**Міністерство освіти і науки України**

**Національний технічний університет України «Ки**

**ївський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт**

з лабораторної роботи № 1 з дисципліни

«Проектування алгоритмів»

„ **Проектування і аналіз алгоритмів зовнішнього сортування**”

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-14 Радзівіло Валерія Артемівна*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.М.*

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc109342184)

[2 ЗаВдання 4](#_Toc109342185)

[3 Виконання 6](#_Toc109342186)

[3.1 Псевдокод алгоритму 6](#_Toc109342187)

[3.2 Програмна реалізація алгоритму 6](#_Toc109342188)

[3.2.1 Вихідний код 6](#_Toc109342189)

[Висновок 7](#_Toc109342190)

[Критерії оцінювання 8](#_Toc109342191)

# Мета лабораторної роботи

Мета роботи – вивчити основні алгоритми зовнішнього сортування та способи їх модифікації, оцінити поріг їх ефективності.

# Завдання

Згідно варіанту (таблиця 2.1), розробити та записати алгоритм зовнішнього сортування за допомогою псевдокоду (чи іншого способу за вибором).

Виконати програмну реалізацію алгоритму на будь-якій мові програмування та відсортувати випадковим чином згенерований масив цілих чисел, що зберігається у файлі (розмір файлу має бути не менше 10 Мб, можна значно більше).

Здійснити модифікацію програми і відсортувати випадковим чином згенерований масив цілих чисел, що зберігається у файлі розміром не менше ніж двократний обсяг ОП вашого ПК. Досягти швидкості сортування з розрахунку 1Гб на 3хв. або менше.

Рекомендується попередньо впорядкувати серії елементів довжиною, що займає не менше 100Мб або використати інші підходи для пришвидшення процесу сортування.

Зробити узагальнений висновок з лабораторної роботи, у якому порівняти базову та модифіковану програми. У висновку деталізувати, які саме модифікації було виконано і який ефект вони дали.

Таблиця 2.1 – Варіанти алгоритмів

|  |  |
| --- | --- |
| **№** | **Алгоритм сортування** |
| 14 | Природне (адаптивне) злиття |

# Виконання

## Псевдокод алгоритму

**function merge( source[], target[], start\_left, start\_right, end\_right)**

**while both arrays contain elements**

**while left\_pos < start\_right && right\_pos < end\_right**

**leftValue = source[left\_pos]**

**rightValue = source[right\_pos]**

**if leftValue <= rightValue**

**target[targetPos++] = leftValue**

**left\_pos++**

**else**

**target[targetPos++] = rightValue**

**right\_pos++**

**while left\_pos < start\_right**

**target[targetPos++] = source[left\_pos++]**

**while right\_pos < end\_right**

**target[targetPos++] = source[right\_pos++]**

**# take the next sorted chunk from the input tapes, and merge into the single given output\_tape.**

**function sort(elements[])**

**for i = 1 i <= num\_elem i++**

**if i == num\_elem || elements[i] < elements[i - 1]**

**st[++run\_c] = i**

**while run counter > 1**

**for i = 0 i < run counter – 1 i += 2**

**merge(from, to, st[i], st[i + 1], st[i + 2])**

**st[new run counter++] = st[i]**

**if run counter is odd**

**copy the last one**

**swap "from" and "to" arrays**

**If final run is not in "elements"**

**copy it there**

## Програмна реалізація алгоритму

### Вихідний код

**Main.java:**

import FileWork.\*;  
import ForArrays.\*;  
import NaturalMergeSort.\*;  
  
import java.io.IOException;  
import java.nio.file.Files;  
import java.nio.file.Path;  
  
  
public class Main {  
 public static void main(String args[]) throws IOException {  
 FileWork data\_file = new FileWork();  
 ForArrays arr = new ForArrays();  
 String filepath = "C:/Users/leraz/OneDrive/Рабочий стол/asd2/lab1.txt";  
 //store file data in string  
 data\_file.fill\_file(filepath);  
 String file\_content = Files.*readString*(Path.*of*(filepath));  
 file\_content = file\_content.substring(1,file\_content.length() - 1);  
 String[] file\_array\_str = file\_content.split(", ", -1);  
  
