**Міністерство освіти і науки України**

**Національний технічний університет України «Київський 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 Громов Владислав*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.М.*

Київ 2022

Зміст

[**1**](#_30j0zll) **Мета лабораторної роботи 3**

[**2**](#_1fob9te) **ЗаВдання 4**

[**3**](#_3znysh7) **Виконання 6**

[3.1](#_2et92p0) Псевдокод алгоритму 6

[3.2](#_tyjcwt) Програмна реалізація алгоритму 6

[*3.2.1*](#_3dy6vkm) *Вихідний код 6*

[**Висновок 7**](#_1t3h5sf)

[**Критерії оцінювання 8**](#_4d34og8)

# 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** |
| 1 | Пряме злиття |
| 2 | Природне (адаптивне) злиття |
| 3 | Збалансоване багатошляхове злиття |
| 4 | Багатофазне сортування |
| 5 | Пряме злиття |
| 6 | Природне (адаптивне) злиття |
| 7 | Збалансоване багатошляхове злиття |
| 8 | Багатофазне сортування |
| 9 | Пряме злиття |
| 10 | Природне (адаптивне) злиття |
| 11 | Збалансоване багатошляхове злиття |
| 12 | Багатофазне сортування |
| 13 | Пряме злиття |
| 14 | Природне (адаптивне) злиття |
| 15 | Збалансоване багатошляхове злиття |
| 16 | Багатофазне сортування |
| 17 | Пряме злиття |
| 18 | Природне (адаптивне) злиття |
| 19 | Збалансоване багатошляхове злиття |
| 20 | Багатофазне сортування |
| 21 | Пряме злиття |
| 22 | Природне (адаптивне) злиття |
| 23 | Збалансоване багатошляхове злиття |
| 24 | Багатофазне сортування |
| 25 | Пряме злиття |
| 26 | Природне (адаптивне) злиття |
| 27 | Збалансоване багатошляхове злиття |
| 28 | Багатофазне сортування |
| 29 | Пряме злиття |
| 30 | Природне (адаптивне) злиття |
| 31 | Збалансоване багатошляхове злиття |
| 32 | Багатофазне сортування |
| 33 | Пряме злиття |
| 34 | Природне (адаптивне) злиття |
| 35 | Збалансоване багатошляхове злиття |

# Виконання

## Псевдокод алгоритму

class MergeSort {

private static void Merge(int[] arr, int l, int m, int r)

{

var n1 = m - l + 1;

var n2 = r - m;

var L = new int[n1];

var R = new int[n2];

int i, j;

for (i = 0; i < n1; ++i)

L[i] = arr[l + i];

for (j = 0; j < n2; ++j)

R[j] = arr[m + 1 + j];

i = 0;

j = 0;

int k = l;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

private static void Merge(string ll, string rr, string directory, int n)

{

var i = 0;

var j = 0;

var lpath = new StreamReader(ll);

var rpath = new StreamReader(rr);

int l;

TryParse(lpath.ReadLine(), out l);

int r;

TryParse(rpath.ReadLine(), out r);

var main = new StreamWriter(directory + $@"\{n}.txt");

while (lpath.Peek() > -1 && rpath.Peek() > -1) {

if (l <= r) {

main.WriteLine(l);

TryParse(lpath.ReadLine(), out l);

}

else {

main.WriteLine(r);

TryParse(rpath.ReadLine(), out r);

}

}

while (lpath.Peek() > -1) {

main.WriteLine(l);

TryParse(lpath.ReadLine(), out l);

}

while (rpath.Peek() > -1) {

main.WriteLine(r);

TryParse(rpath.ReadLine(), out r);

}

main.Close();

lpath.Close();

rpath.Close();

File.Delete(ll);

File.Delete(rr);

}

private static void TempFileMerging(string splittedFiles)

{

var i = 0;

var files = Directory.GetFiles(splittedFiles, "\*", SearchOption.TopDirectoryOnly);

while (files.Length > 1)

