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 Насидлецкая А. Н.

Ссылка на репозиторий: https://github.com/annanas-dev

Приняли: Пысин Максим Дмитриевич

Краснов Дмитрий Олегович

Лобанов Алексей Владимирович

Крашенинников Роман Сергеевич

Дата сдачи: 24.02.2025

Оглавление

[Описание задачи. 2](#_Toc63548272)

[Описание метода/модели. 2](#_Toc63548273)

[Выполнение задачи. 2](#_Toc63548274)

[Заключение. 2](#_Toc63548275)

# Описание задачи.

В лабораторной работе предлагается изучить альтернативные первой лабораторной сортировки, которые обладают меньшей асимтотической сложносью и сравнить их с результатами предыдущей лабораторной работы.

# Описание метода/модели.

**Сортировка слиянием** (англ. Merge sort) — алгоритм сортировки, использующий O(n) дополнительной памяти и работающий за O(nlog(n)) времени.

O(nlog(n)) - лучшее, худшее и среднее время сортировки.

Память O(n).

**Принцип работы:**

Алгоритм использует принцип «разделяй и властвуй»: задача разбивается на подзадачи меньшего размера, которые решаются по отдельности, после чего их решения комбинируются для получения решения исходной задачи. Конкретно процедуру сортировки слиянием можно описать следующим образом:

1. Если в рассматриваемом массиве один элемент, то он уже отсортирован — алгоритм завершает работу.
2. Иначе массив разбивается на две части, которые сортируются рекурсивно.
3. После сортировки двух частей массива к ним применяется процедура слияния, которая по двум отсортированным частям получает исходный отсортированный массив.

**Алгоритм:**

У нас есть два массива a и b (фактически это будут две части одного массива, но для удобства будем писать, что у нас просто два массива). Нам надо получить массив cc размером |a|+|b|. Для этого можно применить процедуру слияния. Эта процедура заключается в том, что мы сравниваем элементы массивов (начиная с начала) и меньший из них записываем в финальный. И затем, в массиве у которого оказался меньший элемент, переходим к следующему элементу и сравниваем теперь его. В конце, если один из массивов закончился, мы просто дописываем в финальный другой массив. После мы наш финальный массив записываем заместо двух исходных и получаем отсортированный участок.

# Выполнение задачи.

**Язык программирования**

Для решения задачи использован язык программирования C++. Этот язык выбран за его высокую производительность, удобство работы с низкоуровневыми операциями и встроенные возможности для работы с динамическими структурами данных, такими как векторы.

**Организация программы**

* **Реализация сортировки слиянием (merge sort):**

1. Сортировка выполняется через рекурсивную функцию mergeSortRecursive. Эта функция делит массив на две части и сортирует каждую из них рекурсивно, а затем сливает отсортированные части в один массив.
2. Для сортировки используется вспомогательный массив helper, чтобы избежать излишних копий данных при слиянии.
3. В процессе работы отслеживаются несколько метрик: количество рекурсивных вызовов (calls), максимальная глубина рекурсии (maxDepth) и количество используемой дополнительной памяти (additionalMemory).

void mergeSortRecursive(vector<double>& arr, vector<double>& helper, size\_t left, size\_t right,

size\_t depth, size\_t& calls, size\_t& maxDepth, size\_t& additionalMemory) {

calls++;

maxDepth = max(maxDepth, depth);

if (left >= right) return;

size\_t mid = left + (right - left) / 2;

mergeSortRecursive(arr, helper, left, mid, depth + 1, calls, maxDepth, additionalMemory);

mergeSortRecursive(arr, helper, mid + 1, right, depth + 1, calls, maxDepth, additionalMemory);

size\_t i = left, j = mid + 1, k = left;

while (i <= mid && j <= right) {

helper[k++] = arr[i] <= arr[j] ? arr[i++] : arr[j++];

}

while (i <= mid) helper[k++] = arr[i++];

while (j <= right) helper[k++] = arr[j++];

const size\_t usedMemory = (right - left + 1) \* sizeof(double);

additionalMemory = max(additionalMemory, usedMemory);

copy(helper.begin() + left, helper.begin() + right + 1, arr.begin() + left);

}

* **Генерация случайных данных:**

Для тестирования используется функция generateArray, которая генерирует массив случайных чисел в диапазоне от -1 до 1.

