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 **ПРОГРАМНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.03 ПРОГРАММНАЯ ИНЖЕНЕРИЯ**

О Т Ч Е Т

По лабораторной работе № 5

Название: Реализация конвейера с использованием параллельных вычислений

Дисциплина: Анализ Алгоритмов

Студент ИУ7-52Б _____ _____ Н.А. Гарасев
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ Л.Л. Волкова
(Подпись, дата) (И.О. Фамилия)

Москва, 2020

Оглавление

Введение

Цель лабораторной работы: изучить и применить на практике возможности конвейера с использованием параллельных вычислений на примере работы с массивами.

Одним из ключевых аспектов в современном программировании является **многопоточность**. Ключевым понятием при работе с многоопоточностью является **поток**. Поток представляет некоторую часть кода программы. При выполнении программы каждому потоку выделяется определенный квант времени. И при помощи многопоточности мы можем выделить в приложении несколько потоков, которые будут выполнять различные задачи одновременно. Если у нас, допустим, графическое приложение, которое посылает запрос к какому-нибудь серверу или считывает и обрабатывает огромный файл, то без многопоточности у нас бы блокировался графический интерфейс на время выполнения задачи. А благодаря потокам мы можем выделить отправку запроса или любую другую задачу, которая может долго обрабатываться, в отдельный поток. Поэтому, к примеру, клиент-серверные приложения (и не только они) практически не мыслимы без многопоточности.

В ходе выполнения лабораторной работы требуется решить следующие **задачи**.

1) Реализовать конвейера выполняющие различные операции над массивами.
2) Организовать очередь элементов.
3) Реализовать подачу элементов на конвейер параллельно и последовательно.
4) Сравнить алгоритмы по затраченным ресурсам.

1. Аналитическая часть

Рассмотрим понятия, с которыми мы столкнемся при выполнении лабораторной работы.

1.1. Многопоточность

Поток выполнения — наименьшая единица обработки, исполнение которой может быть назначено ядром операционной системы. Реализация потоков выполнения и процессов в разных операционных системах отличается друг от друга, но в большинстве случаев поток выполнения находится внутри процесса. Несколько потоков выполнения могут существовать в рамках одного и того же процесса и совместно использовать ресурсы, такие как память, тогда как процессы не разделяют этих ресурсов. В частности, потоки выполнения разделяют инструкции процесса (его код) и его контекст (значения переменных, которые они имеют в любой момент времени).

На одном процессоре многопоточность обычно происходит путём временного мультиплексирования (как и в случае многозадачности): процессор переключается между разными потоками выполнения. Это переключение контекста обычно происходит достаточно часто, чтобы пользователь воспринимал выполнение потоков или задач как одновременное. В многопроцессорных и многоядерных системах потоки или задачи могут реально выполняться одновременно, при этом каждый процессор или ядро обрабатывает отдельный поток или задачу.

Потоки возникли в операционных системах как средство распараллеливания вычислений.

Параллельное выполнение нескольких работ в рамках одного интерактивного приложения повышает эффективность работы пользователя. Так, при работе с текстовым редактором желательно иметь возможность совмещать набор нового текста с такими продолжительными по времени операциями, как переформатирование значительной части текста, печать

документа или его сохранение на локальном или удаленном диске. Еще одним примером необходимости распараллеливания является сетевой сервер баз данных. В этом случае параллелизм желателен как для обслуживания различных запросов к базе данных, так и для более быстрого выполнения отдельного запроса за счет одновременного просмотра различных записей базы. Именно для этих целей современные ОС предлагают механизм многопоточной обработки (multithreading). Понятию «поток» соответствует последовательный переход процессора от одной команды программы к другой. ОС распределяет процессорное время между потоками. Процессу ОС назначает адресное пространство и набор ресурсов, которые совместно используются всеми его потоками.

