Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №4 по курсу "Анализ алгоритмов"

Тема Параллельное программирование

Студент Котляров Н.А.

Группа ИУ7-51Б

Оценка (баллы)

Преподаватель Волкова Л.Л.

Москва — 2022 г.

Оглавление

Введение

В настоящее время компьютерные системы оперируют большими объемами данных. Над этими данными проводится большой объем различного рода вычислений. Для того, чтобы они выполнянлись быстрее, было придумано параллельное программирование.

Его суть заключается в том, чтобы относительно равномерно разделять нагрузку между потоками ядра. Каждое из ядер процессора может обрабатывать по одному потоку, поэтому когда количество потоков на ядро становится больше, происходит квантование времени. Это означает, что на каждый процесс выделяется фиксированная величина времени (квант), после чего в течение кванта обрабатывается следующий процесс. Таким образом создается видимость параллельности. Тем не менее, данная оптимизация может сильно ускорить вычисления.

Целью данной лабораторной работы является получение навыков параллельного программирования. Для достижения поставленной цели необходимо выполнить следующие задачи:

- изучить последовательный и параллельный варианты выбранного алгоритма;
- составить схемы данных алгоритмов;
- реализовать разработанные версии алгоритма;
- провести сравнительный анализ реализаций по затрачиваемым ресурсам (время и память);
- описать и обосновать полученные результаты.

1 Аналитическая часть

В этом разделе будут представлены описания алгоритмов умножения разреженных матриц, последовательного и параллельного.

1.1 Последовательный алгоритм умножения разреженных матриц

Разреженная матрица [1] — в численном анализе и научных вычислениях, разреженная матрица или разреженный массив представляет собой матрицу, в которой большинство элементов равны нулю. Не существует строгого определения того, сколько элементов должно быть нулевым, чтобы матрица считалась разреженной, но общий критерий состоит в том, что количество ненулевых элементов примерно равно количеству строк или столбцов. Напротив, если большинство элементов отличны от нуля, матрица считается плотной. Количество элементов с нулевым знаком, деленное на общее количество элементов, называют разреженностью матрицы.

При хранении разреженных матриц и манипулировании ими на компьютере необходимо использовать специализированные алгоритмы и структуры данных, которые используют разреженную структуру матрицы. Специализированные компьютеры были созданы для разреженных матриц, поскольку они распространены в области машинного обучения. Операции с использованием стандартных структур и алгоритмов с плотной матрицей медленны и неэффективны при применении к большим разреженным матрицам, поскольку обработка и память тратятся на нули. Разреженные данные по своей природе легче сжимать и, следовательно, требуют значительно меньше памяти. Некоторыми очень большими разреженными матрицами невозможно манипулировать с помощью стандартных алгоритмов плотных матриц.

Последовательный алгоритм умножения двух разреженных матриц является набором определенных шагов.

1. Создать матрицу результата.

2. Первый цикл проходит по строкам матриц.

3. Второй цикл идет до тех пор, пока в массиве значений левой матрицы не закончится рассматриваемая строка. В этом цикле выполняются следующие действия:

   (a) Пройти по всем элементам правой матрицы и умножить их на элементы, рассматриваемой на данном шаге строки. Прибавить результат умножения к элементу массива промежуточных значений, номер элемент а определяем по значению столбца правой матрицы, в котором находится, рассматриваемый элемент.

   (b) После умножения всех значений правой матрицы сохранить значение массива в вектор значений матрицы результата. А так же обновить вектор столбцов для каждого элемента.

   (c) Добавить размер вектора столбцов в вектор строк матрицы результата.

4. После завершения первого цикла в матрице результата находится результат умножения переданных двух матриц. Вернуть из метода матрицу ответа.

1.2 Параллельный алгоритм умножения разреженных матриц

Параллельный алгоритм отличается от последовательного возвращаемым результатом. Из-за необходимости независимости работы потоков, результатом умножения двух разреженных матриц будет обычная матрица, так как в последовательном алгоритме используется добавление в вектор строк матрицы результата, в следствие чего в параллельном алгоритме произойдет путаница. В параллельном алгоритме внешний цикл разбивается на несколько циклов, количество которых равно количеству потоков. Каждый поток выполняет свой цикл.

