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 № 3
по дисциплине «Анализ алгоритмов»

Тема Графовые модели алгоритмов

Студент Куликов Н.В.

Группа ИУ7-53Б

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

Москва, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной лабораторной работы был сравнительный анализ рекурсивного и нерекурсивного алгоритмов решения задачи определения количества единиц в последовательности, состоящей из нулей и единиц, заканчивающейся числом 2.

Для достижения поставленной цели необходимо было выполнить следующие задачи:

1) разработать рекурсивный и нерекурсивный алгоритмы решения задачи;
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;
7) оценить трудоёмкость двух алгоритмов в худшем случае;
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 пиковой временной эффективности на материале результатов измерений).

1 Аналитическая часть

1.1 Рекурсия

В математике и программировании рекурсия – это метод определения или выражения функции или процедуры посредством той же функции и процедуры. Рекурсию обычно рассматривают в качестве антипода итерации. Соответственно различают два больших класса алгоритмов: итерационные и рекурсивные.

В основе итерационных алгоритмов лежит итерация – многократное повторение одних и тех же действий. Рекурсивный алгоритм – это алгоритм, определяемый через себя. В основе рекурсивных алгоритмов лежит рекурсия. [1]

В контексте программирования рекурсивной называется функция, которая вызывает сама себя. [2]

1.2 Вывод

В аналитической части были рассмотрены понятия рекурсии, итерационного и рекурсивного алгоритмов, рекурсивной функции.

2 Конструкторская часть

2.1 Функциональные требования

Разработать программное обеспечение с двумя режимами работы: режимом определения количества единиц в последовательности, состоящей из нулей и единиц, заканчивающейся числом 2, и режимом массированного замера времени выполнения реализаций рекурсивного и нерекурсивного алгоритмов.

Входные данные:

— пункт меню (целое число от 0 до 3),

— элементы последовательности – целые числа от 0 до 2.

Выходные данные:

Целое число – количество единиц в последовательности.

2.2 Схемы алгоритмов

На рисунках 2.1 – 2.2 представлены схемы алгоритмов.

