Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

**Лабораторна робота № 3**

по дисципліні «СТП-1»

|  |  |
| --- | --- |
| Виконав:  студент групи ІТ-51  Давиденко Ігор Володимирович  Дата здачі \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Захищено з балом \_\_\_\_\_\_\_\_\_\_\_ | Перевірено:  ac. Федорчук В. В |

Київ 2017

Варіант 2

Завдання

* Create an array of numbers that consists of 150 elements. The contents of the array generate with Random#nextInt.
* Implement five types of data sorts.
* Use two libraries to sort the data.
* Three times perform each of the sorts, including sorts with libraries, with the working time measurement of each of them.
* Gather the performance (resources and time) metrics of your sorting and libraries sorting.
* The time for working of the each sort is output to the console.

**Keep in mind:** Generate an array once, and all sorting algorithms, including libraries for sorting, should use this array.

| **variant** | **task** |
| --- | --- |
| 2 | bubble sort, selection sort, quick sort, odd-even sort, merge sort; |

Короткі теоретичні відомості

Швидке сортування (англ. Quick Sort)

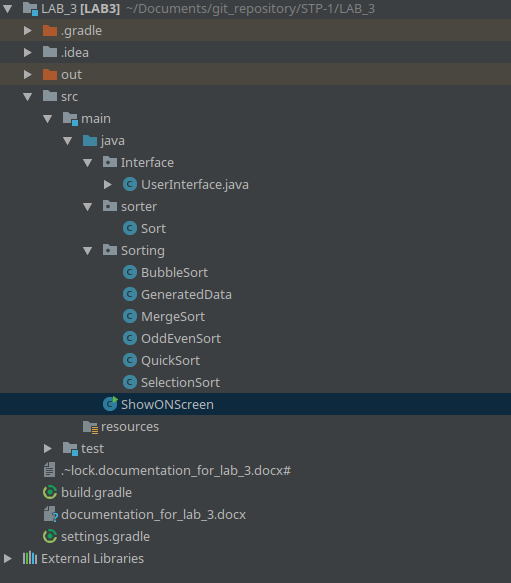
Ідея алгоритму полягає в переставлянні елементів масиву таким чином, щоб його можна було розділити на дві частини і кожний елемент з першої частини був не більший за будь-який елемент з другої. Впорядкування кожної з частин відбувається рекурсивно. Алгоритм швидкого сортування може бути реалізований як у масиві, так і в двозв'язному списку.

Odd-even sort чи Odd-even transposition sort

В інформатиці, парне-непарне сортування (також відоме як сортування цеглинами) є відносно простим алгоритмом сортування, розробленим спочатку для використання на паралельних процесорів з локальними взаємозв'язків. Воно порівнюється з сортуванням бульбашкою, з яким він поділяє багато характеристики. Алгоритм діє наступним чином: порівнюються всі парні / непарні пари проіндексованих суміжних елементів в списку, і якщо пара знаходиться в неправильному порядку (перший більше, ніж другий) елементи міняються місцями. Наступним кроком повторює це для парних / непарних індексованих пар (суміжних елементів). Чергуються парні/непарні та непарні/парні кроки, поки список не буде відсортований.

Xід роботи

1. Створю проект “LAB\_3”, котрий має наступну структуру:



2. Код програми:

import Interface.UserInterface;

public class ShowONScreen {

// main method

public static void main(String []args){

UserInterface.start();

}

}

package Interface;

import Sorting.\*;

import sorter.Sort;

import java.util.Arrays;

import java.util.Scanner;

/\*\*

\* console commands

\*/

enum UICommands{

alltimes,

exit,

help,

memory,

author,

showarray,

bubblesort,

mergesort,

oddevensort,

selectionsort,

quicksort,

arraysort,

insertsort

}

/\*\*

\* it is class who realize work with console

\*/

public class UserInterface {

private static String command = new String(); //value for selecting data

private static GeneratedData array = null;

private static int sizeArray = 0;

private static Scanner scanner = new Scanner(System.in); // get data from keyboard

private static long memoryBegin = Runtime.getRuntime().freeMemory();

/\*\*

\* start the program

\*/

public static void start(){

setSizeArray(scanner);

setCommand(scanner);

}

/\*\*

\* get command from keyboard

\* @param inputData

\*/

public static void setCommand(Scanner inputData){ //

do{

try{

System.out.println("Введите команду: ");

command = inputData.next();

break;

