МИНИСТЕРСТВО СЕЛЬСКОГО ХОЗЯЙСТВА РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«КАЗАНСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ»

Институт экономики

Кафедра цифровых технологий и прикладной информатики

Курсовая работа

По дисциплине «Алгоритмизация и программирование»

На тему: «Сравнение сортировки массива с помощью метода Heap sort».

Выполнили: студенты 1-го курса

группы Б331-8

по направлению «Прикладная информатика»

Зайнуллин Рамис Раисович и Гончаров Никита Олегович

Руководитель: к. т. н., доцент Панков Андрей Олегович

**Содержание**

Введение**3**

1. Глава 1. Обзор литературы**5**

1.1. Массивы5

1.2. Модуль Arrays. Его основные методы 5

2. Глава 2. Основная часть **16**

2.1. Описание выданного алгоритма и расчёт его сложности 16

2.2. Показ работы на массиве из 5-6 алгоритмов 17

2.3. Реализация алгоритма и сравнение со стандартной библиотекой Arrays 20

Заключение **22**

Список источников **23**

**Введение**

Java — это многоплатформенный и объектно-ориентированный язык, разработанный компанией Sun Microsystems, который сам по себе может использоваться как платформа. Это быстрый, безопасный и надежный язык программирования для всего: от мобильных приложений и корпоративного ПО до приложений для работы с большими данными и серверных технологий.

Для использования языка необходим JDK (Java Development Kit).

Java Development Kit — бесплатно распространяемый комплект разработчика приложений на языке Java, включающий в себя компилятор Java, стандартные библиотеки классов Java, примеры, документацию, различные утилиты и исполнительную систему Java.

Где взять JDK?

JDK можно скачать бесплатно с официального сайта(www.oracle.com) без каких-либо ограничений. JDK является незаменимой программой для Java-разработчиков, ведь с ее помощью можно писать программы на Java, работать над созданием виртуального окружения, запускать его и отлаживать.

Актуальная версия JDK.

Новейшая версия JDK 22.0.1 представляет собой обновление для Java Development Kit (JDK), которое включает в себя исправления ошибок, улучшения производительности и новые функциональные возможности. Одним из главных изменений в JDK 22.0.1 является улучшение библиотеки JavaFX, что делает разработку графических приложений более эффективной и простой.

Также в новой версии JDK были добавлены новые инструменты для разработки, улучшения в работе с памятью и увеличение производительности при выполнении различных задач. Кроме того, в JDK 22.0.1 были внесены изменения в работу с безопасностью и исправлены некоторые уязвимости.

Обновление до JDK 22.0.1 рекомендуется для всех разработчиков Java, так как оно предоставляет более стабильное и безопасное окружение для работы с приложениями на этом языке программирования. Для получения более подробной информации о новых возможностях и изменениях в JDK 22.0.1 рекомендуется посетить официальный сайт Oracle. [1]

JDK версии от разных разработчиков.

Oracle JDK - это официальная версия JDK, предоставляемая корпорацией Oracle. Она включает в себя все стандартные функции, библиотеки и инструменты, необходимые для разработки приложений на Java.

OpenJDK - это бесплатная и открытая версия JDK, которая поддерживается сообществом разработчиков. OpenJDK является основой для Oracle JDK и других версий JDK от различных производителей.

Amazon Corretto - это версия JDK, разработанная Amazon с использованием OpenJDK. Она предлагает дополнительную поддержку и оптимизации для работы в облачных средах.

Azul Zulu - это версия JDK от компании Azul, которая предлагает расширенную оптимизацию для работы на серверах и в облачных средах. Она также включает в себя дополнительные инструменты для управления и мониторинга приложений.

IBM JDK - это версия JDK от компании IBM, которая предлагает дополнительные инструменты и оптимизации для работы на серверах и в корпоративных средах. Она также имеет поддержку для различных операционных систем и архитекту

**ГЛАВА 1**

1.1 Массивы

Массивы являются одной из основных структур данных в языке программирования Java. Массив представляет собой упорядоченный набор элементов одного типа, расположенных в памяти компьютера последовательно.

Массивы в Java имеют фиксированную длину, которая определяется при их создании, и элементы массива можно обратиться по их индексу. Индексация в массивах начинается с нуля, то есть первый элемент массива имеет индекс 0, второй - 1 и так далее.

