Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное 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»

Лабораторная работа № 2
по дисциплине «Анализ алгоритмов»

Тема Сравнительный анализ рекурсивного и нерекурсивного алгоритмов

Студент Жаринов М. А.

Группа ИУ7-52Б

Преподаватели Волкова Л. Л., Строганов Д. В.

Москва, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель данной лабораторной работы — сравнительный анализ рекурсивного и нерекурсивного алгоритмов. Для достижения данной цели необходимо выполнить задачи:

1) разработать рекурсивный и нерекурсивный алгоритмы вычисления среднего значения элементов последовательности, оканчивающейся нулём;

2) описать средства разработки и инструменты замера процессорного времени выполнения реализации алгоритмов;

3) реализовать разработанные алгоритмы;

4) выполнить тестирование реализации алгоритмов;

5) выполнить теоретическую оценку затрачиваемой реализацией каждого алгоритма памяти (для рекурсивного алгоритма — на материале анализа высоты дерева рекурсивных вызовов и оценки затрачиваемой на один вызов функции памяти);

6) выполнить замеры процессорного времени выполнения реализации алгоритмов в зависимости от варьируемого входа (одна точка на графике получается делением времени выполнения k идентичных расчётов на k, $k >= 100$);

7) оценить трудоёмкость двух алгоритмов или их реализаций в худшем случае (если случаев несколько);

8) сравнить результаты замеров процессорного времени и оценки трудоёмкости;

9) сделать выводы из сравнительного анализа реализации рекурсивного и нерекурсивного алгоритмов решения одной и той же задачи по критериям ёмкостной эффективности (на материале теоретической оценки) и пиковой временной эффективности (на материале результатов измерений).

1 Аналитическая часть

Рекурсия — способ описания функций или процессов через самих себя. Рекурсивная функция — подпрограмма, которая обращение к самой себе [1]. Хвостовая рекурсия — рекурсия, при которой рекурсивный вызов является последней операцией перед возвратом из функции. Хвостовая рекурсия позволяет компилятору выполнить оптимизацию хвостового вызова. Вместо добавления нового кадра стека для каждого рекурсивного вызова, компилятор может переиспользовать текущий кадр, эффективно превращая рекурсию в цикл.

Среднее арифметическое значение элементов последовательности чисел $x_1, x_2, \ldots, x_n$ определяется как сумма всех элементов последовательности, делённая на их количество и выражается формулой (1.1).

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i, \tag{1.1}$$

где:

1) $\bar{x}$ — это среднее арифметическое значение элементов последовательности;
2) n — число элементов в последовательности;
3) x_i — i-й элемент последовательности.

Вывод

В аналитической части были рассмотрены определения рекурсии, рекурсивной функции, хвостовой рекурсии и среднего арифметического значения элементов последовательности чисел.

2 Конструкторская часть

2.1 Требования к программе

К разрабатываемому программному обеспечению предъявляется следующий набор функциональных требований:

1) программное обеспечение должно предоставлять пользователю текстовый интерфейс для выбора режима работы;

2) необходимо реализовать поддержку двух основных режимов:

— вычисление среднего значения всех элементов последовательности, заканчивающейся нулём, с ручным вводом данных;

— массовое тестирование для замера производительности на последовательностях различных размеров.

3) в ручном режиме программа должна запрашивать элементы последовательности до первого нуля и выполнять валидацию вводимых данных;

4) в режиме массового тестирования должны выполняться замеры процессорного времени для каждого из реализованных алгоритмов. Результаты замеров следует выводить в наглядном табличном формате.

2.2 Разработка алгоритмов

Визуальное представление логики работы алгоритмов представлено в виде блок-схем. На рисунке 2.1 показана схема итеративного алгоритма вычисления среднего значения элементов последовательности. Схема рекурсивного алгоритма вычисления среднего значения элементов последовательности показана на рисунке 2.2.

