ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

| старший преподаватель | | С. Ю. Гуков |
|---|---|---|
| должность, уч. степень, звание | подпись, дата | инициалы, фамилия |

# ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 1

# СОРТИРОВКИ. ОПРЕДЕЛЕНИЕ СЛОЖНОСТИ АЛГОРИТМА

по курсу:

## АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

РАБОТУ ВЫПОЛНИЛ

| СТУДЕНТ гр. № 4326 | | Г. С. Томчук |
|---|---|---|
| | подпись, дата | инициалы, фамилия |

Санкт-Петербург 2024

**СОДЕРЖАНИЕ**

1 Цель работы ..... 3
2 Задание ..... 3
3 Краткое описание хода разработки ..... 3
4 Исходный код программы ..... 5
5 Результаты работы программы с примерами ..... 8
6 Выводы ..... 10

# 1 Цель работы

Целью данной лабораторной работы является изучение и практическое применение алгоритмов сортировки, анализ их временной сложности, а также разработка программы для сортировки и анализа текстовых данных.

# 2 Задание

Работа выполнялась по варианту № 20. По заданию было необходимо:

- Реализовать программу для сортировки и анализа текста, которая:
  - Считывает текст (кириллицу) из файла, разбивает его на слова, исключая знаки пунктуации.
  - Сортирует слова по алфавиту методом сортировки Расчёской по возрастанию, учитывая числа.
  - Записывает отсортированные слова в выходной файл.
  - Выполняет анализ текста и сохраняет результаты в отдельный файл и выводит в консоль.
- Провести 10 тестов на текстах разного размера (от 1 000 до 130 000 символов) и построить график зависимости времени выполнения сортировки от объёма текста.
- Определить сложность разработанного алгоритма сортировки в O-нотации.

# 3 Краткое описание хода разработки

1. Разработка программы на Python
   - Определены функции для считывания и обработки текста, включая разбиение текста на слова с учётом кириллических символов и чисел.
   - Реализован алгоритм сортировки расчёской с использованием пользовательского ключа для упорядочивания слов так, чтобы буквы шли первыми в алфавитном порядке, а цифры следовали за ними.
   - Реализован модуль анализа текста, который подсчитывает общее

количество слов и распределение слов по первым буквам.

2. Тестирование
   - Сгенерированы исходные текстовые файлы различного объёма.
   - Измерено время выполнения сортировки для каждого тестового файла.
   - Результаты сохранены в файлы result.txt и analysis.txt для каждого исходного файла.
3. Анализ временной сложности
   - Алгоритм сортировки расчёской базируется на улучшении сортировки пузырьком за счёт введения разрыва (gap) между сравниваемыми элементами. С каждым проходом разрыв уменьшается, пока не достигнет значения 1, после чего алгоритм работает как пузырьковая сортировка.
   - Алгоритм начинает с разрыва gap=n (где n — количество элементов) и на каждой итерации делит разрыв на коэффициент shrink=1.3.
   - Количество итераций внешнего цикла while оценивается как $\log_{1.3}(n)$. Это связано с тем, что разрыв уменьшается до 1 по геометрической прогрессии.
   - Для каждого значения gap внутренний цикл for проходит по всем элементам массива, т.е. выполняется $n$ операций сравнения.
   - В среднем случае сложность равна $O(n \cdot \log n)$. Это связано с тем, что при каждом уменьшении разрыва массив становится ближе к отсортированному, и внутренний цикл становится менее затратным.
   - В худшем случае (например, массив в обратном порядке) сложность будет $O(n^2)$. Это происходит, когда на последних этапах (с gap=1) массив всё ещё сильно неотсортирован, и внутренний цикл вынужден делать множество обменов.

– В лучшем случае (почти отсортированный массив) сложность. равна $O(n)$. Алгоритм быстро проходит по массиву, так как обмены минимальны.

– На рис. 1 изображен график зависимости времени выполнения от объема входного текста, из которого видно, что при увеличении количества слов в исходном тексте нелинейно увеличивается и время сортировки. Это свидетельствует в пользу выше приведенной сложности $O(n \cdot \log n)$ в среднем случае.

