Федеральное агенство связи

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

по дисциплине: «Методы сортировки.»

Выполнил студент

группы БФИ1902

Рахимов Е.К.

Проверила:

Мосева М.С.

Москва, 2021 г.

Оглавление

[1. Цель лабораторной работы 2](#_Toc58524139)

[2. Задание на лабораторную работу 3](#_Toc58524140)

[3. Ход лабораторной работы 4](#_Toc58524141)

[3.1 Листинг программы 4](#_Toc58524142)

[3.2 Результат выполнения программы 7](#_Toc58524143)

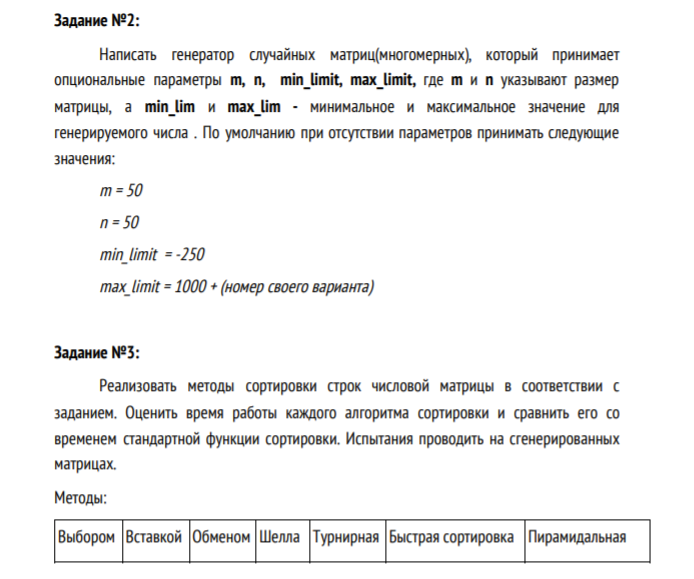
[Список использованных источников 8](#_Toc58524144)

# 1. Цель лабораторной работы

Цель данной лабораторной работы — научиться пользоваться сортировками.

# 2. Задание на лабораторную работу

# 



# 3. Ход лабораторной работы

## 3.1 Листинг программы

package sample;  
  
  
import JJ.HeapSortApp;  
  
public class Main {  
 private static int *MAXINT*=32767;  
 public static void main(String[] args) {  
  
 *hello\_World*();  
 int[][] firstArray = new int[50][50];  
 *generationMatrix*(firstArray);  
 System.*out*.println("default matrix");  
 *printMatrix*(*peredelVodnomer*(firstArray));  
 System.*out*.println("///////");  
 int[][] obmenSortArray = firstArray;  
 int[][] selectSortArray = firstArray;  
 int[][] insertSortArray = firstArray;  
 int[][] shelSortArray = firstArray;  
 int[][] quickSortArray = firstArray;  
 int[][] pyramidSortArray = firstArray;  
 int[][] tourSortArray = firstArray;  
  
 System.*out*.println("SelectSort:");  
 *selectionSort*(selectSortArray);  
 System.*out*.println();  
  
 System.*out*.println("insertSort:");  
 *insertSort*(insertSortArray);  
 System.*out*.println();  
  
 System.*out*.println("ObmenSort:");  
 *obmenSort*(obmenSortArray);  
 System.*out*.println();  
  
 System.*out*.println("ShellSort:");  
 *shellSort*(shelSortArray);  
 System.*out*.println();  
  
 System.*out*.println("QuickSort:");  
 *quickStart*(quickSortArray);  
  
  
  
 System.*out*.println();  
 System.*out*.println("Pyramid");  
 *pyramidStart*(pyramidSortArray);  
  
 System.*out*.println();  
 System.*out*.println("Tournament:");  
 *quickTourn*(tourSortArray);  
  
  
 }  
  
  
  
  
  
 public static int getRandom (int min, int max){  
 max -= min;//делаем число на min меньше чтобы сделать в дальнейшем диапазон от min до max-min, а потом сделать +1 - чтобы включить конечную границу и сделать +min  
 return (int) (Math.*random*() \* (++max)) + min;  
 }  
 public static void generationMatrix(int[][] matrix) {  
 for (int i = 0; i < 50; i++) {  
 for (int j = 0; j < 50; j++) {  
 matrix[i][j] = *getRandom*(-250, 1014);  
 }  
 }  
 }  
 public static int[] peredelVodnomer(int[][] matrix) {  
 int[] mass = new int[2500];  
 for (int i = 0; i < 50; i++)  
 for (int j = 0; j < 50; j++)//циклы прохода двумерного массива, опираясь на которые мы выражаем одномерный  
 {  
 mass[i \* 50 + j] = matrix[i][j];  
 }  
 return mass;  
 }  
 public static void printMatrix(int[] mas) {  
  