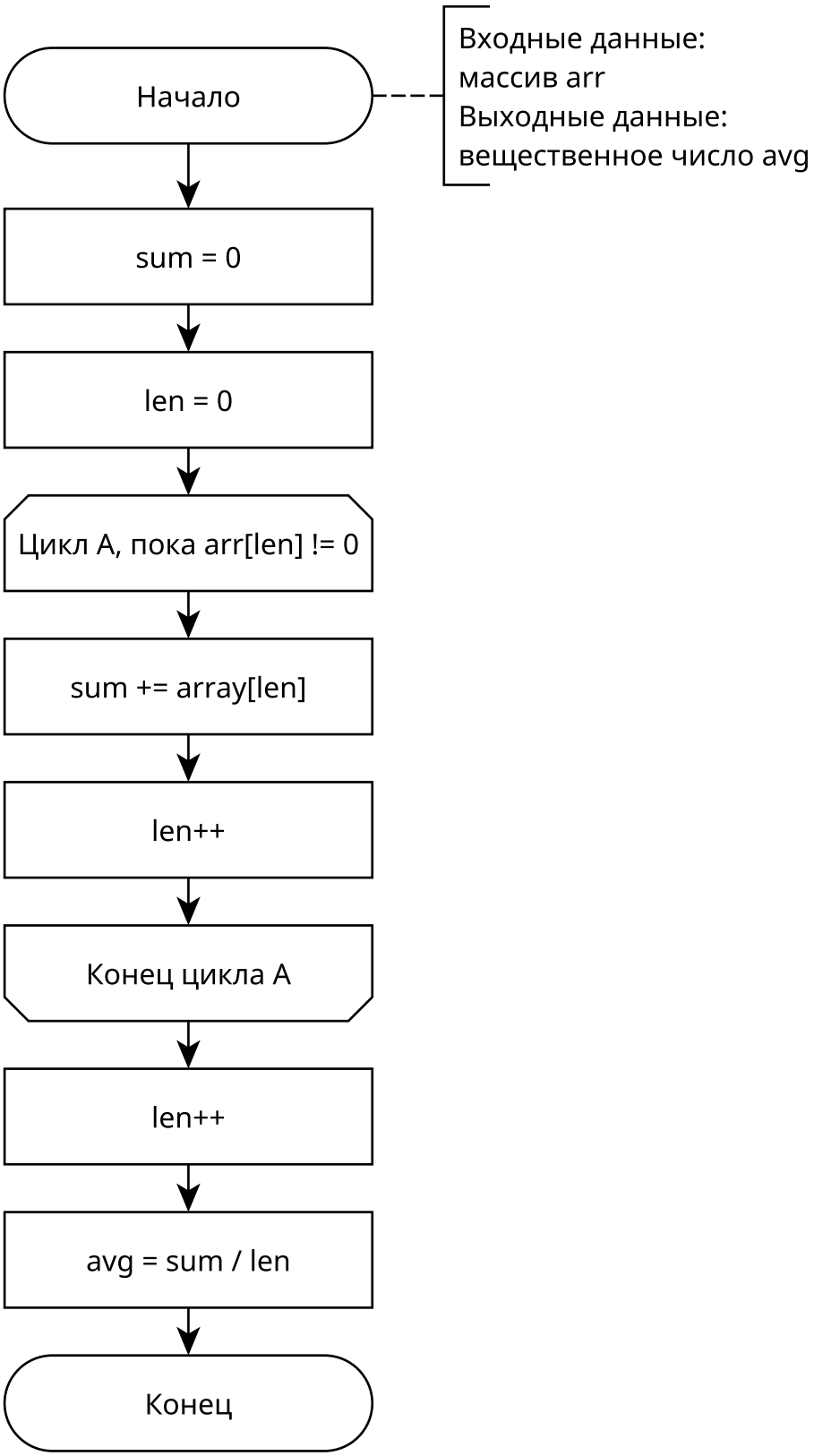

Рисунок 2.1 — Схема итеративного алгоритма вычисления среднего значения элементов последовательности