}catch (Exception e){

inputData = new Scanner(System.in);

System.out.println("Даные введение не верно!!!!");

continue;

}

}while (true);

commandProcessing();

}

/\*\*

\* set size array and then generate them

\* @param inputCommand

\*/

public static void setSizeArray(Scanner inputCommand){

do{

try{

System.out.println("Введите размер масива: ");

sizeArray = inputCommand.nextInt();

break;

}catch (Exception e){

inputCommand = new Scanner(System.in);

System.out.println("Даные введение не верно!!!!");

continue;

}

}while (true);

array = new GeneratedData(sizeArray); // generate array

}

/\*\*

\* show on screen all commands

\*/

public static void showCommands(){

System.out.println("Все команди: ");

for (UICommands comand: UICommands.values()) {

System.out.println(comand.toString());

}

}

/\*\*

\* method who processing commands typed in from keyboard

\*/

public static void commandProcessing(){

long start = 0, end = 0;

long memoryAfter = 0;

UICommands sorting = null;

if(array != null){

try {

sorting = UICommands.valueOf(command);

}catch (Exception e){

System.out.println("Не верно введена команда");

setCommand(scanner);

}

switch (sorting){

case bubblesort:

array.resetData();

start = System.nanoTime();

BubbleSort.sort(array.getArray());

end = System.nanoTime();

System.out.printf("Bubble sort time : %s \n\r", changeTimeFormat(end-start));

break;

case arraysort:

array.resetData();

start = System.nanoTime();

Arrays.sort(array.getArray());

end = System.nanoTime();

System.out.printf("Arrays default sorting time : %s \n\r", changeTimeFormat(end - start));

break;

case mergesort:

array.resetData();

start = System.nanoTime();

MergeSort.sort(array.getArray());

end = System.nanoTime();

System.out.printf("Merge sort time : %s \n\r", changeTimeFormat(end - start));

break;

case quicksort:

array.resetData();

start = System.nanoTime();

MergeSort.sort(array.getArray());

end = System.nanoTime();

System.out.printf("Quick sort time : %s \n\r", changeTimeFormat(end - start));

case insertsort:

array.resetData();

start = System.nanoTime();

Sort.insertion(array.getArray());

end = System.nanoTime();

System.out.printf("Insert sort time : %s \n\r", changeTimeFormat(end - start));

break;

case oddevensort:

array.resetData();

start = System.nanoTime();

OddEvenSort.sort(array.getArray());

end = System.nanoTime();

System.out.printf("odd even sort time: %s \n\r", changeTimeFormat(end - start));

break;

case selectionsort:

array.resetData();

start = System.nanoTime();

SelectionSort.sort(array.getArray());

end = System.nanoTime();

System.out.printf("Selection sort time : %s \n\r", changeTimeFormat(end -start));

break;

case alltimes:

allSortingTimes();

break;

case memory:

memoryAfter = Runtime.getRuntime().freeMemory();

System.out.printf("Memory : %s \n\r", changeTimeFormat(memoryAfter - memoryBegin));

break;

case help:

showCommands();

break;

case exit:

return;

case showarray:

array.showArray();

break;

case author:

System.out.printf("Davidenko Igor \n\r");

return;

default:

break;

}

setCommand(scanner);

}else {

System.out.print("Масив не задан!!!!");

}

}

/\*\*

\* show all sorting

\*/

public static void allSortingTimes(){

long start = 0, end = 0;

array.resetData();

start = System.nanoTime();

BubbleSort.sort(array.getArray());

end = System.nanoTime();

System.out.printf("Bubble sort time : %s \n\r",changeTimeFormat(end-start));

array.resetData();

start = System.nanoTime();

Arrays.sort(array.getArray());

end = System.nanoTime();

System.out.printf("Arrays default sorting time : %s \n\r", changeTimeFormat(end - start));

array.resetData();

start = System.nanoTime();

MergeSort.sort(array.getArray());

end = System.nanoTime();

System.out.printf("Merge sort time : %s \n\r", changeTimeFormat(end - start));

array.resetData();

start = System.nanoTime();

MergeSort.sort(array.getArray());

end = System.nanoTime();

System.out.printf("Quick sort time : %s \n\r", changeTimeFormat(end - start));

array.resetData();

start = System.nanoTime();

Sort.insertion(array.getArray());

end = System.nanoTime();

