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 Ильченко Е. А.

Группа ИУ7-54Б

Преподаватели Волкова Л.Л.

Москва, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является исследование рекурсивного и нерекурсивного алгоритмов подсчета количества единиц в последовательности из нулей и единиц, завершающейся числом 2.

Для достижения поставленной цели необходимо решить следующие задачи:

— разработать рекурсивный и нерекурсивный алгоритмы решения задачи;

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

— реализовать разработанные алгоритмы;

— выполнить тестирование реализации алгоритмов на корректность работы;

— выполнить теоретическую оценку затрачиваемой реализацией каждого алгоритма памяти, включая анализ высоты дерева рекурсивных вызовов;

— выполнить замеры процессорного времени выполнения реализации алгоритмов в зависимости от варьируемого размера входных данных;

— оценить трудоёмкость двух алгоритмов в худшем случае;

— сравнить результаты замеров процессорного времени и оценки трудоемкости;

— провести сравнительный анализ реализации рекурсивного и нерекурсивного алгоритмов по критериям ёмкостной и временной эффективности.

1 Аналитическая часть

1.1 Математическая модель

Пусть дана последовательность $S = \{s_1, s_2, ..., s_n, 2\}$, где $s_i \in \{0, 1\}$. Требуется найти значение:

$$count = \sum_{i=1}^{n} \delta(s_i, 1)$$

где $\delta(x, y)$ – функция Кронекера, равная 1 при $x = y$ и 0 в противном случае.

1.2 Рекурсия

1.2.1 Определение рекурсии

Рекурсия – это метод определения функций или решения задач, при котором функция вызывает саму себя непосредственно или через другие функции [1].

Рекурсивная функция f определяется через [1, 2]:

— **базовый случай** (терминальное условие): $f(x_0) = c$, где x_0 – терминальное значение;

— **рекурсивный случай**: $f(x) = g(x, f(h(x)))$, где $h(x)$ – функция, уменьшающая задачу.

Корректность рекурсивного алгоритма обеспечивается выполнением условий [3]:

— **существование базового случая**: функция завершается при обнаружении терминатора 2 или пустой последовательности;

— **сходимость**: на каждом шаге размер задачи уменьшается ($|rest(S)| < |S|$);

— **правильность**: результат вычисляется как сумма вклада текущего элемента и результата для оставшейся последовательности.

1.2.2 Математическая модель рекурсивного алгоритма подсчёта

Для задачи подсчёта единиц в последовательности рекурсивная функция $count(S)$ определяется как:

$$count(S) = \begin{cases} 0, & \text{если } S = \emptyset \text{ или } first(S) = 2 \\ 1 + count(rest(S)), & \text{если } first(S) = 1 \\ 0 + count(rest(S)), & \text{если } first(S) = 0 \end{cases}$$

где:

— $first(S)$ – первый элемент последовательности S;

— $rest(S)$ – последовательность без первого элемента;
— $\emptyset$ – пустая последовательность.

Вывод

В аналитической части:
— проведён анализ математической модели задачи, формализованной через сумму функций Кронекера;
— представлено строгое математическое описание рекурсивного метода с выделением базового и рекурсивного случаев;
— определены условия корректности рекурсивного алгоритма, обеспечивающие его сходимость и правильность работы.

2 Конструкторская часть

2.1 Требования к реализации

К программе предъявлены ряд функциональных требований: входные данные – последовательность целых чисел (0, 1, 2), выходные данные – количество единиц в последовательности до первого вхождения числа 2.

К программе предъявлены ряд требований:

— наличие интерфейса для выбора действий;

— реализация рекурсивного алгоритма подсчёта единиц в последовательности;

— реализация нерекурсивного алгоритма подсчёта единиц в последовательности;

— наличие функциональности замера процессорного времени выполнения алгоритмов;

— поддержка ввода последовательности чисел с проверкой корректности данных.

Программа работает в трёх режимах:

— выполнение рекурсивного алгоритма;

— выполнение нерекурсивного алгоритма;

— замер процессорного времени выполнения алгоритмов.

При первом и втором режимах на вход поступает последовательность чисел (0, 1), завершающаяся числом 2, на выход – количество единиц в последовательности до первого вхождения числа 2.

