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-36 (Потапов Никита Александрович)

Ссылка на репозиторий: (Ссылка на лабораторную в репозитории)

Приняли: Пысин Максим Дмитриевич

Краснов Дмитрий Олегович

Лобанов Алексей Владимирович

Крашенинников Роман Сергеевич

Дата сдачи: (03.03.2025)

Оглавление

[Описание задачи. 2](#_Toc63548272)

[Описание метода/модели. 3](#_Toc63548273)

[Выполнение задачи. 5](#_Toc63548274)

[Заключение. 9](#_Toc63548275)

# Описание задачи.

Используя предыдущий код посерийного выполнения алгоритма сортировки и измерения времени из лабораторной №1 требуется реализовать метод **быстрой сортировки.**

Реализовать проведения тестирования алгоритма сериями расчетов для измерения параметров времени.

За один расчет выполняется следующие операции:

1. Генерируется массив случайных значений

2. Запоминается время начала расчета алгоритма сортировки

3. Вполняется алгоритм сортировки

4. Вычисляется время затраченное на сортировку: текущее время - время начала

5. Сохраняется время для одной попытки

После этого расчет повторяется до окончания серии.

- Алгоритм вычисляется 8 сериями по 20 раз за серию.

- Алгоритм в каждой серии вычисляется для массива размером M. (1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000)

- Массив заполняется значения числами с плавающей запятой в интервале от -1 до 1.

- Для серии запоминаются все времена которые были замерены

- На основании реализовать проверки негативных случаев и устроить на них серии тестов аналогичные второму пункту:

- Отсортированный массив

- Массив с одинаковыми элементами

- Массив с максимальным количеством сравнений при выборе среднего элемента в качестве опорного

- Массив с максимальным количеством сравнений при детерминированном выборе опорного элемента

При работе сортировки подсчитать:

- колличество вызовов рекурсивной функции

- глубину рекурсии

- колличество операций обмена значений

- времени исполнения сортировки

- По полученным данным времени построить графики зависимости времени от числа элементов в массиве:

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

> Для построения графика вычисляется O большое для каждого размера массива. При этом при вычислении функции O(c \* g(N)) подбирается такая константа с, что бы при значении > 1000 график O(N) был выше графика наихудшего случая, но второй график на его фоне не превращался в прямую линию

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

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

4. Совмещенный график среднего, наилучшего и наихудшего колличества обменов значений.

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

- По результатом расчетов оформляется отчет по предоставленной форме, в отчете:

1. Приводится описание алгоритма.

2. Приводится описания выполнения задачи (Описание кода и спецефических элементов реализации).

3. Приводятся выводы (Графики и их анализ). Требуется ответить на вопрос о поведении алгоритма изученного в процессе выполнения лаборатной работы и зафиксировать его особенности.

# Описание метода/модели.

Быстрая сортировка (QuickSort) — это эффективный алгоритм сортировки, основанный на принципе "разделяй и властвуй". Алгоритм работает следующим образом:

1. Выбор опорного элемента: Из массива выбирается элемент, который будет использоваться в качестве опорного (pivot). В данном случае рассматривается выбор среднего элемента в качестве опорного.

2. Разделение массива: Массив разделяется на две части: элементы, меньшие опорного, и элементы, большие опорного.

3. Рекурсивная сортировка: Рекурсивно применяется быстрая сортировка к каждой из двух частей массива.

4. Слияние: Так как элементы уже разделены на две части относительно опорного, массив оказывается отсортированным после завершения рекурсивных вызовов.

**Сложность алгоритма**

- Средний случай: В среднем случае быстрая сортировка имеет временную сложность (O(n log n)), где (n) — количество элементов в массиве. Это делает алгоритм очень эффективным для большинства практических задач.

- Худший случай: В худшем случае (например, когда опорный элемент всегда выбирается как минимальный или максимальный элемент массива), временная сложность может достигать (O(n^2)). Однако, при правильном выборе опорного элемента (например, среднего элемента), вероятность худшего случая значительно снижается.

**Потребление памяти**

Быстрая сортировка требует (O(\log n)) дополнительной памяти для хранения стека рекурсивных вызовов. Это делает алгоритм достаточно экономичным с точки зрения использования памяти.