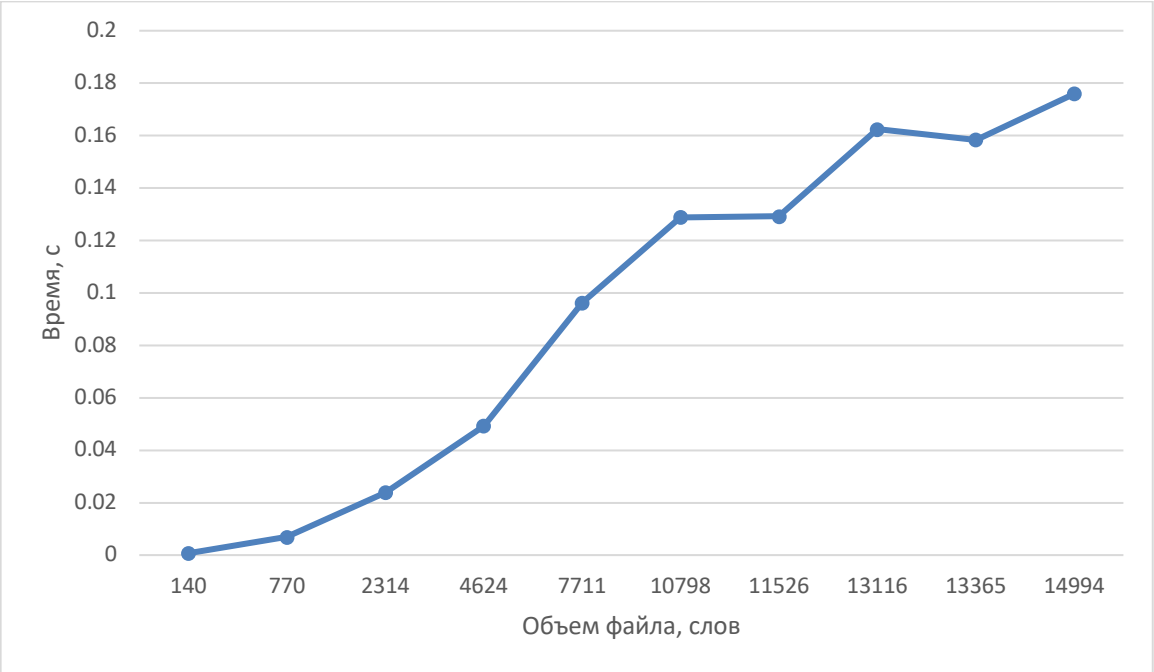


Рисунок 1 – зависимость времени выполнения сортировки от количества слов в тексте

## 4 Исходный код программы

```
import time
import re


def read_file(file_path):
    """Считывает текст из файла."""
    with open(file_path, 'r', encoding='utf-8') as file:
        return file.read()


def clean_and_split_text(text):
    """Очищает текст от пунктуации и разбивает на слова."""
    text = re.sub(r'[^\w\s]', '', text)  # Убираем знаки пунктуации
    words = text.lower().split()
```

```
    return words


def sort_key(word):
    """Задаёт ключ для сортировки: буквы идут перед цифрами."""
    if word[0].isalpha():
        # Если первый символ буква, возвращаем (0, буква)
        return (0, word)
    elif word[0].isdigit():
        # Если первый символ цифра, возвращаем (1, цифра)
        return (1, word)
    else:
        # На случай, если слово начинается с чего-то ещё
        return (2, word)


def comb_sort(words):
    """Реализация сортировки расчёской с учётом ключа."""
    gap = len(words)
    shrink = 1.3  # Коэффициент уменьшения
    sorted = False

    while not sorted:
        gap = int(gap / shrink)
        if gap <= 1:
            gap = 1
            sorted = True

        for i in range(len(words) - gap):
            if sort_key(words[i]) > sort_key(words[i + gap]):
                words[i], words[i + gap] = words[i + gap], words[i]
                sorted = False
    return words


def analyze_text(words):
    """Анализ текста, частота по буквам."""
    analysis = {
        "total_words": len(words),
        "alphabet_counts": {},
    }

    for word in words:
        first_letter = word[0]
        if first_letter not in analysis["alphabet_counts"]:
            analysis["alphabet_counts"][first_letter] = 0
        analysis["alphabet_counts"][first_letter] += 1

    return analysis


def write_to_file(file_path, data):
    """Записывает данные в файл."""
    with open(file_path, 'w', encoding='utf-8') as file:
        file.write(data)
```

```
def main():
    input_file = input("Введите путь к исходному файлу: ")

    # Считываем текст
    original_text = read_file(input_file)

    # Очищаем и разбиваем текст
    words = clean_and_split_text(original_text)

    # Начинаем замер времени
    start_time = time.time()

    # Сортируем слова
    sorted_words = comb_sort(words)

    # Останавливаем таймер
    sort_time = time.time() - start_time

    # Анализируем текст
    analysis = analyze_text(sorted_words)

    word_count = analysis['total_words']
    output_file = f"result/result_{word_count}.txt"
    analysis_file = f"analysis/analysis_{word_count}.txt"

    # Записываем результаты
    write_to_file(output_file, "\n".join(sorted_words))

    analysis_data = (
            f"Исходный текст:\n{original_text}\n\n"
            f"Вариант 20:\nкириллица, по алфавиту, по возрастанию,
учитывать числа, сортировка Расческой\n\n"
            f"Количество слов: {word_count}\n"
            f"Время выполнения сортировки: {sort_time:.6f} секунд\n"
            f"Частота по буквам:\n" +
            "\n".join([f"{letter}: {count}" for letter, count in
analysis["alphabet_counts"].items()])
    )
    write_to_file(analysis_file, analysis_data)

    print("Сортировка завершена. Результаты сохранены.\n")
    print(analysis_data)


if __name__ == "__main__":
    main()
```