Вывод

В данном разделе были рассмотрены принципы работы последовательного и параллельного алгоритмов умножения разреженных матриц.

2 Конструкторская часть

В этом разделе будут приведены схемы алгоритмов и описаны используемые типы данных.

2.1 Разработка алгоритмов

На рисунках 2.1 и 2.2 представлены схемы рассматриваемых алгоритмов.

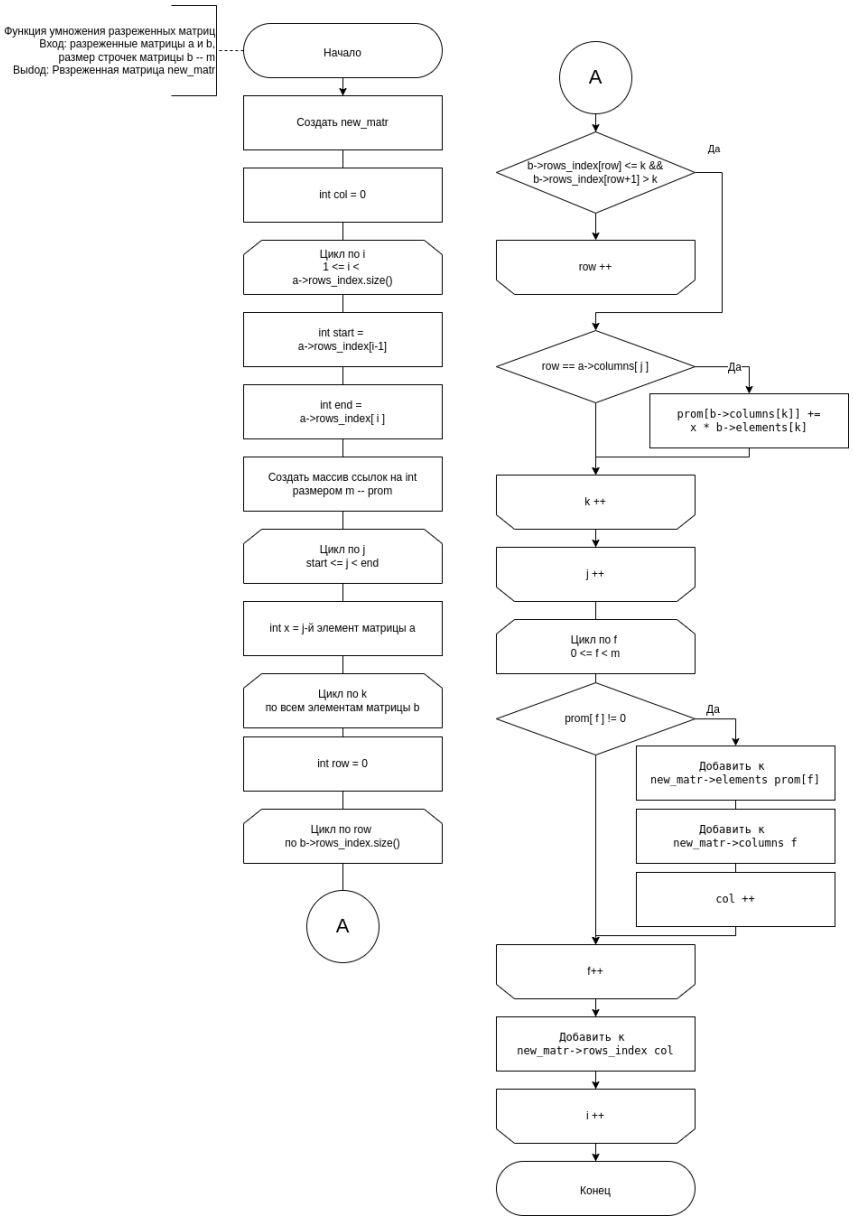

Рисунок 2.1 – Схема последовательного алгоритма умножения разреженных матриц

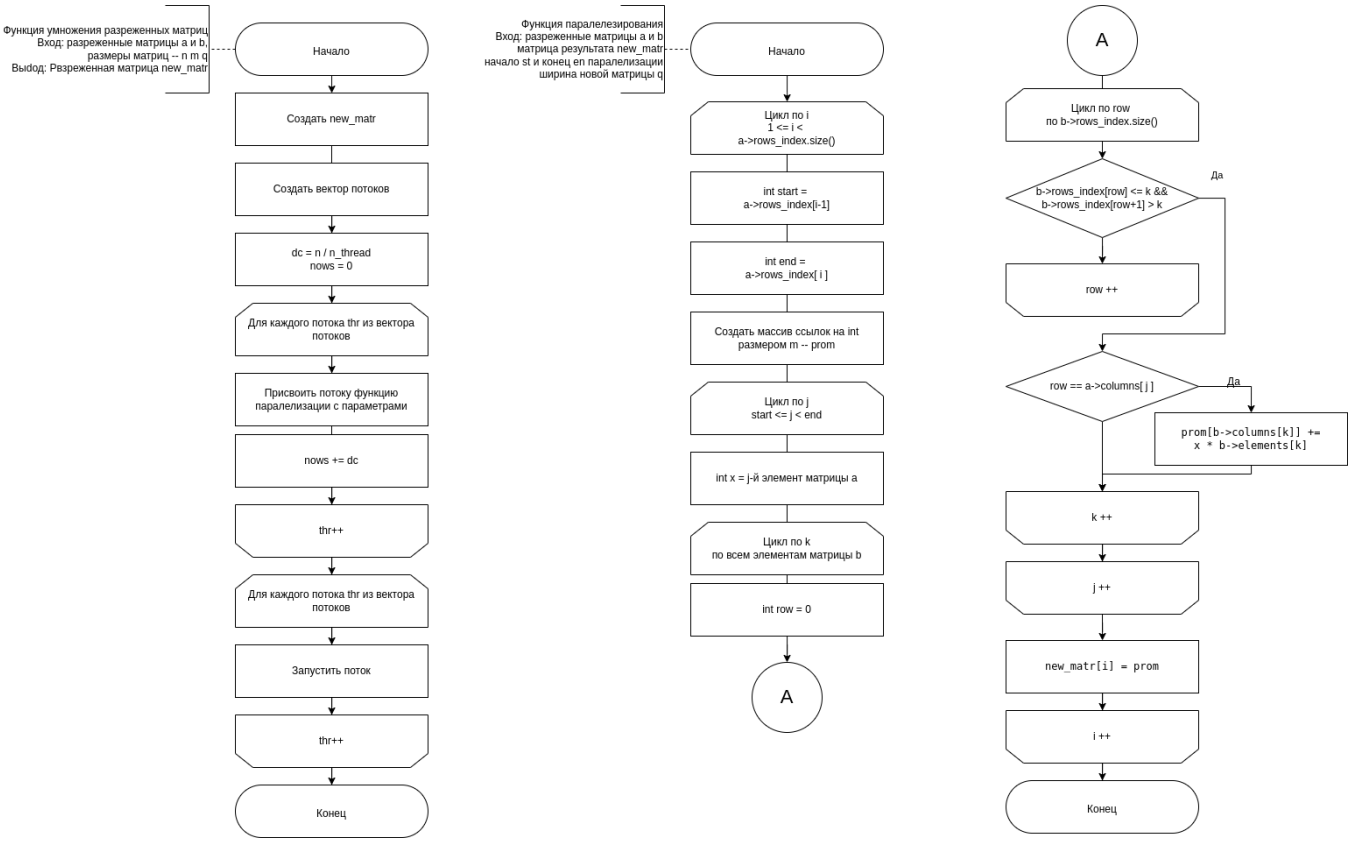

Рисунок 2.2 – Схема параллельного алгоритма умножения разреженных матриц

Вывод

Были разработаны схемы последовательного и параллельного алгоритмов умножения разреженных матриц.

3 Технологическая часть

В данном разделе будут приведены требования к программному обеспечению, средства реализации и листинги кода.

3.1 Требования к ПО

К программе предъявляется ряд требований:

- на вход подаётся массив сравнимых элементов (целые числа);
- на выходе — тот же массив, но в отсортированном порядке.

3.2 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык программирования Си++ [4].

В данном языке есть все требующиеся для данной лабораторной инструменты разработки.