**Преимущества алгоритма**

1. Высокая производительность: В среднем случае быстрая сортировка является одним из самых быстрых алгоритмов сортировки.

2. Экономия памяти: Алгоритм использует небольшое количество дополнительной памяти, что делает его пригодным для использования в системах с ограниченными ресурсами.

3. Простота реализации: Алгоритм относительно прост в реализации, особенно при использовании рекурсии.

**Проблемы алгоритма**

1. Худший случай: В худшем случае производительность алгоритма может значительно ухудшиться, хотя это маловероятно при правильном выборе опорного элемента.

2. Неустойчивость: Быстрая сортировка является неустойчивым алгоритмом, то есть она может изменить относительный порядок элементов с одинаковыми значениями.

# 

# Выполнение задачи.

**Практическое описание решения**

Программа реализует алгоритм быстрой сортировки (Quick Sort) для массивов случайных чисел в интервале от -1 до 1.

**Основные задачи программы:**

* Генерация массивов случайных чисел заданного размера.
* Сортировка массивов с измерением времени выполнения, количества обменов (swaps) и количества проходов (passes).
* Сохранение результатов в текстовые файлы для последующего анализа и построения графиков.

**Используемый язык**

Программа написана на языке C++. Используются стандартные библиотеки:

* <iostream> для ввода-вывода.
* <vector> для работы с динамическими массивами.
* <random> для генерации случайных чисел.
* <chrono> для измерения времени выполнения.
* <fstream> для записи данных в файлы.

Программа состоит из следующих компонентов:

Функция generate\_random\_array: Генерирует массив случайных чисел типа double в интервале от -1 до 1. Использует генератор случайных чисел mt19937 и равномерное распределение uniform\_real\_distribution.

Функция quick\_sort: Реализует алгоритм сортировки вставками. Принимает массив и два счетчика: swaps (количество обменов) и passes (количество проходов). В процессе сортировки обновляет значения счетчиков.

Функция main: Задает параметры для тестирования: количество попыток на серию (attempts\_per\_series) и размеры массивов (array\_sizes). Для каждого размера массива генерирует случайные данные, сортирует их и измеряет время выполнения, количество обменов и проходов. Результаты записываются в три файла: times.txt, swaps.txt, depth.txt.

**Проведенные тесты**

Программа проводит тестирование на массивах следующих размеров:

1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000.

Для каждого размера массива выполняется 20 попыток (задается параметром attempts\_per\_series).

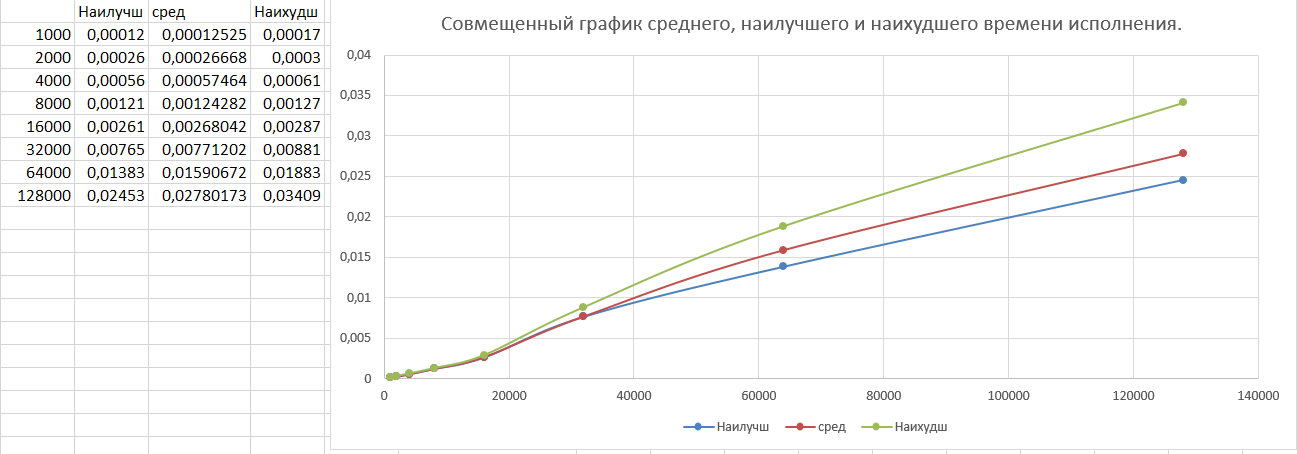
В процессе тестирования измеряются:

* Время выполнения сортировки (в секундах).
* Количество обменов (swaps) — сколько раз элементы массива менялись местами
* Количество вызова рекурсии (passes) — сколько раз алгоритм проходил по массиву.
* Глубина рекурсии

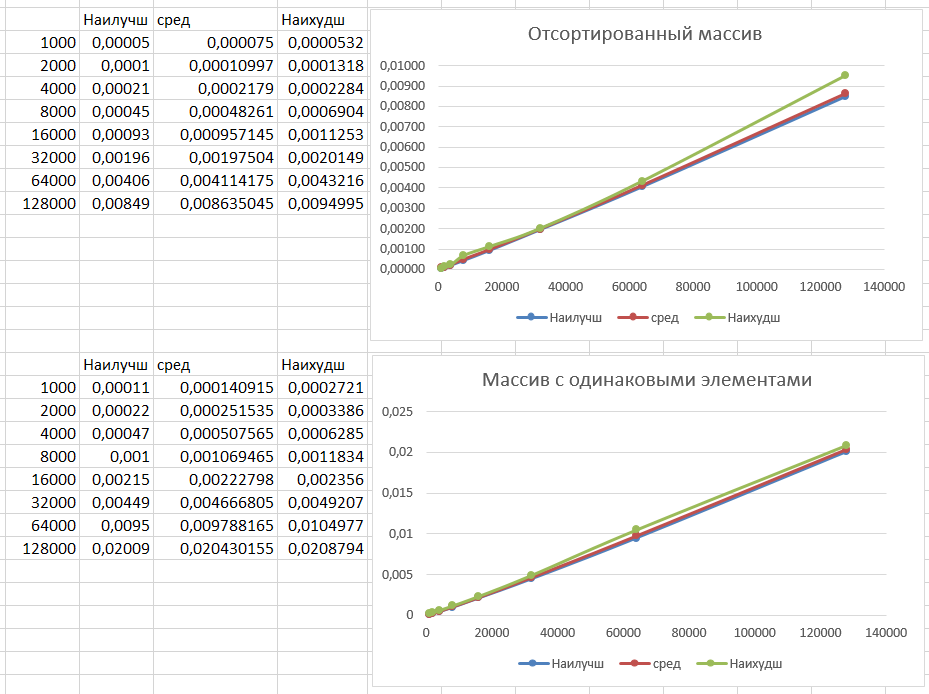
**Анализ результатов**



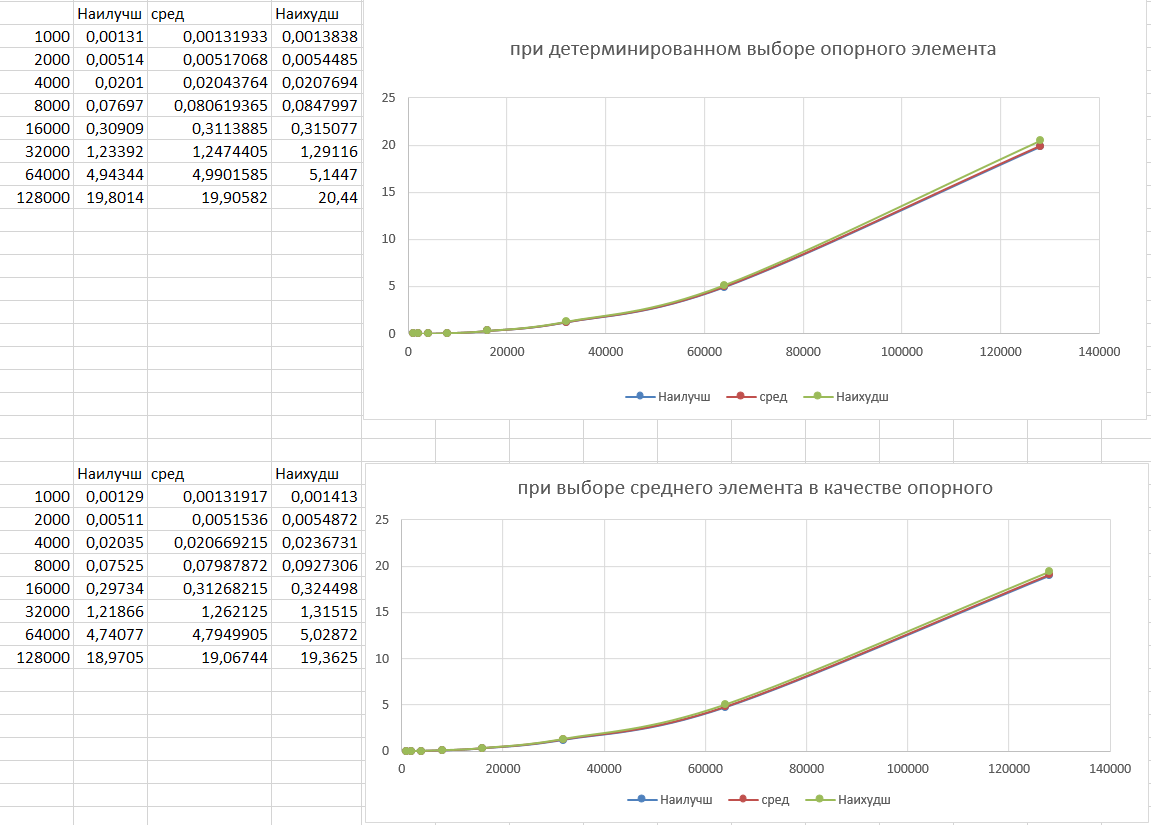