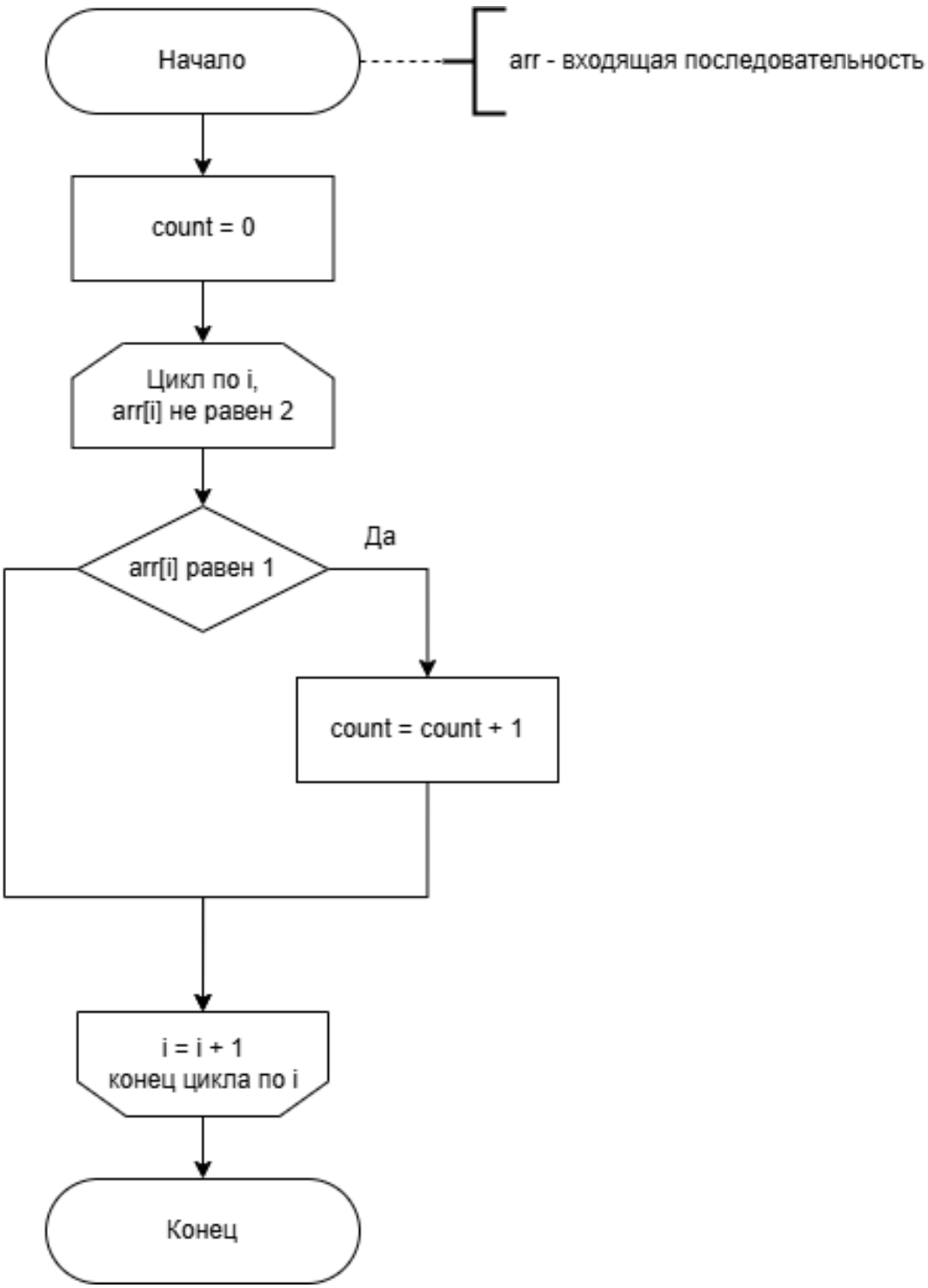

Рисунок 2.1 — Схема нерекурсивного алгоритма

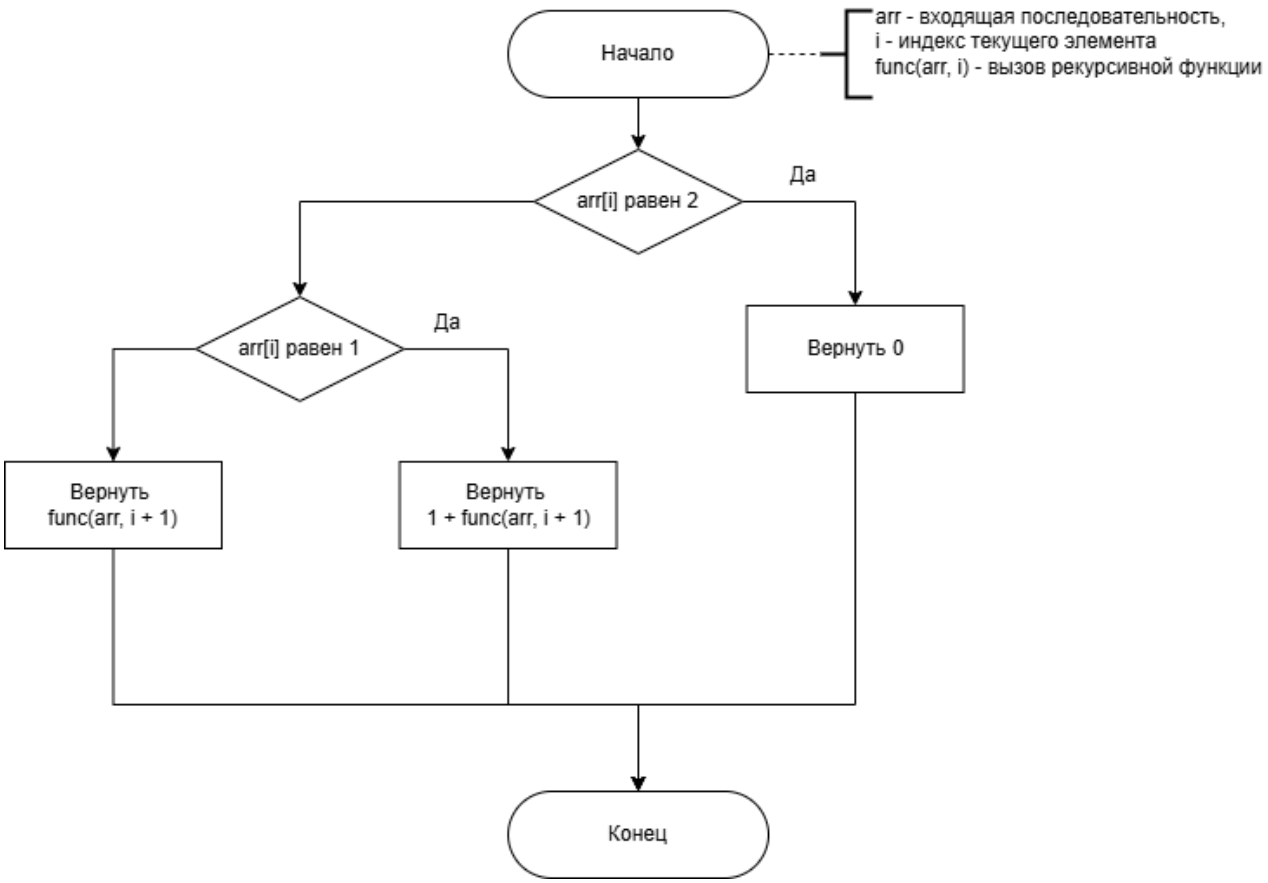

Рисунок 2.2 — Схема рекурсивного алгоритма

2.3 Модель вычислений

Чтобы вычислить трудоёмкость алгоритмов, введена следующая модель вычислений:

1) базовые операции:
   — трудоёмкость операций из списка (2.1) равна 1:

$$=, +, +=, -, -=, ++, --, ==, !=, <, <=, >=, >, [], \&\&, \\ \&, >>, <<, ||, | \tag{2.1}$$

   — трудоёмкость операций из списка (2.2) равна 2:

$$*, *=, /, /=, \%, \%= \tag{2.2}$$

2) трудоёмкость условного перехода равна 0,
3) трудоёмкость условного оператора по формуле (2.3):

$$f_{if} = f_{\text{условия}} + \begin{cases} min(f_A, f_B), & \text{– лучший случай} \\ max(f_A, f_B) & \text{– худший случай} \end{cases} \tag{2.3}$$

4) трудоёмкость цикла по формуле (2.4):

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + M \cdot (f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \tag{2.4}$$

где M – число итераций.

5) трудоёмкость одного рекурсивного вызова-возврата по формуле (2.5):

$$f_R(1) = 2 \cdot (p + k + r + l + 1), \tag{2.5}$$

где p – количество передаваемых фактических параметров,
k – количество возвращаемых по адресной ссылке значений,