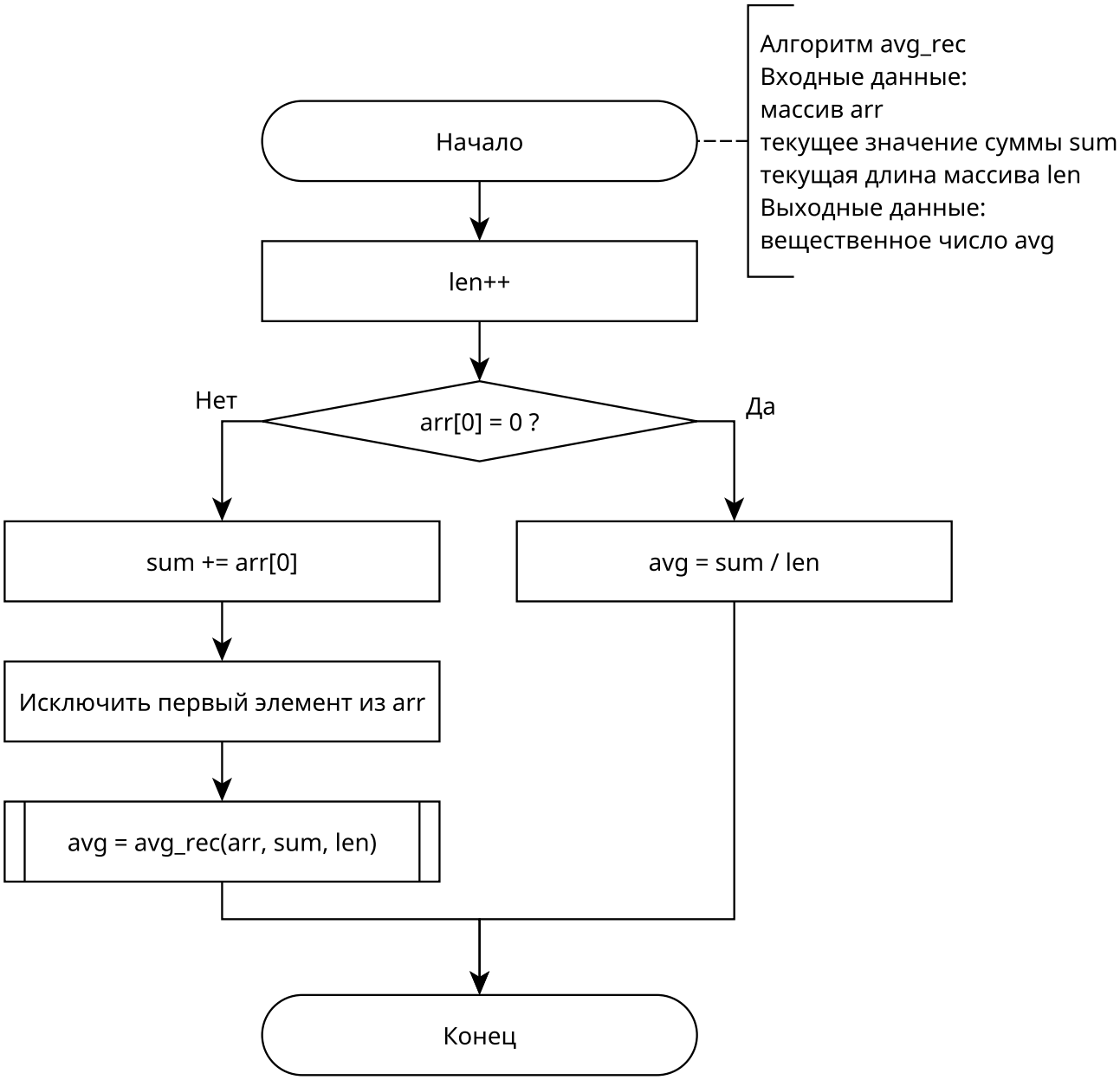

Рисунок 2.2 — Схема рекурсивного алгоритма вычисления среднего значения элементов последовательности

2.3 Расчёт трудоёмкости алгоритмов

2.3.1 Модель вычислений

Для проведения теоретической оценки трудоёмкости алгоритмов вводится следующая модель вычислений:

