Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Российский химико-технологический университет имени Д.И. Менделеева»

Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2

Выполнил студент группы КС-30 Положенцев Аксель Алексеевич

Ссылка на репозиторий: [MUCTR-IKT-CPP/PolozhencevAA\_30](https://github.com/MUCTR-IKT-CPP/PolozhencevAA_30)

Приняли: Пысин Максим Дмитриевич

Краснов Дмитрий Олегович

Лобанов Алексей Владимирович

Крашенинников Роман Сергеевич

Дата сдачи: 17.02.2025

Москва 2025

Оглавление

[Описание задачи 3](#_Toc191294879)

[Описание метода 3](#_Toc191294880)

[Выполнение задачи 4](#_Toc191294881)

[Заключение 9](#_Toc191294882)

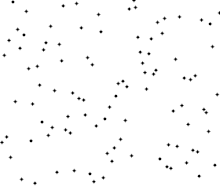
# Описание задачи

В лабораторной работе предлагается изучить альтернативные первой лабораторной сортировки, которые обладают меньшей асимтотической сложносью и сравнить их с результатами предыдущей лабораторной работы. Используя предыдущий код посерийного выполнения алгоритма сортировки и измерения времени требутеся реализовать метод сортировки слиянием.

# Описание метода

**сортировки слиянием**

Подробный алгоритм сортировки

[](https://ru.wikipedia.org/commons.wikimedia.org/wiki/File:Merge_sort_animation2.gif?uselang=ru)

Действие алгоритма на примере сортировки случайных точек.

Для решения задачи сортировки эти три этапа выглядят так:

1. Сортируемый [массив](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%81%D0%B8%D0%B2_(%D1%82%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)) разбивается на две части примерно одинакового размера;
2. Каждая из получившихся частей сортируется отдельно, например — тем же самым алгоритмом;
3. Два упорядоченных массива половинного размера соединяются в один.

1.1. — 2.1. Рекурсивное разбиение задачи на меньшие происходит до тех пор, пока размер массива не достигнет единицы (любой массив длины 1 можно считать упорядоченным).

3.1. Соединение двух упорядоченных массивов в один.  
Основную идею слияния двух отсортированных массивов можно объяснить на следующем примере. Пусть мы имеем два уже отсортированных по возрастанию подмассива. Тогда:

3.2. Слияние двух подмассивов в третий результирующий массив.  
На каждом шаге мы берём меньший из двух первых элементов подмассивов и записываем его в результирующий массив. Счётчики номеров элементов результирующего массива и подмассива, из которого был взят элемент, увеличиваем на 1.

3.3.«Прицепление» остатка. Когда один из подмассивов закончился, мы добавляем все оставшиеся элементы второго подмассива в результирующий массив.

# Выполнение задачи

**Решение задачи**

1. Алгоритм сортировки слиянием основан на принципе "разделяй и властвуй". Он работает следующим образом:

* Если массив содержит 1 или 0 элементов, он уже отсортирован.
* Массив делится пополам, и каждая половина сортируется рекурсивно.
* После сортировки обе половины сливаются в один отсортированный массив.

2. Реализация

Код реализации включает в себя следующие основные части:

Python

import time

import tracemalloc

from random import uniform

recursion\_calls = 0

max\_recursion\_depth = 0

merge\_operations = 0

def merge\_sort(nums, depth=0):

    global recursion\_calls, max\_recursion\_depth, merge\_operations

    recursion\_calls += 1

    max\_recursion\_depth = max(max\_recursion\_depth, depth)

    if len(nums) <= 1:

        return nums

    mid = len(nums) // 2

    left = merge\_sort(nums[:mid], depth + 1)

    right = merge\_sort(nums[mid:], depth + 1)

    return merge(left, right)

def merge(left, right):

    global merge\_operations

    sorted\_list = []

    i = j = 0

    while i < len(left) and j < len(right):

        if left[i] < right[j]:

            sorted\_list.append(left[i])

            i += 1

        else:

            sorted\_list.append(right[j])

            j += 1

        merge\_operations += 1

    sorted\_list.extend(left[i:])

    sorted\_list.extend(right[j:])

    merge\_operations += len(left) - i + len(right) - j

    return sorted\_list

if \_\_name\_\_ == "\_\_main\_\_":

    dimensions = [1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000]

    for dimension in dimensions:

        for approach in range(20):

            nums = [uniform(-1, 1) for \_ in range(dimension)]