r – количество сохраняемых в стек регистров,
l – количество локальных ячеек процедуры, сохранение которых необходимо для обеспечения реентерабельности,

6) трудоёмкость рекурсивного алгоритма по формуле (2.6):

$$f_A(D) = R(D) \cdot f_R(1) + R_L(D) \cdot f_{CL}(v) + R_V(D) \cdot f_{CV}(v), \qquad (2.6)$$

где $R(D)$ – общее количество вершин дерева рекурсивных вызовов,
$R_V(D)$ – количество внутренних вершин дерева рекурсивных вызовов,
$R_L(D)$ – количество листовых вершин дерева рекурсивных вызовов,
$f_R(1)$ – трудоёмкость одного рекурсивного вызова-возврата,
$f_{CL}(v)$ – трудоёмкость вычислений в листовых вершинах дерева рекурсивных вызовов,
$f_{CV}(v)$ – трудоёмкость вычислений во внутренних вершинах дерева рекурсивных вызовов.

2.4 Трудоёмкость алгоритмов

2.4.1 Нерекурсивный алгоритм

Трудоёмкость нерекурсивного алгоритма рассчитана по формулам (2.7 – 2.9):

$$f_{cycle} = 2 + N \cdot (2 + f_{body}), \qquad (2.7)$$

$$f_{body} = 2 + \begin{cases} 0, & \text{– лучший случай} \\ 2 & \text{– худший случай} \end{cases} \qquad (2.8)$$

Итого, для худшего случая по формуле (2.9):

$$f_{algo} = 2 + N \cdot (2 + 4) = 6N + 2. \qquad (2.9)$$

Асимптотическая оценка временной сложности алгоритма: $O(N)$.

