Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Ордена Трудового Красного Знамени федеральное государственное

бюджетное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра Математической кибернетики и

информационных технологий

Лабораторная работа №1

на тему: «Методы сортировки»

Выполнила:

Студентка группы БФИ1902

Кареева В.В.

Вариант 5

Проверил:

Москва, 2021 г.

Оглавление

[1. Цель работы 3](#_Toc72268811)

[2. Задание на лабораторную работу 3](#_Toc72268812)

[3. Ход лабораторной работы 3](#_Toc72268813)

[3.1 Задание 1 3](#_Toc72268814)

[3.2 Задание 2 3](#_Toc72268815)

[3.3 Задание 3 4](#_Toc72268816)

[3.3.1 Сортировка выбором 4](#_Toc72268817)

[3.3.2 Сортировка вставкой 4](#_Toc72268818)

[3.3.3 Сортировка обменом 5](#_Toc72268819)

[3.3.4 Сортировка Шелла 5](#_Toc72268820)

[3.3.5 Быстрая сортировка 6](#_Toc72268821)

[3.3.6 Пирамидальная сортировка 6](#_Toc72268822)

[4. Листинг программы 7](#_Toc72268823)

[5. Вывод 9](#_Toc72268824)

[Список использованных источников 11](#_Toc72268825)

# **1. Цель работы**

Цель работы: рассмотреть и изучить основные методы сортировки.

# **2. Задание на лабораторную работу**

1. Вывести на экран приветствие: «Hello, World!»
2. Написать генератор случайных матриц (многомерных), который принимает опциональные параметры m, n, min\_limit, max\_limit, где m и n указывают размер матрицы, а min\_limit и max\_limit – минимальное и максимальное значение для генерируемого числа. По умолчанию при отсутствии параметров принимать следующие значения:

m = 50;

n = 50;

min\_limit = -250;

max\_limit = 1000 + (номер своего варианта)

1. Реализовать методы сортировки строк в соответствии с заданием. Оценить время работы каждого алгоритма сортировки и сравнить его со временем стандартной функции сортировки.

Необходимо использовать следующие методы: выбором, вставкой, обменом, Шелла, турнирная, быстрая, пирамидальная

1. Создать публичный репозиторий на github и запушить выполненное задание.

# **3. Ход лабораторной работы**

* 1. Задание 1

Чтобы вывести на экран приветствие: «Hello, World!» используем следующую строку:

System.*out*.println("Hello, World!");

* 1. Задание 2

Для того, чтобы написать генератор случайных матриц воспользуемся двойным циклом for и просчитаем каждый элемент матрицы в соответствии с формулой:

Mas[i][j] = (int) (Math.*random*() \* (max - min + 1) + min);

Затем пройдемся еще раз по двойному циклу и заполним получившимися элементами матрицу.

* 1. Задание 3

Поэтапно реализуем все необходимые методы сортировки.

* + 1. Сортировка выбором

Сортировка выбором разделяет массив на сортированный и несортированный подмассивы. Сортированный подмассив формируется вставкой минимального элемента не отсортированного подмассива в конец сортированного, заменой.

Для реализации мы в каждой итерации предполагаем, что первый неотсортированный элемент минимален и итерируем по всем оставшимся элементам в поисках меньшего. После нахождения текущего минимума неотсортированной части массива меняем его местами с первым элементом, и он уже становится частью отсортированного массива.

Временная сложность:

При поиске минимума для длины массива проверяются все элементы, поэтому сложность равна O(n). Поиск минимума для каждого элемента массива равен O(n^2).

* + 1. Сортировка вставкой

Этот алгоритм разделяет оригинальный массив на сортированный и несортированный подмассивы. Длина сортированной части равна 1 в начале и соответствует первому (левому) элементу в массиве. После этого остается итерировать массив и расширять отсортированную часть массива одним элементом с каждой новой итерацией.

После расширения новый элемент помещается на свое место в отсортированном подмассиве. Это происходит путём сдвига всех элементов вправо, пока не встретится элемент, который не нужно двигать.

Временная сложность:

В случае убывания массива каждая итерация сдвигает отсортированный массив на единицу O(n). Придется делать это для каждого элемента в каждом массиве, что приведет к сложности равной O(n ^ 2).

