МИНИСТЕРСТВО НАУКИ И ВЫСШЕГООБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ**

на тему:

**«Сравнительный анализ алгоритмов сортировки kemerge и d-heap»**

**Выполнили:** студент группы 3822Б1ФИ2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Хохлов А.Д./

Подпись

**Проверил:** ассистент кафедры АГДМ,

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Уткин Г.В./

Подпись

Нижний Новгород  
2024

Содержание

[Введение 3](#_Toc185421091)

[Цель 3](#_Toc185421092)

[1. Описание алгоритмов 4](#_Toc185421093)

[1.1.1 Сортировка к-слиянием 4](#_Toc185421094)

[1.1.2 Временная сложность сортировка к-слиянием 4](#_Toc185421095)

[1.2.1 Сортировка d-кучей 5](#_Toc185421096)

[1.2.2 Временная сложность сортировки d-кучей 5](#_Toc185421097)

[2 Описание алгоритмов программы 6](#_Toc185421098)

[2.1 Сортировка к-слиянием 6](#_Toc185421099)

[2.2 Сортировка d-кучей 7](#_Toc185421100)

[3 Проведенные эксперименты 10](#_Toc185421101)

[4 Сравнение алгоритмов 14](#_Toc185421102)

[5 Вывод 15](#_Toc185421103)

# Введение

Лабораторная работа направлена на изучение и реализацию двух алгоритмов: к-слияние и сортировка на д-кучах.

# Цель

Целью данной работы является реализация сортировки 4-слиянием и 3-кучи. После чего необходимо проанализировать сложности алгоритмов, а также сравнить их экспериментально. Первая программа должна читать из файла входную информацию и создавать два новых, каждый из которых должен содержать результаты работы соответствующего алгоритма и время его работы. Вторая программа предназначена для проведения экспериментов, параметры для массива данных будут также считываться из файла, а время работы алгоритмов должны записываться в новые, соответствующие два файла.

Задачи:

1. Реализовать структуру данных d-куча.

2. Реализовать сортировку кучей.

3. Реализовать алгоритм сортировки к-слиянием.

4. Написать программу, проверяющую работу алгоритмов, зафиксировать результат и время их работы

5. Провести эксперименты для тестирования программы на различных входных данных.

Эксперименты будут проводится на машине со следующими характеристиками:

1. Процессор - 10th Gen Intel(R) Core(TM) i7-10700K 2.90 GHz

2. Оперативная память - 32,0 ГБ (доступно: 31,7 ГБ)

3. Тип системы - 64-разрядная операционная система, процессор x64

## Описание алгоритмов

## Сортировка к-слиянием

Сортировка k-слиянием, как и быстрая сортировка, является алгоритмом типа «разделяй и властвуй». Ее рекурсивная реализация состоит в разбиении массива на k приблизительно одинаковых частей, их сортировкой с последующим объединением отсортированных фрагментов. Для этого используются две процедуры: SORT\_MERGE(a,i,j,k) и MERGE(a,i,m,j).

SORT\_MERGE(a,i,j,k):

Если длина входного массива меньше, чем n/k, то массив сортируется пузырьком, иначе рекурсивно вызывается SORT\_MERGE(a, i + step \* numb, i + step \* (numb + 1) - 1, k), где step = (j - i + 1) / k, numb = при условии что numb < k – 1. Иначе вызывается SORT\_MERGE(a, i + (k - 1) \* step, j, k). После чего вызывается SORT\_MERGE(a, i + (k - 1) \* step, j, k) на остатке массива. После этих действий мы получаем к отсортированных массивов, их осталось слиять функцией MERGE.

В цикле вызывается MERGE(a, i, i + step \* (numb + 1) - 1, i + step \* (numb + 2) - 1), если numb < k – 2. Иначе MERGE(a, i, i + (k - 1) \* step - 1, j).

На выходе мы получаем отсортированный массив.