* **Вычисление среднего значения:**

Функция average вычисляет среднее значение для переданного вектора данных.

* **Бенчмаркинг:**

Функция runBenchmark запускает сортировку для нескольких размеров массивов, выполняя заданное количество попыток для каждого размера. Для каждой попытки измеряется время выполнения сортировки и собираются данные о рекурсивных вызовах, глубине рекурсии и объеме дополнительной памяти.

Результаты (среднее, минимальное и максимальное значения для времени, количества вызовов, глубины и памяти) записываются в файл merge\_sort\_results.txt.

* **Основная программа:**

В функции main задаются размеры массивов для тестирования и количество попыток для каждого размера. Затем вызывается функция runBenchmark, которая выполняет тестирование и записывает результаты в файл.

**Размер массива: 1000**

**Среднее время (мс): 1.440 | Лучшее: 1.398 | Худшее: 1.710**

**Кол-во вызовов рекурсии Сред.: 1999 | Лучшее: 1999 | Худшее: 1999**

**Глубина рекурсии Сред.: 11 | Лучшая: 11 | Худшая: 11**

**Доп. память (байт) Сред.: 8000 | Лучшая: 8000 | Худшая: 8000**

**Размер массива: 2000**

**Среднее время (мс): 2.879 | Лучшее: 2.843 | Худшее: 2.960**

**Кол-во вызовов рекурсии Сред.: 3999 | Лучшее: 3999 | Худшее: 3999**

**Глубина рекурсии Сред.: 12 | Лучшая: 12 | Худшая: 12**

**Доп. память (байт) Сред.: 16000 | Лучшая: 16000 | Худшая: 16000**

**Размер массива: 4000**

**Среднее время (мс): 6.024 | Лучшее: 5.783 | Худшее: 7.462**

**Кол-во вызовов рекурсии Сред.: 7999 | Лучшее: 7999 | Худшее: 7999**

**Глубина рекурсии Сред.: 13 | Лучшая: 13 | Худшая: 13**

**Доп. память (байт) Сред.: 32000 | Лучшая: 32000 | Худшая: 32000**

**Размер массива: 8000**

**Среднее время (мс): 11.978 | Лучшее: 11.811 | Худшее: 12.435**

**Кол-во вызовов рекурсии Сред.: 15999 | Лучшее: 15999 | Худшее: 15999**

**Глубина рекурсии Сред.: 14 | Лучшая: 14 | Худшая: 14**

**Доп. память (байт) Сред.: 64000 | Лучшая: 64000 | Худшая: 64000**

**Размер массива: 16000**

**Среднее время (мс): 24.120 | Лучшее: 23.937 | Худшее: 24.667**

**Кол-во вызовов рекурсии Сред.: 31999 | Лучшее: 31999 | Худшее: 31999**

**Глубина рекурсии Сред.: 15 | Лучшая: 15 | Худшая: 15**

**Доп. память (байт) Сред.: 128000 | Лучшая: 128000 | Худшая: 128000**

**Размер массива: 32000**

**Среднее время (мс): 49.282 | Лучшее: 48.547 | Худшее: 52.889**

**Кол-во вызовов рекурсии Сред.: 63999 | Лучшее: 63999 | Худшее: 63999**

**Глубина рекурсии Сред.: 16 | Лучшая: 16 | Худшая: 16**

**Доп. память (байт) Сред.: 256000 | Лучшая: 256000 | Худшая: 256000**

**Размер массива: 64000**

**Среднее время (мс): 100.108 | Лучшее: 99.104 | Худшее: 102.880**

**Кол-во вызовов рекурсии Сред.: 127999 | Лучшее: 127999 | Худшее: 127999**

**Глубина рекурсии Сред.: 17 | Лучшая: 17 | Худшая: 17**

**Доп. память (байт) Сред.: 512000 | Лучшая: 512000 | Худшая: 512000**

**Размер массива: 128000**

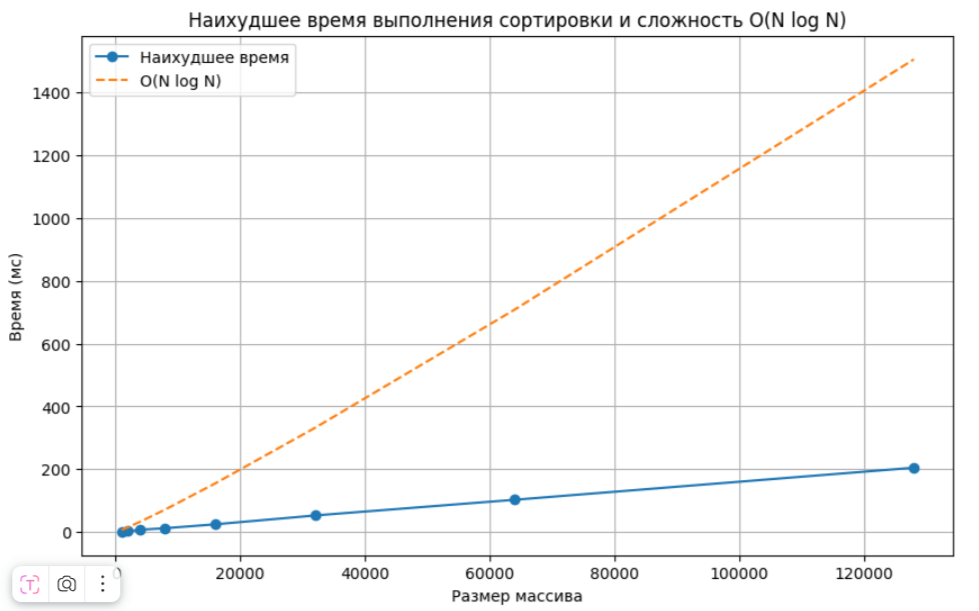
**Среднее время (мс): 203.025 | Лучшее: 202.105 | Худшее: 205.001**

**Кол-во вызовов рекурсии Сред.: 255999 | Лучшее: 255999 | Худшее: 255999**

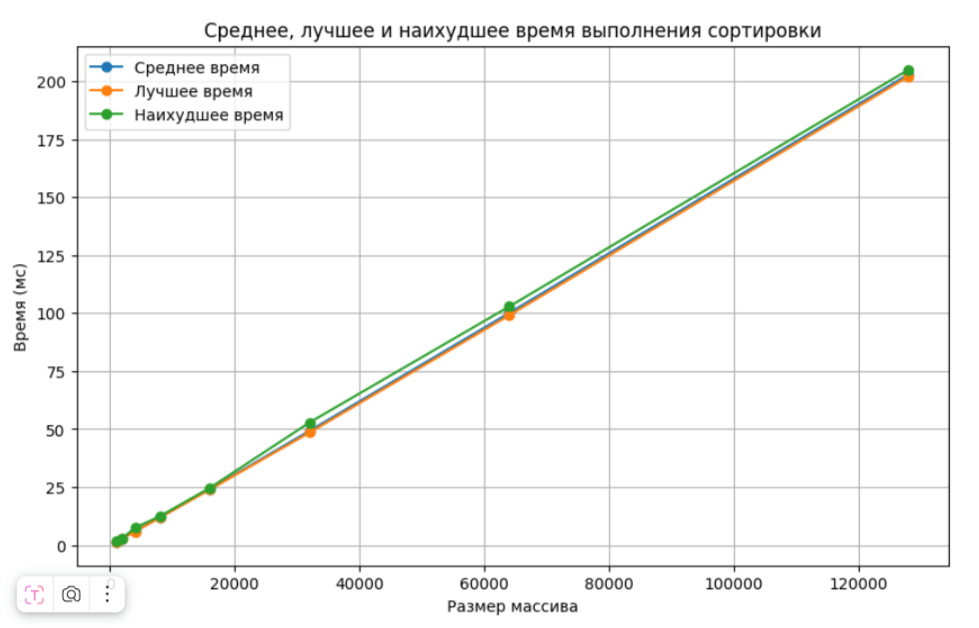
**Глубина рекурсии Сред.: 18 | Лучшая: 18 | Худшая: 18**

**Доп. память (байт) Сред.: 1024000 | Лучшая: 1024000 | Худшая: 102400**

1. Совмеotyysq график наихудшего времени выполнения сортировки и сложности алгоритма указанной в нотации O большое.



1. Совмещенный график среднего, наилучшего и наихудшего времени исполнения.



Между размером массива и временем работы наблюдается зависимость близкая к линейной. Время работы почти удваивается при удваивании размера массива (например, 1.44 мс для 1000 элементов и 2.88 мс для 2000 элементов).Поскольку сортировка слиянием имеет сложность O(n log n), время роста увеличивается быстрее, чем линейно, но медленнее, чем квадратично.