 int j = 0;  
 for (int i = 0; i < 2500; i++) {  
 if (j == 50) {  
 System.*out*.println();  
 j = 0;  
 }  
 System.*out*.print(mas[i] + " ");  
 j++;  
 }  
 }  
 public static void obmenSort(int[][] matrix) {  
 int[] mass = *peredelVodnomer*(matrix);  
 boolean needIteration = true;  
 while (needIteration) {  
 needIteration = false;  
 for (int i = 1; i < mass.length; i++) {  
 if (mass[i] < mass[i - 1]) {  
 *swap*(mass, i, i - 1);  
 needIteration = true;  
 }  
 }  
 }  
 *printMatrix*(mass);  
 }  
 public static void selectionSort(int[][] matrix) {  
 int[] mas = *peredelVodnomer*(matrix);  
 for (int left = 0; left < mas.length; left++) {  
 int minInd = left;  
 for (int i = left; i < mas.length; i++) {  
 if (mas[i] < mas[minInd]) {  
 minInd = i;  
 }  
 }  
 *swap*(mas, left, minInd);  
 }  
 *printMatrix*(mas);  
 }  
 public static void insertSort(int[][] matrix) {  
 int[] mas = *peredelVodnomer*(matrix);  
 for (int left = 0; left < mas.length; left++) {  
 // Вытаскиваем значение элемента  
 int value = mas[left];  
 // Перемещаемся по элементам, которые перед вытащенным элементом  
 int i = left - 1;  
 for (; i >= 0; i--) {  
 // Если вытащили значение меньшее — передвигаем больший элемент дальше  
 if (value < mas[i]) {  
 mas[i + 1] = mas[i];  
 } else {  
 // Если вытащенный элемент больше — останавливаемся  
 break;  
 }  
 }  
 // В освободившееся место вставляем вытащенное значение  
 mas[i + 1] = value;  
  
 }  
 *printMatrix*(mas);  
 }  
 public static void shellSort(int [] [] matrix){  
 int [] mas = *peredelVodnomer*(matrix);  
 // Высчитываем промежуток между проверяемыми элементами  
 int gap = mas.length / 2;  
 // Пока разница между элементами есть  
 while (gap >= 1) {  
 for (int right = 0; right < mas.length; right++) {  
 // Смещаем правый указатель, пока не сможем найти такой, что  
 // между ним и элементом до него не будет нужного промежутка  
 for (int c = right - gap; c >= 0; c -= gap) {  
 if (mas[c] > mas[c + gap]) {  
 *swap*(mas, c, c + gap);  
 }  
 }  
 }  
 // Пересчитываем разрыв  
 gap = gap / 2;  
 }  
 *printMatrix*(mas);  
 }  
 public static void quickStart(int [] [] matrix){  
 int [] mas =*peredelVodnomer*(matrix);  
 *quickSort*(mas,0,2499);  
 *printMatrix*(mas);  
 }  
 public static void pyramidStart(int [] [] matrix){  
 int [] mas =*peredelVodnomer*(matrix);  
 HeapSort d= new HeapSort();  
 d.sort(mas);  
 *printMatrix*(mas);  
 }  
 public static void quickSort(int[] source, int leftBorder, int rightBorder) {  
 int leftMarker = leftBorder;  
 int rightMarker = rightBorder;  
 int pivot = source[(leftMarker + rightMarker) / 2];//определяется опорный элемент  
 do {  
 // Двигаем левый маркер слева направо пока элемент меньше, чем pivot  
 while (source[leftMarker] < pivot) {  
 leftMarker++;  
 }  
 // Двигаем правый маркер, пока элемент больше, чем pivot  
 while (source[rightMarker] > pivot) {  
 rightMarker--;  
 }  
 // Проверим, не нужно обменять местами элементы, на которые указывают маркеры  
 if (leftMarker <= rightMarker) {  
 // Левый маркер будет меньше правого только если мы должны выполнить swap  
 if (leftMarker < rightMarker) {  
 *swap*(source,leftMarker,rightMarker);  
 }  
 // Сдвигаем маркеры, чтобы получить новые границы  
 leftMarker++;  
 rightMarker--;  
 }  
 } while (leftMarker <= rightMarker);  
  
 // Выполняем рекурсивно для частей  
 if (leftMarker < rightBorder) {  
 *quickSort*(source, leftMarker, rightBorder);  
 }  
 if (leftBorder < rightMarker) {  
 *quickSort*(source, leftBorder, rightMarker);  
 }  
 }  
 public static void swap ( int[] array, int ind1, int ind2){  
 int tmp = array[ind1];  
 array[ind1] = array[ind2];  
 array[ind2] = tmp;  
 }  
 public static void hello\_World (){  
 System.*out*.printf("Hello,World!\n");  
 }  
 public static void quickTourn(int [] [] matrix){  
 int [] mas =*peredelVodnomer*(matrix);  
 TournamentSort s = new TournamentSort();  
 s.Sort(mas);  
 *printMatrix*(mas);  
 }  
}

package sample;  
  