**Время исполнения**



Для обычной сортировки график времени исполнения имеет вид n log n

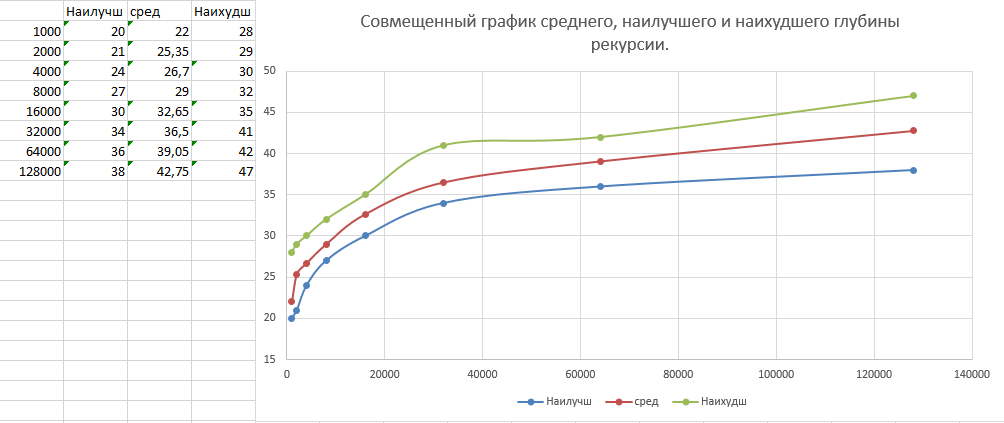


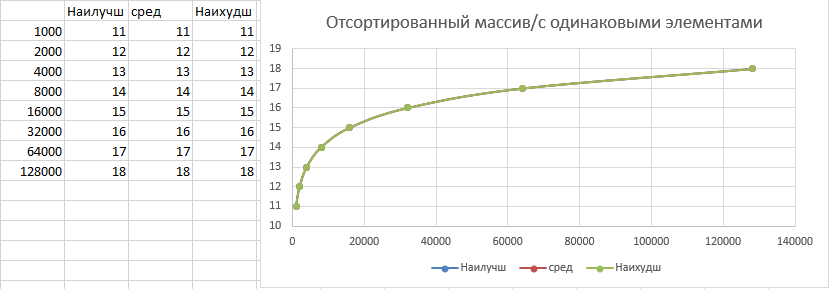
Для худших случаев время выполнения растет согласно графику O(n)



Для самых худших случаев время выполнения вырастает до O(n^2)