При третьем режиме работы на вход поступают минимальный размер последовательности, максимальный размер, шаг изменения размера и число итераций k ($k \geq 100$), на выход — результаты замера времени в виде таблицы, где каждая точка получается делением времени выполнения k идентичных расчётов на k.

2.2 Разработка алгоритмов

Раздел содержит блок-схемы, описывающие следующие алгоритмы: рекурсивный алгоритм подсчёта единиц в последовательности, нерекурсивный алгоритм подсчёта единиц в последовательности.

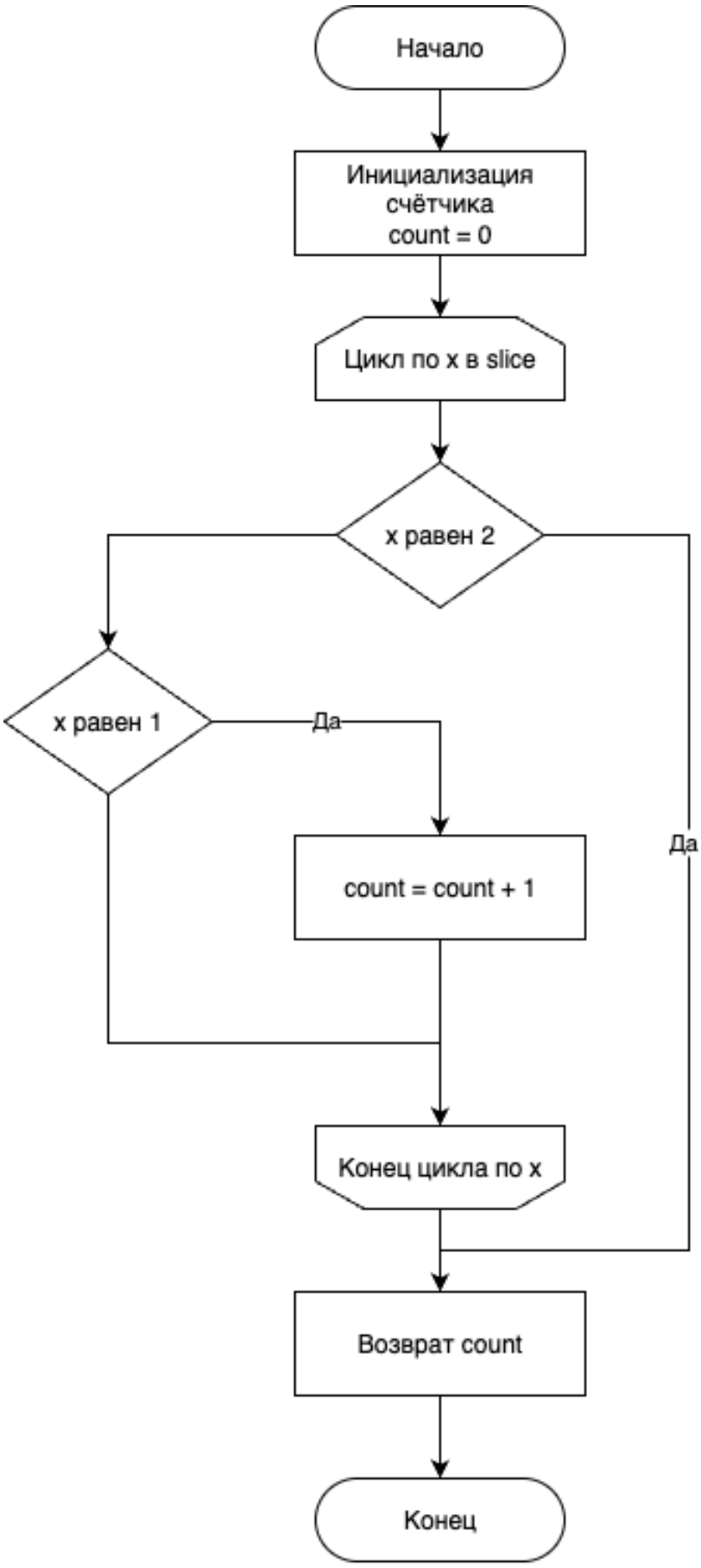

Рисунок 2.1 — Схема нерекурсивного алгоритма

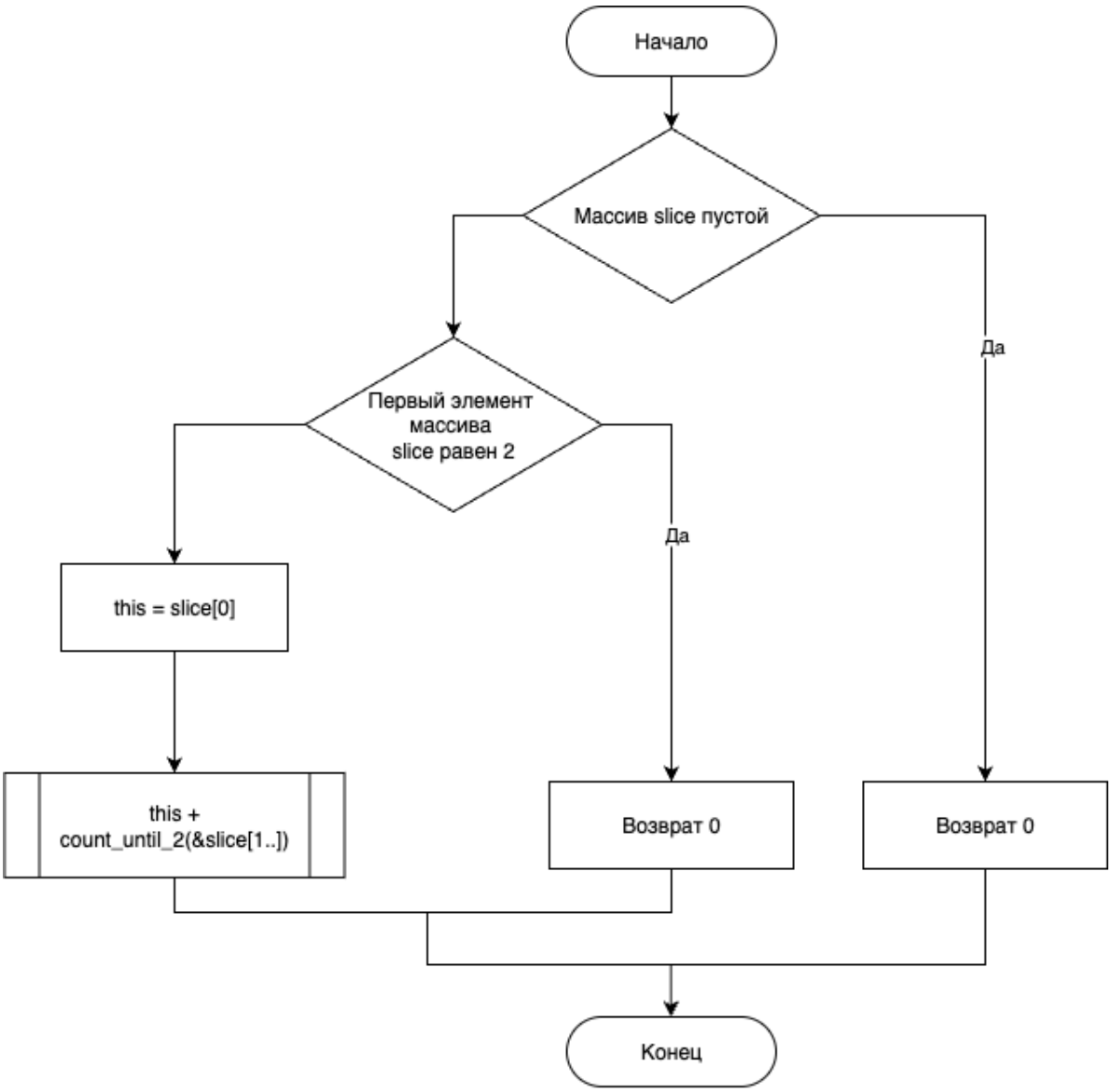

Рисунок 2.2 — Схема рекурсивного алгоритма

2.3 Модель вычислений

Для проведения оценки трудоёмкости была введена модель вычислений.