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  
 Main.*swap*(arr,0,i);  
  
 // 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)  
 { Main.*swap*(arr,i,largest);  
 // Рекурсивно преобразуем в двоичную кучу затронутое поддерево  
 heapify(arr, n, largest);  
 }  
 }  
  
}

package sample;  
  
public class Node {  
 public int iData; //Данные(ключ)  
 public int idd;  
 public Node leftChild; // Левый потомок узла  
 public Node rightChild; // Правый потомок узла  
 public Node(int key){  
 iData=key;  
 }  
 public Node(int key, int id){  
 iData=key;  
 idd=id;  
 }  
 public Node (){}  
  
 public int getKey() {  
 return iData;  
 }  
}

package sample;  
  
public class TournamentSort {  
 public void Adjust(Node[] data, int idx)  
 {  
 while(idx != 0)  
 {  
 if(idx % 2 == 1)  
 {  
 if(data[idx].iData < data[idx + 1].iData)  
 {  
 data[(idx - 1)/2] = data[idx];  
 }  
 else  
 {  
 data[(idx-1)/2] = data[idx + 1];  
 }  
 idx = (idx - 1)/2;  
 }  
 else  
 {  
 if(data[idx-1].iData < data[idx].iData)  
 {  
 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];  
 //initialize the data  
  
 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].iData < nodes[i \* 2 + 2].iData)  
 {  
 nodes[i] = nodes[i\*2 + 1];  
 }  
 else  
 {  
 nodes[i] = nodes[i\*2 + 2];  
 }  
 }  
 //the real sorting procedure  
 for( i = 0; i < data.length; i++)//ʵ������Ĺ���  
 {  
 data[i] = nodes[0].iData;//ȡ����С��  
 nodes[nodes[0].idd].iData = Integer.*MAX\_VALUE*;  
 Adjust(nodes, nodes[0].idd);  
  
 }  
  
  
  