## Временная сложность сортировка к-слиянием

Для этого составим рекуррентное соотношение. Пускай *T(n)* — время сортировки массива длины n, тогда для сортировки слиянием справедливо *T(n)=2\*T(n/2)+O(n)*  (O(n) — это время, необходимое на то, чтобы слить два массива). Распишем это соотношение:

*T(n)=2\*T(n/2)+O(n)=4\*T(n/4)+2\*O(n)=…=2l\*T(w)+l\*O(n*)

Где w – число элементов, которые сортируются пузырьком.

Значит справедливо: T(w)=O(w2)

*2l=*w- в зависимости от n может изменяться от 1 до k-1 (Сортировка k-слиянием).

Таким образом, l=log =logn – logw. Т.к. w=const для конкретного n, то O(l)=O(logn), и аналогично для O(*2l)=O()=O(n).*

Итак, *T(n)= O(nw2)+log n \*O(n)= O(n)+log n \*O(n)=*O(n\* log n)

Таким образом временная сложность сортировки к-слиянием будет O(n\* log n).

## Сортировка d-кучей

Сортировка кучей — алгоритм сортировки, использующий структуру данных d-куча. Это неустойчивый алгоритм сортировки с временем работы O(nlogn), где n — количество элементов для сортировки, и использующий O(1) дополнительной памяти.

Алгоритм сортировки:

1. Построим на базе массива a за O(n) кучу для максимума.

2. Минимальный элемент находится в корне, поменяем его местами с a[n−1].

3. Далее вызовем процедуру погружения для 0 элемента для поддержания свойств кучи, предварительно уменьшив размер кучи на 1. Она за O(logn) просеет a[0] на нужное место и сформирует новую кучу.

4. Повторим эту процедуру для новой кучи, корень будет менять местами с a[n−2].

5. Делая аналогичные действия, пока размер кучи не станет равен 1, мы будем ставить наименьшее из оставшихся чисел в конец не отсортированной части.

6. Таким образом, мы получим отсортированный массив.

## Временная сложность сортировки d-кучей

Временная сложность сортировки d-кучей Временная сложность этого алгоритма может быть оценена на основе двух основных этапов: построения d-кучи и процесса сортировки.

Построение d-кучи требует O(n) времени, что можно показать следующим образом:

* Для узлов на нижних уровнях кучи (где h близка к 0) требуется меньше времени, так как у них меньше дочерних узлов.
* Для узлов на верхних уровнях (где h близка к logd(n)) требуется больше времени, но таких узлов меньше.

В результате, суммируя время для всех узлов, мы получаем O(n) для построения d-кучи. Оценка процесса сортировки:

* Производится извлечение корня кучи (минимального элемента) и последующая перестановка элементов в конце массива. Для этого корень заменяется последним элементом и происходит "погружение".
* Каждое извлечение корня занимает O(logd(n)) времени, и поскольку необходимо извлечь n элементов, это дает общее время O(nlogd(n)).

Итоговая временная сложность равна O(nlogd(n)).

## Описание алгоритмов программы

## Сортировка к-слиянием

В ходе данной лабораторной работы был реализован алгоритм сортировки к-слиянием следующим образом:

template <typename T>

void bubbleSort(std::vector<T>& a, int i, int j) {

for (int k = i; k <= j; ++k) {

for (int l = i; l < j - (k - i); ++l) {

if (a[l] > a[l + 1]) {

std::swap(a[l], a[l + 1]);

}

}

}

}

template <typename T>

void MERGE(std::vector<T>& a, int i, int m, int j) {

int n1 = m - i + 1; // Размер первой половины

int n2 = j - m; // Размер второй половины

std::vector<T> b(j - i + 1); // Временный массив для слияния

int i1 = i, i2 = m + 1; // Индексы для двух половин

while (i1 + i2 < j + m + 2) {

if ((i1 <= m && i2 < j + 1 && a[i1] <= a[i2]) || i2 == j + 1) {

b[i1 + i2 - i - m - 1] = a[i1];

i1++;