1) стоимость базовых операций. Операции присваивания, сравнения, инкремента/декремента, простого сложения/вычитания, индексации массива, разыменования ($=, ==, != , <, >, <=, >=, ++, --, +, -, [], *$) имеют условную стоимость, равную 1;

2) стоимость арифметических операций. Операции умножения, деления и взятия остатка ($\cdot, /, \%$) считаются более затратными и имеют условную стоимость, равную 2;

3) трудоёмкость условного оператора. Для оператора *if (условие)* {блок X} *else* {блок Y} общая трудоёмкость (f_{if}) складывается из стоимости вычисления условия ($f_{\text{условия}}$) и стоимости выполнения соответствующей ветви по формуле (2.1).

$$f_{\text{if}} = f_{\text{условия}} + \begin{cases} \min(f_X, f_Y), & \text{в лучшем случае} \\ \max(f_X, f_Y), & \text{в худшем случае} \end{cases} \tag{2.1}$$

4) трудоёмкость цикла. Для цикла *for* с N итерациями общая трудоёмкость (f_{for}) рассчитывается как сумма стоимости инициализации (f_{init}), всех проверок условия ($N + 1$ раз) и выполнения тела цикла и инкремента (N раз) по формуле (2.2).

$$f_{\text{for}} = f_{\text{init}} + (N + 1) \cdot f_{\text{условия}} + N \cdot (f_{\text{тело}} + f_{\text{инкремент}}) \tag{2.2}$$

2.3.2 Расчёт трудоёмкости итеративного алгоритма

Трудоёмкость итеративного алгоритма рассчитывается по формуле (2.2) и приведена в формуле (2.3).

$$f_{\text{iter}} = 1 + 1 + 2 + N \cdot (2 + 2 + 1) + 1 + 3 = 5N + 8 \approx 5N \tag{2.3}$$

Асимптотическая сложность — $O(N)$.

2.3.3 Расчёт трудоёмкости рекурсивного алгоритма

Трудоёмкость рекурсивного алгоритма рассчитывается по формуле (2.4).

$$f_A(D) = R(D) \cdot f_R(1) + R_L(D) \cdot f_{CL}(1) + R_V(D) \cdot f_{CV}(v), \tag{2.4}$$

где:

1) $R(D)$ — это общее количество вершин дерева рекурсии для входа D;

2) $f_R(1)$ — это количество базовых операций для обслуживания одного рекурсивного вызова;

3) $R_L(D)$ — это количество листьев дерева рекурсии для входа D;

4) $f_{CL}(1)$ — это трудоёмкость алгоритма при останове рекурсии;

5) $R_V(D)$ — это количество внутренних вершин дерева для входа D;

6) $f_{CV}(v)$ — это трудоёмкость вычислений в любой внутренней вершине.

Для данного алгоритма количество листьев дерева рекурсии всегда равно 1, то есть $R_L(D) = 1$. Количество внутренних вершин дерева рекурсии $R_V(D) = N-1$. Общее количество вершин дерева рекурсии $R(D) = N$.

Трудоёмкость алгоритма при останове рекурсии приведена в формуле (2.5).

$$f_{CL}(1) = 2 + 2 + 5 = 9 \tag{2.5}$$

Трудоёмкость вычислений в любой внутренней вершине приведена в формуле (2.6).

$$f_{CV}(v) = 2 + 2 + 3 + 1 + 1 = 9 \tag{2.6}$$

Количество базовых операций для обслуживания одного рекурсивного вызова рассчитывается по формуле (2.7).

$$f_R(1) = 2 \cdot (p + k + r + f + l + 1), f = 1, \tag{2.7}$$

где:

1) p — количество передаваемых фактических параметров;

2) r — количество сохраняемых в стеке регистров;

3) k — количество возвращаемых по адресной ссылке значений;

4) l — количество локальных ячеек процедуры, сохранение которых необходимо для обеспечения реентерабельности.