  
 //make int array out of string array  
 int[] file\_array\_int = new int[file\_array\_str.length];  
 file\_array\_int = arr.*make\_str\_arr\_to\_int*(file\_array\_str);  
 // print the array before sort  
 System.*out*.println("Basic array: ");  
 arr.*printArray*(file\_array\_int);  
  
 // natural merge sort  
 NaturalMergeSort arr\_to\_sort = new NaturalMergeSort();  
 System.*out*.println("Sorted array: ");  
 arr\_to\_sort.sort(file\_array\_int);  
 arr.*printArray*(file\_array\_int);  
  
  
  
 }  
}

NaturalMergeSort.java:

package NaturalMergeSort;  
  
public class NaturalMergeSort {  
  
 public void sort(int[] elements) {  
 // find array length  
 int num\_elem = elements.length;  
  
 int[] tmp = new int[num\_elem]; // temporary array for sorting  
 int[] st = new int[num\_elem + 1]; // starts  
  
 // identify runs  
 int run\_c = 0; // counter  
 st[0] = 0;  
 for (int i = 1; i <= num\_elem; i++) {  
  
 if (i == num\_elem || elements[i] < elements[i - 1]) {  
 st[++run\_c] = i;  
 }  
 }  
  
 // merge runs, until only 1 run is left  
 int[] from = elements;  
 int[] to = tmp;  
  
 while (run\_c > 1) {  
 // new counter  
 int run\_c\_new = 0;  
  
 // Merge two runs each  
 for (int i = 0; i < run\_c - 1; i += 2) {  
 merge(from, to, st[i], st[i + 1], st[i + 2]);  
 st[run\_c\_new++] = st[i];  
 }  
  
 // if there is odd number of runs - copy the last one  
 if (isOdd(run\_c)) {  
 int lastStart = st[run\_c - 1];  
 System.*arraycopy*(from, lastStart, to, lastStart, num\_elem - lastStart);  
 st[run\_c\_new++] = lastStart;  
 }  
  
 // Prepare for next round  
 st[run\_c\_new] = num\_elem;  
 run\_c = run\_c\_new;  
  
 // Swap "from" and "to" arrays  
 int[] help = from;  
 from = to;  
 to = help;  
 }  
  
 // if final run is not in "elements", copy it there  
 if (isNotSameArray(from, elements)) {  
 System.*arraycopy*(from, 0, elements, 0, num\_elem);  
 }  
 }  
  
 private void merge(int[] source, int[] target, int start\_left, int start\_right, int end\_right) {  
 int left\_pos = start\_left;  
 int right\_pos = start\_right;  
 int target\_pos = start\_left;  
  
 // As long as both arrays contain elements  
 while (left\_pos < start\_right && right\_pos < end\_right) {  
 // find out which one is smaller  
 int leftValue = source[left\_pos];  
 int rightValue = source[right\_pos];  
 if (leftValue <= rightValue) {  
 target[target\_pos++] = leftValue;  
 left\_pos++;  
 } else {  
 target[target\_pos++] = rightValue;  
 right\_pos++;  
 }  
 }  
 // Copy the rest  
 while (left\_pos < start\_right) {  
 target[target\_pos++] = source[left\_pos++];  
 }  
 while (right\_pos < end\_right) {  
 target[target\_pos++] = source[right\_pos++];  
 }  
 }  
  
 private boolean isOdd(int number) {  
 return number % 2 != 0;  
 }  
  
 // check the same instance  
 private boolean isNotSameArray(int[] array1, int[] array2) {  
 return array1 != array2;  
 }  
  
  
}

ForArrays.java:

package ForArrays;  
  
public class ForArrays {  
  
 /\* A utility function to print array of size n \*/  
 public static void printArray(int arr[]) {  
 int n = arr.length;  
 for (int i = 0; i < n; ++i)  
 System.*out*.print(arr[i] + " ");  
 System.*out*.println();  
 }  
  
