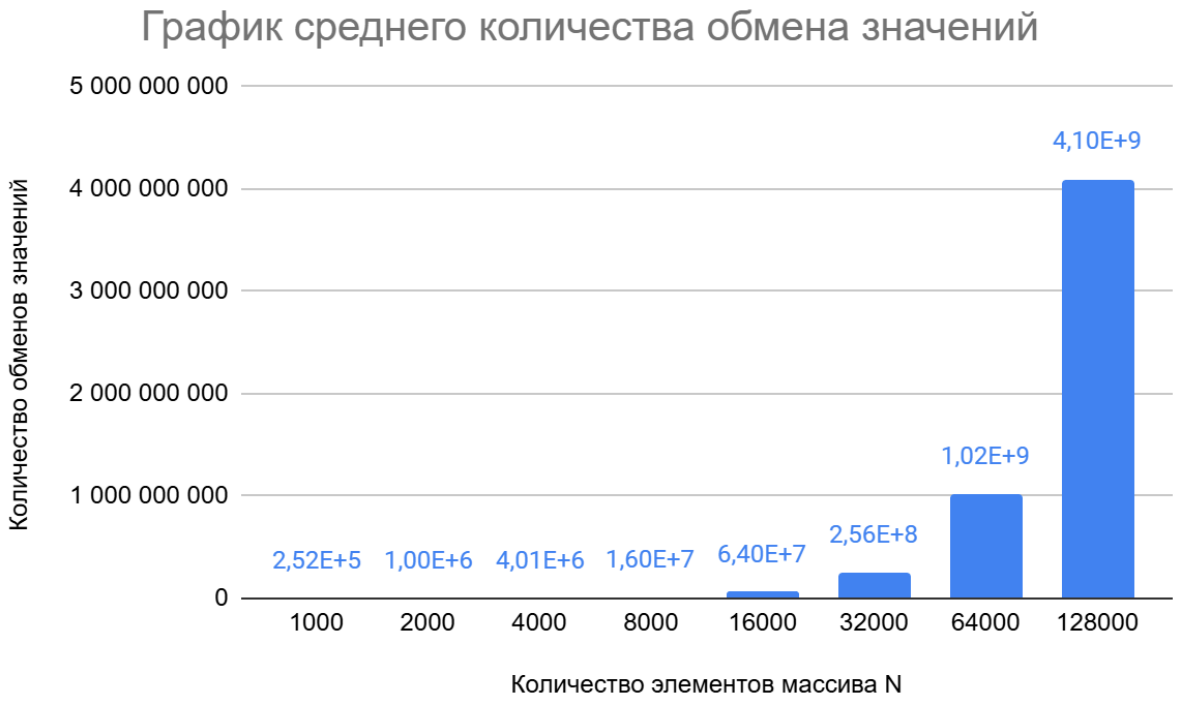

Рисунок 3 - График среднего количества обменов значений (п. 3)

Рисунок 4 - График повторных обходов массива (п. 4)

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.

Таким образом, шейкерная сортировка демонстрирует улучшение по сравнению с классической пузырьковой сортировкой за счет двунаправленного прохода. Однако при увеличении размера массива её эффективность снижается, поэтому для работы с большими объемами данных целесообразно использовать более быстрые алгоритмы сортировки.