Применив формулу (2.7) к рассматриваемому алгоритму, получается формула (2.8).

$$f_R(1) = 2 \cdot (1 + 0 + 2 + 1 + 1 + 1) = 2 \cdot 6 = 12 \tag{2.8}$$

Итоговая трудоёмкость выражается по формуле (2.9).

$$f_A(D) = N \cdot 12 + 1 \cdot 9 + (N - 1) \cdot 9 = 12N + 9 + 9N - 9 = 21N \tag{2.9}$$

Асимптотическая сложность — $O(N)$.

2.4 Оценка используемой алгоритмами памяти

2.4.1 Оценка используемой памяти для итеративного алгоритма

Так как в итеративном алгоритме память используется только для создания трёх локальных переменных вне цикла, то использование памяти итеративным алгоритмом не зависит от входных данных и асимптотически оценивается как $O(1)$.

2.4.2 Оценка используемой памяти для рекурсивного алгоритма

Количество используемых ячеек памяти для обслуживания одного рекурсивного вызова рассчитывается по формуле (2.10).

$$V_R(1) = p + k + f + l + 1 = 1 + 2 + 1 + 1 + 1 = 6 \tag{2.10}$$

Объём используемой памяти рассчитывается по формуле (2.11).

$$V_R(D) = V_R(1) \cdot H_R(D) = 6 \cdot N = 6N, \tag{2.11}$$

где $H_R(D) = R(D)$ — это максимальная глубина дерева рекурсии.

Асимптотическая сложность — $O(N)$.

Вывод

В конструкторской части были сформулированы требования к программной реализации и представлены блок-схемы, описывающие логику работы итеративного и рекурсивного алгоритмов вычисления среднего значения последовательности. Была введена модель вычислений, на основе которой проведена теоретическая оценка и анализ трудоёмкости и затрачиваемой памяти.

Результаты анализа трудоёмкости памяти показывают, что асимптотическая сложность алгоритмов одинаковая, однако трудоёмкость рекурсивного алгоритма больше в 4.2 раза.

Результаты оценки использования памяти показывают, что использование памяти итеративным алгоритмом асимптотически оценивается как $O(1)$, а рекурсивного оценивается как $O(N)$.

3 Технологическая часть

3.1 Средства реализации

Для реализации алгоритмов был выбран язык C. Для измерения процессорного времени процесса используется функция clock [2] из заголовочного файла time.h. Для построения графиков используется утилита gnuplot.

3.2 Реализации алгоритмов

В листинге 3.1 представлен исходный код реализации итеративного алгоритма вычисления среднего значения элементов последовательности.

```
double calc_avg_iter(int *array) {
  long long sum = 0;
  size_t len = 0;
  while (array[len] != 0) {
    sum += array[len];
    len++;
  }
  len++;
  double avg = (double)sum / (double)len;
  return avg;
}
```

Листинг 3.1 — Реализация итеративного алгоритма вычисления среднего значения

В листинге 3.2 представлен исходный код реализации рекурсивного алгоритма вычисления среднего значения элементов последовательности.

```
double calc_avg_rec(int *array, long long *sum, size_t *len) {
  double avg;
  (*len)++;
  if (*array == 0) {
    avg = (double)*sum / (double)*len;
    return avg;
  }
  *sum += *array;
  array++;
  avg = calc_avg_rec(array, sum, len);
  return avg;
}
```

Листинг 3.2 — Реализация рекурсивного алгоритма вычисления среднего значения

3.3 Функциональные тесты

В таблице 3.1 представлены данные и результаты тестирования реализаций алгоритмов вычисления среднего значения элементов последовательности.

Таблица 3.1 — Функциональные тесты реализаций алгоритмов вычисления среднего значения элементов последовательности

Последовательность	Ожидаемый результат	Фактический результат
$(1, 2, 3, a)$	Сообщение об ошибке	Ошибка ввода
(0)	0	0
$(1, -1, -1, 1, 2, -3, 4, -3, 0)$	0	0
$(1, 2, 3, 3, 4, 5, 4, 5, 6, 2, 1, 7, 5, 6, 0)$	3.6	3.6

Все тесты пройдены успешно для функций, реализующих рассматриваемые алгоритмы.