Процессорное время работы реализаций алгоритмов было замерено с помощью функции chrono::high_resolution_clock::now(); из библиотеки <chrono> [5].

3.3 Сведения о модулях программы

Программа состоит из трех модулей:

1. main.cpp - главный файл программы, в котором располагается меню;
2. matr.cpp - файл программы, в котором располагаются все варианты умножения;
3. iog.cpp - файл с вводом, выводом и генерацией матриц.

3.4 Реализация алгоритмов

В листингах 3.1, 3.2, **??** представлены реализации алгоритмов умножения разреженных матриц (последовательный и параллельный).

Листинг 3.1 – Реализация последовательного алгоритма умножения разреженных матриц

```
1  matr *mutl(matr *a, matr *b, const int m)
2  {
3      matr *new_matr = new(matr); // 1
4      new_matr->rows_index.push_back(0);
5      int col = 0;
6      for (int i = 1; i < int(a->rows_index.size()); i++) // 2
7      {
8          int start = a->rows_index[i-1];
9          int end = a->rows_index[i];
10         int *prom = (int *)calloc(m, sizeof(int));
11         for (int j = start; j < end; j++) { // 3
12             int x = a->elements[j];
13             for (int k = 0; k < int(b->elements.size()); k++) {
14                 int row;
15                 for (row = 0; row < int(b->rows_index.size() -
                       1); ++row) {
16                   if (b->rows_index[row] <= k &&
                         b->rows_index[row + 1] > k)
17                   break;
18                 }
19                 if (row == a->columns[j])
20                 prom[b->columns[k]] += x * b->elements[k]; // 3.a
21             }
22         }
23         for (int f = 0; f < m; f++) // 3.b
24         if (prom[f]) {
25             new_matr->elements.push_back(prom[f]);
26             new_matr->columns.push_back(f);
27             col++;
28         }
29         new_matr->rows_index.push_back(col); // 3.c
30         free(prom);
31     }
```

```
32      return new_matr; // 4
33  }
```

Листинг 3.2 – Реализация паралельного алгоритма умножения разреженных матриц

```
1  void parallel_func(int **res, int st, int en, matr *a, matr *b,
       const int q)
2  {
3      for (int i = st; i < en; i++) // 2
4      {
5          int start = a->rows_index[i-1];
6          int end = a->rows_index[i];
7          int *prom = (int *)calloc(q, sizeof(int));
8          for (int j = start; j < end; j++) { // 3
9              int x = a->elements[j];
10
11             for (int k = 0; k < int(b->elements.size()); k++) {
12                 int row;
13                 for (row = 0; row < int(b->rows_index.size() -
                       1); ++row) {
14                     if (b->rows_index[row] <= k &&
                           b->rows_index[row + 1] > k)
15                     break;
16                 }
17                 if (row == a->columns[j])
18                 prom[b->columns[k]] += x * b->elements[k]; // 3.a
19             }
20         }
21         res[i - 1] = prom;
22     }
23 }
24
25 int **mutl_parallel(matr *a, matr *b, const int n, const int m,
       const int q, int thread_count)
26 {
27     int **new_matr = (int**)calloc(n, sizeof(int*));
28     vector<thread> threads(thread_count);
29     double dc = double(n) / thread_count;
30     int nows = 1;
31     cout << thread_count << '\n';
32     auto start = chrono::high_resolution_clock::now();
```

```
33      for (auto& thr: threads) {
34          thr = thread(parallel_func, new_matr, nows,
                int(round(nows + dc)), a, b, q);
35          nows += dc;
36      }
37      auto stop = chrono::high_resolution_clock::now();
38      auto dur1 = chrono::duration_cast<chrono::microseconds>((stop
          - start));
39      cout << "Zapusk:␣" << dur1.count() << '\n';
40
41      auto start2 = chrono::high_resolution_clock::now();
42      for (auto& thr: threads)
43      thr.join();
44      auto stop2 = chrono::high_resolution_clock::now();
45      auto dur2 =
            chrono::duration_cast<chrono::microseconds>((stop2 -
            start2));
46      cout << "Simhro:␣" << dur2.count() << '\n';
47      return new_matr; // 4
48  }
```

3.5 Функциональные тесты

В таблице ?? приведены тесты для функций, реализующих алгоритмы сортировки. Тесты пройдены успешно.

