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

Цель данной лабораторной работы — научиться пользоваться сортировками.

2. Задание на лабораторную работу

Задание №1:

1. Создать Jupyter Notebook со следующим наименованием: Lab1_Группа_ФИО
2. Создать новую ячейку с помощью кнопки 
3. В созданной ячейке по указанной ниже форме заполните оглавление файла, заменив наименование группы и вписав свое ФИО,

```
# Лабораторная работа №1
## Выполнил студент(ка) группы ГРУППА ФИО
### Задание №1
Описание условия задания
```

Рисунок 1 - Форма оглавления файла в ячейке

после чего создайте еще одну ячейку и напишите следующий код:

```
print("Hello, World!")
```

4. С помощью кнопки 

запустите выполнение всех ячеек.
5. После выполнения у вас должна отформатироваться ячейка с оглавлением и должен выполниться "Hello, World!" (см. рис. 2).

Задание №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_lim** и **max_lim** - 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)

Задание №3:

Реализовать методы сортировки строк числовой матрицы в соответствии с заданием. Оценить время работы каждого алгоритма сортировки и сравнить его со временем стандартной функции сортировки. Испытания проводить на сгенерированных матрицах.

Методы:

Выбором	Вставкой	Обменом	Шелла	Турнирная	Быстрая сортировка	Пирамидальная

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++)//ґ??????Ĺ???
        {
            data[i] = nodes[0].iData;//d?????C??
            nodes[nodes[0].idd].iData = Integer.MAX_VALUE;
            Adjust(nodes, nodes[0].idd);