1.2 Модуль Arrays и его основные методы

Класс arrays представляет собой коллекцию элементов одного типа данных, организованных в виде массива. Этот класс предоставляет различные методы для работы с массивами, такие как добавление, удаление, изменение элементов, сортировка, поиск и т. д.

Это удобный способ для работы с данными, хранящимися в виде массива, и предоставляющий ряд методов для упрощения работы с ними. Например, можно отсортировать элементы массива, или найти индекс определенного элемента в массиве. Основные методы: fill, sort, copyOf, compare, equals, mismatch.

Fill

Метод fill в Java используется для заполнения всех элементов определенного массива одним и тем же значением. Этот метод удобно использовать, когда необходимо инициализировать все элементы массива одним значением.

Примеры:

Fill(Char[], Int32, Int32, Char) - присваивает указанное значение char каждому элементу указанного диапазона указанного массива символов.

Fill(Double[], Int32, Int32, Double) - присваивает заданное значение типа double каждому элементу указанного диапазона указанного массива doubles.

Fill(Int16[], Int32, Int32, Int16) - присваивает заданное короткое значение каждому элементу указанного диапазона указанного массива коротких значений.

Fill(Byte[], Int32, Int32, SByte) - присваивает указанное значение байта каждому элементу указанного диапазона указанного массива байтов.

Fill(Int64[], Int32, Int32, Int64) - присваивает заданное значение long каждому элементу указанного диапазона указанного массива longs.

Fill(Single[], Int32, Int32, Single) - присваивает заданное значение float каждому элементу указанного диапазона указанного массива floats.

Fill(Int32[], Int32, Int32, Int32) - присваивает заданное значение int каждому элементу указанного диапазона указанного массива ints.

Fill(Object[], Int32, Int32, Object) - назначает указанную ссылку На объект каждому элементу указанного диапазона указанного массива Объектов.

Fill(Boolean[], Int32, Int32, Boolean) - присваивает заданное логическое значение каждому элементу указанного диапазона указанного массива логических значений.

Fill(Int64[], Int64) - присваивает заданное значение long каждому элементу указанного массива longs.

Fill(Single[], Single) - присваивает указанное значение float каждому элементу указанного массива floats.

Fill(Int32[], Int32) - присваивает заданное значение int каждому элементу указанного массива ints.

Fill(Int16[], Int16) - присваивает заданное короткое значение каждому элементу указанного массива шорт.

Fill(Double[], Double) - присваивает заданное значение double каждому элементу указанного массива двойников.

Fill(Char[], Char) - присваивает указанное значение char каждому элементу указанного массива символов.

Fill(Byte[], SByte) - присваивает заданное значение байта каждому элементу указанного массива байтов.

Fill(Boolean[], Boolean) - присваивает заданное логическое значение каждому элементу указанного массива логических значений.

Fill(Object[], Object) - назначает указанную ссылку На объект каждому элементу указанного массива Объектов.

Sort

Метод sort в Java используется для сортировки элементов массива или списка по их естественному порядку или с помощью заданного компаратора. Этот метод может быть использован для сортировки массива примитивных типов данных и объектов. Он реализует алгоритм быстрой сортировки (quicksort) и обеспечивает эффективную сортировку элементов.

Примеры:

Sort(Object[], Int32, Int32, IComparator) - сортирует указанный диапазон указанного массива объектов в соответствии с порядком, вызванным указанным компаратором.

Sort(Byte[], Int32, Int32) - сортирует указанный диапазон массива по возрастанию.

Sort(Single[], Int32, Int32) - сортирует указанный диапазон массива по возрастанию.

Sort(Int64[], Int32, Int32) - сортирует указанный диапазон массива по возрастанию.

Sort(Int32[], Int32, Int32) - сортирует указанный диапазон массива по возрастанию.

Sort(Int16[], Int32, Int32) - сортирует указанный диапазон массива по возрастанию.

Sort(Double[], Int32, Int32) - сортирует указанный диапазон массива по возрастанию.

Sort(Char[], Int32, Int32) - сортирует указанный диапазон массива по возрастанию.

Sort(Object[], Int32, Int32) - сортирует указанный диапазон указанного массива объектов в порядке возрастания в соответствии с естественным порядком сопоставимых элементов.

Sort(Single[]) - сортирует указанный массив по возрастанию числового порядка.

Sort(Int64[]) - сортирует указанный массив по возрастанию числового порядка.

Sort(Int32[]) - сортирует указанный массив по возрастанию числового порядка.

Sort(Int16[]) - сортирует указанный массив по возрастанию числового порядка.