* + 1. Сортировка обменом

Метод сортировки, который многие обычно осваивают раньше других из-за его исключительной простоты, называется пузырьковой сортировкой (bubble sort), в рамках которой выполняются следующие действия: проход по файлу с обменом местами соседних элементов, нарушающих заданный порядок, до тех пор, пока файл не будет окончательно отсортирован. Основное достоинство пузырьковой сортировки заключается в том, что его легко реализовать в виде программы. Для понимания и реализации этот алгоритм — простейший, но эффективен он лишь для небольших массивов. Сложность алгоритма: O(n^2). Суть алгоритма пузырьковой сортировки состоит в сравнении соседних элементов и их обмене, если они находятся не в надлежащем порядке. Неоднократно выполняя это действие, мы заставляем наибольший элемент "всплывать" к концу массива. Следующий проход приведет к всплыванию второго наибольшего элемента, и так до тех пор, пока после n-1 итерации массив не будет полностью отсортирован.

* + 1. Сортировка Шелла

Идея метода заключается в сравнение разделенных на группы элементов последовательности, находящихся друг от друга на некотором расстоянии. Изначально это расстояние равно d или N/2, где N — общее число элементов.

На первом шаге каждая группа включает в себя два элемента расположенных

друг от друга на расстоянии N/2; они сравниваются между собой, и, в случае необходимости, меняются местами. На последующих шагах также происходят проверка и обмен, но расстояние d сокращается на d/2, и количество групп, соответственно, уменьшается. Постепенно расстояние между элементами уменьшается, и на d=1 проход по массиву происходит в последний раз.

* + 1. Быстрая сортировка

Выбираем один элемент массива в качестве стержня и

сортируем остальные элементы вокруг (меньшие элементы налево, большие направо). Так соблюдается правильная позиция самого «стержня».

Затем рекурсивно повторяем сортировку для правой и левой частей.

* + 1. Пирамидальная сортировка

Общая идея пирамидальной сортировки заключается в том, что сначала строится пирамида из элементов исходного массива, а затем осуществляется сортировка элементов.

Выполнение алгоритма разбивается на два этапа.

1 этап

Построение пирамиды. Определяем правую часть дерева, начиная с n/2-1 (нижний уровень дерева). Берем элемент левее этой части массива и просеиваем его сквозь пирамиду по пути, где находятся меньшие его элементы, которые одновременно поднимаются вверх; из двух возможных путей выбираете путь через меньший элемент.

2 этап

Сортировка на построенной пирамиде. Берем последний элемент массива в качестве текущего. Меняем верхний (наименьший) элемент массива и текущий местами. Текущий элемент (он теперь верхний) просеиваем сквозь n-1 элементную пирамиду. Затем берем предпоследний элемент и т.д.

В худшем случае требуется n·log2n шагов, сдвигающих элементы.

# **4. Листинг программы**

package com.company;  
import java.util.\*;  
import java.lang.\*;  
import java.util.Arrays;  
import java.io.\*;  
  
public class Main {  
  
 public static void main(String[] args) {  
 System.out.println("hello world");  
 matrout();  
  
 }  
  
 public static void matrout() {  
 int rows, col, min, max;  
  
 Scanner scan1 = new Scanner(System.in);  
  
 System.out.println("Введите количество строк и столбцов: ");  
 rows = scan1.nextInt();  
 col = scan1.nextInt();  
 int[][] a = new int[rows][col];  
 System.out.println("Введите минимальное и максимальное значение: ");  
 min = scan1.nextInt();  
 max = scan1.nextInt();  
  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < col; j++) {  
 a[i][j] = (int) (Math.random() \* (max - min + 1) + min);  
 }  
 }  
  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < col; j++) {  
 System.out.print(a[i][j] + " ");  
 }  
 System.out.println();  
 }  
  
 int[] b = vodnomern(a, rows, col);  
 int low = 0;  
 int high = b.length - 1;  
  
 int[][] a1 = a;  
 viborom(rows, col, a1);  
 int[][] a2 = a;  
 vstavkoy(rows, col, a2);  
 int[][] a3 = a;  
 obmenom(rows, col, a3);  
 int[][] a4 = a;  
 shellom(rows, col, a4);  
 int[] b1 = b;  
 pyramid(b1);  
 System.out.println("Пирамидальная сортировка:");  
 int[][] a5 = vdvumern(b1, rows, col);  
  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < col; j++) {  
 System.out.print(a5[i][j] + " ");  
 }  
 System.out.println();  
 }  
 int[] b2 = b;  
 quickSort(rows, col, b2, low, high);  
  
