**Федеральное агентство связи**

**Государственное бюджетное образовательное учреждение**

**высшего образования**

**ордена Трудового Красного Знамени**

**«Московский технический университет связи и информатики»**

Кафедра

«Математическая кибернетика и информационные технологии»

**Лабораторная работа №1**

**по дисциплине «Структура и алгоритмы обработки данных»**

**По теме «Методы сортировки»**

Выполнил студент группы БФИ 1902

Леонов Н.Н.

Научный руководитель:

Мкртчян Г.М.

Москва 2021

**Задание на лабораторную работу**: создать генератор случайных матриц по указанным требованиям и реализовать методы сортировки строк числовой матрицы.

**Ход выполнения лабораторной работы**

Задание выполнено путем создания класса Matrix, в котором создается двумерный массив, конструктор создания требуемой матрицы, необходимые для этого методы. Все действия с данной матрицой будут выполняться через экземпляр класса Matrix.

Также были созданы нестатические методы, выполняющие сортировку строк матрицы и необходимые для них статические методы.

Листинг:

import java.util.Random;  
public class Matrix {  
 int [][] matrixA;  
 int len;  
 int width;  
 int minValue;  
 int maxValue;  
 public Matrix (int m, int n, int min\_limit, int max\_limit) {  
 len=m;  
 width=n;  
 minValue=min\_limit;  
 maxValue=max\_limit;  
 matrixA = new int [m][n];  
 for (int i=0; i<len;i++) {  
 for (int j = 0; j < width; j++) {  
 matrixA[i][j]=*randomNum*(minValue,maxValue);  
 //System.out.print(matrixA[i][j] + "\t\t");  
 }  
 System.*out*.println();  
 }  
 }  
 public int getSingleValue(int m, int n) {  
 return this.matrixA[m][n];  
 }  
 public int[][] getMatrix() {  
 return this.matrixA;  
 }  
 public static int randomNum(int min, int max) {  
 Random random = new Random();  
 int randomValue = random.nextInt();  
 if (randomValue<min && min<0) {  
 while (randomValue<min) {  
 randomValue /= 10;  
 }  
 }  
 if (randomValue<min && min>0) {  
 while (randomValue<min) {  
 randomValue\*=10;  
 }  
 }  
 if (randomValue>max&&max<0) {  
 while (randomValue>max) {  
 randomValue\*=10;  
 }  
 }  
 if (randomValue>max&&max>0) {  
 while (randomValue>max) {  
 randomValue/=10;  
 }  
 }  
 return randomValue;  
 }  
 public void sortChoose () {  
 int index;  
 int min;  
 for (int i = 0; i < len; i++) {  
 for (int j = 0; j < width; j++) {  
 min = matrixA[i][j]; // записываем в максимум каждый эелемент строки в матрице  
 index = j; // находим его индекс  
 for (int c = j + 1; c < len; c++) { // находим максимум и индекс элемент  
 if (matrixA[i][c] < min) {  
 min = matrixA[i][c];  
 index = c;  
 }  
 }  
 if (j != index) { //если текущий элемент не совпадает с максимальным  
 int zero = matrixA[i][j];  
 matrixA[i][j] = min; //меняем местами  
 matrixA[i][index] = zero;  
 }  
 }  
 }  
 }  
 public void sortInsert () {  
 for (int i = 0; i < len; i++) {  
 for (int j = 1; j < width; j++) {  
 for (int c = j; c > 0 && matrixA[i][c - 1] > matrixA[i][c]; c--) { //если предыдущий больше текущего  
 int z = matrixA[i][c];  
 matrixA[i][c] = matrixA[i][c - 1]; //меняем местами  
 matrixA[i][c - 1] = z;  
 }  
 }  
 }  
 }  
 public void sortExchange () {  
 for (int i = 0; i < len; i++) {  
 boolean needIteration = true; //конструкция для перехода к след строке  
 while (needIteration) {  
 needIteration = false;  
 for (int j = 1; j < width; j++) {  
 if (matrixA[i][j] < matrixA[i][j - 1]) { //сравнивает соседние элементы и меняет местами если правый больше  
 int z = matrixA[i][j];  
 matrixA[i][j] = matrixA[i][j - 1];  
 matrixA[i][j - 1] = z;  
 needIteration = true;  
 }  
 }  
 }  
 }  
 }  
 public void sortShell() {  
 int d = len / 2; //шаг  
 for (int i = 0; i < len; i++) {  
 while (d > 0) { //если шаг больше нуля  
 for (int j = 0; j < width - d; j++) {  
 int q = j;  
 while (q >= 0 && matrixA[i][q] > matrixA[i][q + d]) {  
 int temp = matrixA[i][q]; //меняет местами элементы с шагом d который уменьшается в половину с каждым проходом цикла  
 matrixA[i][q] = matrixA[i][q + d];  
 matrixA[i][q + d] = temp;  
 q--;  
 }  
 }  
 d = d / 2;  
 }  
  