1) трудоёмкость базовых операций:
   - — равна 1 для: =, !=, ==, <, >, <=, >=, |, ||, &, &&, |=, &=, +, ++, -, –, +=, -=, [], $<<$, $>>$;
   - — равна 2 для: *, /, *=, /=, %, %=.
2) трудоёмкость условного оператора:

$$f_{if} = f_{\text{условия}} + \begin{cases} min(f_1, f_2), & \text{л. с.} \\ max(f_1, f_2), & \text{х. с.} \end{cases}$$

3) трудоёмкость цикла:

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + M_{\text{итераций}} \cdot (f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}})$$

2.4 Трудоёмкость и затраты памяти алгоритмов

В этом разделе приведены оценки трудоёмкостей и затрат памяти алгоритмов согласно реализациям.

2.4.1 Оценка трудоемкости алгоритмов

Нерекурсивный алгоритм

Пусть длина последовательности n. В худшем случае в последовательности только единицы.

1) инициализация переменной `count`: $f_{init} = 1$;
2) цикл по элементам: $f_{loop} = 2 + n \cdot (2 + f_{body})$;
3) тело цикла: $f_{body} = f_{2\_check} + f_{1\_check}$;
4) проверка на равенство двойке: $f_{2\_check} = 1$;
5) проверка на равенство единице: $f_{1\_check} = 1 + 1$;
6) возврат результат: $f_{return} = 1$.

Результирующее выражение:

$$f_{iterative} = f_{init} + f_{loop} + f_{return} = 1 + 2 + n \cdot (2 + 1 + 2) + 1 = 4 + 5 \cdot n \approx O(n)$$

Рекурсивный алгоритм

Пусть длина последовательности n. В худшем случае в последовательности только единицы, то есть будет n вызовов.

1) проверка на пустоту: $f_{is\_empty} = 1$;

2) проверка первого элемента: $f_{first} = 1 + 1$;

3) рекурсия: $f_{recursive\_case} = 2 + 1 + f_{call}$;

4) вызов функции: $f_{call} = 2 \cdot (1 + 1 + 1 + 1) = 8$.

Результирующее выражение: $f_{recursive} = n \cdot (1 + 2 + 2 + 1 + 8) + 1 + 1 = 14n + 2 \approx O(n)$

2.4.2 Оценка затрат памяти алгоритмов

Нерекурсивный алгоритм

Затраты памяти нерекурсивного алгоритма:

— результат: $O(1)$ – одна целочисленная переменная;

— локальные переменные: $O(1)$ – переменные count, x, итератор цикла;

— параметры функции: $O(1)$ – ссылка на срез.

Общие затраты памяти: $O(1)$

Рекурсивный алгоритм

Затраты памяти рекурсивного алгоритма:

— результат: $O(1)$ – одно целое число;

— стек вызовов: O(n), где n – длина среза до первого элемента 2;

— память на один вызов функции:

  – адрес возврата: 8 байт;

  – сохранение регистров: 16 байт;

  – параметры функции: 16 байт;

  – локальные переменные: 8 байт;

  – служебная информация: 8 байт.

Итого на один вызов 56 байт

Общие затраты памяти: $O(n)$ Высота дерева рекурсивных вызовов: n. Максимальный объём памяти стека: $56 \cdot n$ байт.

Вывод

В данном разделе были разработаны и проанализированы два алгоритма подсчёта единиц в последовательности: рекурсивный и нерекурсивный. Для каждого из них представлены блок-схемы, описывающие логику работы. На основе введённой модели вычислений проведена оценка трудоёмкости алгоритмов, которая показала, что оба алгоритма имеют линейную временную сложность $O(n)$. Оценка затрат памяти выявила, что нерекурсивный алгоритм требует $O(1)$ памяти, в то время как рекурсивный, вследствие использования стека вызовов, – $O(n)$. Таким образом, с точки зрения эффективности использования памяти нерекурсивный алгоритм

является более предпочтительным.

3 Технологическая часть

3.1 Средства реализации

Для реализации алгоритмов подсчёта единиц в последовательности был выбран язык программирования `Rust`, поскольку он соответствует требованиям лабораторной работы. Измерение времени выполнения алгоритмов осуществлялось с использованием структуры `Instant` из стандартной библиотеки `std::time`. Разработка программного обеспечения проводилась в интегрированной среде разработки `RustRover`.