 System.out.println("Сортировка методом быстрой сортировки:");  
 int[][] a6 = vdvumern(b2, rows, col);  
  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < col; j++) {  
 System.out.print(a6[i][j] + " ");  
 }  
 System.out.println();  
 }  
  
 int[] b3 = b;  
 Sort(b3);  
  
 System.out.println("Турнирная сортировка:");  
 int[][] a7 = vdvumern(b3, rows, col);  
  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < col; j++) {  
 System.out.print(a7[i][j] + " ");  
 }  
 System.out.println();  
 }  
  
  
  
 }  
  
 public static int[] vodnomern(int[][] a, int rows, int col) {  
 int[] b = new int[rows \* col];  
 int t = 0;  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < col; j++) {  
 b[t] = a[i][j];  
 t++;  
 }  
 }  
 return b;  
 }  
  
 public static int[][] vdvumern(int[] b, int rows, int col) {  
 int[][] a = new int[rows][col];  
 int t = 0;  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < col; j++) {  
 a[i][j] = b[t];  
 t++;  
 }  
 }  
 return a;  
 }  
  
 public static void viborom(int rows, int col, int[][] a) {  
 System.out.println("Сортировка выбором: ");  
  
 int[] b = vodnomern(a, rows, col);  
  
 for (int i = 0; i < rows \* col; i++) {  
 int pos = i;  
 int min = b[i];  
 for (int j = i + 1; j < rows \* col; j++) {  
 if (b[j] < min) {  
 pos = j;  
 min = b[j];  
 }  
  
 }  
 b[pos] = b[i];  
 b[i] = min;  
 }  
  
 a = vdvumern(b, rows, col);  
  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < col; j++) {  
 System.out.print(a[i][j] + " ");  
 }  
 System.out.println();  
 }  
 // System.out.print(Arrays.toString(a));  
  
 }  
  
 public static void vstavkoy (int rows, int col, int[][] a){  
 System.out.println("Сортировка вставкой:");  
  
 int[] b = vodnomern(a, rows, col);  
  
 for (int i = 1; i < b.length; i++) {  
 int current = b[i];  
 int j = i - 1;  
 while (j >= 0 && current < b[j]) {  
 b[j + 1] = b[j];  
 j--;  
 }  
 b[j + 1] = current;  
 }  
  
 a = vdvumern(b, rows, col);  
  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < col; j++) {  
 System.out.print(a[i][j] + " ");  
 }  
 System.out.println();  
 }  
 }  
  
 public static void obmenom(int rows, int col, int[][] a){  
 System.out.println("Сортировка обменом:");  
  
 int[] b = vodnomern(a, rows, col);  
  
 for (int i = 0; i < b.length - 1; i++) {  
 for (int j = b.length - 1; j > i; j--) {  
 if (b[j - 1] > b[j]) {  
 int tmp = b[j - 1];  
 b[j - 1] = b[j];  
 b[j] = tmp;  
 }  
 }  
 }  
  
 a = vdvumern(b, rows, col);  
  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < col; j++) {  
 System.out.print(a[i][j] + " ");  
 }  
 System.out.println();  
 }  
  