2.4.2 Рекурсивный алгоритм

Трудоёмкость рекурсивного алгоритма рассчитана по формулам (2.10 – 2.16):

— количество передаваемых параметров p равно 2, количество сохраняемых регистров r равно 19 [3], тогда трудоёмкость одного рекурсивного

вызова-возврата по формуле (2.10):

$$f_R(1) = 2 \cdot (2 + 0 + 19 + 0 + 1) = 44, \tag{2.10}$$

— трудоёмкость вычислений в листовых вершинах дерева рекурсивных вызовов по формуле (2.11):

$$f_{CL}(v) = 2, \tag{2.11}$$

— трудоёмкость вычислений во внутренних вершинах дерева рекурсивных вызовов по формуле (2.12):

$$f_{CV}(v) = 2 + 2 + 1 = 5, \tag{2.12}$$

— общее количество вершин дерева рекурсивных вызовов по формуле (2.13):

$$R(D) = N, \tag{2.13}$$

— количество листовых вершин дерева рекурсивных вызовов по формуле (2.14):

$$R_L(D) = 1, \tag{2.14}$$

— количество внутренних вершин дерева рекурсивных вызовов по формуле (2.15):

$$R_V(D) = N - 1, \tag{2.15}$$

— трудоёмкость алгоритма по формуле (2.16):

$$f_A(D) = N \cdot 44 + 1 \cdot 2 + (N - 1) \cdot 5 = 49N - 3. \tag{2.16}$$

Асимптотическая оценка временной сложности алгоритма: $O(N)$.

2.5 Вывод

В данном разделе были описаны функциональные требования к программе, построены схемы алгоритмов: нерекурсивного и рекурсивного.

Была введена модель вычислений, в соответствии с которой были рассчитаны трудоёмкости алгоритмов. Проведённая оценка трудоёмкости алгоритмов показала, что трудоёмкость выполнения рекурсивного алгоритма в 8.2 раза больше, чем у нерекурсивного.

3 Технологическая часть

3.1 Средства реализации

Для реализации алгоритмов был выбран язык C. Выбор обусловлен тем, что C – статически типизированный язык программирования, в нём нет сборщика мусора и имеется стандартная библиотека для замера процессорного времени.

Для замера процессорного времени использовалась функция clock() из модуля time. [4]

3.2 Реализация алгоритмов

В листингах 3.1 - 3.2 показаны реализации алгоритмов: рекурсивного и нерекурсивного.

```
size_t count_ones_recursive(const int *arr, size_t i) {
  if (arr[i] == 2)
    return 0;

  if (arr[i] == 1)
    return 1 + count_ones_recursive(arr, i + 1);

  return count_ones_recursive(arr, i + 1);
}
```

Листинг 3.1 — Реализация рекурсивного алгоритма

```
size_t count_ones_iterative(const int *arr) {
  size_t count = 0;
  for (size_t i = 0; arr[i] != 2; i++)
    if (arr[i] == 1)
      count++;

  return count;
}
```

Листинг 3.2 — Реализация нерекурсивного алгоритма

3.3 Функциональные тесты

В таблице 3.1 представлены результаты функционального тестирования реализаций: рекурсивного и нерекурсивного алгоритмов. Каждая реализация каждого алгоритма прошла тесты успешно.

Таблица 3.1 — Результаты функционального тестирования алгоритмов подсчёта единиц

Входные данные		**Результат**	
Последовательность	**Завершающий символ**	**Ожидаемый**	**Фактический**
$(1, 0, 1, 1, 0)$	2	3	3
$(1, 1, 1, 1, 1)$	2	5	5
$(0, 0, 0, 0, 0)$	2	0	0
(1)	2	1	1
(0)	2	0	0
$()$	2	0	0
(2)	2	0	0
$(1, 0, 1, 0, 1)$	-	Ошибка	Ошибка
$(1, 0, 0, 1)$	3	Ошибка	Ошибка

3.4 Вывод

В этом разделе были описаны средства реализации алгоритмов. Также были продемонстрированы листинги реализаций алгоритмов: рекурсивного и нерекурсивного. Приведены результаты функционального тестирования.

4 Исследовательская часть

4.1 Технические характеристики

Замеры процессорного времени проводились на ноутбуке ACER Predator со следующими техническими характеристиками:

— процессор: Intel(R) Core(TM) i7-10750H CPU @ 2.60ГГц 2.59 ГГц,

— ОЗУ: 16 ГБ,

— ОС: Windows 10 Pro 64-разрядная.

Во время замеров процессорного времени ноутбук был подключен к электропитанию, сторонними приложениями нагружен не был.

4.2 Замеры процессорного времени

Были проведены замеры времени работы реализаций алгоритмов для данных, соответствующих худшему случаю: когда последовательность полностью состоит из единиц.