Создание потоков требует от ОС меньших накладных расходов, чем процессов. В отличие от процессов, которые принадлежат разным, вообще говоря, конкурирующим приложениям, все потоки одного процесса всегда принадлежат одному приложению, поэтому ОС изолирует потоки в гораздо меньшей степени, нежели процессы в традиционной мультипрограммной системе. Все потоки одного процесса используют общие файлы, таймеры, устройства, одну и 5 ту же область оперативной памяти, одно и то же адресное пространство. Это означает, что они разделяют одни и те же глобальные переменные. Поскольку каждый поток может иметь доступ к любому виртуальному адресу процесса, один поток может использовать стек другого потока. Между потоками одного процесса нет полной защиты, потому что, во-первых, это невозможно, а во-вторых, не нужно. Чтобы организовать взаимодействие и обмен данными, потокам вовсе не требуется обращаться к ОС, им достаточно использовать общую память — один поток записывает данные, а другой читает их. С другой стороны, потоки разных процессов по-прежнему хорошо защищены друг от друга.

Широко используемый подход состоит и в применении тех или иных библиотек, обеспечивающих определенный программный интерфейс

(application programming interface, API) для разработки параллельных программ. В рамках такого подхода наиболее известны Windows Thread API. Однако первый способ применим только для ОС семейства Microsoft Windows, а второй вариант API является достаточно трудоемким для использования и имеет низкоуровневый характер [1].

2. Конструкторская часть

Существует три конвейера, каждый из которого выполняет свою задачу.

2.1. Схемы алгоритмов

На рис. 1 приведена схема главного по потокам алгоритмов.