}

else {

b[i1 + i2 - i - m - 1] = a[i2];

i2++;

}

}

// Копируем отсортированные элементы обратно в оригинальный массив

for (int k = 0; k < b.size(); ++k) {

a[i + k] = b[k];

}

}

template <typename T>

void SORT\_MERGE(std::vector<T>& a, int i, int j, int k) {

if (j - i + 1 < k) {

bubbleSort(a, i, j);

}

else {

int step = (j - i + 1) / k;

for (int numb = 0; numb < k; ++numb) {

if (numb < k - 1) {

SORT\_MERGE(a, i + step \* numb, i + step \* (numb + 1) - 1, k);

}

else {

SORT\_MERGE(a, i + (k - 1) \* step, j, k);

}

}

for (int numb = 0; numb < k - 1; ++numb) {

if (numb < k - 2) {

MERGE(a, i, i + step \* (numb + 1) - 1, i + step \* (numb + 2) - 1);

}

else {

MERGE(a, i, i + (k - 1) \* step - 1, j);

}

}

}

}

Метод bubbleSort

Сортируем массив длины n/k пузырьком. Принимает 3 параметра: массив а, индекс начала сортируемой части, индекс конца сортируемой части. Последовательно обходит все элементы и сравнивает со всеми после него. Если элемент после него оказался меньше, то меняет их местами.

Метод SORT\_MERGE

Является основной рекурсивной функцией, которая сортирует массив.

• Принимает четыре параметра: указатель на массив a, левую границу i, правую границу j и параметр k.

• Если длина массива меньше n/k (j-i < n/k), то:

• Вызывается функция bubbleSort, сортирует данный массив пузырьком.

• Рекурсивно вызывается SORT\_MERGE для всех массивов длины n/k.

• Рекурсивно вызывается SORT\_MERGE для оставшихся элементов массива.

• Рекурсивно вызывается MERGE для отсортированных массивов длины n/k.

• Рекурсивно вызывается MERGE для оставшихся элементов массива.

Метод MERGE

Сливает два массива в один.

Рассчитывается размер первого массива = m – i + 1 и размер второго массива = j – m. Индекс первой половины i1= i, второй половины i2 = m + 1. После чего в цикле пока сумма индексов половин меньше j + m + 2 если i1 <= m && i2 < j + 1 && a[i1] <= a[i2]) || i2 == j + 1, то в выходной массив записывается элемент первого массива, иначе элемент второго.

## Сортировка d-кучей

В ходе данной лабораторной работы был реализован алгоритм сортировки с помощью d-кучи. Для этого был реализован шаблонный класс DHeap, в котором содержатся методы, необходимые для сортировки.

template<class T>

class DHeap {

private:

std::vector<T> heap{};

int d{};

//operations for heap

int father(int id);

int first\_child(int id);

int last\_child(int id);

int min\_child(int id);

public:

DHeap();

DHeap(const int& d\_, const std::vector<T>& keys); // ~build\_heap

void sift\_down(int id);

void sift\_up(int id);

void insert(T elem);

T extract\_min();

std::vector<T> sort\_d(const int& d\_, const std::vector<T>& a);

void print\_heap();

};

Метод сортировки

template<class T> std::vector<T> DHeap<T>::sort\_d(const int& d\_, const std::vector<T>& a)

{

// DHeap(d\_, a);

int n = heap.size();

std::vector<T> sort\_a;

while (n > 0)

{

for (int i = 0; i < n; i++)

{

sort\_a.push\_back(extract\_min());

n = heap.size();

}

}

heap = sort\_a;

return sort\_a;

}

• Принимает степень кучи d\_ и вектор a, который нужно отсортировать. • Создает d-кучу, инициализируя её с помощью конструктора DHeap.

• Создает пустой вектор sort\_a для хранения отсортированных элементов.