**ГЛУБИНА РЕКУРСИИ**



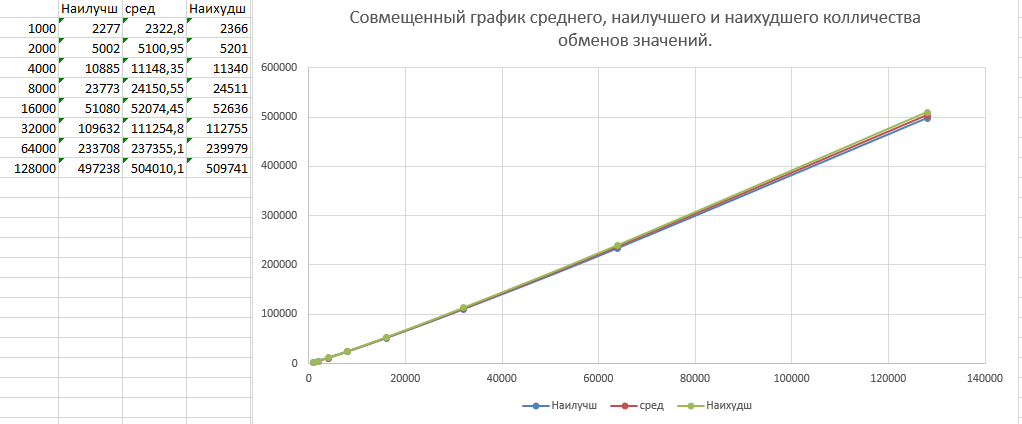


Глубина рекурсии для данных случаев имеет вид n log n



При самых худших случаях глубина рекурсии становится равной количеству элементов в массиве

**ОБМЕН**



Для всех случаев, за исключением, когда массив уже отсортирован, количество обменов имеет вид графика O(n)

**РЕКУРСИИ**



Во всех случаях количество вызова рекурсии равно графику O(n)

# Заключение

В ходе выполнения лабораторной работы был реализован и протестирован алгоритм быстрой сортировки (QuickSort) с различными стратегиями выбора опорного элемента. На основании проведённых экспериментов и анализа результатов можно сделать следующие выводы:

Применимость алгоритма быстрой сортировки

- Быстрая сортировка является одним из наиболее эффективных алгоритмов сортировки для большинства практических задач. Её средняя временная сложность составляет (O(n log n)), что делает её предпочтительной для работы с большими массивами данных.

- Однако в худших случаях (например, при неудачном выборе опорного элемента) время выполнения может деградировать до (O(n^2)). Это особенно заметно при использовании детерминированных стратегий выбора опорного элемента (например, первого или последнего элемента) на уже отсортированных или почти отсортированных массивах.

Влияние выбора опорного элемента

- Стратегия выбора опорного элемента играет ключевую роль в эффективности алгоритма. Использование медианы трёх элементов (первого, последнего и среднего) позволяет значительно снизить вероятность худшего случая и улучшить баланс разделения массива.

- В то же время, детерминированный выбор опорного элемента (например, всегда первого или последнего элемента) может привести к значительному увеличению времени выполнения и глубины рекурсии, особенно на специально подготовленных данных.

Сложность реализации

- Реализация алгоритма быстрой сортировки требует внимательного подхода к выбору опорного элемента и обработке граничных случаев. Неправильная реализация может привести к переполнению стека (stack overflow) или некорректной работе алгоритма.

Результаты тестирования

- На массивах случайных данных быстрая сортировка показала высокую эффективность, подтверждая свою среднюю временную сложность (O(n log n)).

- На отсортированных массивах и массивах с одинаковыми элементами производительность алгоритма падала до (O(n^2)).

Имплементационные аспекты

- Реализация алгоритма потребовала учёта множества нюансов, таких как подсчёт количества рекурсивных вызовов, глубины рекурсии и операций обмена. Это позволило получить детальную статистику работы алгоритма и проанализировать его поведение на разных типах данных.

Личные впечатления

- Реализация быстрой сортировки оказалась интересным и полезным опытом. Алгоритм демонстрирует высокую эффективность на практике, но требует тщательной проработки деталей, таких как выбор опорного элемента и обработка граничных случаев.

Для практического использования быстрой сортировки рекомендуется:

- Использовать стратегию "медиана трёх" для выбора опорного элемента, чтобы минимизировать вероятность худшего случая.

- Ограничивать глубину рекурсии или использовать итеративную версию алгоритма для предотвращения переполнения стека.