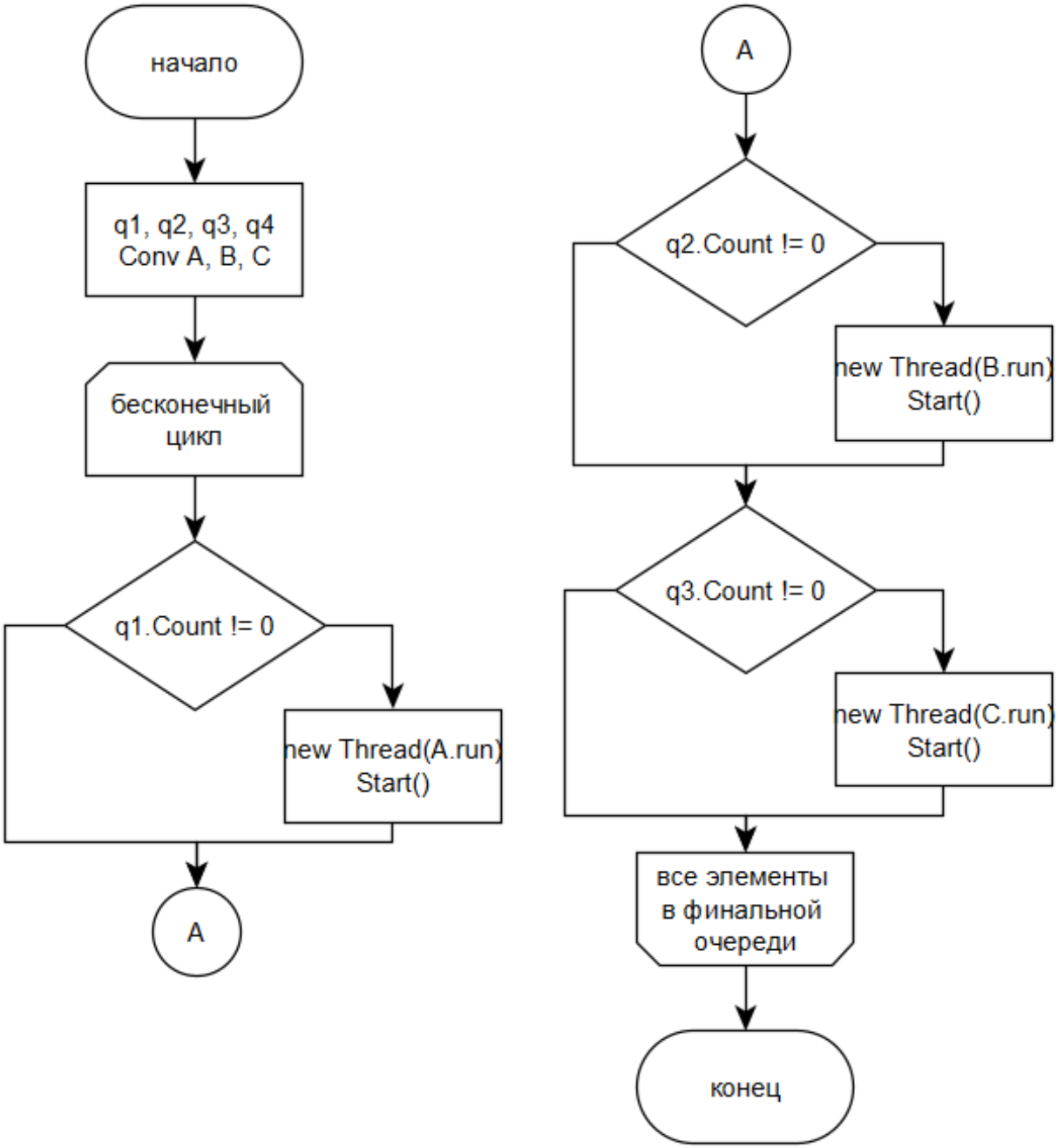

Рисунок 1. *Главный алгоритм вызова конвейеров*

2.2. Распараллеливание конвейеров

В программе представлены три класса конвейера. У каждого конвейера своя задача. Первый считает количество положительных чисел в массиве. Второй подсчитывает количество отрицательных. Третий считает количество нулей и выводит на экран равна ли сумма положительных, отрицательных и нулей количеству чисел в массиве.

3. Технологическая часть

В качестве языка программирования был выбран C#, т.к. на сегодняшний момент язык программирования C# один из самых мощных, быстро развивающихся и востребованных языков в ИТ-отрасли. В настоящий момент на нем пишутся самые различные приложения: от небольших десктопных программок до крупных веб-порталов и веб-сервисов, обслуживающих ежедневно миллионы пользователей. Основной функционал для использования потоков в приложении сосредоточен в пространстве имен **System.Threading.** Так же для лабораторной работы использовалось **System.Diagnostics**.

3.1 Реализация алгоритмов

В листингах 1-3 представлена реализация алгоритмов умножения матриц. В листинге 4 представлена функция для замера времени выполнения заданной функции.

Листинг 1. Реализация алгоритма с потоками

```
static public int MainLoop(int elem_cnt, int elem_len, int repeat)
        {
            Queue<Element> q1 = new Queue<Element>();
            Queue<Element> q2 = new Queue<Element>();
            Queue<Element> q3 = new Queue<Element>();
            Queue<Element> q_finish = new Queue<Element>();

            FillQueue(ref q1, elem_cnt, elem_len);

            ConvA ca = new ConvA(ref q1, ref q2, repeat);
            ConvB cb = new ConvB(ref q2, ref q3, repeat);
            ConvC cc = new ConvC(ref q3, ref q_finish, repeat);

            int conv_cnt = 3;

            Thread[] t = new Thread[conv_cnt];

            while (true)
            {
                if (t[0] is null)
                {
                    t[0] = new Thread(ca.run);
                    t[0].Start();
                }
                else if (!t[0].IsAlive && q1.Count != 0)
                {
                    t[0] = new Thread(ca.run);
```

```
                t[0].Start();
            }

            if (t[1] is null)
            {
                if (q2.Count != 0)
                {
                    t[1] = new Thread(cb.run);
                    t[1].Start();
                }
            }
            else if (!t[1].IsAlive && q2.Count != 0)
            {
                t[1] = new Thread(cb.run);
                t[1].Start();
            }

            if (t[2] is null)
            {
                if (q3.Count != 0)
                {
                    t[2] = new Thread(cc.run);
                    t[2].Start();
                }
            }
            else if (!t[2].IsAlive && q3.Count != 0)
            {
                t[2] = new Thread(cc.run);
                t[2].Start();
            }
            if (q_finish.Count == elem_cnt)
                break;
        }
        return 0;
    }
```