3.2 Реализация алгоритмов

Были реализованы нерекурсивный алгоритм подсчёта единиц в последовательности (листинг 3.1), рекурсивный алгоритм подсчёта единиц в последовательности (листинг 3.2).

Листинг 3.1 — Нерекурсивный алгоритм

```
pub(crate) fn count_until_2(slice: &[i32]) -> usize {
  let mut count = 0;
  for &x in slice {
    if x == 2 {
      break;
    }
    if x == 1 {
      count += 1;
    }
  }
  count
}
```

Листинг 3.2 — Рекурсивный алгоритм

```
pub(crate) fn count_until_2(slice: &[i32]) -> usize {
  if slice.is_empty() {
    0
  } else if slice[0] == 2 {
    0
  } else {
    let this = slice[0] as usize;
    this + count_until_2(&slice[1..])
  }
}
```

3.3 Функциональные тесты

В данном разделе представлены функциональные тесты для разработанных алгоритмов: нерекурсивный алгоритм подсчёта единиц в последовательности (таблица 3.1), рекурсивный алгоритм подсчёта единиц в последовательности (таблица 3.2).

Таблица 3.1 — Функциональные тесты для нерекурсивного алгоритма

№	Входная последовательность	Ожидаемый результат	Полученный результат
1	2	0	0
2	1 2	1	1
3	0 2	0	0
4	1 1 1 2	3	3
5	0 0 0 2	0	0
6	1 0 1 0 2	2	2
7	0 1 0 1 0 2	2	2
8	1 1 0 2 1 1	2	2
9	1 0 1 0 1 0 1 2	4	4
10	a	Некорректный ввод: 'a'	Некорректный ввод: 'a'
11	1 3	Недопустимое число 3. Разрешены только 0, 1, 2	Недопустимое число 3. Разрешены только 0, 1, 2

Таблица 3.2 — Функциональные тесты для рекурсивного алгоритма

№	Входная последовательность	Ожидаемый результат	Полученный результат
1	2	0	0
2	1 2	1	1
3	0 2	0	0
4	1 1 1 2	3	3
5	0 0 0 2	0	0
6	1 0 1 0 2	2	2
7	0 1 0 1 0 2	2	2
8	1 1 0 2 1 1	2	2
9	1 0 1 0 1 0 1 2	4	4
10	a	Некорректный ввод: 'a'	Некорректный ввод: 'a'
11	1 3	Недопустимое число 3. Разрешены только 0, 1, 2	Недопустимое число 3. Разрешены только 0, 1, 2

Все функциональные тесты были успешно пройдены.

Вывод

В данном разделе были реализованы алгоритмы нерекурсивный алгоритм подсчёта единиц в последовательности и рекурсивный алгоритм подсчёта единиц в последовательности. Все алгоритмы включают проверку корректности входных данных и обработку ошибок.

Проведённые функциональные тесты подтвердили корректность работы всех реализованных алгоритмов.

4 Исследовательская часть

4.1 Характеристики ЭВМ

Замеры проводились на устройстве со следующими характеристиками:

— процессор: Apple M4 Pro;

— оперативная память: 24 Гб;

— операционная система: macOS Sequoia 15.6.1.

Замеры времени проводились, когда ноутбук был загружен только системными приложениями.

4.2 Результаты замера времени

В данном разделе представлены результаты измерения времени выполнения алгоритмов подсчёта единиц в последовательности.

Замеры проводились для последовательностей размером от 100 до 2000 символов. Каждый замер выполнялся 1000 раз.

Размер	**Рекурсия (мкс)**	**Итерация (мкс)**
100	2.649	0.889
200	5.395	1.453
300	6.700	1.695
400	7.555	2.109
500	8.273	2.068
600	8.626	2.362
700	9.629	2.488
800	10.247	2.783
900	10.489	2.816
1000	11.585	2.957
1100	12.170	3.235
1200	12.382	3.464
1300	14.295	3.764
1400	14.875	4.018
1500	16.320	4.551
1600	17.899	4.787
1700	18.396	5.036
1800	20.388	5.435
1900	20.858	5.766
2000	22.215	6.014