 }  
}

## 3.2 Результат выполнения программы

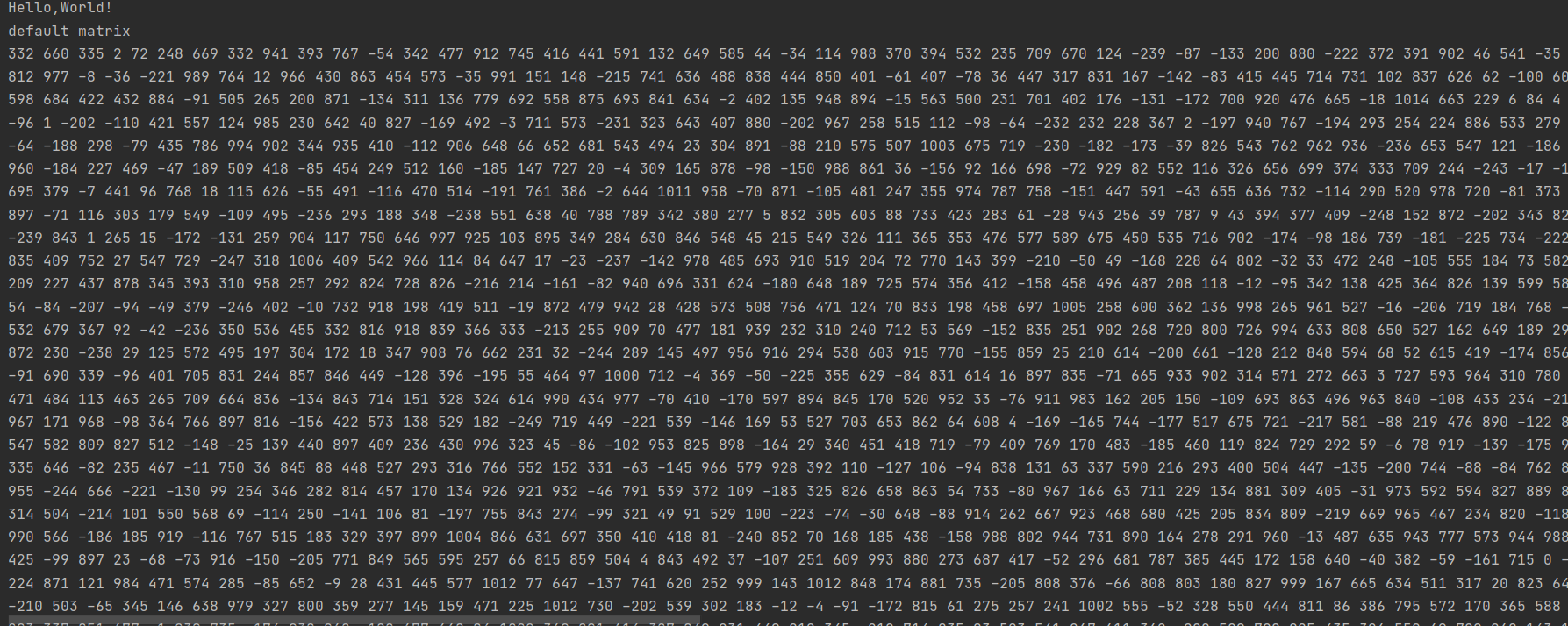


Рисунок 1 – результат выполнения