Листинг 2. Реализация алгоритма без потоков

```
static public int SimpleLoop(int elem_cnt, int elem_len, int repeat)
    {
        Queue<Element> q1 = new Queue<Element>();
        Queue<Element> q2 = new Queue<Element>();
        Queue<Element> q3 = new Queue<Element>();
        Queue<Element> q_finish = new Queue<Element>();

        FillQueue(ref q1, elem_cnt, elem_len);

        ConvA ca = new ConvA(ref q1, ref q2, repeat);
        ConvB cb = new ConvB(ref q2, ref q3, repeat);
        ConvC cc = new ConvC(ref q3, ref q_finish, repeat);

        while (true)
        {
            if (q1.Count != 0)
            {
                ca.run();
            }
            else if (q2.Count != 0)
            {
                cb.run();
            }
            else if (q3.Count != 0)
            {
                cc.run();
```

```
                }
                else if (q_finish.Count == elem_cnt)
                    break;
            }

            return 0;
        }
```

Листинг 3. Реализация класс конвейера

```
class ConvA
    {
        public Queue<Element> q_in = new Queue<Element>();
        public Queue<Element> q_out = new Queue<Element>();

        public int repeat;
        public int pass_cnt;

        public ConvA(ref Queue<Element> q_in, ref Queue<Element> q_out, int repeat)
        {
            this.q_in = q_in;
            this.q_out = q_out;

            this.repeat = repeat;
            this.pass_cnt = 0;
        }
        public void run()
        {
            int t1 = DateTime.Now.Minute;
            int t2 = DateTime.Now.Second;
            int t3 = DateTime.Now.Millisecond / 10;
            Element a = q_in.Dequeue();
            this.pass_cnt += 1;
            for (int repeat = 1; repeat <= this.repeat; repeat++)
            {
                a.positive = 0;
                for (int i = 0; i < a.array.Length; i++)
                {
                    if (a.array[i] > 0)
                    {
                        a.positive += 1;
                    }
                }
            }
            Console.WriteLine("[{0:00}:{1:00}.{2:00} - {3:00}:{4:00}.{5:00}] Conv #1:
element #{6:00} count {7} pos number",
                t1, t2, t3,
                DateTime.Now.Minute, DateTime.Now.Second, DateTime.Now.Millisecond / 10,
this.pass_cnt, a.positive);
            q_out.Enqueue(a);
        }
    }
```

Листинг 4. Функция подсчета среднего времени выполнения алгоритма умножения матриц.

```
static public void Time(Func<int, int, int, int> func, int elem_cnt, int elem_len, int
repeat)
        {
            Stopwatch stopWatch = new Stopwatch();
            TimeSpan ts;
            stopWatch.Start();
            func(elem_cnt, elem_len, repeat);
```

```
            stopWatch.Stop();
            ts = stopWatch.Elapsed;
            string elapsedTime = String.Format("{0:00}:{1:00}.{2:00}",
                    ts.Minutes, ts.Seconds,
                    ts.Milliseconds / 10);
            Console.WriteLine("Время: " + elapsedTime);
        }
```

3.2. Характеристика машины

Эксперименты проводились на компьютере со следующими характеристиками:

- ОС – Windows 10, 64bit
- Процессор – Intel(R) Core(TM) i3-8100 CPU @ 3.60GHz
- ОЗУ – 16 Gb

4. Экспериментальная часть

Сравним реализованные алгоритмы по времени.