  
 // make int array  
 public static int[] make\_str\_arr\_to\_int(String arr1[])  
 {  
 int[] intList = new int[arr1.length];  
  
 // parsing the String argument to make int array  
 for (int i = 0; i < arr1.length; i++) {  
 intList[i] = Integer.*valueOf*(arr1[i]);  
 }  
 return intList;  
 }  
}

FileWork.java:

package FileWork;  
import java.io.\*;  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
import java.util.Random;  
  
  
public class FileWork {  
  
  
 // create random array  
 int[] make\_arr(int len)  
 {  
 Random random = new Random();  
  
 int[] arr = new int[len];  
  
 for (int i = 0; i < len; i++)  
 arr[i] = random.nextInt(1000000000);  
  
 return arr;  
 }  
  
 public void fill\_file(String filepath) {  
 // get random size  
 int len = (int) (Math.*random*() \* (10000000 - 9000000)) + 9000000;  
 int[] arr = make\_arr(len);  
 String text\_for\_file = Arrays.*toString*(arr);  
 try {  
  
 // Create a FileWriter object  
 // to write in the file  
 FileWriter fWriter = new FileWriter(filepath);  
  
 // Writing into file  
 // Note: The content taken above inside the  
 // string  
 fWriter.write(text\_for\_file);  
  
  
 // Closing the file writing connection  
 fWriter.close();  
  
 // Display message for successful execution of  
 // program on the console  
 System.*out*.println(  
 "File is created successfully with the content.");  
 }  
  
 // Catch block to handle if exception occurs  
 catch (  
 IOException e) {  
  
 // Print the exception  
 System.*out*.print(e.getMessage());  
 }  
 }  
  
 public String[] get\_array(String filepath) throws IOException {  
 List<String> listOfStrings = new ArrayList<String>();  
  
 // load data from file  
 BufferedReader bf = new BufferedReader(new FileReader(filepath));  
  
 // read entire line as string  
 String line = bf.readLine();  
  
 // checking for end of file  
 while (line != null) {  
 listOfStrings.add(line);  
 line = bf.readLine();  
 }  
  
 // closing bufferreader object  
 bf.close();  
  
 // storing the data in arraylist to array  
 String[] array = listOfStrings.toArray(new String[0]);  
  
 // printing each line of file  
 // which is stored in array  
 for (String str : array) {  
 System.*out*.println(str);  
 }  
  
 return array;  
 }  
}

Модифікація:

Main.java:

import FileWork.\*;  
import ForArrays.\*;  
import NaturalMergeSort.\*;  
  
import java.io.IOException;  
import java.lang.reflect.Array;  
import java.util.ArrayList;  
import java.util.Collections;  
import java.util.List;  
  
  
public class Main {  
 public static void main(String args[]) throws IOException {  
 FileWork base\_file = new FileWork();  
 NaturalMergeSort sort\_arr = new NaturalMergeSort();  
 ForArrays arr = new ForArrays();  
 String filepath = "Big\_Rand\_Numbers.txt";  
  
  
 arr.make\_large\_arr();  
 System.*out*.println("\n");  
  
  
 int count\_lines = base\_file.count\_lines(filepath)/3374;  
 System.*out*.println(count\_lines);  
 int add = count\_lines/3;  
 int counter\_files = 0;  
 int[] sizes = new int[3];  
 for(int i =0; counter\_files<3;i+=add) {  
 // розберись з add  
 // об'єднай ці файли в 3, а потім в 1  
 int[] int\_text\_arr = base\_file.text\_in\_arr(filepath, i, add\*(counter\_files+1), count\_lines);  
 sort\_arr.sort(int\_text\_arr);  
 String new\_file\_name = "A" + Integer.*toString*(counter\_files)+".txt";  
 base\_file.fill\_file(new\_file\_name, int\_text\_arr);  
 if(i+add>count\_lines)  
 add=count\_lines-i;  
 sizes[counter\_files] = int\_text\_arr.length;  
 counter\_files++;  
 }  
  
 int size\_for\_three\_files = sizes[0]/3+sizes[1]/3+sizes[2]/3;  
  
 base\_file.divide\_and\_sort(size\_for\_three\_files, sizes,base\_file, arr, sort\_arr, 3,"A","B");  
   