        }



    }
}
```

3.2 Результат выполнения программы

```
Hello,World!
default matrix
332 660 335 2 72 248 669 332 941 393 767 -54 342 477 912 745 416 441 591 132 649 585 44 -34 114 988 370 394 532 235 709 670 124 -239 -87 -133 200 880 -222 372 391 902 46 541 -35
812 977 -8 -36 -221 989 764 12 966 430 863 454 573 -35 991 151 148 -215 741 636 488 838 444 850 401 -61 407 -78 36 447 317 831 167 -142 -83 415 445 714 731 102 837 626 62 -100 60
598 684 422 432 884 -91 505 265 200 871 -134 311 136 779 692 558 875 693 841 634 -2 402 135 948 894 -15 563 500 231 701 402 176 -131 -172 700 920 476 665 -18 1014 663 229 6 84 4
-96 1 -202 -110 421 557 124 985 230 642 40 827 -169 492 -3 711 573 -231 323 643 407 880 -202 967 258 515 112 -98 -64 -232 232 228 367 2 -197 940 767 -194 293 254 224 886 533 279
-64 -188 298 -79 435 786 994 902 344 935 410 -112 906 648 66 652 681 543 494 23 304 891 -88 210 575 507 1003 675 719 -230 -182 -173 -39 826 543 762 962 936 -236 653 547 121 -186