4.1. Сравнение алгоритмов по времени работы реализаций

Для сравнения алгоритмов в программе необходимо провести замеры процессорного времени. На рисунке 2 представлен вывод алгоритма с потоками. В квадратных скобках указано время начала и конца выполнения операции, далее идет номер конвейера, номер элемента и информация о операции.

```
[51:08.55 - 51:10.38] Conv #1: element #01 count 4558 pos number
[51:10.54 - 51:11.49] Conv #1: element #02 count 4488 pos number
[51:10.54 - 51:11.89] Conv #2: element #01 count 4964 neg number
[51:11.91 - 51:12.38] Conv #3: element #01 count 478 zero number. It's True
[51:11.50 - 51:12.59] Conv #1: element #03 count 4553 pos number
[51:11.94 - 51:13.11] Conv #2: element #02 count 5012 neg number
[51:13.15 - 51:13.64] Conv #3: element #02 count 500 zero number. It's True
[51:12.59 - 51:13.85] Conv #1: element #04 count 4567 pos number
[51:13.19 - 51:14.43] Conv #2: element #03 count 4939 neg number
[51:14.44 - 51:14.95] Conv #3: element #03 count 508 zero number. It's True
[51:13.86 - 51:15.05] Conv #1: element #05 count 4534 pos number
[51:14.45 - 51:15.67] Conv #2: element #04 count 4945 neg number
[51:15.71 - 51:16.22] Conv #3: element #04 count 488 zero number. It's True
[51:15.11 - 51:16.33] Conv #1: element #06 count 4531 pos number
[51:15.77 - 51:16.89] Conv #2: element #05 count 5000 neg number
[51:16.91 - 51:17.31] Conv #3: element #05 count 466 zero number. It's True
[51:16.34 - 51:17.36] Conv #1: element #07 count 4554 pos number
[51:16.90 - 51:17.97] Conv #2: element #06 count 4997 neg number
[51:17.97 - 51:18.47] Conv #3: element #06 count 472 zero number. It's True
[51:17.38 - 51:18.52] Conv #1: element #08 count 4587 pos number
[51:17.97 - 51:19.14] Conv #2: element #07 count 4989 neg number
[51:19.15 - 51:19.60] Conv #3: element #07 count 457 zero number. It's True
[51:18.55 - 51:19.64] Conv #1: element #09 count 4490 pos number
[51:19.16 - 51:20.32] Conv #2: element #08 count 4909 neg number
[51:20.33 - 51:20.75] Conv #3: element #08 count 504 zero number. It's True
[51:19.67 - 51:20.75] Conv #1: element #10 count 4472 pos number
[51:20.33 - 51:21.37] Conv #2: element #09 count 5040 neg number
[51:21.38 - 51:21.74] Conv #3: element #09 count 470 zero number. It's True
[51:21.38 - 51:22.29] Conv #2: element #10 count 4997 neg number
[51:22.30 - 51:22.63] Conv #3: element #10 count 531 zero number. It's True
Время: 00:14.33
```

Рисунок 2. *Пример работы алгоритма с потоками*

На рисунке 3 продемонстрирован результат работы алгоритма с той же задачей, что и предыдущий алгоритм, но в этом алгоритме не используются потоки.