Результаты замеров процессорного времени приведены в таблице 4.1:

Таблица 4.1 — Результаты замеров времени работы реализаций алгоритмов подсчёта единиц

Длина последовательности	Рекурсивный алгоритм, нс	Нерекурсивный алгоритм, нс
1000	6431.51	1879.79
2000	12415.00	3916.74
3000	19764.65	6402.30
4000	25434.50	8060.47
5000	30984.43	9787.86
6000	37403.94	11666.33
7000	44202.55	13603.21
8000	50720.17	15071.54
9000	57141.32	17016.54
10000	64082.05	19018.39

На рисунке 4.1 изображён график зависимости времени работы алгоритмов от длины последовательности.

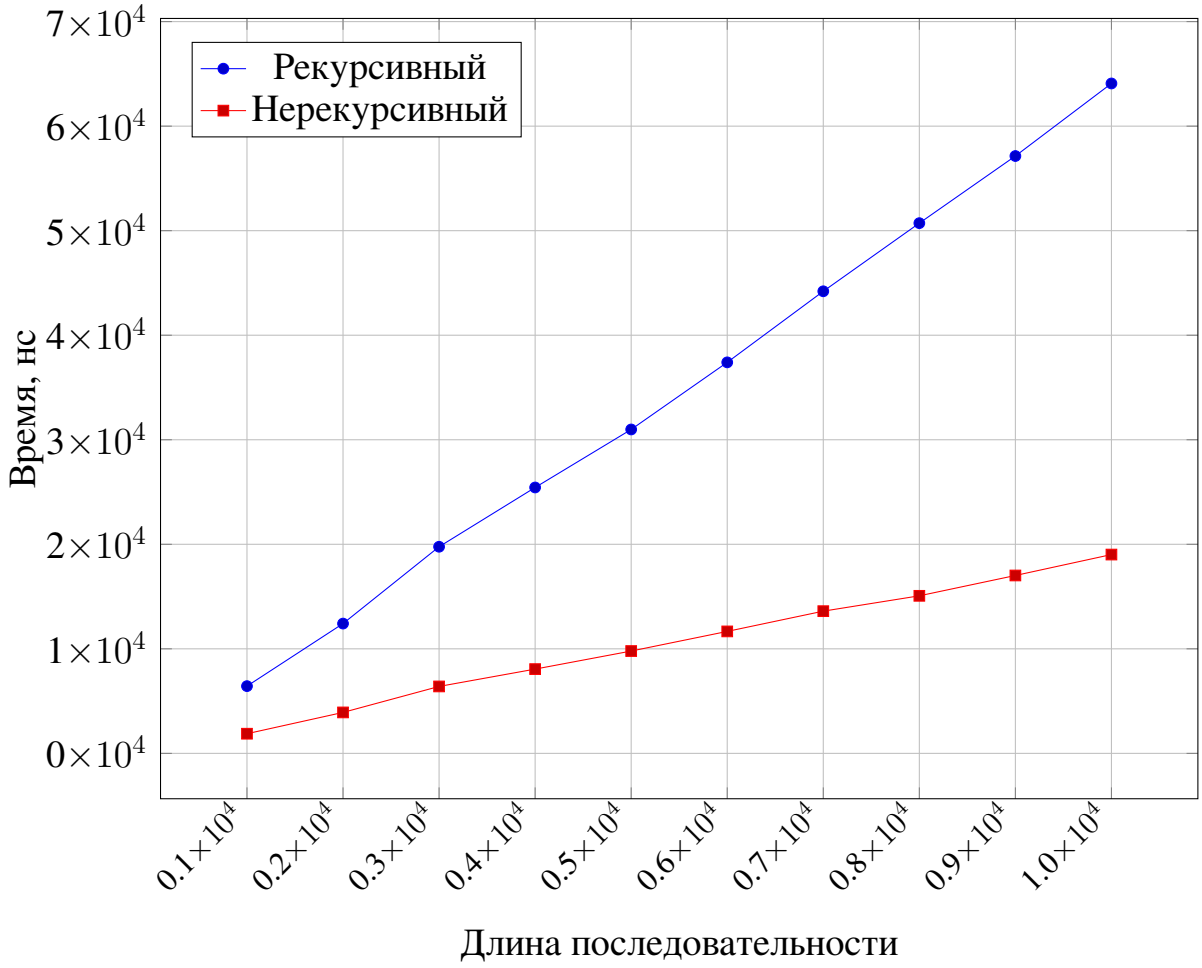

Рисунок 4.1 — Сравнение времени работы рекурсивного и нерекурсивного алгоритмов

Из графика 4.1 следует, что реализация нерекурсивного алгоритма работает быстрее реализации рекурсивного алгоритма примерно в 3.2 раза.