960 -184 227 469 -47 189 509 418 -85 454 249 512 160 -185 147 727 20 -4 309 165 878 -98 -150 988 861 36 -156 92 166 698 -72 929 82 552 116 326 656 699 374 333 709 244 -243 -17 -1
695 379 -7 441 96 768 18 115 626 -55 491 -116 470 514 -191 761 386 -2 644 1011 958 -70 871 -105 481 247 355 974 787 758 -151 447 591 -43 655 636 732 -114 290 520 978 720 -81 373
897 -71 116 303 179 549 -109 495 -236 293 188 348 -238 551 638 40 788 789 342 380 277 5 832 305 603 88 733 423 283 61 -28 943 256 39 787 9 43 394 377 409 -248 152 872 -202 343 82
-239 843 1 265 15 -172 -131 259 904 117 750 646 997 925 103 895 349 284 630 846 548 45 215 549 326 111 365 353 476 577 589 675 450 535 716 902 -174 -98 186 739 -181 -225 734 -222
835 409 752 27 547 729 -247 318 1006 409 542 966 114 84 647 17 -23 -237 -142 978 485 693 910 519 204 72 770 143 399 -210 -50 49 -168 228 64 802 -32 33 472 248 -105 555 184 73 582
209 227 437 878 345 393 310 958 257 292 824 728 826 -216 214 -161 -82 940 696 331 624 -180 648 189 725 574 356 412 -158 458 496 487 208 118 -12 -95 342 138 425 364 826 139 599 58