• В цикле извлекает минимальные элементы из кучи с помощью extract\_min и добавляет их в sort\_a, пока куча не станет пустой.

• Возвращает отсортированный вектор sort\_a.

Рассмотрим методы, используемые в данной реализации:

Конструктор DHeap:

• Принимает два параметра: d\_ (степень кучи) и keys (вектор элементов для инициализации кучи).

• Инициализирует переменные d и heap, где heap — это вектор, содержащий элементы.

• Запускает цикл, который вызывает функцию sift\_down для каждого узла, начиная с последнего родительского узла и заканчивая корнем. Это позволяет построить d-кучу из неупорядоченного массива.

Функция min\_child:

• Принимает индекс узла id и находит индекс минимального дочернего узла среди всех дочерних узлов данного узла.

• Вычисляет границы для дочерних узлов и сравнивает их значения, чтобы определить, какой из них минимален.

• Возвращает индекс дочернего узла с минимальным значением.

Функция sift\_down:

• Принимает индекс узла id и восстанавливает свойства d-кучи, начиная с этого узла.

• Внутри цикла проверяет, есть ли дочерние узлы, и находит минимальный дочерний узел с помощью min\_child.

• Если значение текущего узла больше значения минимального дочернего узла, они меняются местами, и процесс продолжается для нового узла, пока не будет восстановлено свойство кучи.

Функция extract\_min:

• Извлекает минимальный элемент из кучи (корень).

• Сохраняет минимальный элемент, заменяет корень последним элементом в куче, уменьшает размер кучи и вызывает sift\_down для восстановления свойств кучи.

• Возвращает извлеченный минимальный элемент.

**Итог**

Алгоритм сортировки sort\_d работает следующим образом:

• Сначала создается d-куча из входного массива.

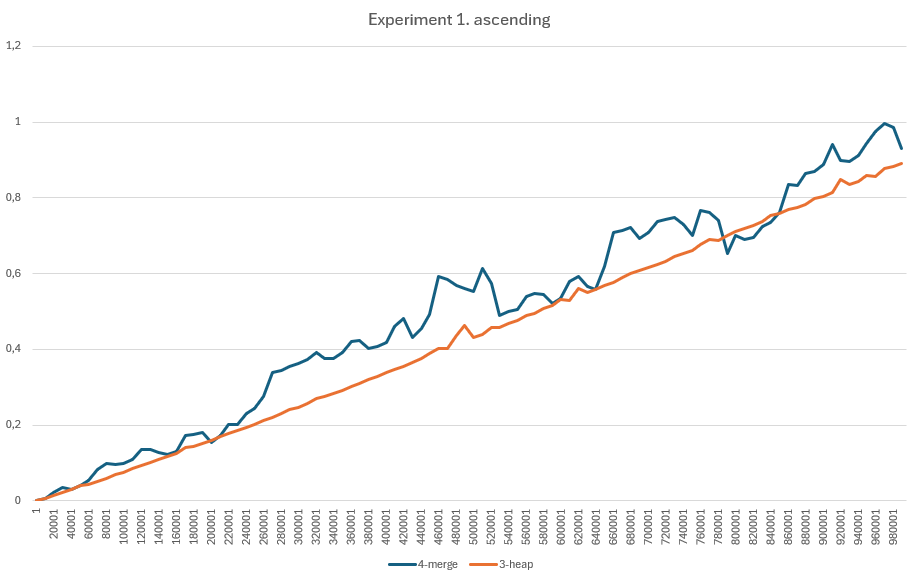
• Затем, используя метод извлечения минимального элемента, элементы извлекаются из кучи и добавляются в новый вектор, который будет отсортирован.

• В результате получается отсортированный массив, где элементы расположены в порядке возрастания.

