Министерство цифрового развития, связи и массовых коммуникаций Государственное образовательного учреждение высшего образования

Ордена Трудового Красного Знамени

«Московский технический университет связи и информатики»

Лабораторная работа № 1

«Методы сортировки»

Выполнил студент

группы БВТ1902

Шульпина Полина

Москва

2021

# Задание

Реализовать методы сортировок строк числовой матрицы. Оценить время работы каждого алгоритма сортировки.

Сортировки:

* Выбором
* Вставкой
* Обменом
* Шелла
* Быстрая
* Пирамидальная

# Выполнение

Листинг программы:

Класс «Main»

import java.util.Scanner;  
  
public class Main  
{  
 public static void main(String[] args)  
 {  
 Scanner in = new Scanner(System.*in*);  
  
 int[][] matrix;  
 int[][] sorted\_matrix;  
 int[] qsort\_line;  
  
 long start\_time;  
  
 String ans = "";  
 boolean correct = false;  
  
 int n = 50;  
 int m = 50;  
 int min\_limit = -50; //минимальный лимит -1, т.к. иначе он не видит нижнюю границу  
 int max\_limit = 50;  
  
  
 Sorts sorts = new Sorts();  
  
 System.*out*.println("Будете менять значения по умолчанию (+/-)");  
 while (!correct)  
 {  
 ans = in.nextLine();  
 if (!ans.equals("+") && !ans.equals("-"))  
 {  
 System.*out*.println("допустимые символы- + и -, дайте корректный ответ");  
 } else  
 {  
 correct = true;  
 }  
 }  
 if (ans.equals("+"))  
 {  
 System.*out*.println("Введите кол-во столбцов матрицы");  
 m = in.nextInt();  
 System.*out*.println("Введите кол-во строк матрицы");  
 n = in.nextInt();  
 System.*out*.println("Введите минимальное возможное значение");  
 min\_limit = in.nextInt();  
 min\_limit--;  
 System.*out*.println("Введите максимальное возможное значение");  
 max\_limit = in.nextInt();  
 }  
  
 qsort\_line = new int[n];  
 sorted\_matrix = new int[m][n];  
  
 do  
 {  
 matrix = Generator.*matrix*(m, n, min\_limit, max\_limit);  
  
 System.*out*.println("Выберите сортировку (1-6), для выхода, напишите 0");  
 ans = in.nextLine();  
  
  
 switch (ans)  
 {  
 case "1":  
 sorted\_matrix = sorts.selection\_sort(matrix, n, m);  
 break;  
  
 case "2":  
 sorted\_matrix = sorts.insertion\_sort(matrix, n, m);  
 break;  
  
 case "3":  
 sorted\_matrix = sorts.bubble\_sort(matrix, n, m);  
 break;  
  
 case "4":  
 sorted\_matrix = sorts.shell\_sort(matrix, n, m);  
 break;  
  
 case "5":  
 start\_time = System.*currentTimeMillis*();  
 for (int i = 0; i < n; i++)  
 {  
 for (int j = 0; j < m; j++)  
 {  
 qsort\_line[j] = matrix[i][j];  
 }  
 sorts.quick\_sort(qsort\_line, 0, m - 1);  
  
 System.*arraycopy*(qsort\_line, 0, sorted\_matrix[i], 0, qsort\_line.length);  
 }  
 System.*out*.println("quick sort: " + (System.*currentTimeMillis*() - start\_time));  
 break;  
  
 case "6":  
 start\_time = System.*currentTimeMillis*();  
 for (int i = 0; i < n; i++)  
 {  
 for (int j = 0; j < m; j++)  
 {  
 qsort\_line[j] = matrix[i][j];  
 }  
 sorts.heap\_sort(qsort\_line);  
  
 System.*arraycopy*(qsort\_line, 0, sorted\_matrix[i], 0, qsort\_line.length);  
 }  
 System.*out*.println("heap sort: " + (System.*currentTimeMillis*() - start\_time));  
 break;  
 }  
 if (!ans.equals("0"))  
 {  
 System.*out*.println("Вывести результат?(+/-)");  
 ans = in.nextLine();  
  
 if (ans.equals("+"))  
 {  
 for (int i = 0; i < m; i++)  
 {  
 for (int j = 0; j < n; j++)  
 {  
 System.*out*.print(sorted\_matrix[i][j] + " ");  
 }  
 System.*out*.println();  
 }  
 }  
 }  
 } while (!ans.equals("0"));  
 }  
}

Класс «Generator»

import java.util.Random;  
  
public class Generator  
{  
 public static int[][] matrix(int m, int n, int min, int max)  
 {  
 int[][] matrix= new int[m][n];  
 double rand=0;  
 Random random=new Random();  
  
 for (int i=0; i<n; i++)  
 {  
 for (int j=0; j<m; j++)  
 {  
 rand=Math.*random*()\*((max-min)+1)+min;  
 matrix[i][j]=(int) rand;  
 }  
 }  
 return matrix;  
 }  
}

Класс «Sorts»

public class Sorts  
{  
 private long start\_time;  
 private int temp;  
  
 public int[][] selection\_sort(int[][] matrix, int n, int m)  
 {  
 int min;  
  
 this.start\_time = System.*currentTimeMillis*();  
  
 for (int i = 0; i < n; i++)  
 {  
 for (int j = 0; j < m - 1; j++)  
 {  
 min = j;  
 for (int k = min + 1; k < m; k++)  
 {  
 if (matrix[i][k] < matrix[i][min])  
 {  
 min = k;  
 }  
 }  
 temp = matrix[i][min];  
 matrix[i][min] = matrix[i][j];  
 matrix[i][j] = temp;  
 }  
 }  
  