  
  
  
  
  
 }  
  
  
}

ForArrays.java:

package ForArrays;  
  
import java.io.\*;  
import java.nio.charset.StandardCharsets;  
import java.nio.file.Files;  
import java.nio.file.Paths;  
import java.util.List;  
import java.util.Random;  
import java.util.Scanner;  
  
public class ForArrays {  
  
 /\* A utility function to print array of size n \*/  
 public void printArray(int arr[]) {  
 int n = arr.length;  
 for (int j : arr) System.*out*.print(j + " ");  
 System.*out*.println();  
 }  
  
 public void printStrArray(String arr[]) {  
 int n = arr.length;  
 for (String s : arr) System.*out*.print(s + " ");  
 System.*out*.println();  
 }  
  
  
 // make int array  
 public int[] make\_str\_arr\_to\_int(String arr1[])  
 {  
 int[] intList = new int[arr1.length];  
  
 // parsing the String argument to make int array  
 for (int i = 0; i < arr1.length; i++) {  
 intList[i] = Integer.*valueOf*(arr1[i]);  
 }  
 return intList;  
 }  
  
  
  
 public void make\_large\_arr() throws FileNotFoundException, UnsupportedEncodingException {  
 //Size in Gbs of my file that I want  
 double wantedSize = 1.0;  
 int line\_count = 0;  
 Random random = new Random();  
 File file = new File("Big\_Rand\_Numbers.txt");  
 long start = System.*currentTimeMillis*();  
 PrintWriter writer = new PrintWriter(new BufferedWriter(new OutputStreamWriter(new FileOutputStream(file), "UTF-8")), false);  
 int counter = 0;  
 while (true) {  
 for (int i = 0; i < 500; i++) {  
 int number = random.nextInt(1000) + 1;  
 writer.print(number);  
 writer.print("\n");  
 }  
  
// writer.println();  
 //Check to see if the current size is what we want it to be  
 if (++counter == 20000) {  
 System.*out*.printf("Size: %.3f GB%n", file.length() / 1e9);  
 if (file.length() >= wantedSize \* 1e9) {  
 writer.close();  
 break;  
 } else {  
 counter = 0;  
 }  
 }  
 }  
 long time = System.*currentTimeMillis*() - start;  
 System.*out*.printf("Took %.1f seconds to create a file of %.3f GB", time / 1e3, file.length() / 1e9);  
 }  
  
  
 public Integer[] make\_from\_int\_Intenger(int[] arr)  
 {  
 Integer array[] = new Integer[arr.length];  
 for (int i = 0; i < arr.length; i++) {  
 array[i] = Integer.*valueOf*(arr[i]);  
 }  
 return array;  
 }  
  
 public int[] make\_from\_Intenger\_int(Integer[] arr)  
 {  
 int array[] = new int[arr.length];  
 for (int i = 0; i < arr.length; i++) {  
 array[i] = (int)arr[i];  
 }  
 return array;  
 }  
  
 public int[] delete\_values\_from\_array(int amount, int [] arr)  
 {  
 int[] arr\_new = new int[arr.length-amount];  
 for (int i = 0, k = 0; i < arr.length; i++) {  
 if (i>amount) {  
 arr\_new[k] = arr[i];  
 k++;  
 }  
  
 }  
 return arr\_new;  
 }  
}

NaturalMergeSort.java:

package NaturalMergeSort;  
  
public class NaturalMergeSort {  
  
 public void sort(int[] elements) {  
 // find array length  
 int num\_elem = elements.length;  
  
 int[] tmp = new int[num\_elem]; // temporary array for sorting  
 int[] st = new int[num\_elem + 1]; // starts  
  
 // identify runs  
 int run\_c = 0; // counter  
 st[0] = 0;  
 for (int i = 1; i <= num\_elem; i++) {  
  
 if (i == num\_elem || elements[i] < elements[i - 1]) {  
 st[++run\_c] = i;  
 }  
 }  
  