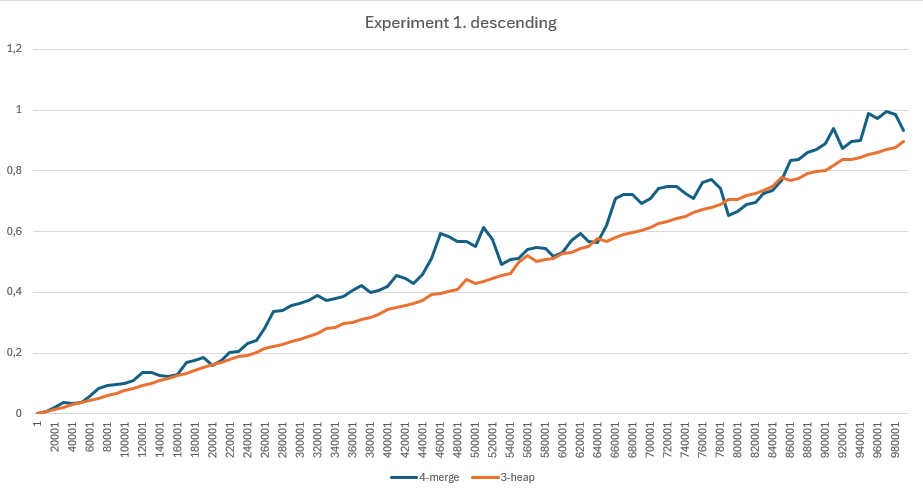
### Проведенные эксперименты

В данной лабораторной работе было проведено два эксперимента: суть первого заключался в том, чтобы понять как разные размеры массивов с различными характеристиками случайных чисел влияют на производительность алгоритмов сортировки, а второго - как изменения в генерации случайных чисел влияют на производительность сортировки для фиксированного размера массива, размер массива неизменен.

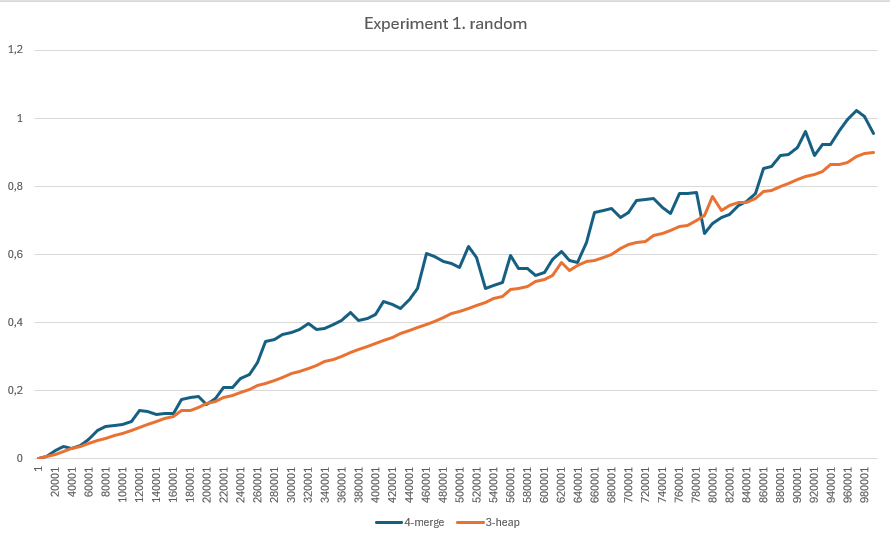
В каждом эксперименте массив создавался тремя разными способами: псевдослучайная генерация чисел, массив по убыванию, массив по возрастанию. Проведя эксперименты и проанализировав полученные данные имеем следующие графики:



1. Первый эксперимент с массивом по возростанию.