54 -84 -207 -94 -49 379 -246 402 -10 732 918 198 419 511 -19 872 479 942 28 428 573 508 756 471 124 70 833 198 458 697 1005 258 600 362 136 998 265 961 527 -16 -206 719 184 768 -
532 679 367 92 -42 -236 350 536 455 332 816 918 839 366 333 -213 255 909 70 477 181 939 232 310 240 712 53 569 -152 835 251 902 268 720 800 726 994 633 808 650 527 162 649 189 29
872 230 -238 29 125 572 495 197 304 172 18 347 908 76 662 231 32 -244 289 145 497 956 916 294 538 603 915 770 -155 859 25 210 614 -200 661 -128 212 848 594 68 52 615 419 -174 856
-91 690 339 -96 401 705 831 244 857 846 449 -128 396 -195 55 464 97 1000 712 -4 369 -50 -225 355 629 -84 831 614 16 897 835 -71 665 933 902 314 571 272 663 3 727 593 964 310 780
471 484 113 463 265 709 664 836 -134 843 714 151 328 324 614 990 434 977 -70 410 -170 597 894 845 170 520 952 33 -76 911 983 162 205 150 -109 693 863 496 963 840 -108 433 234 -21
967 171 968 -98 364 766 897 816 -156 422 573 138 529 182 -249 719 449 -221 539 -146 169 53 527 703 653 862 64 608 4 -169 -165 744 -177 517 675 721 -217 581 -88 219 476 890 -122 8