Вывод

В технологической части определены необходимые средства реализации, и с их помощью реализованы алгоритмы вычисления среднего значения элементов последовательности. Успешно проведено функциональное тестирование реализаций алгоритмов.

4 Исследовательская часть

4.1 Технические характеристики

Замеры процессорного времени проводились на ноутбуке с характеристиками:

— операционная система — Debian Unstable GNU/Linux с ядром версии 6.16.8-1;

— процессор — AMD Ryzen 5 7535HS с 6 физическими ядрами и 12 логическими ядрами с максимальной тактовой частотой 4.60 ГГц [3];

— оперативная память — 16 Гбайт типа LPDDR5 в 4 каналах с тактовой частотой 6400 МГц и задержками CL 19 tRCD 15 tRP 17 tRAS 34 tRC 51.

4.2 Замеры процессорного времени

Замеры процессорного времени проводились для последовательностей случайных чисел, оканчивающихся нулём, длиной от 1000 до 40000 с шагом изменения 1000. Для каждого размера производилось $k = 100$ измерений, результатом замера является среднее арифметическое из этих измерений. Время измерялось в микросекундах. Во время выполнения замеров ноутбук был подключён к сети, никаких прочих программ, кроме системных, запущено не было.

Результаты измерений представлены в таблице 4.1.

Таблица 4.1 — Результаты измерений процессорного времени вычисления среднего значения элементов последовательности, мкс

Размер	Итеративный алгоритм	Рекурсивный алгоритм
1000	1.606	4.378
2000	2.686	8.024
3000	3.334	10.314
4000	4.300	13.568
5000	5.166	16.968
6000	6.172	20.354
7000	7.326	24.964
8000	8.110	28.038
9000	9.118	31.246
10000	9.994	34.642
11000	11.360	39.780
12000	12.046	42.532
13000	12.776	44.722
14000	13.716	47.770
15000	14.662	52.448
16000	15.526	54.394
17000	16.600	58.046
18000	17.392	61.196
19000	18.498	64.738
20000	19.268	68.016
21000	20.372	71.144
22000	21.096	75.098
23000	22.090	78.104
24000	23.250	80.812
25000	24.404	85.992
26000	25.186	89.728
27000	26.004	92.408
28000	27.002	95.992
29000	27.932	99.458
30000	28.840	102.264
31000	29.896	105.660
32000	30.936	108.360
33000	31.626	111.804
34000	32.660	114.964
35000	33.742	118.504
36000	34.680	121.636
37000	35.376	125.138
38000	36.484	128.992
39000	37.368	131.644
40000	38.290	136.262

Графики зависимости времени выполнения от длины последовательности приведены на рисунке 4.1.