Sort(Double[]) - сортирует указанный массив по возрастанию числового порядка.

Sort(Char[]) - сортирует указанный массив по возрастанию числового порядка.

Sort(Byte[]) - сортирует указанный массив по возрастанию числового порядка.

Sort(Object[]) - сортирует указанный массив объектов в порядке возрастания в соответствии с естественным порядком сопоставимых элементов.

Sort(Object[], IComparator) - сортирует указанный массив объектов в соответствии с порядком, вызванным указанным компаратором.

CopyOf

Метод copyOf в Java используется для создания копии массива или списка определенной длины. Он создает новый массив или список, копируя элементы из исходного массива или списка. Если новая длина больше длины исходного массива или списка, то новый массив (список) будет заполнен нулевыми значениями или значениями по умолчанию для данного типа данных. Метод copyOf также может быть использован для создания копии части исходного массива или списка, указав начальный и конечный индексы элементов, которые нужно скопировать.

Примеры:

CopyOf(Int32[], Int32) - копирует указанный массив, усечение или заполнение нулями (при необходимости), чтобы копия была указанной длины.

CopyOf(Object[], Int32, Class) - копирует указанный массив, усечение или заполнение значениями NULL (при необходимости), чтобы копия была указанной длины.

CopyOf(Single[], Int32) - копирует указанный массив, усечение или заполнение нулями (при необходимости), чтобы копия была указанной длины.

CopyOf(Int64[], Int32) - копирует указанный массив, усечение или заполнение нулями (при необходимости), чтобы копия была указанной длины.

CopyOf(Int16[], Int32) - копирует указанный массив, усечение или заполнение нулями (при необходимости), чтобы копия была указанной длины.

CopyOf(Boolean[], Int32) - копирует указанный массив, усечение или заполнение ( false при необходимости), чтобы копия была указанной длины.

CopyOf(Char[], Int32) - копирует указанный массив, усечение или заполнение символами NULL (при необходимости), чтобы копия была указанной длины.

CopyOf(Byte[], Int32) - копирует указанный массив, усечение или заполнение нулями (при необходимости), чтобы копия была указанной длины.

CopyOf(Double[], Int32) - копирует указанный массив, усечение или заполнение нулями (при необходимости), чтобы копия была указанной длины.

CopyOf(Object[], Int32) - копирует указанный массив, усечение или заполнение значениями NULL (при необходимости), чтобы копия была указанной длины.

Equals

Метод equals в Java используется для сравнения двух объектов на равенство. Переопределение этого метода позволяет определить логику сравнения объектов на основе их содержимого, а не ссылок на них.

По умолчанию метод equals в классе Object сравнивает объекты по ссылкам, т.е. возвращает true только в случае, если две ссылки указывают на один и тот же объект. Для большинства классов необходимо переопределить метод equals(), чтобы он сравнивал объекты на основе их содержимого.

Примеры:

Equals(Byte[], Int32, Int32, Byte[], Int32, Int32) - возвращает значение true, если два указанных массива байтов в указанных диапазонах равны друг другу.

Equals(Char[], Int32, Int32, Char[], Int32, Int32) - возвращает значение true, если два указанных массива символов в указанных диапазонах равны друг другу.

Equals(Object[], Int32, Int32, Object[], Int32, Int32, IComparator) - возвращает значение true, если два указанных массива Объектов в указанных диапазонах равны друг другу.

Equals(Double[], Int32, Int32, Double[], Int32, Int32) - возвращает значение true, если два указанных массива двойников в указанных диапазонах равны друг другу.

Equals(Single[], Int32, Int32, Single[], Int32, Int32) - возвращает значение true, если два указанных массива с плавающей точкой в указанных диапазонах равны друг другу.

Equals(Int32[], Int32, Int32, Int32[], Int32, Int32) - возвращает значение true, если два указанных массива значений ints в указанных диапазонах равны друг другу.

Equals(Int64[], Int32, Int32, Int64[], Int32, Int32) - возвращает значение true, если два указанных массива значений long в указанных диапазонах равны друг другу.

Equals(Boolean[], Int32, Int32, Boolean[], Int32, Int32) - возвращает значение true, если два указанных массива логических значений в указанных диапазонах равны друг другу.

Equals(Int16[], Int32, Int32, Int16[], Int32, Int32) - возвращает значение true, если два указанных массива шорт в указанных диапазонах равны друг другу.