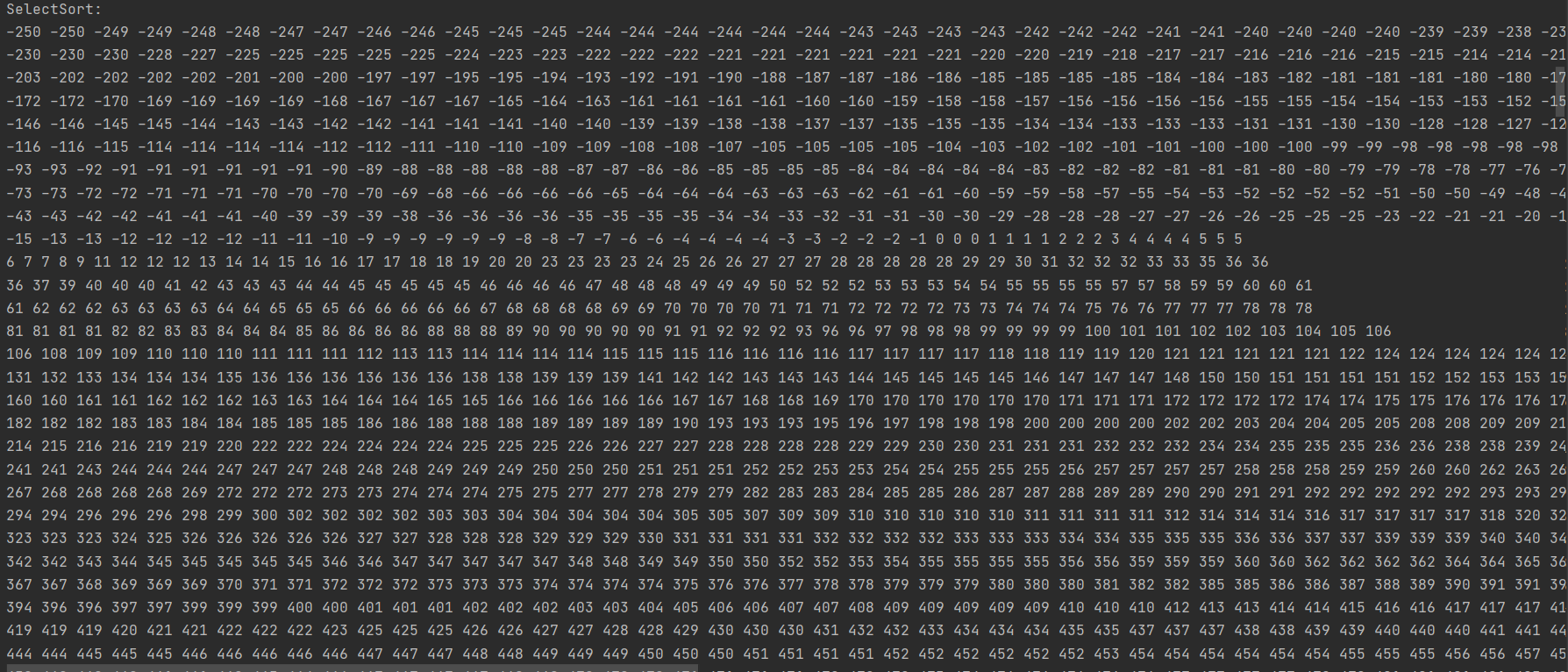


Рисунок 2 – результат выполнения

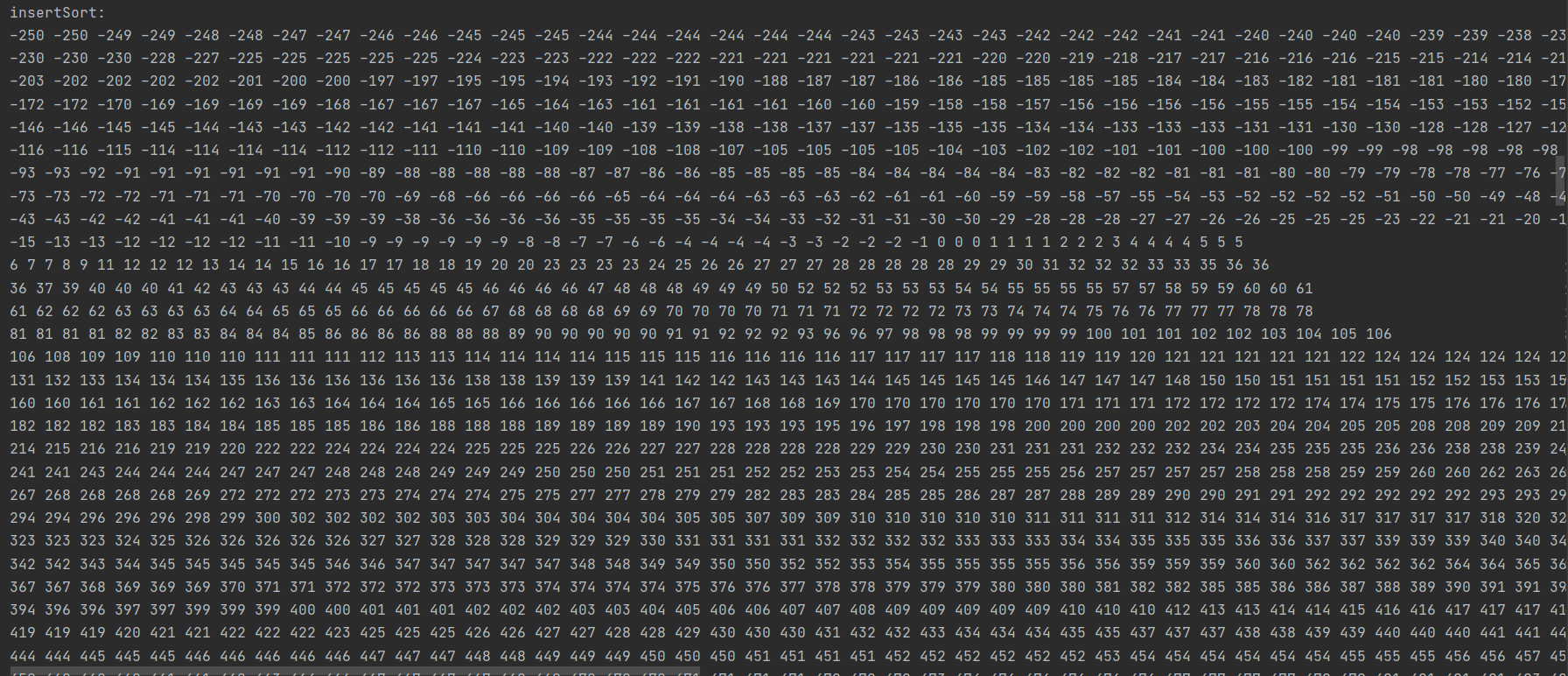


Рисунок 3 – результат выполнения