Разница между лучшим и худшим временем работы в каждой группе незначительна, что говорит о стабильности алгоритма. Сортировка слиянием всегда работает с предсказуемым временем, даже в худших случаях.

1. Совмещенный график среднего, наилучшего и наихудшего глубины рекурсии.

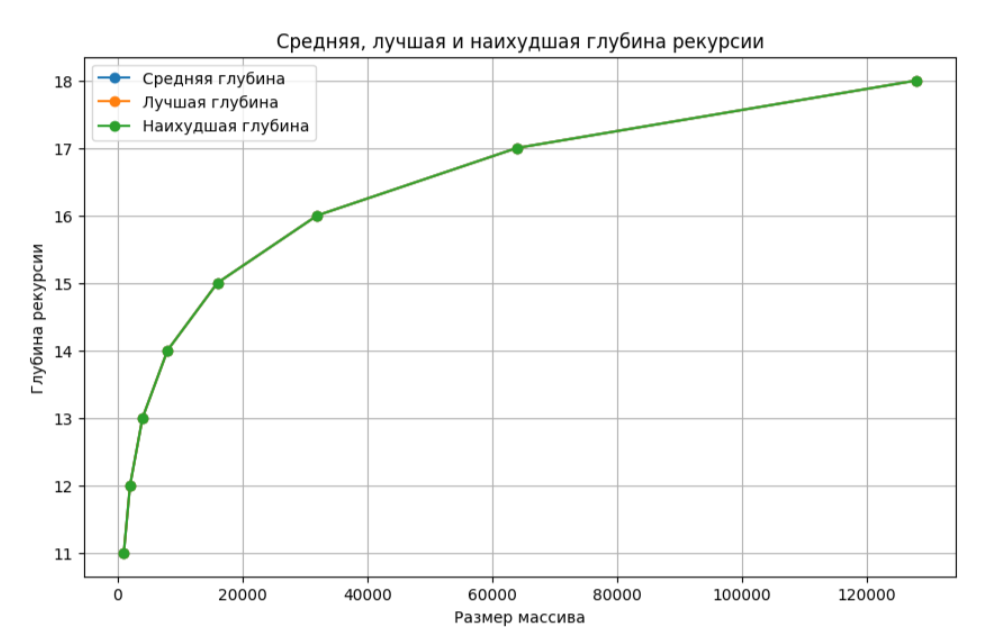
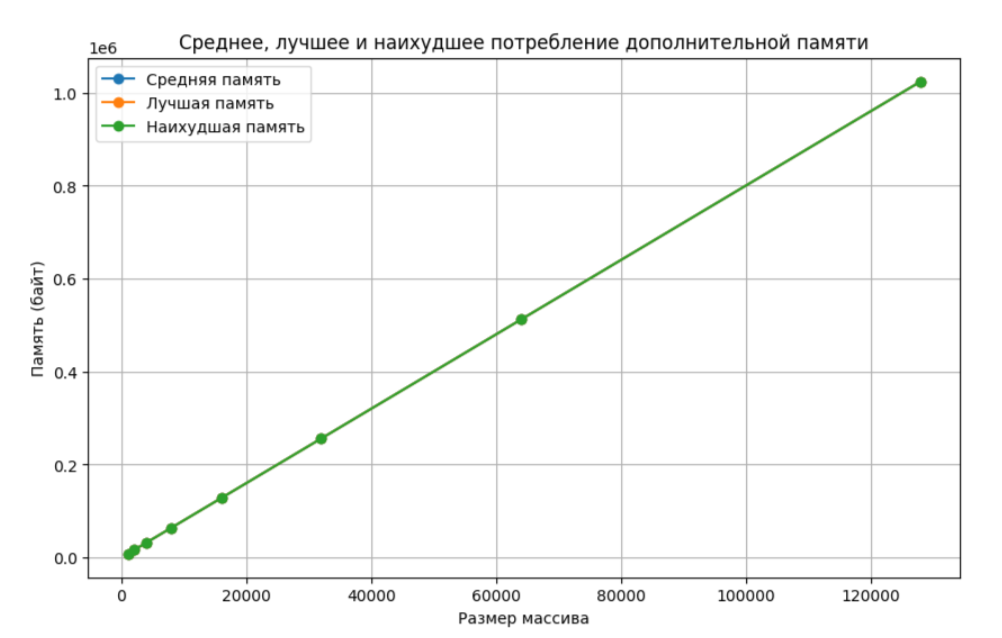


