НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ІМ. ІГОРЯ СІКОРСЬКОГО

КАФЕДРА АВТОМАТИКИ ТА УПРАВЛІННЯ В ТЕХНІЧНИХ СИСТЕМАХ

Звіт

з дисципліни «Сучасні технології програмування - 1»

за результатами виконання лабораторної роботи № 3

Виконав:

студент групи ІТ-51

Сопов О. О.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Перевірив:

ас. Федорчук В. В.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Київ – 2017

**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.

|  |  |
| --- | --- |
| 7 | bubble sort, insertion sort, quick sort, cycle sort, shell sort; |

**Вихідний код:**

***Sortable.java***

package ArraySorting;

import java.util.Comparator;

public abstract class Sortable {

public <T extends Comparable<T>> void sort(T[] source) {

processSorting(source, null);

}

public <T> void sort(T[] source, Comparator<T> comparator) {

processSorting(source, comparator);

}

public abstract String getSortingMethodName();

abstract <T> void processSorting(T[] source, Comparator<T> comparator);

<T> int compare(T a, T b, Comparator comparator) {

if (comparator == null) {

return ((Comparable)a).compareTo(b);

}

return comparator.compare(a, b);

}

<T> int compare(T[] source, int a, int b, Comparator comparator) {

if (comparator == null) {

return ((Comparable)source[a]).compareTo(source[b]);

}

return comparator.compare(source[a], source[b]);

}

<T> void swap(T[] x, int a, int b) {

T temp = x[a];

x[a] = x[b];

x[b] = temp;

}

}

***BubbleSorter.java***

package ArraySorting;

import java.util.Comparator;

public class BubbleSorter extends Sortable {

@Override

<T> void processSorting(T[] source, Comparator<T> comparator) {

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

for (int j = 0; j < i; j++) {

if (compare(source, j, j + 1, comparator) > 0) {

swap(source, j, j + 1);

}

}

}

}

@Override

public String getSortingMethodName() {

return "Bubble sort";

}

}

***InsertionSort.java***

package ArraySorting;

import java.util.Comparator;

public class InsertionSort extends Sortable {

@Override

<T> void processSorting(T[] source, Comparator<T> comparator) {

for (int i = 1; i < source.length; i++) {

int j = i - 1;

while (j >= 0 && compare(source, j, i, comparator) > 0) {

source[j + 1] = source[j];

j--;

}

source[j + 1] = source[i];

}

}

@Override

public String getSortingMethodName() {

return "Insertion sort";

}

}

***QuickSorter.java***

package ArraySorting;

import java.util.Comparator;

public class QuickSorter extends Sortable {

@Override

<T> void processSorting(T[] source, Comparator<T> comparator) {

quickSort(source, 0, source.length - 1, comparator);

}

private <T> void quickSort(T[] source, int leftLimit, int rightLimit, Comparator<T> comparator) {

T middleValue = source[(leftLimit + rightLimit) / 2];

int index1 = leftLimit;

int index2 = rightLimit;

while (index1 <= index2)

{

while (index1 < rightLimit && compare(source[index1], middleValue, comparator) < 0) {

index1++;

}

while (index2 > leftLimit && compare(source[index2], middleValue, comparator) > 0) {

index2--;

}

if (index1 <= index2) {

if (index1 < index2) {

swap(source, index1, index2);

}

index1++;

index2--;

}

}

if (index1 < rightLimit) {

quickSort(source, index1, rightLimit, comparator);

}

if (index2 > leftLimit) {

quickSort(source, leftLimit, index2, comparator);

}

}

@Override

public String getSortingMethodName() {

return "Quick sort";

}

}

***ShellSort.java***

package ArraySorting;

import java.util.Comparator;

public class ShellSort extends Sortable {

@Override

<T> void processSorting(T[] source, Comparator<T> comparator) {

int length = source.length;

for (int gap = source.length / 2; gap > 0; gap /= 2) {

for (int i = gap; i < length; i += 1) {

T temp = source[i];

int j;

for (j = i; j >= gap && compare(source[j - gap], temp, comparator) > 0; j -= gap) {

source[j] = source[j - gap];

}

source[j] = temp;

}

}

}

@Override

public String getSortingMethodName() {

return "Shell sort";

}

}

***JavaNativeSort.java***

package ArraySorting;

import java.util.Arrays;

import java.util.Comparator;

public class JavaNativeSort extends Sortable {

@Override

<T> void processSorting(T[] source, Comparator<T> comparator) {

Arrays.sort(source, comparator);

}

@Override

public String getSortingMethodName() {

return "Arrays sort";

}

}