Таблица 3.1 – Тестовые случаи

N	Матрица 1	Матрица 2	Результат
1	$\begin{pmatrix} 1 & 1 & 1 \\ 5 & 5 & 5 \\ 2 & 2 & 2 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 3 \\ 15 \\ 6 \end{pmatrix}$
2	$\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$
3	$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$
4	$\begin{pmatrix} 2 \end{pmatrix}$	$\begin{pmatrix} 2 \end{pmatrix}$	$\begin{pmatrix} 4 \end{pmatrix}$
5	$\begin{pmatrix} 1 & -2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} -1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 0 & 4 & 6 \\ 4 & 12 & 18 \\ 4 & 12 & 18 \end{pmatrix}$
6	$\begin{pmatrix} 1 & 2 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \end{pmatrix}$	Некорректный размер

Вывод

Были реализованы алгоритмы умножения разреженных матриц: последовательный и параллельный.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программ, постановка эксперимента и сравнительный анализ алгоритмов на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- операционная система: Manjaro xfce [6] Linux [7] x86_64;
- память: 8 Гб;
- мобильный процессор AMD Ryzen™ 7 3700U @ 2.3 ГГц [8].

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно системой тестирования.

4.2 Демонстрация работы программы

На рисунке 4.1 представлен результат работы программы.

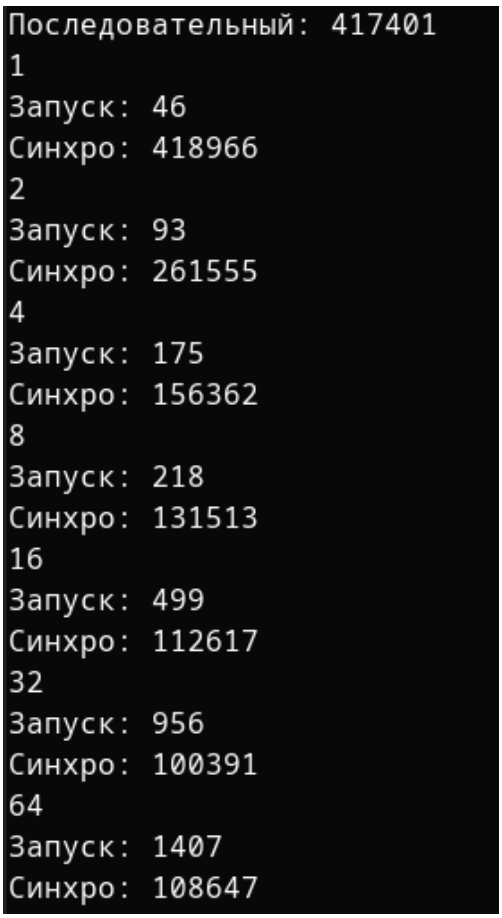

Рисунок 4.1 – Пример работы программы

4.3 Время выполнения реализации алгоритмов

Время работы реализации алгоритмов измерялось при помощи функции chrono::high_resolution_clock::now() из библиотеки <chrono> языка C++.

Сравнение работы последовательного и параллельного алгоритмов проводилось при 16 потоках.

Таблица 4.1 – Результаты замеров реализаций алгоритмов умножения разреженных матриц

Размер	Последовательный	Параллельный
400	245643	76332
500	197758	58654
600		
700		
800		
900		
1000		

Таблица 4.2 – Время выполнения реализации параллельного алгоритма умножения разреженных матриц (мкс)

Количество потоков	Время
1	419012
2	261648
4	156537
8	131731
16	113116
32	116371
64	131898