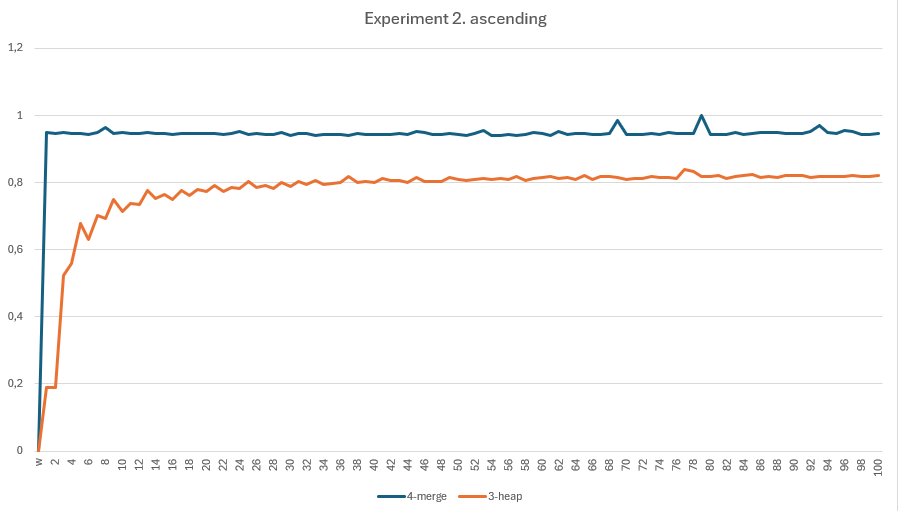


1. Первый эксперимент с массивом по убыванию.



1. Первый эксперимент с псевдослучайным массивом.

Во всех экспериментах обе линии показывают линейный рост времени выполнения при увеличении размера массива. Сортировка 3-кучей демонстрирует более низкое время выполнения по сравнению с сортировкой методом 4-слияния.



1. Второй эксперимент с массивом по возростанию.

Изображение выглядит как линия, График, текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

1. Второй эксперимент с массивом по убыванию.

Изображение выглядит как текст, линия, снимок экрана, График

Контент, сгенерированный ИИ, может содержать ошибки.

1. Второй эксперимент с псевдослучайным массивом.

По графикам видно, что во всех экспериментах выигрывает алгоритм сортировки на d-куче, так как он не тратит время на слияние массивов, а сразу достает нужный элемент из кучи.

### Сравнение алгоритмов

1. **Временная сложность**

Временная сложность алгоритма сортировки к-слиянием O(nlogn). В случае сортировки d-кучей временная сложность равна O(nlogd(n)), зависит от значения d. Чем больше d, тем меньше количество уровней в куче, но большее количество дочерних узлов может увеличить время на операции с кучей.

1. **Стабильность**

Сортировка к-слиянием является стабильной, в отличие от сортировки d-кучей. Порядок равных элементов может измениться.

1. **Использование памяти**

Сортировка к-слиянием использует O(log n) дополнительной памяти для хранения стека вызовов рекурсии. В случае не-рекурсивной реализации может использовать O(1) дополнительной памяти. Сортировка d-кучей использует O(n) памяти для хранения кучи, так как все элементы должны быть в куче во время сортировки.

1. **Простота реализации**

Основная сложность в сортировке к-слиянием заключается в реализации рекурсивного разбиения на к частей и последующего слияния. Реализация сортировки d-кучей может быть более сложной из-за необходимости управления кучей и операций с ней (например, sift\_down и extract\_min).

1. **Производительность на различных типах данных**

Во всех случаях по производительности выиграла сортировка d-кучей. Однако сортировка к-слиянием является более стабильной.

### 5 Вывод

В ходе данной лабораторной работы были реализованы алгоритмы сортировки 4-слиянием и 3-кучей, проведены эксперименты, для анализа поведения и эффективности работы алгоритмов. Оба алгоритма имеют свои преимущества и недостатки. Сортировка 3-кучей оказалась быстрее, но сложнее в реализации. Сортировка 4-слиянием более устойчива к худшим случаям, но требует меньше памяти и может быть быстрее на практике. Опираясь на результаты экспериментов, можно сделать вывод, что сортировка 3-кучей, несмотря на сложную реализацию, выполняет свою работу эффективнее и стабильнее, чем сортировка 4-слиянием.