 }  
 }  
 public static void heapify(int[] array, int length, int i) {  
 int leftChild = 2 \* i + 1; //позиции дочерних эллементов  
 int rightChild = 2 \* i + 2;  
 int largest = i; //индекс самого большого изначально записываем как текущий  
  
 // если левый дочерний больше родительского  
 if (leftChild < length && array[leftChild] > array[largest]) {  
 largest = leftChild;  
 }  
  
 // если правый дочерний больше родительского  
 if (rightChild < length && array[rightChild] > array[largest]) {  
 largest = rightChild;  
 }  
  
 // если должна произойти замена  
 if (largest != i) {  
 int temp = array[i];  
 array[i] = array[largest];  
 array[largest] = temp;  
 *heapify*(array, length, largest);  
 }  
 }  
 public static void heapSort(int[] array) {  
 if (array.length == 0) return;  
 int length = array.length;  
 // Построение кучи (перегруппируем массив)  
 for (int i = length / 2 - 1; i >= 0; i--)  
 *heapify*(array, length, i);  
 // Один за другим извлекаем элементы из кучи  
 for (int i = length - 1; i >= 0; i--) {  
 // Перемещаем текущий корень в конец  
 int temp = array[0];  
 array[0] = array[i];  
 array[i] = temp;  
 // Вызываем процедуру heapify на уменьшенной куче  
 *heapify*(array, i, 0);  
 }  
 }  
 public void sortPyramid() {  
 int[] arr1 = new int[len];  
 for (int i = 0; i < len; i++) {  
 for (int j = 0; j < width; j++) {  
 arr1[j] = matrixA[i][j];  
 }  
 *heapSort*(arr1);  
 for (int j = 0; j < width; j++) {  
 matrixA[i][j] = arr1[j];  
 }  
 }  
 }  
 static int partition(int[] array, int begin, int end) {  
 int pivot = end;  
  
 int counter = begin;  
 for (int i = begin; i < end; i++) {  
 if (array[i] < array[pivot]) { //перемещает значения меньшие чем pivot в левую от него часть, большие в правую  
 int temp = array[counter];  
 array[counter] = array[i];  
 array[i] = temp;  
 counter++;  
 }  
 }  
 int temp = array[pivot];  
 array[pivot] = array[counter];  
 array[counter] = temp;  
  
 return counter;  
 }  
  
 public static void quickSort(int[] array, int begin, int end) {  
 if (end <= begin) return;  
 int pivot = *partition*(array, begin, end); //выбираем опорный элемент  
 *quickSort*(array, begin, pivot - 1); //левая часть  
 *quickSort*(array, pivot + 1, end); //правая часть  
 }  
 public void sortFast() {  
 int[] arr1 = new int[len];  
 for (int i = 0; i < len; i++) {  
 for (int j = 0; j < width; j++) {  
 arr1[j] = matrixA[i][j];  
 }  
  
 *quickSort*(arr1, 0, len - 1);  
 for (int j = 0; j < width; j++) {  
 matrixA[i][j] = arr1[j];  
 }  
 }  
 }  
 private class Node {  
 public int data;  
 public int id;  
  
 public Node() {  
  
 }  
  
 public Node(int \_data, int \_id)//  
 {  
 data = \_data;  
 id = \_id;  
 }  
 }  
  
 public 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 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);  
  
 }  
 }  
 public void sortTour() {  
 int[] arr1 = new int[len];  
 for (int i = 0; i < len; i++) {  
 for (int j = 0; j < width; j++) {  
 arr1[j] = matrixA[i][j];  
 }  
  