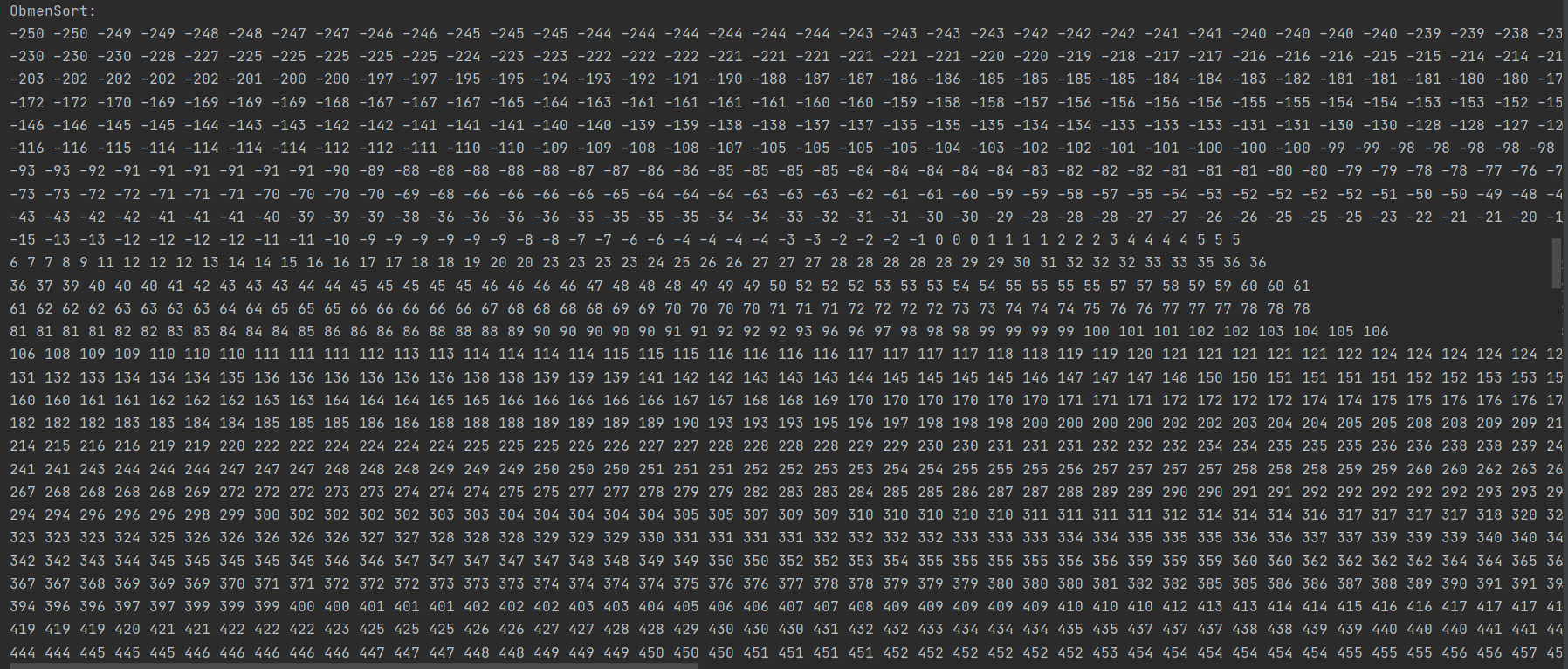


Рисунок 4 – результат выполнения

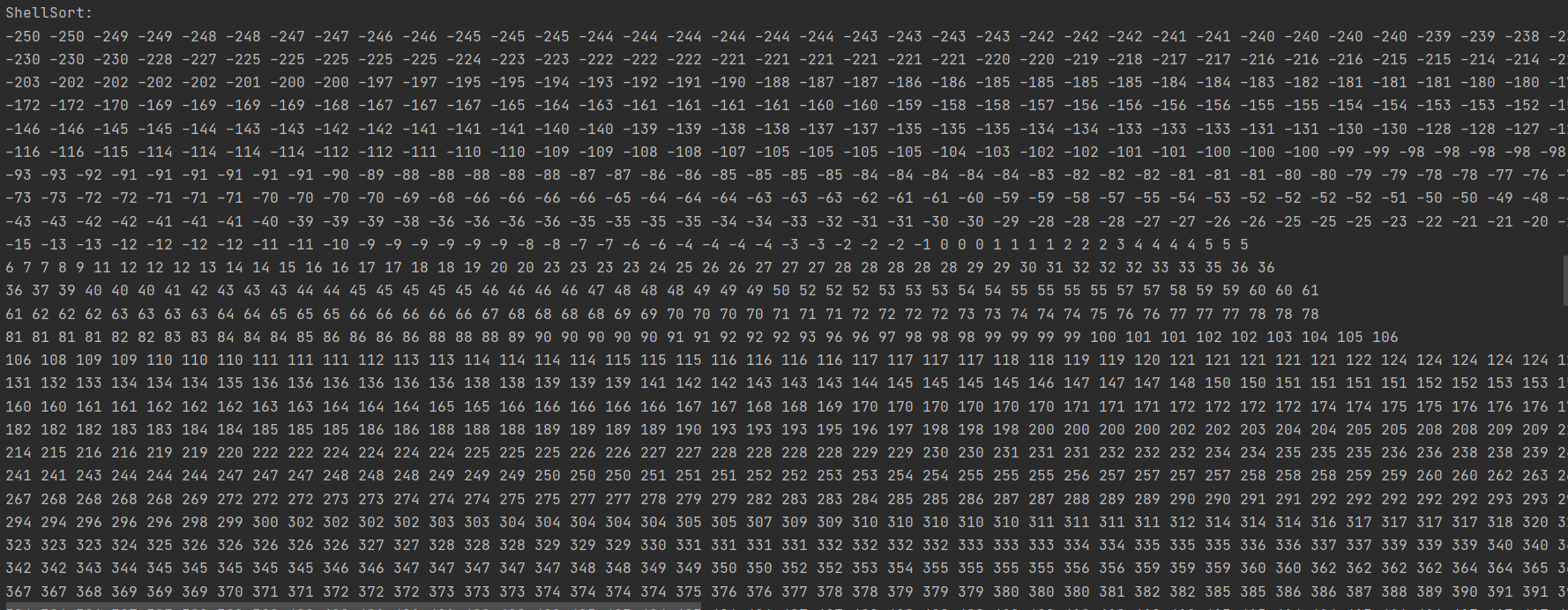


Рисунок 5 – результат выполнения