547 582 809 827 512 -148 -25 139 440 897 409 236 430 996 323 45 -86 -102 953 825 898 -164 29 340 451 418 719 -79 409 769 170 483 -185 460 119 824 729 292 59 -6 78 919 -139 -175 9
335 646 -82 235 467 -11 750 36 845 88 448 527 293 316 766 552 152 331 -63 -145 966 579 928 392 110 -127 106 -94 838 131 63 337 590 216 293 400 504 447 -135 -200 744 -88 -84 762 8
955 -244 666 -221 -130 99 254 346 282 814 457 170 134 926 921 932 -46 791 539 372 109 -183 325 826 658 863 54 733 -80 967 166 63 711 229 134 881 309 405 -31 973 592 594 827 889 8
314 504 -214 101 550 568 69 -114 250 -141 106 81 -197 755 843 274 -99 321 49 91 529 100 -223 -74 -30 648 -88 914 262 667 923 468 680 425 205 834 809 -219 669 965 467 234 820 -118
990 566 -186 185 919 -116 767 515 183 329 397 899 1004 866 631 697 350 410 418 81 -240 852 70 168 185 438 -158 988 802 944 731 890 164 278 291 960 -13 487 635 943 777 573 944 988
425 -99 897 23 -68 -73 916 -150 -205 771 849 565 595 257 66 815 859 504 4 843 492 37 -107 251 609 993 880 273 687 417 -52 296 681 787 385 445 172 158 640 -40 382 -59 -161 715 0 -