System.out.printf("Insert sort time : %s \n\r", changeTimeFormat(end - start));

array.resetData();

start = System.nanoTime();

OddEvenSort.sort(array.getArray());

end = System.nanoTime();

System.out.printf("odd even sort time: %s \n\r", changeTimeFormat(end - start));

array.resetData();

start = System.nanoTime();

SelectionSort.sort(array.getArray());

end = System.nanoTime();

System.out.printf("Selection sort time : %s \n\r", changeTimeFormat(end -start));

}

/\*\*

\* method who represent numbers in next format : "1 000 000"

\* @param time

\* @return

\*/

public static String changeTimeFormat(long time){

char [] textArray = Long.toString(time).toCharArray();

int iter = 0;

String finalyTime = new String();

for (int i = textArray.length - 1; i >= 0; i--) {

finalyTime+= textArray[iter];

if (i%3 == 0)

finalyTime+=" ";

iter++;

}

return finalyTime;

}

}

package Sorting;

import java.util.Arrays;

import java.util.Random;

/\*\*

\* it is class who generate array

\*/

public class GeneratedData {

private int arraySize;

private int []arrayElements = null; // array elements

private Random generatorElements = new Random(); // create object

private int [] arrayCopy = null;

/\*\*

\*

\* @return size array

\*/

public int getSize() { // get size array

return arraySize;

}

/\*\*

\*

\* @return array

\*/

public int[] getArray() { return arrayElements; }

/\*\*

\* initialize array

\* class constructor

\* @param size array

\*/

public GeneratedData(int size){

this.arraySize = size;

arrayElements = new int[size];

generateArray(); // after create array, generate him

}

/\*\*

\* populating an array with random data

\*/

public void generateArray(){ // method whose addition data to our array

int generatedNumber = 0;

for (int i = 0; i < this.arraySize; i++){

generatedNumber = generatorElements.nextInt();

arrayElements[i] = generatedNumber;

}

arrayCopy = arrayElements.clone();

}

/\*\*

\* method who return all changes to default values

\*/

public void resetData(){ // repair default data

arrayElements = arrayCopy;

}

/\*\*

\* method who return element array using her index

\* @param position - position element

\* @return retun element using her position

\*/

public int getElement(int position){

return arrayElements[position];

}

/\*\*

\* set data for element in array

\* @param position element

\* @param newData new value array element

\*/

public void setElement(int position, int newData){

arrayElements[position] = newData;

}

/\*\*

\* method who print the value to the console

\*/

public void showArray(){ // method whose showing array data

System.out.printf("Array: %s \n", Arrays.toString(arrayElements));

}

}

package Sorting;

/\*\*

\* it is class who realize bubble sorting

\*/

public class BubbleSort {

/\*\*

\* method who sorting our array using bubble sort

\*@param array numbers

\*

\*/

public static void sort(int [] arrayNumbers){

int size = arrayNumbers.length; // get size array

int timeValue = 0;

for (int i = 0; i < size; i++)

{

for (int j = 1; j < size - i; j++ ){

if (arrayNumbers[j-1] > arrayNumbers[j])

{

timeValue = arrayNumbers[j-1]; // get element from array

arrayNumbers[j-1] = arrayNumbers[j]; //change next element

arrayNumbers[j] = timeValue;

}

}

}

}

}

package Sorting;

/\*\*

\* it is class who realize selection sorting

\*/

public class SelectionSort {

/\*\*

\* a method which sorts by exchanging a min element with an element of the first non-sorted position

\* @param arrayNumbers

\*/

public static void sort(int[] arrayNumbers){

int size = arrayNumbers.length;

int minValueIndex = 0;

int timeValue = 0;

for (int i = 0; i < size - 1; i++){

minValueIndex = i;

for (int j = i+1; j < size; j++){

if(arrayNumbers[j] < arrayNumbers[minValueIndex])

minValueIndex = j;

}

timeValue = arrayNumbers[i]; // get element from array

arrayNumbers[i] = arrayNumbers[minValueIndex]; //change next element

arrayNumbers[minValueIndex] = timeValue;

}

}

}

package Sorting;

/\*\*

\* it is class who realize quick sorting

\*/

public class QuickSort {

/\*\*

\* method who sorting our array using quick sort

\* @param arrayNumbers

\*/

public static void sort(int[] arrayNumbers){

int startArray = 0;

int endArray = arrayNumbers.length-1;

quickSort(arrayNumbers, startArray, endArray);