            # Сброс счетчиков

            recursion\_calls = 0

            max\_recursion\_depth = 0

            merge\_operations = 0

            # Замер памяти до сортировки

            tracemalloc.start()

            before\_memory = tracemalloc.get\_traced\_memory()[1]  # Пик до начала

            # Замер времени

            start\_time = time.time()

            sorted\_nums = merge\_sort(nums)

            end\_time = time.time()

            execution\_time = end\_time - start\_time

            # Замер памяти после сортировки

            after\_memory = tracemalloc.get\_traced\_memory()[1]  # Пик после завершения

            tracemalloc.stop()

            additional\_memory = after\_memory - before\_memory

            # Запись результатов в файл

            with open("results.txt", "a", encoding='UTF-8') as file:

                file.write(

                    f"Размерность: {dimension} "

                    f"Время: {execution\_time:.6f} "

                    f"Количество вызовов рекурсии: {recursion\_calls} "

                    f"Глубина рекурсии: {max\_recursion\_depth} "

                    f"Количество операций слияния: {merge\_operations} "

                    f"Доп. память: {additional\_memory} байт\n"

                )

3. Запуск и результаты

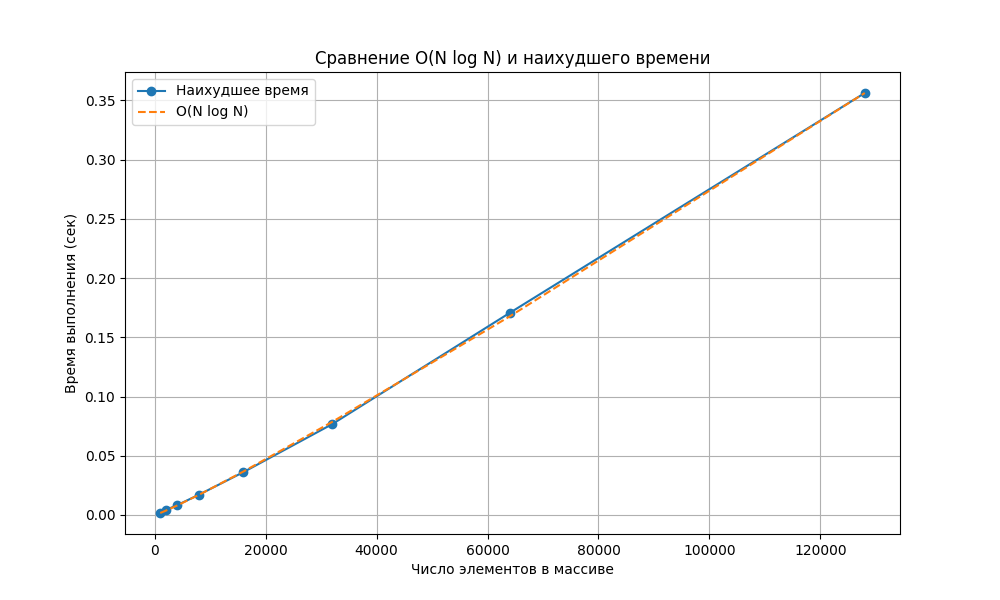
* Код выполняется для массивов различных размерностей: от 1000 до 128000 элементов.
* Для каждой размерности производится 20 итераций генерации случайных чисел, что позволяет оценить стабильность алгоритма и его производительность.
* Результаты записываются в файл results.txt, где указаны размерность массива, время выполнения, количество рекурсивных вызовов, максимальная глубина рекурсии, количество операций слияния и дополнительная память, использованная алгоритмом.