224 871 121 984 471 574 285 -85 652 -9 28 431 445 577 1012 77 647 -137 741 620 252 999 143 1012 848 174 881 735 -205 808 376 -66 808 803 180 827 999 167 665 634 511 317 20 823 64
-210 503 -65 345 146 638 979 327 800 359 277 145 159 471 225 1012 730 -202 539 302 183 -12 -4 -91 -172 815 61 275 257 241 1002 555 -52 328 550 444 811 86 386 795 572 170 365 588
```

Рисунок 1 – результат выполнения

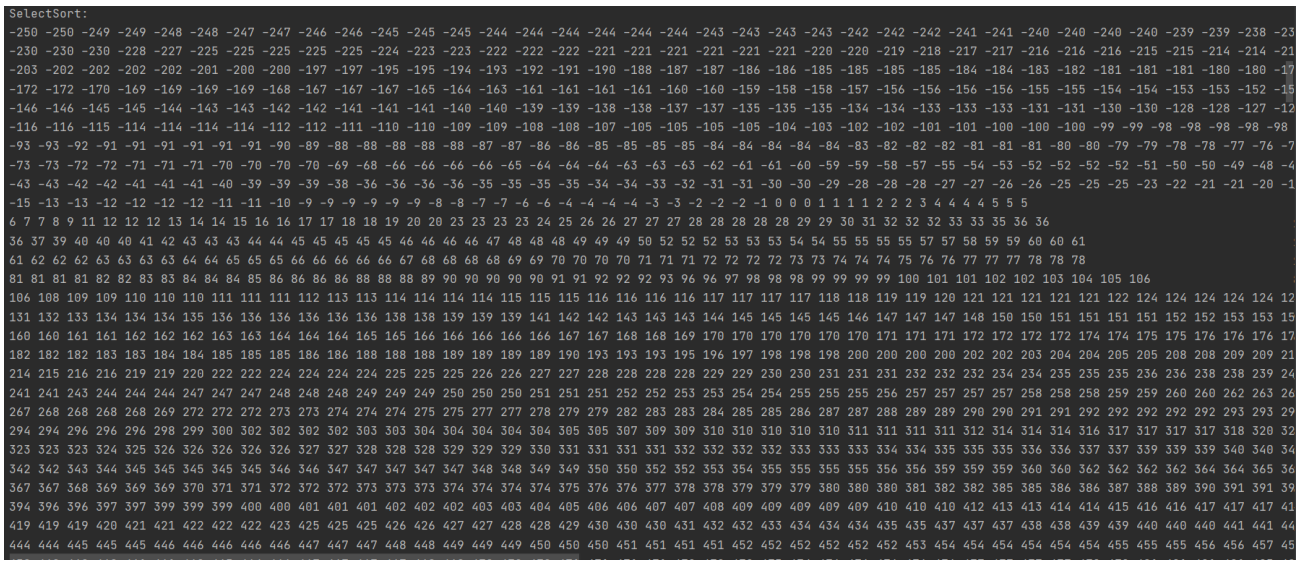

Рисунок 2 – результат выполнения

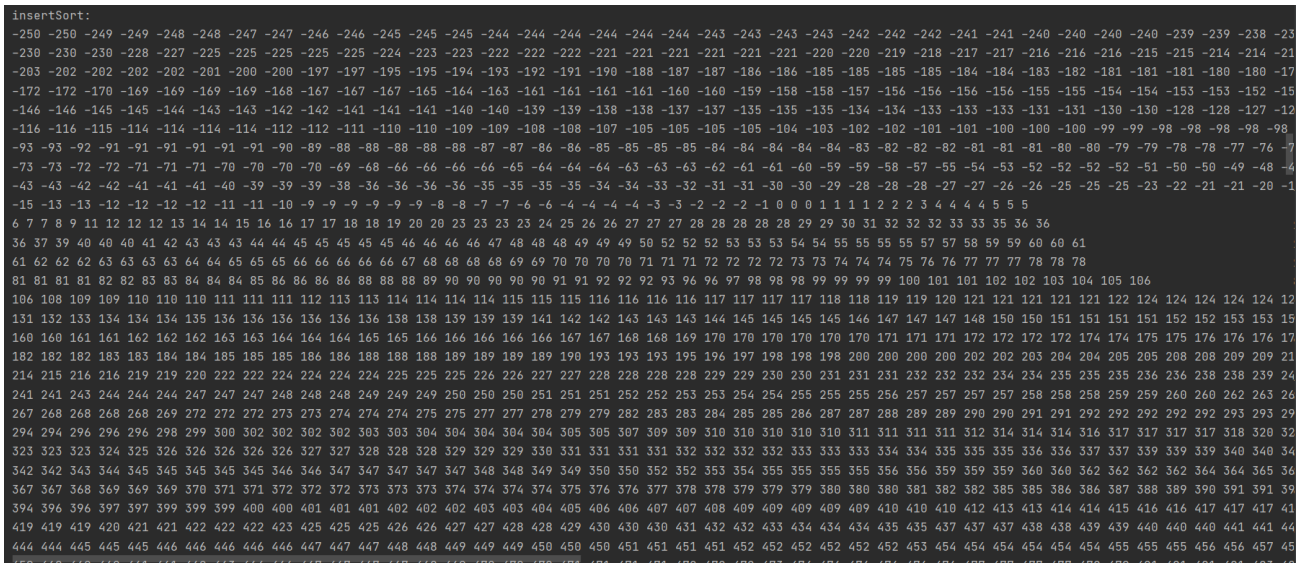

Рисунок 3 – результат выполнения

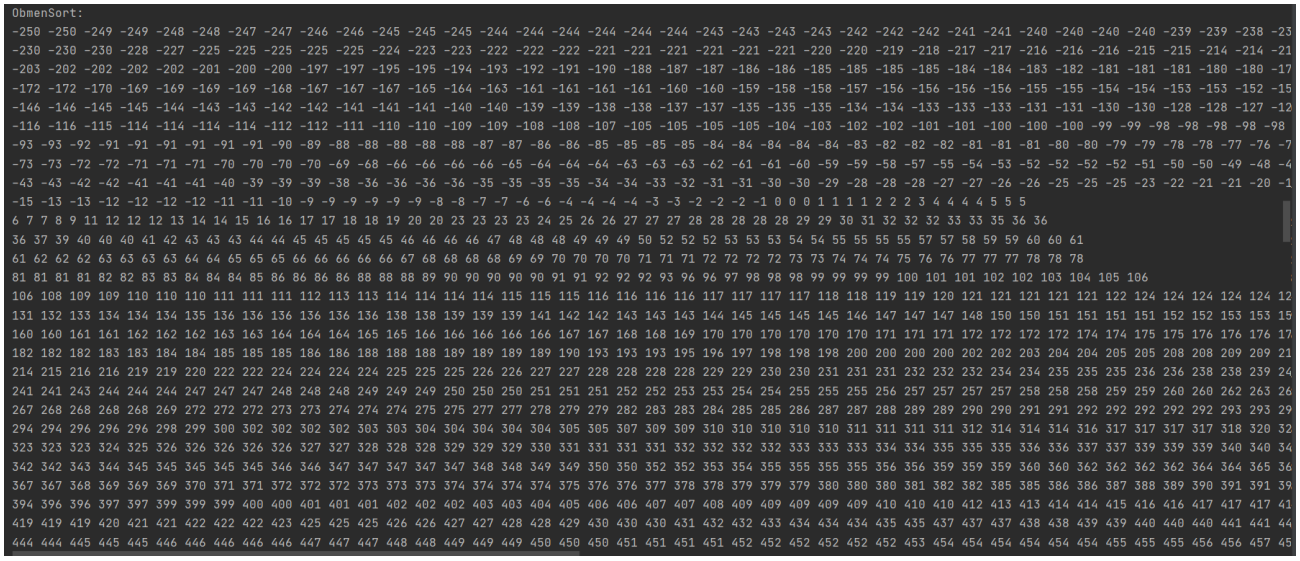

Рисунок 4 – результат выполнения

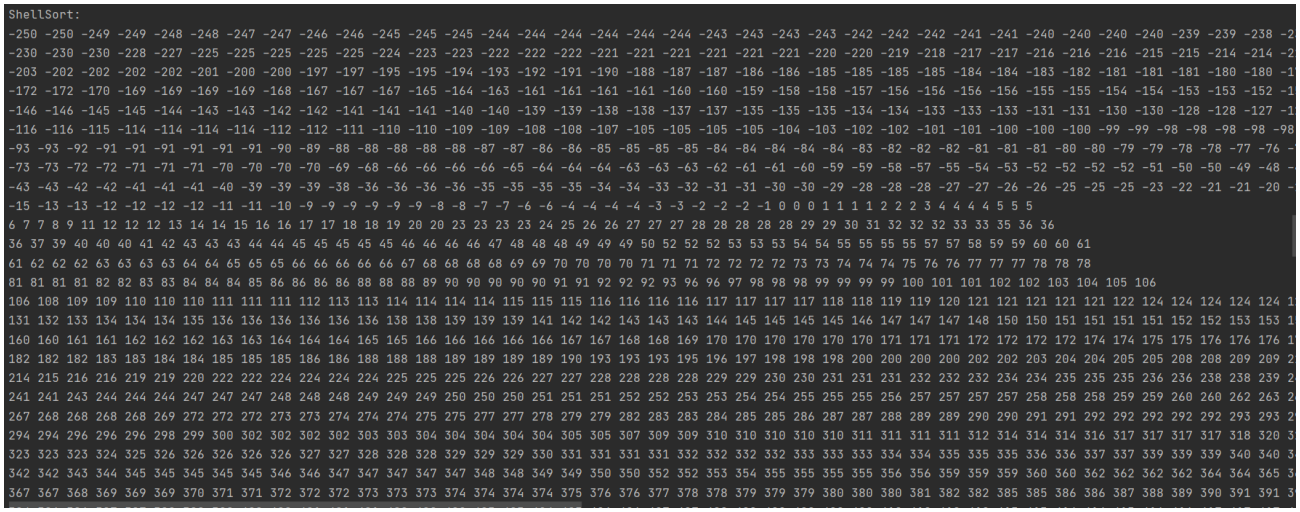

Рисунок 5 – результат выполнения

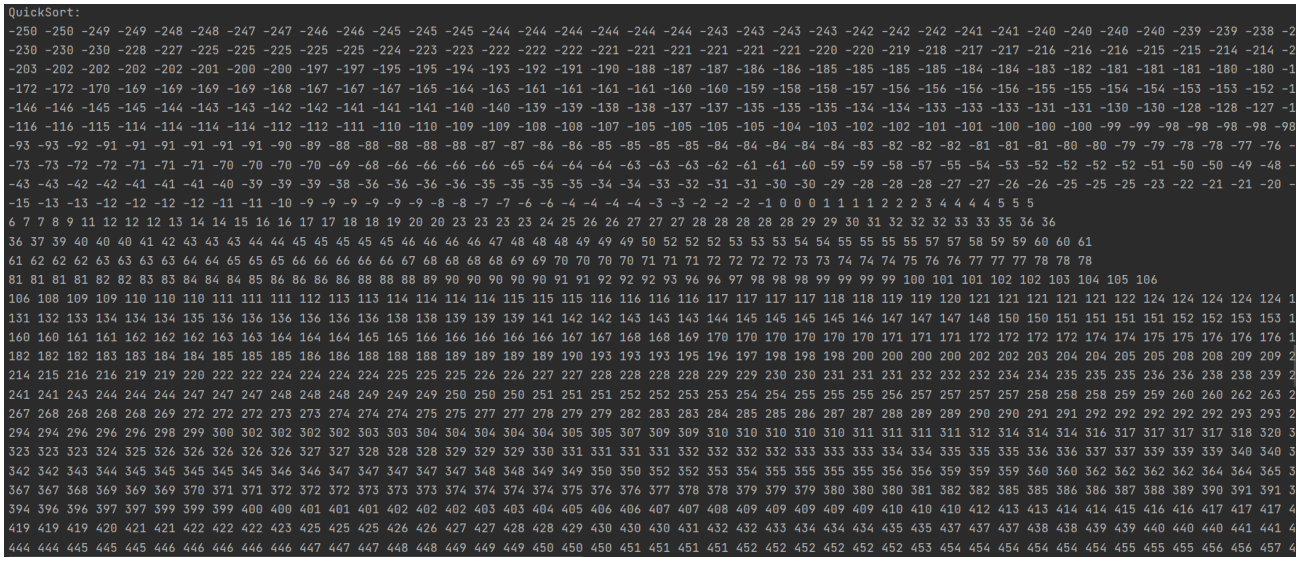

Рисунок 6 – результат выполнения

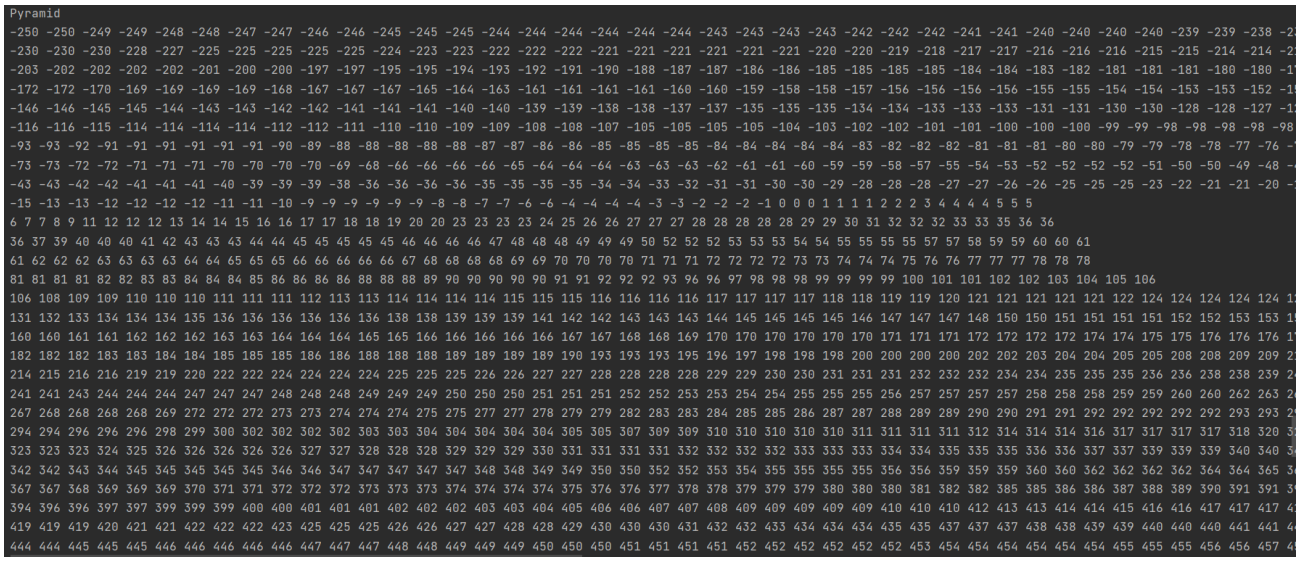

Рисунок 7 – результат выполнения

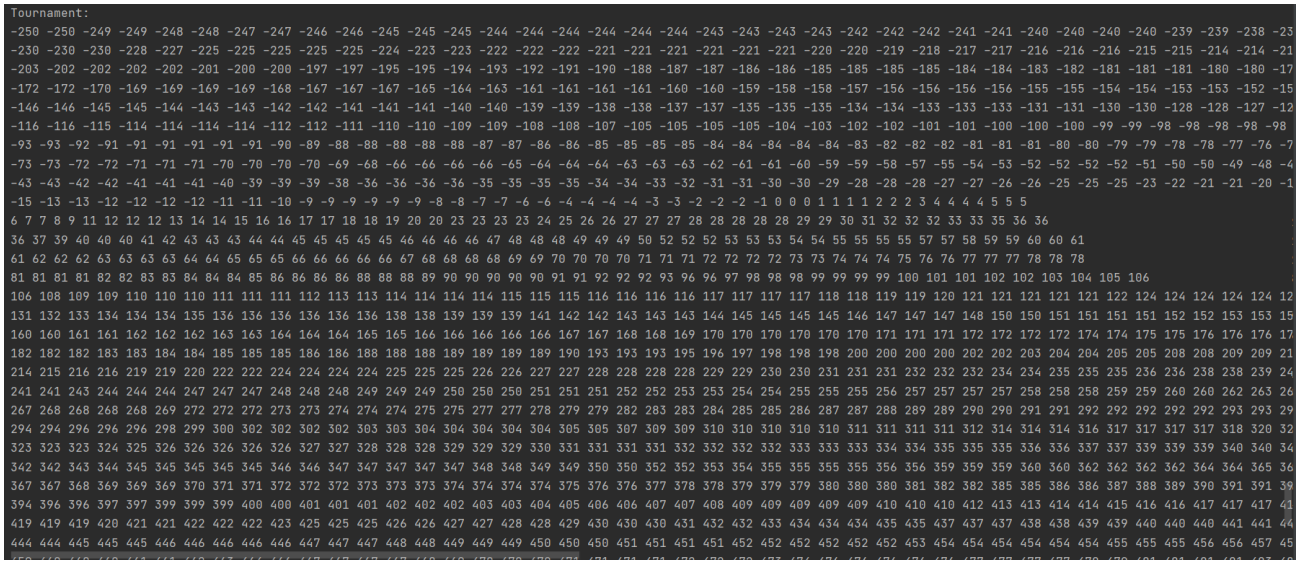

Рисунок 8 – результат выполнения

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