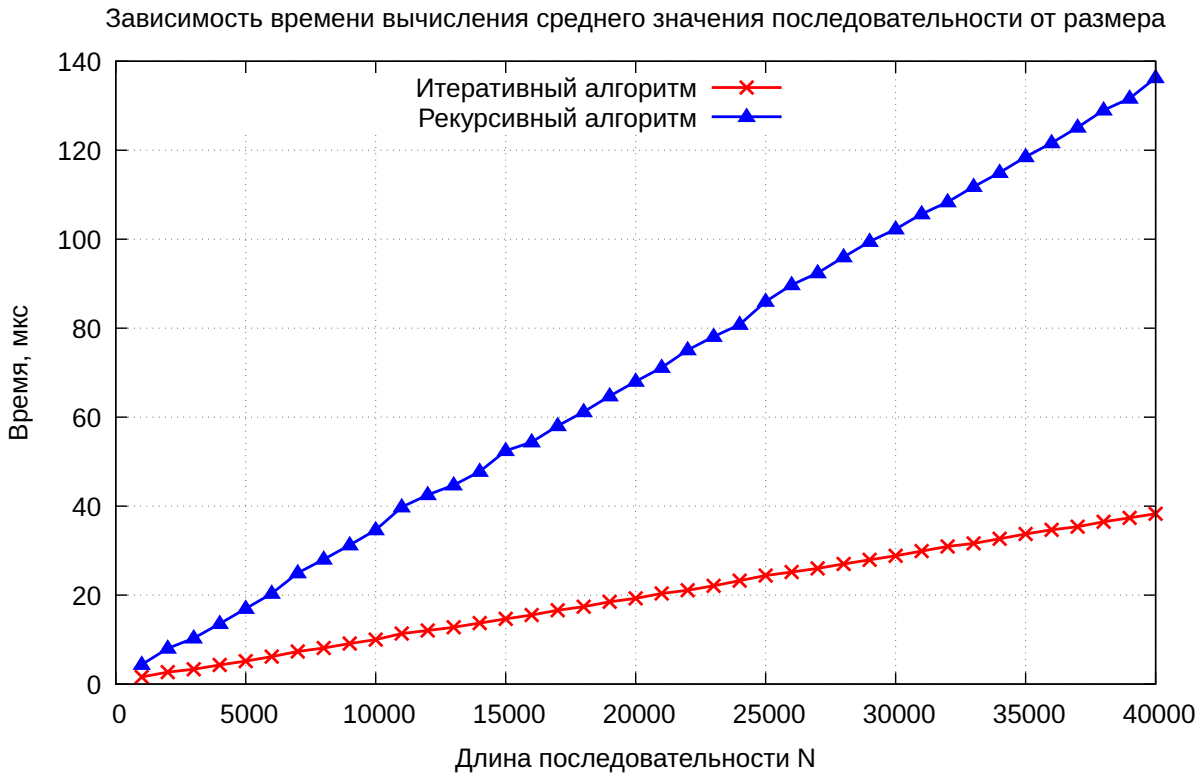

Рисунок 4.1 — Зависимость времени выполнения от длины последовательности

Вывод

В исследовательской части были произведены измерения зависимости процессорного времени работы реализации алгоритмов от длины последовательности. В результате, реализация итеративного алгоритма работает быстрее реализации рекурсивного в среднем в 3.5 раза.

ЗАКЛЮЧЕНИЕ

Цель данной лабораторной работы была достигнута: проведён сравнительный анализ рекурсивного и нерекурсивного алгоритмов. Все задачи решены:

1) разработаны рекурсивный и нерекурсивный алгоритмы решения задачи;

2) описаны средства разработки и инструменты замера процессорного времени выполнения реализации алгоритмов;

3) реализованы разработанные алгоритмы;

4) выполнено тестирование реализации алгоритмов;

5) выполнена теоретическая оценка затрачиваемой реализацией каждого алгоритма памяти (для рекурсивного алгоритма — на материале анализа высоты дерева рекурсивных вызовов и оценки затрачиваемой на один вызов функции памяти);

6) выполнены замеры процессорного времени выполнения реализации алгоритмов в зависимости от варьируемого входа;

7) оценена трудоёмкость реализаций двух алгоритмов;

8) проведено сравнение результатов замеров процессорного времени и оценки трудоёмкости;

9) сделаны выводы из сравнительного анализа рекурсивного и нерекурсивного алгоритмов решения одной и той же задачи по критериям ёмкостной эффективности (на материале теоретической оценки) и пиковой временной эффективности (на материале результатов измерений) — реализация итеративного алгоритма работает быстрее реализации рекурсивного в среднем в 3.5 раза, использование памяти итеративным алгоритмом асимптотически оценивается как $O(1)$, а рекурсивного оценивается как $O(N)$.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Рекурсивные методы в программировании / Д. Баррон — Москва: Мир, 2003 - 79 с.

2. clock(3) — Linux manual page [Электронный ресурс]. Режим доступа: `https://man7.org/linux/man-pages/man3/clock.3.html` (Дата обращения: 02.10.2025).

3. AMD Ryzen 5 7535HS [Электронный ресурс]. Режим доступа: `https://www.amd.com/en/products/processors/laptop/ryzen/7000-series/amd-ryzen-5-7535hs.html` (Дата обращения: 04.10.2025).