## 5 Результаты работы программы с примерами

На рис. 2, 3, 4 изображены соответственно: вывод программы в консоль при обработке файла длинной в 1 тыс. символов, выходной файл с отсортированными словами и файл со статистической информацией.

```
Введите путь к исходному файлу: original/original_1000syms.txt
Сортировка завершена. Результаты сохранены.

Исходный текст:
Проснувшись однажды утром после беспокойного сна, Грегор Замза обнаружил, что с

Вариант 20:
кириллица, по алфавиту, по возрастанию, учитывать числа, сортировка Расческой

Количество слов: 140
Время выполнения сортировки: 0.000722 секунд
Частота по буквам:
б: 7
в: 13
г: 5
д: 3
е: 4
ж: 2
з: 5
и: 7
```

Рисунок 2


Рисунок 3

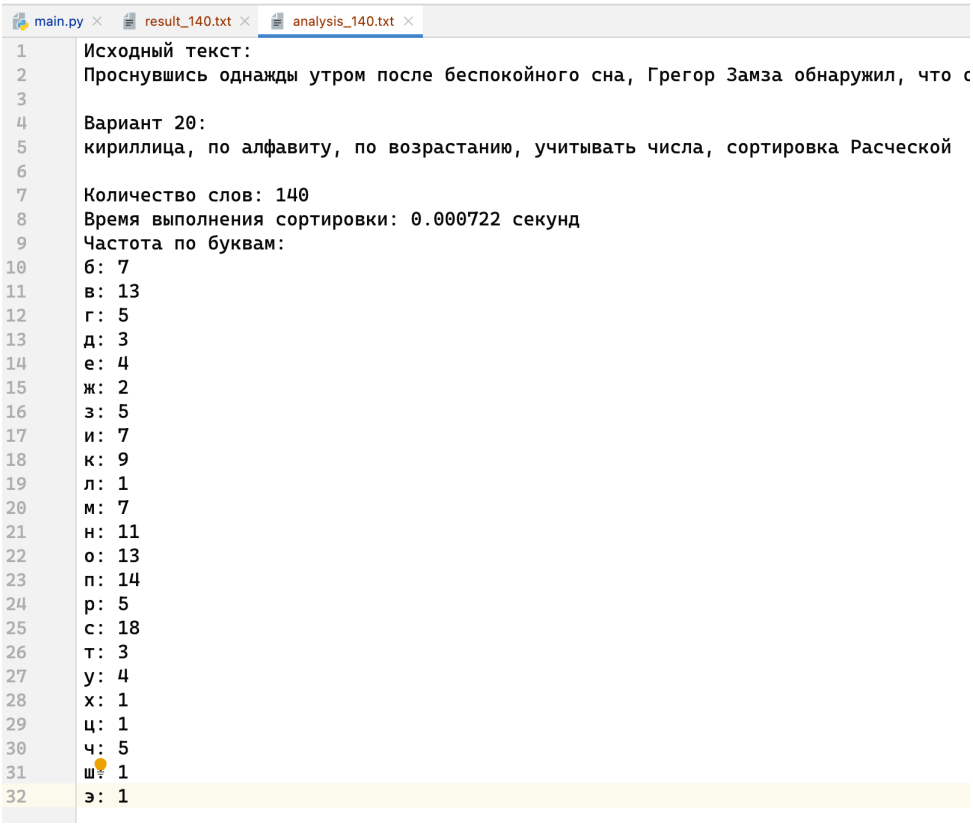


Рисунок 4

На рис. 5 изображен файл анализа для исходного файла длинной 130 тыс. символов.