4.3 Теоретическая оценка затрачиваемой памяти

Размеры типов данных в байтах:

— $S_{reg} = 8$ – размер регистра,

— $S_{ret} = 8$ – размер адреса возврата,

— $S_{int} = 8$ – размер типа int.

4.3.1 Рекурсивный алгоритм

Память, затрачиваемая реализацией рекурсивного алгоритма, в байтах рассчитана по формулам (4.1 – 4.4):

— глубина дерева рекурсивных вызовов рассчитана по формуле (4.1):

$$R(D) = N, \tag{4.1}$$

— память в байтах, затраченная на один рекурсивный вызов, рассчитана по формуле (4.2):

$$M_{reccall} = 19 \cdot S_{reg} + S_{ret} = 160, \tag{4.2}$$

— память в байтах, затраченная на локальные переменные, рассчитана по формуле (4.3):

$$M_{local} = 0, \tag{4.3}$$

— общая память в байтах для рекурсивного алгоритма рассчитана по формуле (4.4):

$$M_{total} = (M_{reccall} + M_{local}) \cdot (R(D) + 1) = 160N + 160. \tag{4.4}$$

4.3.2 Нерекурсивный алгоритм

Память, затрачиваемая реализацией нерекурсивного алгоритма, в байтах рассчитана по формулам (4.5 – 4.7):

— память в байтах, затраченная на вызов функции, рассчитана по формуле (4.5):

$$M_{itercall} = 19 \cdot S_{reg} + S_{ret} = 160, \tag{4.5}$$

— память в байтах, затраченная на локальные переменные, рассчитана по формуле (4.6):

$$M_{local} = 2 \cdot S_{int} = 2 \cdot 8 = 16, \tag{4.6}$$

— общая память в байтах для нерекурсивного алгоритма рассчитана по

формуле (4.7):

$$M_{total} = M_{itercall} + M_{local} = 160 + 16 = 176. \quad (4.7)$$

4.4 Вывод

В данном разделе были описаны технические характеристики машины, на которой проводились замеры времени. Продемонстрированы результаты замеров процессорного времени, был проведён сравнительный анализ времени работы алгоритмов умножения матриц.

Также была проведена теоретическая оценка затрачиваемой реализациями алгоритмов памяти. Для работы рекурсивного алгоритма требуется больше памяти, чем для работы нерекурсивного алгоритма: асимптотика рекурсивного алгоритма – $O(n)$, а нерекурсивного – $O(1)$.

ЗАКЛЮЧЕНИЕ

В данной лабораторной работе был проведён сравнительный анализ рекурсивного и нерекурсивного алгоритмов решения задачи определения количества единиц в последовательности, состоящей из нулей и единиц, заканчивающейся числом 2.

Выполнены задачи:

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 двух алгоритмов в худшем случае;
8) сравнены результаты замеров процессорного времени и оценки трудоёмкости;
9) сделаны выводы из сравнительного анализа реализации рекурсивного и нерекурсивного алгоритмов решения одной и той же задачи по критериям ёмкостной эффективности (на материале теоретической оценки пиковой временной эффективности на материале результатов измерений).

В результате лабораторной работы было выявлено, что реализация нерекурсивного алгоритма быстрее реализации рекурсивного примерно в 3.2 раза. Для работы рекурсивного алгоритма требуется больше памяти, чем для работы нерекурсивного алгоритма: асимптотика рекурсивного алгоритма – $O(n)$, а нерекурсивного – $O(1)$.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Быкова В. Математические методы анализа рекурсивных алгоритмов. Журн. СФУ. Сер. Матем. и физ., том 1, выпуск 3, 2008. — С. 236.

2. Свейгарт Э. Рекурсивная книга о рекурсии. — Санкт-Петербург: Изд-во ПИТЕР, 2023. — С. 28.

3. Соглашение о вызовах для 64-разрядных систем. [Электронный ресурс]. — Режим доступа: `https://learn.microsoft.com/ru-ru/cpp/build/x64-calling-convention?view=msvc-170#callercallee-saved-registers`. (Дата обращения: 2025-10-10).

4. ISO/IEC 9899:1999. 2007. — С. 7.23.2.1.