Equals(Object[], Int32, Int32, Object[], Int32, Int32) - возвращает значение true, если два указанных массива Объектов в указанных диапазонах равны друг другу.

Equals(Int16[], Int16[]) - возвращает значение true , если два указанных массива коротких значений равны друг другу.

Equals(Single[], Single[]) - возвращает значение true , если два указанных массива с плавающей точкой равны друг другу.

Equals(Int64[], Int64[]) - возвращает значение true , если два указанных массива значений long равны друг другу.

Equals(Int32[], Int32[]) - возвращает значение true , если два указанных массива ints равны друг другу.

Equals(Double[], Double[]) - возвращает значение true , если два указанных массива двойников равны друг другу.

Equals(Char[], Char[]) - возвращает значение true , если два указанных массива символов равны друг другу.

Equals(Byte[], Byte[]) - возвращает значение true , если два указанных массива байтов равны друг другу.

Equals(Boolean[], Boolean[]) - возвращает значение true , если два указанных массива логических значений равны друг другу.

Equals(Object[], Object[]) - возвращает значение true , если два указанных массива Объектов равны друг другу.

Equals(Object[], Object[], IComparator) - возвращает значение true , если два указанных массива Объектов равны друг другу.

Mismatch

Метод mismatch в Java используется для сравнения двух итераторов и возвращает индекс первого несовпадающего элемента. Если итераторы имеют различное количество элементов, то будет возвращено индекс первого элемента, после которого итераторы различаются. Если все элементы итераторов совпадают, метод возвращает -1.

Примеры:

Mismatch(Object[], Int32, Int32, Object[], Int32, Int32, IComparator) - находит и возвращает относительный индекс первого несоответствия между двумя Object массивами в указанных диапазонах, в противном случае возвращается значение -1, если несоответствие не найдено.

Mismatch(Single[], Int32, Int32, Single[], Int32, Int32) - находит и возвращает относительный индекс первого несоответствия между двумя float массивами в указанных диапазонах, в противном случае возвращается значение -1, если несоответствие не найдено.

Mismatch(Int64[], Int32, Int32, Int64[], Int32, Int32) - находит и возвращает относительный индекс первого несоответствия между двумя long массивами в указанных диапазонах, в противном случае возвращается значение -1, если несоответствие не найдено.

Mismatch(Int32[], Int32, Int32, Int32[], Int32, Int32) - находит и возвращает относительный индекс первого несоответствия между двумя int массивами в указанных диапазонах, в противном случае возвращается значение -1, если несоответствие не найдено.

Mismatch(Int16[], Int32, Int32, Int16[], Int32, Int32) - находит и возвращает относительный индекс первого несоответствия между двумя short массивами в указанных диапазонах, в противном случае возвращается значение -1, если несоответствие не найдено.

Mismatch(Double[], Int32, Int32, Double[], Int32, Int32) - находит и возвращает относительный индекс первого несоответствия между двумя double массивами в указанных диапазонах, в противном случае возвращается значение -1, если несоответствие не найдено.

Mismatch(Char[], Int32, Int32, Char[], Int32, Int32) - находит и возвращает относительный индекс первого несоответствия между двумя char массивами в указанных диапазонах, в противном случае возвращается значение -1, если несоответствие не найдено.

Mismatch(Byte[], Int32, Int32, Byte[], Int32, Int32) - находит и возвращает относительный индекс первого несоответствия между двумя byte массивами в указанных диапазонах, в противном случае возвращается значение -1, если несоответствие не найдено.

Mismatch(Boolean[], Int32, Int32, Boolean[], Int32, Int32) - находит и возвращает относительный индекс первого несоответствия между двумя boolean массивами в указанных диапазонах, в противном случае возвращается значение -1, если несоответствие не найдено.

Mismatch(Object[], Int32, Int32, Object[], Int32, Int32) - находит и возвращает относительный индекс первого несоответствия между двумя Object массивами в указанных диапазонах, в противном случае возвращается значение -1, если несоответствие не найдено.

Mismatch(Single[], Single[]) - находит и возвращает индекс первого несоответствия между двумя float массивами, в противном случае возвращает значение -1, если несоответствие не найдено.

Mismatch(Int64[], Int64[]) - находит и возвращает индекс первого несоответствия между двумя long массивами, в противном случае возвращает значение -1, если несоответствие не найдено.

Mismatch(Int32[], Int32[]) - находит и возвращает индекс первого несоответствия между двумя int массивами, в противном случае возвращает значение -1, если несоответствие не найдено.