}

/\*\*

\* a method which sorts the recursive parts of the array

\* @param arrayNumbers

\* @param start

\* @param end

\*/

private static void quickSort(int[] arrayNumbers, int start, int end){ //

int timeVale = 0;

int i = start;

int j = end;

int randVal = arrayNumbers[start+(end -start)/2];

do {

while (arrayNumbers[i] < randVal) ++i;

while (arrayNumbers[j] > randVal) --j;

if ( i <= j){

timeVale = arrayNumbers[i];

arrayNumbers[i] = arrayNumbers[j];

arrayNumbers[j] = timeVale;

i++;

j--;

}

}while( i <= j );

if (start < j) quickSort(arrayNumbers, start, j);

if (i < end) quickSort(arrayNumbers, i,end);

}

}

package Sorting;

/\*\*

\* it is class who realize odd even sorting

\*/

public class OddEvenSort {

/\*\*

\* The method which sorts by comparing all the pair / odd pairs of items in the list

\* @param array

\*/

public static void sort(int [] array) {

int arraySize = array.length;

boolean sorted = false;

while (!sorted) {

sorted = true;

for (int i = 0; i < arraySize - 1; i += 2) {

if (array[i] > array[i+1]) {

int temp = array[i];

array[i] = array[i+1];

array[i+1] = temp;

sorted = false;

}

}

for (int i = 1; i < arraySize - 1; i += 2) {

if (array[i] > array[i+1]) {

int temp = array[i];

array[i] = array[i+1];

array[i+1] = temp;

sorted = false;

}

}

}

}

}

package Sorting;

/\*\*

\* it is class who realize merge sorting

\*/

public class MergeSort {

private static int[] array; // our array

private static int[] tempMergArr; // time array

private static int length; // size array

/\*\*

\* method who sort our array using merge sort

\* @param inputArr the array who will be sorting

\*/

public static void sort(int inputArr[]) {

array = inputArr;

length = inputArr.length;

tempMergArr = new int[length];

doMergeSort(0, length - 1);

}

/\*\*

\* method who separate our array on parts then sorting data and merge two sorted arrays

\* @param lowerIndex

\* @param higherIndex

\*/

private static void doMergeSort(int lowerIndex, int higherIndex) {

if (lowerIndex < higherIndex) {

int middle = lowerIndex + (higherIndex - lowerIndex) / 2;

// Below step sorts the left side of the array

doMergeSort(lowerIndex, middle);

// Below step sorts the right side of the array

doMergeSort(middle + 1, higherIndex);

// Now merge both sides

mergeParts(lowerIndex, middle, higherIndex);

}

}

/\*\*

\* method who merge parts

\* @param lowerIndex

\* @param middle

\* @param higherIndex

\*/

private static void mergeParts(int lowerIndex, int middle, int higherIndex) {

for (int i = lowerIndex; i <= higherIndex; i++) {

tempMergArr[i] = array[i];

}

int i = lowerIndex;

int j = middle + 1;

int k = lowerIndex;

while (i <= middle && j <= higherIndex) {

if (tempMergArr[i] <= tempMergArr[j]) {

array[k] = tempMergArr[i];

i++;

} else {

array[k] = tempMergArr[j];

j++;

}

k++;

}

while (i <= middle) {

array[k] = tempMergArr[i];

k++;

i++;

}

}

}

3. В результаті отримаю наступне:

Введите размер масива:

150

Введите команду:

alltimes

Bubble sort time : 825 606

Arrays default sorting time : 260 256

Merge sort time : 692 053

Quick sort time : 135 621

Insert sort time : 560 396

odd even sort time: 337 487

Selection sort time : 596 167

Введите команду:

alltimes

Bubble sort time : 580 841

Arrays default sorting time : 23 018

Merge sort time : 72 898

Quick sort time : 23 136

Insert sort time : 11 063

odd even sort time: 8 222

Selection sort time : 447 182

Введите команду:

alltimes

Bubble sort time : 526 150

Arrays default sorting time : 40 782

Merge sort time : 31 701

Quick sort time : 22 830

Insert sort time : 13 941

odd even sort time: 7 688

Selection sort time : 409 569

Висновок: Під час лабораторної роботи я навчився працювати з алогитмами сортуватння і також навчився вимірювати час виконання того чи іншого алгоритму.