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 Акулинин Андрей Игоревич

Ссылка на репозиторий: <https://github.com/MUCTR-IKT-CPP/AkulininAI_36/tree/main/algorithms>

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. 2](#_Toc63548273)

[Выполнение задачи. 2](#_Toc63548274)

[Заключение. 2](#_Toc63548275)

# Описание задачи.

В рамках лабораторной работы №2 поставлена задача изучения и анализа временной сложности алгоритмов сортировки. Основная цель работы заключается в исследовании наихудшего, наилучшего и среднего времени выполнения выбранного алгоритма сортировки, а также в сопоставлении полученных результатов с теоретической оценкой сложности алгоритма, выраженной в нотации O-большое.

# Описание метода/модели.

**Пирамидальная сортировка (Heap Sort)** — это алгоритм сортировки, который использует структуру данных "куча" (heap). Куча — это двоичное дерево, в котором каждый родительский узел больше (или меньше) своих дочерних узлов. Алгоритм состоит из двух основных этапов:

1. **Построение кучи**: Преобразование массива в кучу.
2. **Сортировка**: Постепенное извлечение максимального (или минимального) элемента из кучи и его перемещение в конец массива.

**Основные шаги алгоритма:**

1. **Построение кучи**:
   * Начинаем с последнего нелистового узла и просеиваем его вниз, чтобы он удовлетворял свойствам кучи.
   * Повторяем процесс для всех узлов до корня.
2. **Сортировка**:
   * Перемещаем корневой элемент (максимальный) в конец массива.
   * Уменьшаем размер кучи на 1.
   * Просеиваем новый корневой элемент, чтобы восстановить свойства кучи.
   * Повторяем процесс, пока куча не станет пустой.

**Временная сложность:**

* **Наихудший случай**: O(n log n). Возникает, когда массив отсортирован в обратном порядке. В этом случае требуется максимальное количество сравнений и обменов.
* **Средний случай**: O(n log n). В среднем, для случайного массива, количество операций также логарифмически зависит от размера массива.
* **Наилучший случай**: O(n). Даже если массив уже отсортирован, алгоритм все равно выполняет построение кучи и извлечение элементов, что требует времени O(n).

# Выполнение задачи.

Для выполнения поставленной задачи был использован язык программирования C++. Программа предназначена для анализа времени работы алгоритма пирамидальной сортировки (Heap Sort) при различных размерах входных массивов. Входные данные генерируются случайным образом, а затем производится их сортировка с подсчетом количества вызовов функций и внутренних вызовов.

**Организация программы:**

Программа состоит из нескольких ключевых функций:

1. **generateRandomArray** – генерация массива заданного размера со случайными значениями в диапазоне от -1.0 до 1.0.
2. **makeHeap** – функция для построения кучи, которая также подсчитывает количество вызовов функции и внутренних вызовов.
3. **heapSort** – реализация алгоритма пирамидальной сортировки, которая использует функцию makeHeap для построения и восстановления кучи.
4. **saveDataToFile** – сохранение результатов тестирования (время выполнения, количество вызовов функций, количество внутренних вызовов) в файлы формата CSV для последующего построения графиков.
5. **main** – главная функция, которая управляет процессом тестирования. Она последовательно увеличивает размер массива и запускает серию тестов для каждого размера, собирая статистику о времени выполнения, количестве вызовов функций и внутренних вызовов.

**Проведённые тесты:**

Тестирование проводилось для размеров массива от 1000 до 128000 элементов с шагом удвоения. Для каждого размера массива выполнялось 20 попыток сортировки, чтобы получить усреднённое значение времени выполнения и других показателей.

**Результаты:**

В результате были собраны следующие данные:

* Время выполнения алгоритма для каждого размера массива.
* Количество вызовов функции makeHeap.
* Количество внутренних вызовов функции makeHeap.

Эти данные сохранены в файлах times.csv, funcCalls.csv и inFuncCalls.csv. Построение графиков на основе этих данных позволяет наглядно оценить зависимость времени выполнения алгоритма от размера массива, а также понять динамику количества вызовов функций и внутренних вызовов.

Изображение выглядит как линия, текст, График, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как линия, текст, График, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, линия, График, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как линия, График, текст, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, линия, График, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки.

# Заключение.

В ходе выполнения лабораторной работы был проведен анализ временной сложности алгоритма пирамидальной сортировки (Heap Sort). Результаты тестирования подтвердили теоретическую оценку сложности алгоритма, которая составляет O(n log n) в наихудшем, среднем и O(n) наилучшем случаях.

**Основные выводы:**

1. **Время выполнения**: Время выполнения алгоритма растет пропорционально n log n, что соответствует теоретической оценке. На графиках видно, что с увеличением размера массива время выполнения увеличивается, но не так быстро, как в случае квадратичных алгоритмов (например, сортировки пузырьком).
2. **Количество вызовов функций**: Количество вызовов функции makeHeap и внутренних вызовов также растет с увеличением размера массива, что подтверждает логарифмическую зависимость.
3. **Эффективность алгоритма**: Пирамидальная сортировка является эффективным алгоритмом для больших массивов данных, так как её сложность O(n log n) делает её более предпочтительной по сравнению с квадратичными алгоритмами для больших объемов данных.

**Рекомендации:**

* Для малых массивов (до 1000 элементов) можно использовать более простые алгоритмы сортировки, такие как сортировка вставками, так как их накладные расходы на вызовы функций могут быть меньше.
* Для больших массивов пирамидальная сортировка является одним из наиболее эффективных алгоритмов, особенно если требуется стабильность и предсказуемость времени выполнения.

Таким образом, пирамидальная сортировка является мощным инструментом для сортировки больших объемов данных, и её использование оправдано в случаях, когда требуется гарантированное время выполнения и устойчивость к наихудшим случаям.