{

Merge(files[0], files[1], splittedFiles, i);

i++;

files = Directory.GetFiles(splittedFiles, "\*", SearchOption.TopDirectoryOnly);

}

}

private static void Sort(int[] arr, int l, int r)

{

if (l >= r) return;

var m = l + (r - l) / 2;

Sort(arr, l, m);

Sort(arr, m + 1, r);

Merge(arr, l, m, r);

}

private static void FileSplit(string path, string splitFile, int count)

{

var size = File.ReadLines(path).Count();

var buf = size / count;

var reader = new StreamReader(path);

var k = 0;

for (int i = 1; i <= count; i++)

{

var writer = new StreamWriter(string.Format(splitFile, i));

for (int j = k ; j < size - buf \* (count - i) - 1; j++)

{

writer.WriteLine(reader.ReadLine());

}

writer.Write(reader.ReadLine());

writer.Close();

k = size - buf \* (count - i);

}

reader.Close();

}

private static void SortSplittedFiles(string splittedFiles)

{

var fCount = Directory.GetFiles(splittedFiles, "\*", SearchOption.TopDirectoryOnly).Length;

string path;

int[] arr;

for (int i = 1; i <= fCount; i++)

{

path = splittedFiles + $@"\file{i}.txt";

arr = Array.ConvertAll(File.ReadAllLines(path), Parse);

Sort(arr, 0, arr.Length - 1);

File.WriteAllText(path, string.Join("\n", arr));

}

}

private static void OptimizedSort(string genPath)

{

var splittedFiles = @"C:\Users\Владислав\Desktop\SplittedFiles";

var splittedFile = @"C:\Users\Владислав\Desktop\SplittedFiles\file{0}.txt";

FileSplit(genPath, splittedFile, 10);

SortSplittedFiles(splittedFiles);

TempFileMerging(splittedFiles);

}

public static void Main(String[] args)

{

var sss = DateTime.Now;

OptimizedSort(@"C:\Users\Владислав\Desktop\ss.txt");

Console.WriteLine((DateTime.Now - sss));

Console.ReadLine();

var arr = Array.ConvertAll(File.ReadAllLines(@"C:\Users\Владислав\Desktop\sss.txt"), Parse);;

var ss = DateTime.Now;

Sort(arr, 0, arr.Length - 1);

Console.WriteLine((DateTime.Now - ss).Seconds);

}

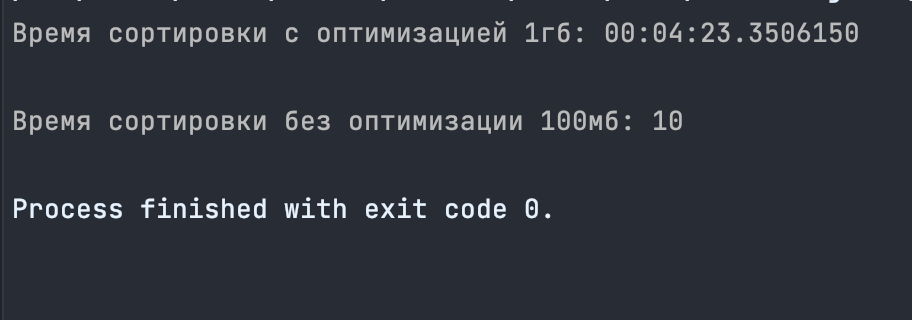
}

## Програмна реалізація алгоритму

### Вихідний код

1. using static System.Int32;  
     
   class MergeSort {  
     
    private static void Merge(int[] arr, int l, int m, int r)  
    {  
    var n1 = m - l + 1;  
    var n2 = r - m;  
     
    var L = new int[n1];  
    var R = new int[n2];  
    int i, j;  
     
    for (i = 0; i < n1; ++i)  
    L[i] = arr[l + i];  
    for (j = 0; j < n2; ++j)  
    R[j] = arr[m + 1 + j];  
     