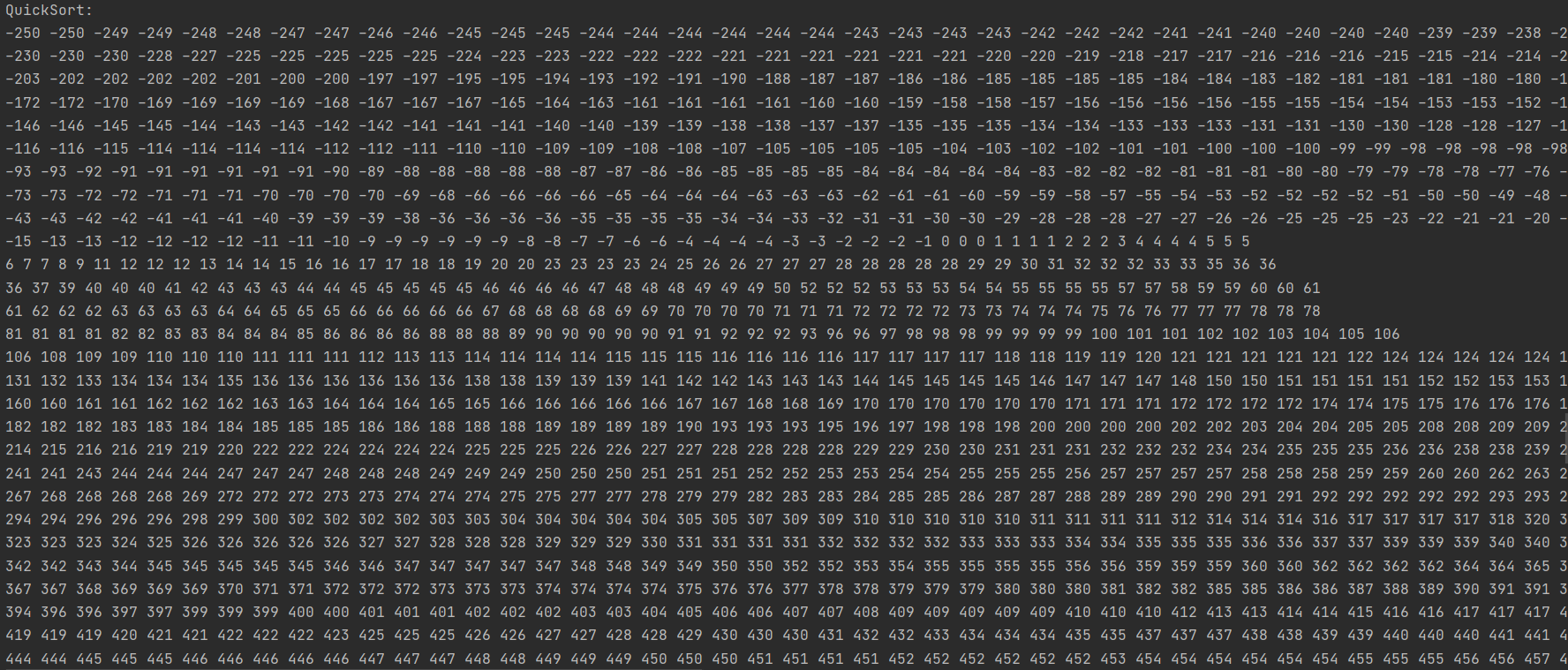


Рисунок 6 – результат выполнения

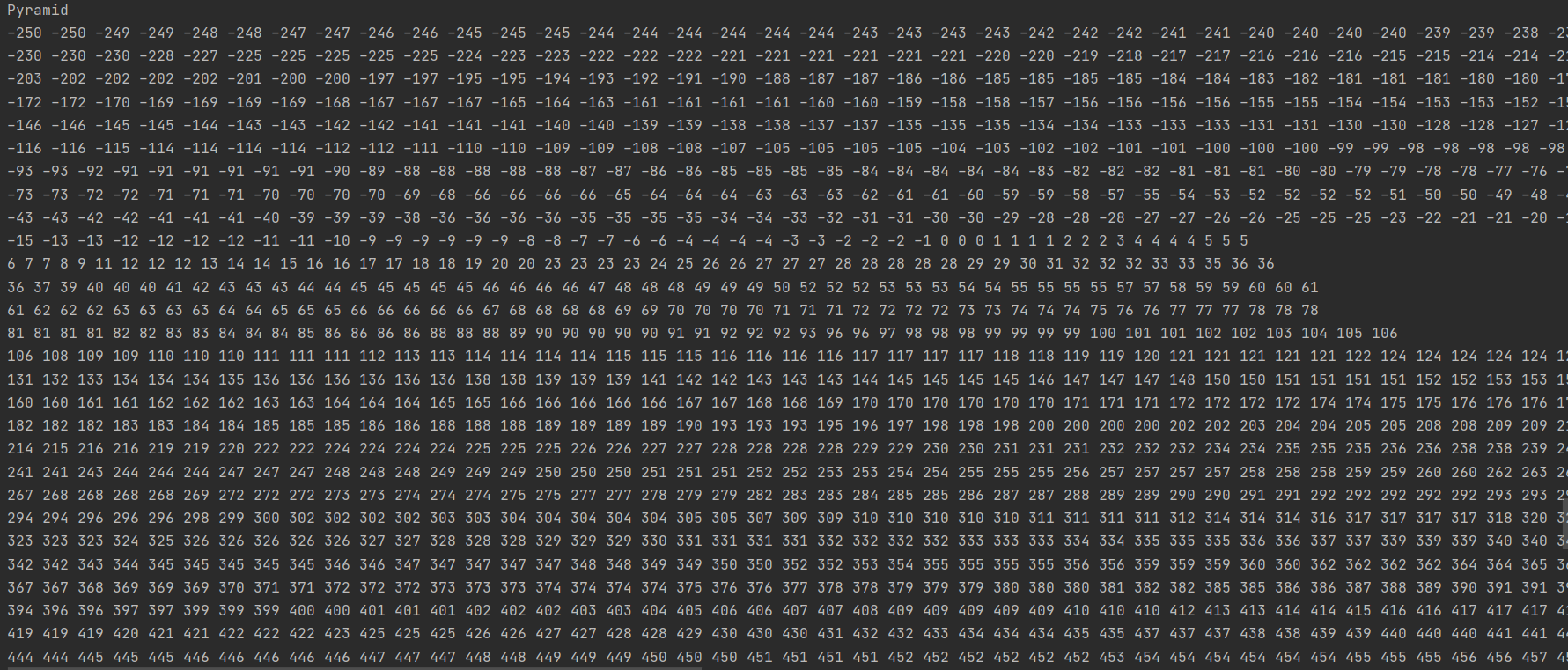


Рисунок 7 – результат выполнения