 }  
  
 public static void shellom(int rows, int col, int[][] a) {  
 System.out.println("Сортировка методом Шелла:");  
  
 int[] b = vodnomern(a, rows, col);  
  
 int h = 0;  
  
  
 while (h <= b.length / 3)  
 h = 3 \* h + 1;  
  
 for (int k = h; k > 0; k = (k - 1) / 3) {  
 for (int i = k; i < b.length; i++) {  
 int temp = b[i];  
 int j;  
 for (j = i; j >= k; j -= k) {  
 if (temp < b[j - k])  
 b[j] = b[j - k];  
 else  
 break;  
 }  
 b[j] = temp;  
 }  
 }  
  
 a = vdvumern(b, rows, col);  
  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < col; j++) {  
 System.out.print(a[i][j] + " ");  
 }  
 System.out.println();  
 }  
 }  
  
 public static void quickSort(int rows, int col, int[] b, int low, int high){  
  
  
 if (b.length == 0)  
 return;//завершить выполнение если длина массива равна 0  
  
 if (low >= high)  
 return;  
  
 int middle = low + (high - low) / 2;  
 int opora = b[middle];  
  
 // разделить на подмассивы, который больше и меньше опорного элемента  
 int i = low, j = high;  
 while (i <= j) {  
 while (b[i] < opora) {  
 i++;  
 }  
  
 while (b[j] > opora) {  
 j--;  
 }  
  
 if (i <= j) {//меняем местами  
 int temp = b[i];  
 b[i] = b[j];  
 b[j] = temp;  
 i++;  
 j--;  
 }  
 }  
  
  
 // вызов рекурсии для сортировки левой и правой части  
 if (low < j)  
 quickSort(rows, col, b, low, j);  
  
 if (high > i)  
 quickSort(rows, col, b, i, high);  
  
  
 }  
  
  
 public static void pyramid(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  
 public static 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);  
 }  
 }  
  
 public static class Node  
 {  
 public int data;  
 public int id;  
  
 public Node()  
 {  
  
 }  
 public Node(int \_data, int \_id)//  
 {  
 data = \_data;  
 id = \_id;  
 }  
 }  
  
 public static void Adjust(Node[] data, int idx)  
 {  
 while(idx != 0)  
 {  
 if(idx % 2 == 1)  
 {  
 if(data[idx].data < data[idx + 1].data)  
 {  
 data[(idx - 1)/2] = data[idx];  
 }  
 else  
 {  
 data[(idx-1)/2] = data[idx + 1];  
 }  
 idx = (idx - 1)/2;  
 }  
 else  
 {  
 if(data[idx-1].data < data[idx].data)  
 {  
 data[idx/2 - 1] = data[idx-1];  
 }  
 else  
 {  
 data[idx/2 - 1] = data[idx];  
 }  
 idx = (idx/2 - 1);  
 }  
 }  
 }  
  
 public static void Sort(int[] data)  
 {  
  
 int nNodes = 1;  
 int nTreeSize;  
 while(nNodes < data.length)  
 {  
 nNodes \*= 2;  
 }  
 nTreeSize = 2 \* nNodes - 1;  
  
 Node[] nodes = new Node[nTreeSize];  
  
 int i, j;  
 int idx;  
 for( i = nNodes - 1; i < nTreeSize; i++)  
 {  
 idx = i - (nNodes - 1);  
 if(idx < data.length)  
 {  
 nodes[i] = new Node(data[idx], i);  
 }  
 else  
 {  
 nodes[i] = new Node(Integer.MAX\_VALUE, -1);  
 }  
  
 }  
  
 for( i = nNodes - 2; i >= 0; i--)  
 {  
 nodes[i] = new Node();  
 if(nodes[i \* 2 + 1].data < nodes[i \* 2 + 2].data)  
 {  
 nodes[i] = nodes[i\*2 + 1];  
 }  
 else  
 {  
 nodes[i] = nodes[i\*2 + 2];  
 }  
 }  
 for( i = 0; i < data.length; i++)  
 {  
 data[i] = nodes[0].data;  
 nodes[nodes[0].id].data = Integer.MAX\_VALUE;  
 Adjust(nodes, nodes[0].id);  
  
 }  
 }  
  
}

# **5. Вывод**

Вывод: Мы рассмотрели и изучили основные методы сортировки, такие как сортировка выбором, вставкой, обменом, Шелла, быстрая сортировка и пирамидальная.

# **Список использованных источников**

1) ГОСТ 7.32-2017 Система стандартов по информации, библиотечному и издательскому делу. Отчёт о научно-исследовательской работе. Структура и правила оформления

2) ГОСТ 7.1-2003 Библиографическая запись. Библиографическое описание. Общие требования и правила составления