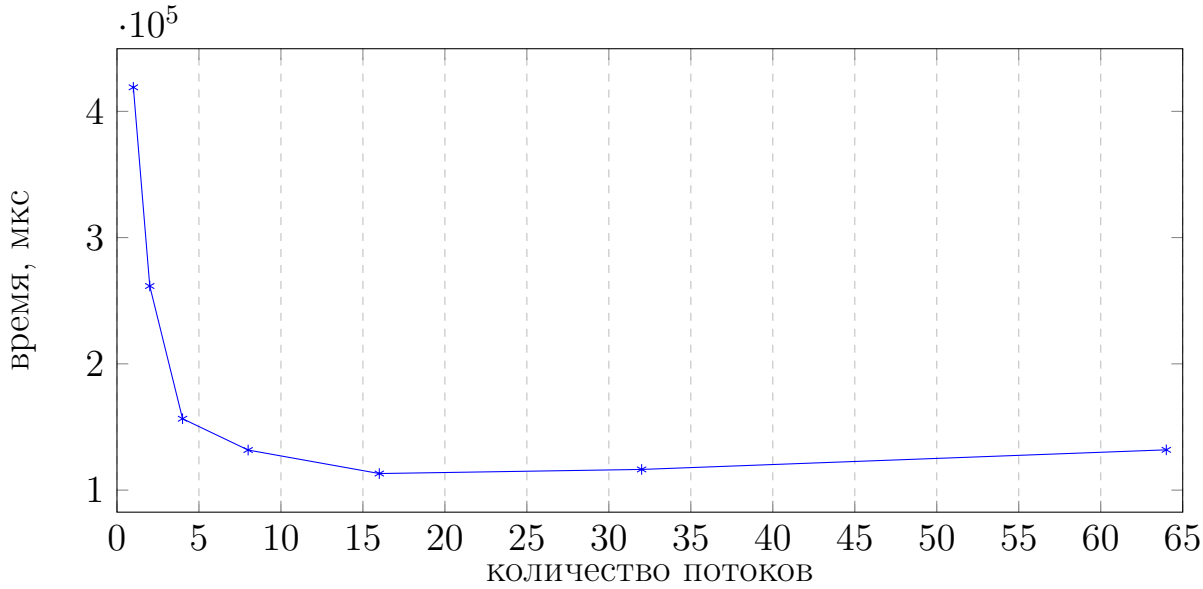

Рисунок 4.2 – Сравнение времени работы параллельного алгоритма при разном количестве потоков

Вывод

Реализация алгоритма сортировки подсчетом работает быстрее остальных двух во всех трех случаях, что произошло благодаря небольшому диапазону значений генерируемых массивов — не более 1000. Это привело к линейному характеру зависимости трудоемкости данного алгоритма от размера массива и линейному от мощность диапазона значений в массиве.

Заключение

В ходе выполнения лабораторной работы была достигнута цель работы: были разработаны алгоритмы сортировки массивов.

Все поставленные задачи решены:

- изучены и реализованы 3 алгоритма сортировки: перемешиванием, посчетом, быстрой;
- проведен сравнительный анализ трудоёмкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;
- проведен сравнительный анализ алгоритмов на основе экспериментальных данных;
- подготовлен отчет о лабораторной работе.

Список использованных источников

1. Шейкерная сортировка (перемешиванием) [Электронный ресурс]. Режим доступа: `https://kvodo.ru/shaker-sort.html` (дата обращения: 18.10.2022).

2. Сортировка вставками [Электронный ресурс]. Режим доступа: `https://kvodo.ru/quicksort.html` (дата обращения: 18.010.2022).

3. Сортировка выбором [Электронный ресурс]. Режим доступа: `https://kvodo.ru/sortirovka-vyiborom-2.html` (дата обращения: 12.09.2021).

4. Welcome to Python [Электронный ресурс]. Режим доступа: `https://www.python.org` (дата обращения: 04.09.2021).

5. time — Time access and conversions [Электронный ресурс]. Режим доступа: `https://docs.python.org/3/library/time.html#functions` (дата обращения: 04.09.2021).

6. Manjaro [Электронный ресурс]. Режим доступа: `https://manjaro.org/` (дата обращения: 03.10.2022).

7. Linux – Википедия [Электронный ресурс]. Режим доступа: `https://ru.wikipedia.org/wiki/Linux` (дата обращения: 04.10.2022).

8. Мобильный процессор AMD Ryzen™ 7 3700U [Электронный ресурс]. Режим доступа: `https://www.amd.com/ru/products/apu/amd-ryzen-7-3700u` (дата обращения: 04.10.2022).