 // merge runs, until only 1 run is left  
 int[] from = elements;  
 int[] to = tmp;  
  
 while (run\_c > 1) {  
 // new counter  
 int run\_c\_new = 0;  
  
 // Merge two runs each  
 for (int i = 0; i < run\_c - 1; i += 2) {  
 merge(from, to, st[i], st[i + 1], st[i + 2]);  
 st[run\_c\_new++] = st[i];  
 }  
  
 // if there is odd number of runs - copy the last one  
 if (isOdd(run\_c)) {  
 int lastStart = st[run\_c - 1];  
 System.*arraycopy*(from, lastStart, to, lastStart, num\_elem - lastStart);  
 st[run\_c\_new++] = lastStart;  
 }  
  
 // Prepare for next round  
 st[run\_c\_new] = num\_elem;  
 run\_c = run\_c\_new;  
  
 // Swap "from" and "to" arrays  
 int[] help = from;  
 from = to;  
 to = help;  
 }  
  
 // if final run is not in "elements", copy it there  
 if (isNotSameArray(from, elements)) {  
 System.*arraycopy*(from, 0, elements, 0, num\_elem);  
 }  
 }  
  
 private void merge(int[] source, int[] target, int start\_left, int start\_right, int end\_right) {  
 int left\_pos = start\_left;  
 int right\_pos = start\_right;  
 int target\_pos = start\_left;  
  
 // As long as both arrays contain elements  
 while (left\_pos < start\_right && right\_pos < end\_right) {  
 // find out which one is smaller  
 int leftValue = source[left\_pos];  
 int rightValue = source[right\_pos];  
 if (leftValue <= rightValue) {  
 target[target\_pos++] = leftValue;  
 left\_pos++;  
 } else {  
 target[target\_pos++] = rightValue;  
 right\_pos++;  
 }  
 }  
 // Copy the rest  
 while (left\_pos < start\_right) {  
 target[target\_pos++] = source[left\_pos++];  
 }  
 while (right\_pos < end\_right) {  
 target[target\_pos++] = source[right\_pos++];  
 }  
 }  
  
 private boolean isOdd(int number) {  
 return number % 2 != 0;  
 }  
  
 // check the same instance  
 private boolean isNotSameArray(int[] array1, int[] array2) {  
 return array1 != array2;  
 }  
  
  
}

Filework.java:

package FileWork;  
import ForArrays.ForArrays;  
import NaturalMergeSort.NaturalMergeSort;  
  
import java.io.\*;  
import java.lang.reflect.Array;  
import java.nio.charset.Charset;  
import java.nio.file.Files;  
import java.nio.file.Path;  
import java.nio.file.Paths;  
import java.util.\*;  
import java.util.stream.Stream;  
  
  
public class FileWork {  
  
  
 public String[] get\_array(String filepath) throws IOException {  
 List<String> listOfStrings = new ArrayList<String>();  
  
 // load data from file  
 BufferedReader bf = new BufferedReader(new FileReader(filepath));  
  
 // read entire line as string  
 String line = bf.readLine();  
  
 // checking for end of file  
 while (line != null) {  
 listOfStrings.add(line);  
 line = bf.readLine();  
 }  
  
 // closing bufferreader object  
 bf.close();  
  
 // storing the data in arraylist to array  
 String[] array = listOfStrings.toArray(new String[0]);  
  
 // printing each line of file  
 // which is stored in array  
 for (String str : array) {  
 System.*out*.println(str);  
 }  
  
 return array;  
 }  
  
  
 public void fill\_file(String filepath, int[] arr) throws IOException {  
  
 FileWriter writer = new FileWriter(filepath);  
 int len = arr.length;  
 for (int i = 0; i < len; i++) {  
 writer.write(arr[i] + "\n");  
 }  
 writer.close();  
 System.*out*.println("File was filled successfully!\n");  
  