Mismatch(Int16[], Int16[]) - находит и возвращает индекс первого несоответствия между двумя short массивами, в противном случае возвращает значение -1, если несоответствие не найдено.

Mismatch(Double[], Double[]) - находит и возвращает индекс первого несоответствия между двумя double массивами, в противном случае возвращает значение -1, если несоответствие не найдено.

Mismatch(Char[], Char[]) - находит и возвращает индекс первого несоответствия между двумя char массивами, в противном случае возвращает значение -1, если несоответствие не найдено.

Mismatch(Byte[], Byte[]) - находит и возвращает индекс первого несоответствия между двумя byte массивами, в противном случае возвращает значение -1, если несоответствие не найдено.

Mismatch(Boolean[], Boolean[]) - находит и возвращает индекс первого несоответствия между двумя boolean массивами, в противном случае возвращает значение -1, если несоответствие не найдено.

Mismatch(Object[], Object[]) - находит и возвращает индекс первого несоответствия между двумя Object массивами, в противном случае возвращает значение -1, если несоответствие не найдено.

Mismatch(Object[], Object[], IComparator) - находит и возвращает индекс первого несоответствия между двумя Object массивами, в противном случае возвращает значение -1, если несоответствие не найдено.

Это лишь небольшой набор методов, предоставляемых классом Arrays для работы с массивами в Java. Не смотря на то, что массивы - это одна из простейших структур данных, их возможности и способы обработки в Java очень многообразны. Умение работать с массивами является неотъемлемой частью разработки программ на этом языке

**ГЛАВА 2**

**2.1** Описание пирамидальной сортировки

Пирамидальная сортировка (или сортировка кучей) - это алгоритм сортировки массива данных, который основан на структуре данных, называемой кучей. Куча представляет собой древовидную структуру, в которой каждый элемент имеет значение не меньше (для кучи максимум) или не больше (для кучи минимум) значений его потомков.

Принцип работы пирамидальной сортировки заключается в построении из массива кучи, затем извлечении элементов из кучи с сохранением порядка сортировки.

Расчет сложности алгоритма.

Для массива из 5-6 элементов пирамидальная сортировка будет иметь следующую сложность:

- Сложность построения первоначальной пирамиды (heapify): O(n)

- Сложность сортировки пирамиды (heapify down): O(log n)

- Итоговая сложность сортировки всего массива: O(n log n)

Таким образом, для массива из 5-6 элементов общая сложность пирамидальной сортировки будет O((n+n) log n) = O(2n log n) = O(n log n), где n - количество элементов в массиве, в данном случае 5-6 элементов.

**2.2** Пример алгоритма

Пример реализации пирамидальной сортировки на Java:

public class HeapSort

{

public void sort(int arr[])

{

int n = arr.length;

// Построение кучи (перегруппируем массив)

for (int i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

// Один за другим извлекаем элементы из кучи

for (int i=n-1; i>=0; i--)

{

// Перемещаем текущий корень в конец

int temp = arr[0];

arr[0] = arr[i];

arr[i] = temp;

// Вызываем процедуру heapify на уменьшенной куче

heapify(arr, i, 0);

}

}

// Процедура для преобразования в двоичную кучу поддерева с корневым узлом i, что является

// индексом в arr[]. n - размер кучи

void heapify(int arr[], int n, int i)

{

int largest = i; // Инициализируем наибольший элемент как корень

int l = 2\*i + 1; // левый = 2\*i + 1

int r = 2\*i + 2; // правый = 2\*i + 2

// Если левый дочерний элемент больше корня

if (l < n && arr[l] > arr[largest])

largest = l;

// Если правый дочерний элемент больше, чем самый большой элемент на данный момент

if (r < n && arr[r] > arr[largest])

largest = r;

// Если самый большой элемент не корень

if (largest != i)

{

int swap = arr[i];

arr[i] = arr[largest];

arr[largest] = swap;

// Рекурсивно преобразуем в двоичную кучу затронутое поддерево

heapify(arr, n, largest);

}

}

/\* Вспомогательная функция для вывода на экран массива размера n \*/

static void printArray(int arr[])

{

int n = arr.length;

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

System.out.print(arr[i]+" ");

System.out.println();

}

// Управляющая программа

public static void main(String args[])

{

int arr[] = {12, 11, 13, 5, 6, 7};

int n = arr.length;

HeapSort ob = new HeapSort();

ob.sort(arr);

System.out.println("Сортед аррей ис");

printArray(arr);

}

}

Выводится:

Сортед аррей ис

5 6 7 11 12 13

Сортировка заняла 1 миллисекунду.