Таблица 4.1 — Результаты измерения времени

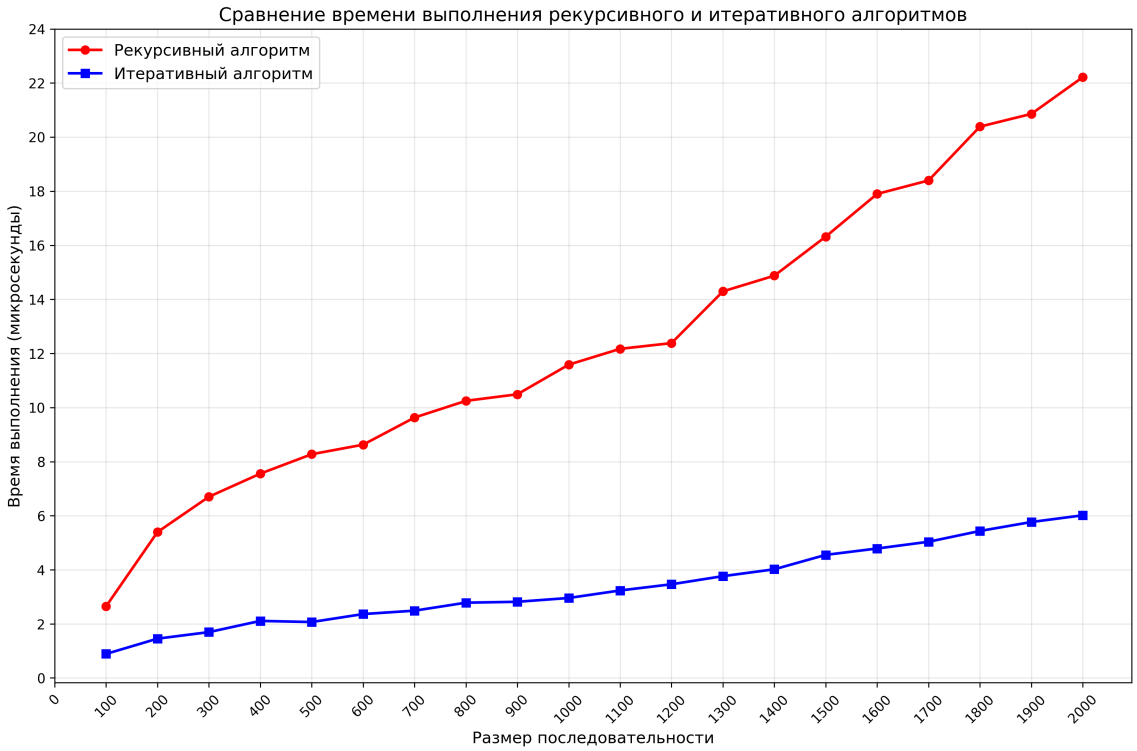

Рисунок 4.1 — График замера времени

Вывод

Согласно результатм исследования, нерекурсивный алгоритм работает быстрее рекурсивного алгоритма. Эта разница становится заметнее с увеличением размера последовательности.

ЗАКЛЮЧЕНИЕ

В рамках данной лабораторной работы была успешно достигнута основная цель.
Все поставленные задачи выполнены в полном объёме:

— разработаны рекурсивный и нерекурсивный алгоритмы решения задачи;

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

— реализованы разработанные алгоритмы на языке программирования Rust;

— выполнено тестирование реализации алгоритмов на корректность работы;

— выполнена теоретическая оценка затрачиваемой памяти для каждого алгоритма, включая анализ высоты дерева рекурсивных вызовов. Оценка затрат памяти выявила, что нерекурсивный алгоритм требует больше памяти;

— оценена трудоёмкость алгоритмов в худшем случае. Оценка выявила, что рекурсивный алгоритм более трудоемкий в худшем случае;

— выполнено сравнение результатов замеров процессорного времени. Оценка выявила, что рекурсивный алгоритм работает медленнее нерекурсивного.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Головешкин В.А., Ульянов М.В. Теория рекурсии для программистов. – М.: ФИЗМАТЛИТ, 2006. – 296 с.

2. Баррон Д. Рекурсивные методы в программировании. – М.: Мир, 1974. – 79 с.

3. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ, 2-ое издание : Пер. с англ. – М.: Издательский дом «Вильямс», 2005. – 1296 с.