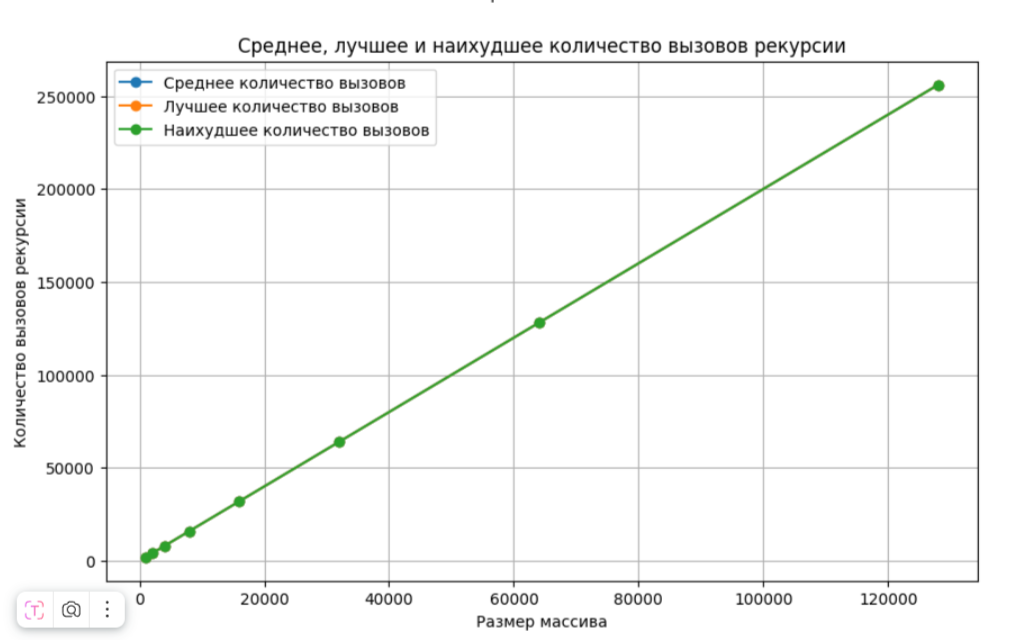
График подтверждает логарифмическую зависимость глубины рекурсии от размера массива, что также отражает сложность O(n log n) сортировки слиянием. Глубина рекурсии равна log2n, количество раз, которое массив делется на два, степень двойки. Это отражает особенности работы алгоритма.

1. Совмещенный график среднего, наилучшего и наихудшего потребления дополнительной памяти.



Между доп. памятью и размером массива наблюдается линейная зависимость. С увеличением размера массива на два, увеличивается и используемая доп. память. Это обусловлено работой алгоритма - во избежании излишнего копирования создается вспомогательный массив (helper), размер которого равен раземру сортируемого массива. Это подтверждает теоретическое значение используемой доп. памяти O(n),

1. Совмещенный график среднего, наилучшего и наихудшего вызовов рекурсивной функции.



Между количеством вызовов рекурсии и размером массива наблюдается линейная зависимость. Количество вызовов рекурсии равно 2\*n - 1. Суть в том, что на каждом уровне рекурсии количество вызовов удваивается: если на одном уровне рекурсии у нас 1 массив, то на следующем уровне будет 2 массива, потом 4, и так далее. Поскольку деление продолжается до тех пор, пока не будут достигнуты массивы размера 1, мы видим, что для массива размера n будет 2n−1 вызовов рекурсии (так как на последнем уровне рекурсии будет происходить n вызовов слияния, а выше — соответственно меньше).

# Заключение.

В заключение, сортировка слиянием (Merge Sort) — это эффективный алгоритм сортировки с гарантированной сложностью O(n log n), который работает по принципу «разделяй и властвуй». В ходе выполнения алгоритм разделяет массив на подмассивы, сортирует их и сливает обратно в отсортированную последовательность. Основными характеристиками алгоритма являются стабильность работы в различных случаях, линейная зависимость дополнительной памяти от размера массива и логарифмическая глубина рекурсии. Однако из-за использования дополнительной памяти алгоритм может быть менее эффективен в некоторых ограниченных по памяти системах.