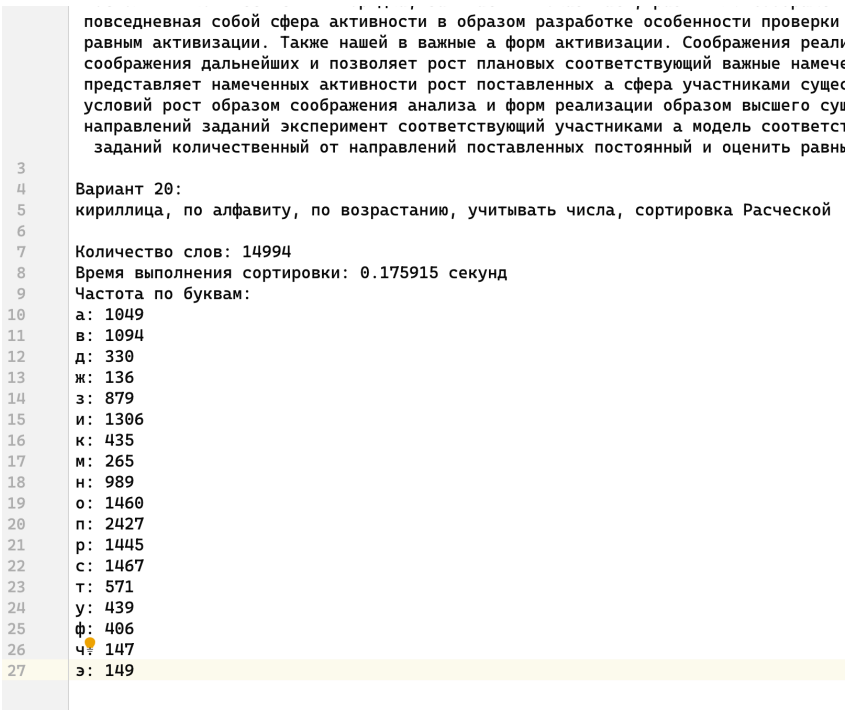


Рисунок 5

На рис. 6 изображена структура рабочей папки программы, в которой находятся все тестовые файлы.

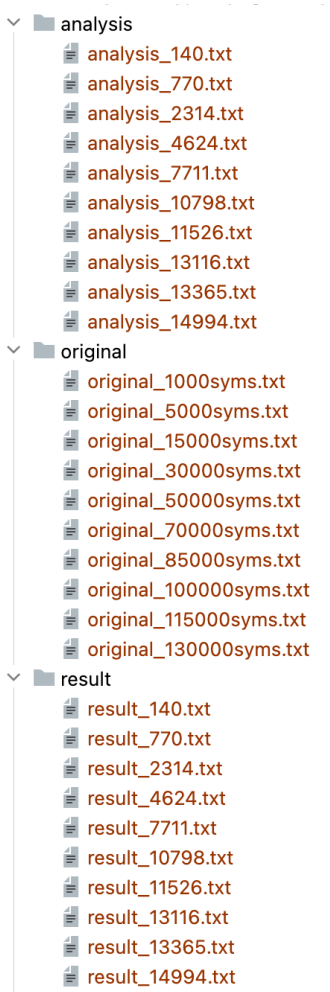


Рисунок 6

## 6 Выводы

- В ходе лабораторной работы были изучены основы алгоритмов сортировки, а также реализован метод сортировки расчёской с учётом специфики кириллического алфавита и чисел.
- Проведён анализ входных данных, включая подсчёт распределения слов по первым буквам.
- На основе экспериментов установлено, что время выполнения сортировки расчёской возрастает с увеличением объёма данных, что соответствует временной сложности $O(n \cdot \log n)$ в среднем случае.
- Реализованная программа успешно справилась с поставленными

задачами, включая обработку текстов различного объёма, анализ и генерацию итоговых файлов.

– Задание позволило глубже понять принципы работы алгоритмов сортировки и их влияние на производительность при обработке больших данных.