**2.3** Реализация алгоритма и сравнение со стандартным методом

Реализация():

1. generateRandomArray(int size): Создает случайный массив целых чисел заданного размера.

2. heapSort(int[] arr): Реализует алгоритм HeapSort:

- heapify(arr, n, i): Строит кучу из массива arr, где n - размер массива, а i - индекс корневого элемента.

- Сортировка происходит путем извлечения корневого элемента кучи (наибольшего) и его перестановки с последним элементом кучи, после чего выполняется heapify для восстановления кучи.

3. main(String[] args):

- Создает два массива: arr - для сортировки HeapSort, arrCopy - для сравнения с Arrays.sort.

- Измеряет время выполнения heapSort и Arrays.sort.

- Сравнивает результаты сортировки для проверки корректности HeapSort.

4. Time:

- С помощью startTime и endTime выясняем время работы каждого алгоритма.

Сравнение с Arrays.sort:

- Arrays.sort использует алгоритм быстрой сортировки (Quicksort) по умолчанию.

- HeapSort обычно работает медленнее, чем Quicksort.

- Преимущества HeapSort:

- Стабильность времени: HeapSort гарантирует время O(n log n) во всех случаях, в то время как Quicksort в худшем случае может иметь время O(n^2).

- Возможность сортировки "по месту": HeapSort может работать "по месту", не требуя дополнительной памяти для сортировки.

- Используется в алгоритмах приоритетных очередей: HeapSort может быть использован для реализации алгоритмов приоритетных очередей, где требуется быстрое извлечение минимального или максимального элемента.

Таким образом, пирамидальная сортировка и обычные алгоритмы сортировки имеют разные подходы к упорядочиванию элементов массива, и выбор конкретного алгоритма может зависеть от характеристик входных данных и требуемой производительности

**Заключение**

Пирамидальная сортировка – это алгоритм сортировки, который работает по следующему принципу: Сначала мы строим бинарную кучу из нашего массива. Для этого мы проходим по массиву снизу вверх, сравнивая каждый элемент с его потомками и меняя их местами, если нарушается свойство кучи. Затем мы берем самый большой (или самый маленький) элемент и меняем его местами с последним элементом кучи. После этого мы снова делаем кучу из оставшихся элементов, исключая последний. Повторяем это действие, пока не будут отсортированы все элементы.

Сложность алгоритма - это мера ресурсов, которые алгоритм потребляет при обработке данных. Она позволяет нам оценить, насколько быстро или эффективно работает алгоритм при увеличении размера входных данных.

Пирамидальная сортировка имеет гарантированную сложность:  O(nlogn).

Нам удалось узнать, что алгоритм пирамидальной сортировки работает эффективнее всего на больших 𝑛, из-за сложности алгоритма.

**Список источников**

1. Habr: [https://habr.com/ru/companies/otus/articles/460087/](https://vk.com/away.php?utf=1&to=https%3A%2F%2Fhabr.com%2Fru%2Fcompanies%2Fotus%2Farticles%2F460087%2F)
2. JDK 22.0.1 release note <https://jdk.java.net/22/release-notes>
3. Microsoft Learn: [https://learn.microsoft.com/ru-ru/dotnet/api/java.util.arrays.sort?view=net-android-34.0](https://vk.com/away.php?to=https%3A%2F%2Flearn.microsoft.com%2Fru-ru%2Fdotnet%2Fapi%2Fjava.util.arrays.sort%3Fview%3Dnet-android-34.0&cc_key=)
4. Oracle: [www.oracle.com](http://www.oracle.com)
5. Wiki: [https://ru.wikipedia.org/wiki/Пирамидальная\_сортировка](https://vk.com/away.php?utf=1&to=https%3A%2F%2Fru.wikipedia.org%2Fwiki%2F%25D0%259F%25D0%25B8%25D1%2580%25D0%25B0%25D0%25BC%25D0%25B8%25D0%25B4%25D0%25B0%25D0%25BB%25D1%258C%25D0%25BD%25D0%25B0%25D1%258F_%25D1%2581%25D0%25BE%25D1%2580%25D1%2582%25D0%25B8%25D1%2580%25D0%25BE%25D0%25B2%25D0%25BA%25D0%25B0)