 System.*out*.println("Selection sort: " + (System.*currentTimeMillis*() - start\_time));  
  
 return matrix;  
 }  
  
 public int[][] insertion\_sort(int[][] matrix, int n, int m)  
 {  
 this.start\_time = System.*currentTimeMillis*();  
 int current;  
 int j;  
  
 for (int k = 0; k < n; k++)  
 {  
 for (int i = 1; i < m - 1; i++)  
 {  
 current = matrix[k][i];  
 j = i - 1;  
 while (j >= 0 && current < matrix[k][j])  
 {  
 matrix[k][j + 1] = matrix[k][j];  
 j--;  
 }  
 matrix[k][j + 1] = current;  
 }  
 }  
  
 System.*out*.println("Insertion sort: " + (System.*currentTimeMillis*() - start\_time));  
 return matrix;  
 }  
  
 public int[][] bubble\_sort(int[][] matrix, int n, int m)  
 {  
 boolean isSorted = false;  
  
 this.start\_time = System.*currentTimeMillis*();  
 for (int k = 0; k < n; k++)  
 {  
 for (int i = 0; i < n - 1; i++)  
 {  
 for (int j = i + 1; j < n; j++)  
 {  
 if (matrix[k][i] > matrix[k][j])  
 {  
 temp = matrix[k][i];  
 matrix[k][i] = matrix[k][j];  
 matrix[k][j] = temp;  
 }  
 }  
 }  
 }  
 System.*out*.println("bubble sort: " + (System.*currentTimeMillis*() - start\_time));  
  
 return matrix;  
 }  
  
 public int[][] shell\_sort(int[][] matrix, int n, int m)  
 {  
 this.start\_time = System.*currentTimeMillis*();  
  
  
 for (int k = 0; k < n; k++)  
 {  
 for (int step = m / 2; step > 0; step /= 2)  
 {  
 for (int i = step; i < m; i++)  
 {  
 for (int j = i - step; j >= 0 && matrix[k][j] > matrix[k][j + step]; j -= step)  
 {  
 temp = matrix[k][j];  
 matrix[k][j] = matrix[k][j + step];  
 matrix[k][j + step] = temp;  
 }  
 }  
 }  
 }  
 System.*out*.println("shell sort: " + (System.*currentTimeMillis*() - start\_time));  
  
 return matrix;  
 }  
  
 public void quick\_sort(int[] mas, int low, int high)  
 {  
 int i = low;  
 int j = high;  
 int x = mas[low + (high - low) / 2];  
 do  
 {  
 while (mas[i] < x) i++;  
 while (mas[j] > x) j--;  
 if (i <= j)  
 {  
 temp = mas[i];  
 mas[i] = mas[j];  
 mas[j] = temp;  
 i++;  
 j--;  
 }  
 } while (i<=j);  
  
 if (low<j) quick\_sort(mas, low,j);  
 if (i<high) quick\_sort(mas,i,high);  
 }  
  
 public void heap\_sort(int[] arr)  
 {  
 int n = arr.length;  
  
 for (int i = n / 2 - 1; i >= 0; i--)  
 heapify(arr, n, i);  
  
 for (int i = n - 1; i >= 0; i--)  
 {  
 int temp = arr[0];  
 arr[0] = arr[i];  
 arr[i] = temp;  
  
 heapify(arr, i, 0);  
 }  
 }  
  
 void heapify(int[] arr, int n, int i)  
 {  
 int largest = i;  
 int l = 2 \* i + 1;  
 int r = 2 \* i + 2;  
  
 if (l < n && arr[l] > arr[largest])  
 largest = l;  
  
 if (r < n && arr[r] > arr[largest])  
 largest = r;  
  
 if (largest != i)  
 {  
 int swap = arr[i];  
 arr[i] = arr[largest];  
 arr[largest] = swap;  
   
 heapify(arr, n, largest);  
 }  
 }  
}

Снимки экрана выполнения программы



Рис. 1 Исходная матрица

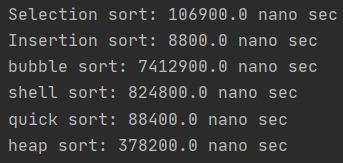


Рис.2 Реализация сортировок выбором, вставками, обменом, Шелла, быстрой и пирамидальной

# Вывод

В ходе выполнения лабораторной работы, я реализовала сортировки выбором, вставками, обменом, Шелла, быстрой и пирамидальной.

Сортировка выбором разделяет массив на сортированный и неотсортированный, минимальный эл-т неотсортированного подмассива вставляется в конец сортированного, меняя его местами с первым неотсортированным эл-том.

Сортировка вставками постепенно перебирает массив слева направо, при этом каждый последующий эл-т размещается так, чтобы он оказался между ближайшими эл-тами с минимальным и максимальным значениями.

Сортировка обменом просматривает массив и сравнивает каждую пару соседних эл-тов, заменяя пары, расположенные не по порядку.

Сортировка Шелла похожа на сортировку обменом, однако на каждую итерацию мы имеем разный промежуток между сравниваемыми эл-тами, каждую итерацию он уменьшается вдвое.

Быстрая сортировка выбирает эл-т в массиве в качестве опорного, все меньшие эл-ты помещаем слева, большие- справа. Затем рекурсивно повторяем для левой и правой частей.

Пирамидальная сортировка реализуется через пирамидальное бинарное дерево и сортируем его.