 Sort(arr1);  
 for (int j = 0; j < width; j++) {  
 matrixA[i][j] = arr1[j];  
 }  
 }  
 }  
 public static void main(String[] args) {  
 //int [][] matrixM;  
 //matrixM = new int [10][10];  
 Matrix mat = new Matrix(10,10,250,1000);  
 for (int i=0; i<mat.len; i++) {  
 for (int j=0; j<mat.width; j++) {  
 System.*out*.print(mat.getSingleValue(i,j) + "\t\t");  
 }  
 System.*out*.println();  
 }  
 System.*out*.println();  
 mat.sortTour();  
 for (int i=0; i<mat.len; i++) {  
 for (int j=0; j<mat.width; j++) {  
 System.*out*.print(mat.getSingleValue(i,j) + "\t\t");  
 }  
 System.*out*.println();  
 }  
 }  
}

**Результат выполнения лабораторной работы.**

Результат показан на примере использования турнирной сортировки, путем вызова соответствующего метода.

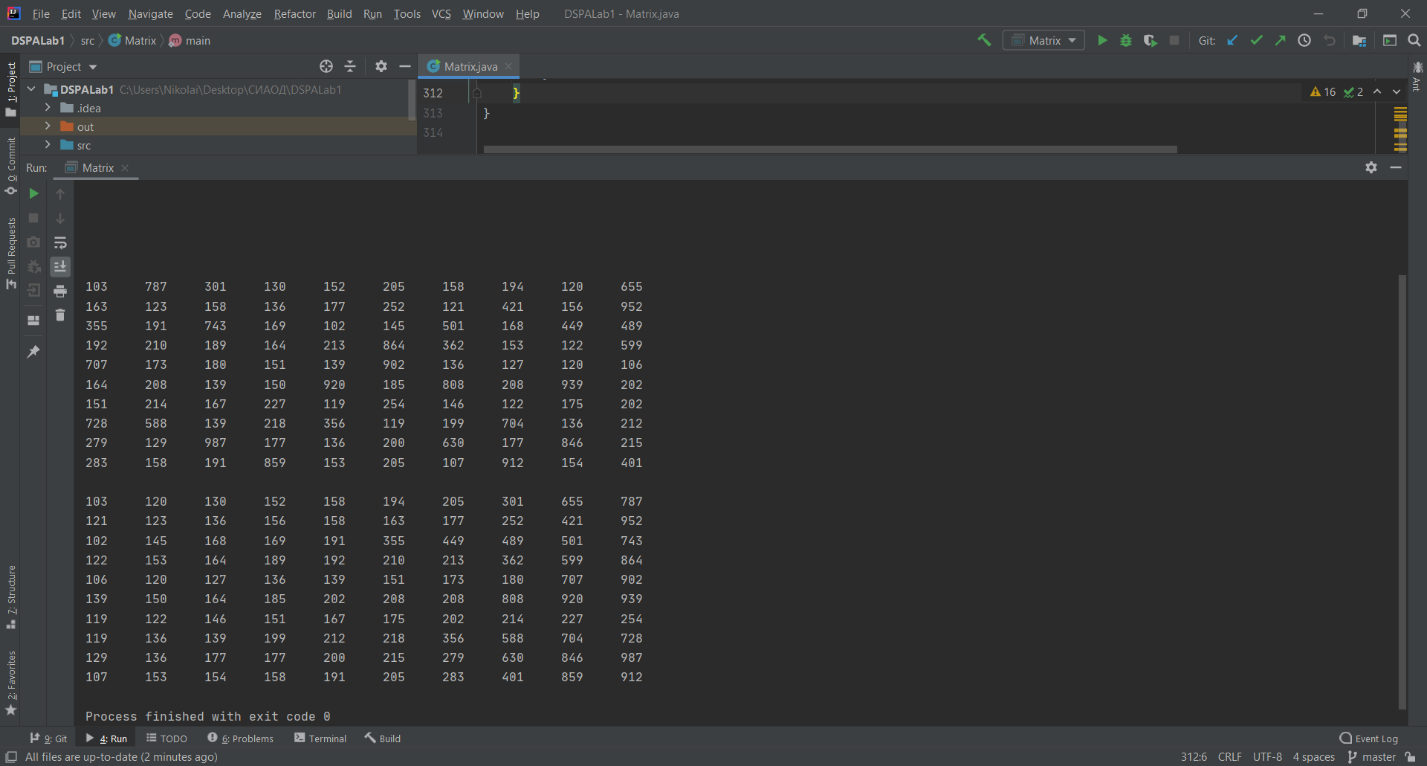


Рисунок 1 – Результат выполнения лабораторной работы