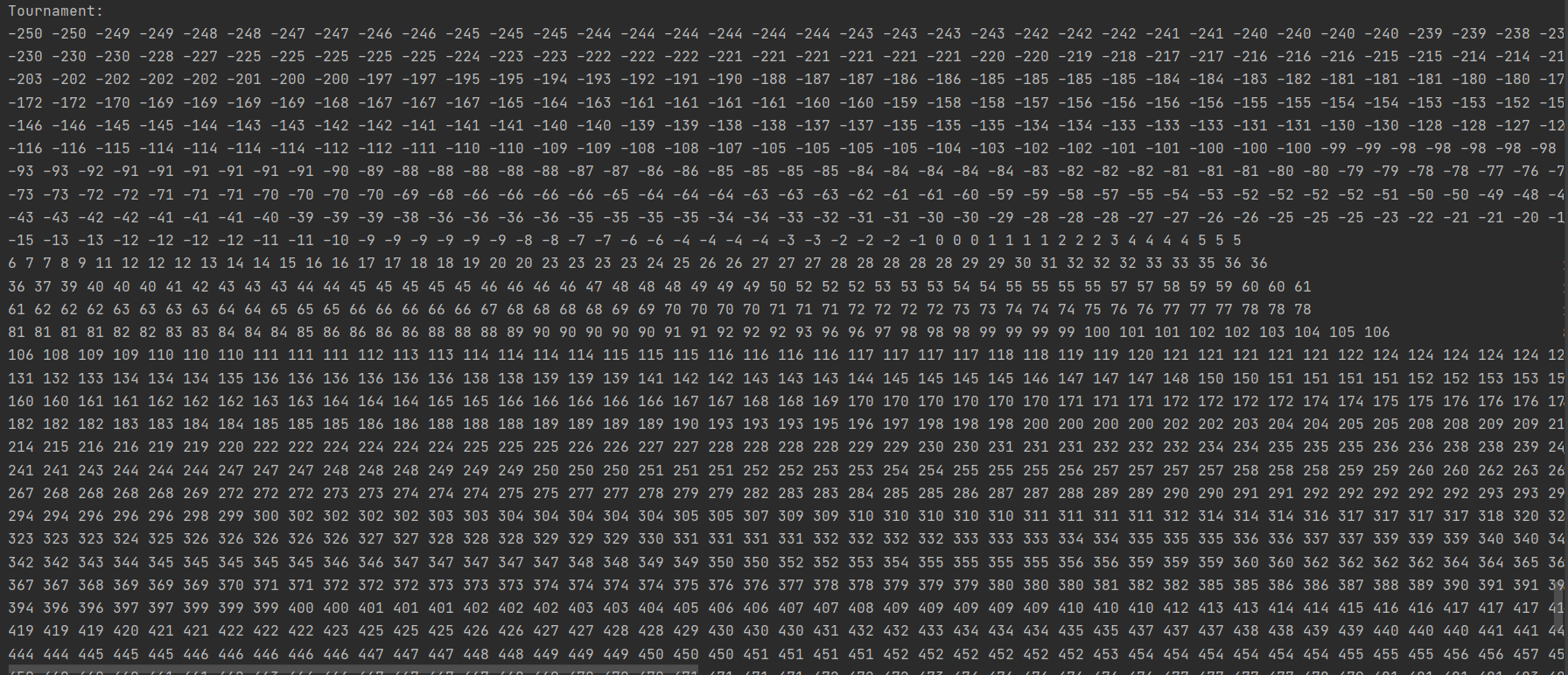


Рисунок 8 – результат выполнения

# 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