  
 }  
  
 public int[] text\_in\_arr(String filepath\_old, int start\_point, int add, int size\_file) throws IOException {  
 if (start\_point!=0)  
 start\_point++;  
 BufferedReader reader;  
 if(size\_file-add<1000)  
 add = size\_file;  
 int[] int\_text;  
 if(add>start\_point)  
 int\_text = new int[add-start\_point+1];  
 else  
 int\_text = new int[add+1];  
  
 try {  
 reader = new BufferedReader(new FileReader(filepath\_old));  
 String line = "1";  
 int i = 0;  
 int counter = 0;  
 while (line != null && i < add) {  
 // read next line  
 line = reader.readLine();  
 if(i>=start\_point) {  
 int\_text[counter] = Integer.*parseInt*(line);  
 counter++;  
 }  
 i++;  
 }  
 reader.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return int\_text;  
 }  
  
  
 public int count\_lines(String filepath) {  
 long numOfLines;  
 try (Stream<String> lines = Files.*lines*(Path.*of*(filepath), Charset.*defaultCharset*())) {  
 numOfLines = lines.count();  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 }  
 return (int) numOfLines;  
  
  
 }  
  
 public void divide\_and\_sort(int size\_for\_three\_files, int[] sizes, FileWork base\_file, ForArrays arr, NaturalMergeSort sort\_arr, int amount\_files, String old\_letter, String new\_letter) throws IOException {  
 int[] full\_array = new int[size\_for\_three\_files+amount\_files];  
 int start\_point = 0;  
 for(int j =0; j<amount\_files; j++) {  
 for (int i = 0; i < 3; i++) {  
 String file\_name = old\_letter + Integer.*toString*(i) + ".txt";  
 if (j!=0)  
 start\_point=(sizes[i]/amount\_files)\*j;  
 int[] half\_array = base\_file.text\_in\_arr(file\_name, start\_point, (sizes[i] / amount\_files)\*(j+1), sizes[i]);  
 if (i!=0)  
 full\_array = arr.make\_from\_Intenger\_int(*concatWithCollection*(arr.make\_from\_int\_Intenger(full\_array), arr.make\_from\_int\_Intenger(half\_array)));  
 else  
 full\_array = half\_array;  
 sort\_arr.sort(full\_array);  
 }  
 full\_array = arr.delete\_values\_from\_array(amount\_files, full\_array);  
 String new\_file\_name = new\_letter + Integer.*toString*(j) + ".txt";  
 base\_file.fill\_file(new\_file\_name, full\_array);  
 }  
 }  
  
 static Integer[] concatWithCollection(Integer[] array1, Integer[] array2) {  
 List<Integer> resultList = new ArrayList<>(array1.length + array2.length);  
 Collections.*addAll*(resultList, array1);  
 Collections.*addAll*(resultList, array2);  
  
 @SuppressWarnings("unchecked")  
//the type cast is safe as the array1 has the type T[]  
 Integer[] resultArray = (Integer[]) Array.*newInstance*(array1.getClass().getComponentType(), 0);  
 return resultList.toArray(resultArray);  
 }  
}

Висновок

При виконанні даної лабораторної роботи було вивчено основні алгоритми зовнішнього сортування та способи їх модифікації. В ході роботи була створена програма з роботою з зовнішнім файлом та випадковим масивом чисел, поміщеним у цей файл. Створена програма здатна сортувати та створювати масиви різних величин за відносно невеликий час. Також було оцінено поріг ефективності методу сортування Natural Merge Sort та порівняно його з раніше вивченими методами сортування. У базовому сортуванні злиттям початкова точка передбачає, що кожен запуск складається з одного елемента. На практиці випадкові вхідні дані матимуть багато коротких циклів, які просто сортуються. У модернізоване сортування злиттям може не потребувати стільки проходів, оскільки потрібно менше циклів для злиття. Модернізована програма здатна працювати з файлами великого обсягу за доволі короткий час (приблизно 3 хвилини).

Критерії оцінювання

У випадку здачі лабораторної роботи до 09.10.2022 включно максимальний бал дорівнює – 5. Після 09.10.2022 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

* псевдокод алгоритму – 15%;
* програмна реалізація алгоритму – 40%;
* програмна реалізація модифікацій – 40%;
* висновок – 5%.