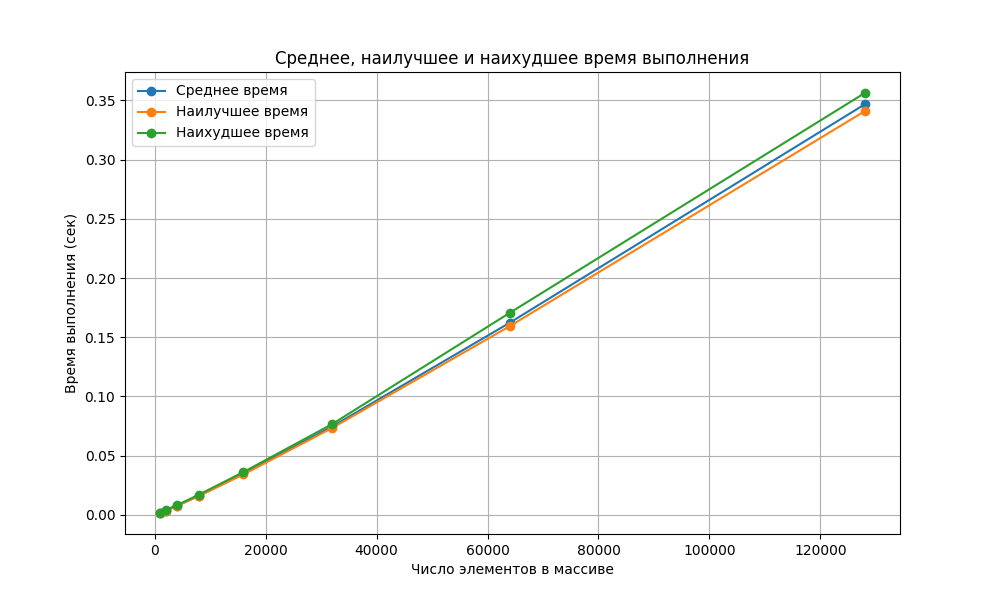
4. Анализ производительности

На основании собранных данных провели анализ производительности алгоритма. Важно обратить внимание на следующие аспекты:

* Совмещенный график среднего, наилучшего и наихудшего времени исполнения.
* Совмещенный график среднего, наилучшего и наихудшего глубины рекурсии.
* Совмещенный график среднего, наилучшего и наихудшего потребления дополнительной памяти.
* Совмещенный график среднего, наилучшего и наихудшего вызовов рекурсивной функции.

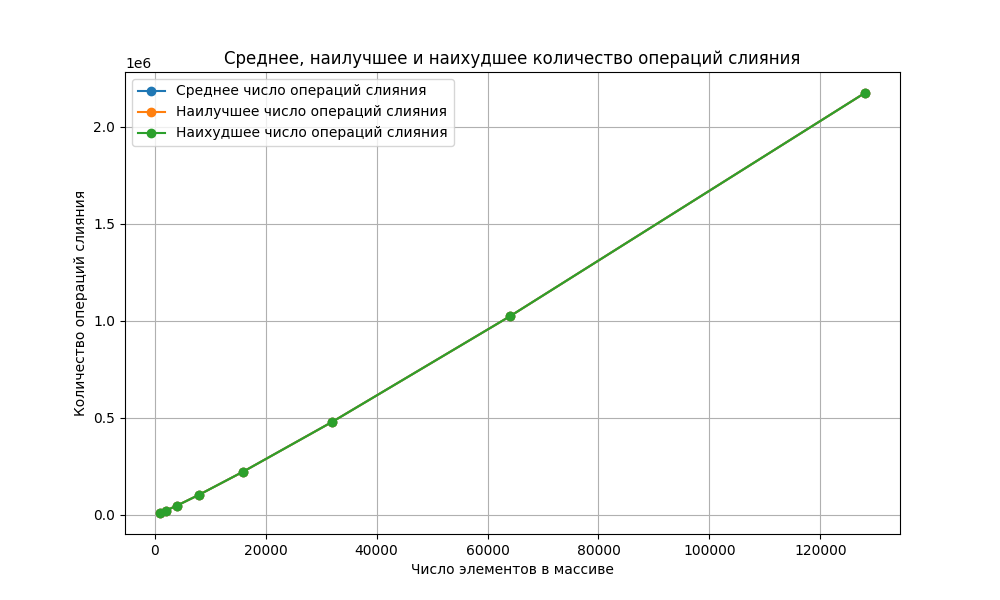
**Графики приведены ниже.**

****

****

****

****

****

# Заключение

В ходе выполнения работы был реализован и протестирован алгоритм сортировки слиянием. Проведенные эксперименты позволили оценить его производительность по ключевым параметрам: время выполнения, количество рекурсивных вызовов, максимальная глубина рекурсии, количество операций слияния и объем дополнительной памяти.

Результаты показали, что алгоритм демонстрирует ожидаемую временную сложность **O(n log n)**. Однако с увеличением размерности входных данных время выполнения увеличивается не только за счет роста количества операций, но и из-за затрат на рекурсивные вызовы и выделение дополнительной памяти для хранения временных массивов.

На основе собранных данных можно сделать следующие выводы:

1. **Эффективность сортировки** – алгоритм стабильно выполняет сортировку даже для больших массивов, подтверждая свою надежность.
2. **Рекурсия** – увеличение размерности массива приводит к росту числа рекурсивных вызовов и глубины рекурсии, что следует учитывать при использовании в ограниченных по стеку средах.
3. **Операции слияния** – основной вклад в вычислительную сложность алгоритма вносит этап слияния, что подтверждается высоким числом операций слияния.
4. **Использование памяти** – дополнительная память необходима для временных массивов, и ее потребление также растет с увеличением размерности входных данных.

Таким образом, алгоритм сортировки слиянием является хорошим выбором для задач, требующих стабильной сортировки с предсказуемым временем выполнения, однако для случаев, когда критично минимальное использование памяти, могут потребоваться альтернативные методы, например, быстрая сортировка или сортировка слиянием без выделения дополнительных массивов.