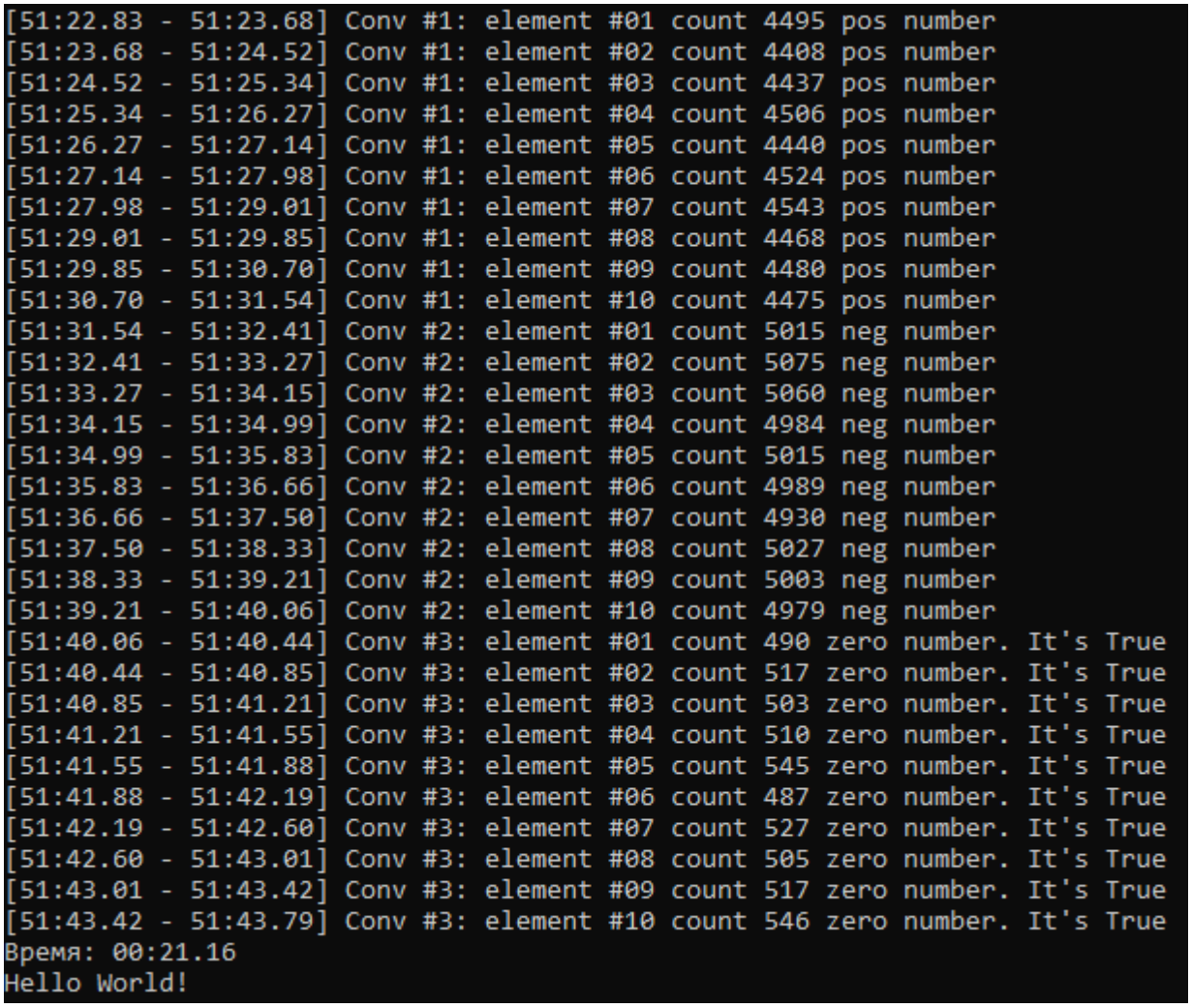

Рисунок 3. *Пример работы алгоритма без потоков*

Время работы параллельных конвейеров на данном примере быстрее на 30%, чем последовательный вызов конвейеров.

Вывод: наглядно видно, что алгоритм с потоками работает быстрее за счет параллельной работы трех конвейеров.

Заключение

В ходе работы были изучены и реализованы конвейера с использованием параллельных вычислений.

Был сделан вывод для использования распараллеливания для алгоритмов, основанный на временных замерах реализованных алгоритмов

Цель работы достигнута. Получены практические навыки реализации алгоритмов распараллеливания, а также проведена исследовательская работа по вычислению и сравнению этих алгоритмов.

Список литературы

1. Дж. Макконнелл. Анализ алгоритмов. Активный обучающий подход. –М.: Техносфера, 2017. – 267 с.

2. Сайт о программировании, C# [Электронный ресурс]. Режим доступа: https://metanit.com/, свободный (дата обращения: 07.12.20).