     
    i = 0;  
    j = 0;  
     
    int k = l;  
    while (i < n1 && j < n2) {  
    if (L[i] <= R[j]) {  
    arr[k] = L[i];  
    i++;  
    }  
    else {  
    arr[k] = R[j];  
    j++;  
    }  
    k++;  
    }  
     
    while (i < n1) {  
    arr[k] = L[i];  
    i++;  
    k++;  
    }  
    while (j < n2) {  
    arr[k] = R[j];  
    j++;  
    k++;  
    }  
    }  
     
    private static void Merge(string ll, string rr, string directory, int n)  
    {  
    var i = 0;  
    var j = 0;  
     
    var lpath = new StreamReader(ll);  
    var rpath = new StreamReader(rr);  
     
    int l;  
    TryParse(lpath.ReadLine(), out l);  
    int r;  
    TryParse(rpath.ReadLine(), out r);  
     
    var main = new StreamWriter(directory + $@"\{n}.txt");  
      
    while (lpath.Peek() > -1 && rpath.Peek() > -1) {  
    if (l <= r) {  
    main.WriteLine(l);  
    TryParse(lpath.ReadLine(), out l);  
    }  
    else {  
    main.WriteLine(r);  
    TryParse(rpath.ReadLine(), out r);  
    }  
    }  
     
    while (lpath.Peek() > -1) {  
    main.WriteLine(l);  
    TryParse(lpath.ReadLine(), out l);  
    }  
    while (rpath.Peek() > -1) {  
    main.WriteLine(r);  
    TryParse(rpath.ReadLine(), out r);  
    }  
      
    main.Close();  
    lpath.Close();  
    rpath.Close();  
      
    File.Delete(ll);  
    File.Delete(rr);  
    }  
     
    private static void TempFileMerging(string splittedFiles)  
    {  
    var i = 0;  
    var files = Directory.GetFiles(splittedFiles, "\*", SearchOption.**TopDirectoryOnly**);  
    while (files.Length > 1)  
    {  
    Merge(files[0], files[1], splittedFiles, i);  
    i++;  
    files = Directory.GetFiles(splittedFiles, "\*", SearchOption.**TopDirectoryOnly**);  
    }  
    }  
     
    private static void Sort(int[] arr, int l, int r)  
    {  
    if (l >= r) return;  
    var m = l + (r - l) / 2;  
     
    Sort(arr, l, m);  
    Sort(arr, m + 1, r);  
     
    Merge(arr, l, m, r);  
    }  
     
    private static void FileSplit(string path, string splitFile, int count)  
    {  
    var size = File.ReadLines(path).Count();  
    var buf = size / count;  
    var reader = new StreamReader(path);  
    var k = 0;  
    for (int i = 1; i <= count; i++)  
    {  
    var writer = new StreamWriter(string.Format(splitFile, i));  
      
    for (int j = k ; j < size - buf \* (count - i) - 1; j++)  
    {  
    writer.WriteLine(reader.ReadLine());  
    }  
    writer.Write(reader.ReadLine());  
    writer.Close();  
    k = size - buf \* (count - i);  
    }  
    reader.Close();  
      
    }  
     
    private static void SortSplittedFiles(string splittedFiles)  
    {  
    var fCount = Directory.GetFiles(splittedFiles, "\*", SearchOption.**TopDirectoryOnly**).Length;  
    string path;  
    int[] arr;  
    for (int i = 1; i <= fCount; i++)  
    {  
    path = splittedFiles + $@"\file{i}.txt";  
    arr = Array.ConvertAll(File.ReadAllLines(path), Parse);  
    Sort(arr, 0, arr.Length - 1);  
    File.WriteAllText(path, string.Join("\n", arr));  
    }  
    }  
     
    private static void OptimizedSort(string genPath)  
    {  
    var splittedFiles = @"C:\Users\Владислав\Desktop\SplittedFiles";  
    var splittedFile = @"C:\Users\Владислав\Desktop\SplittedFiles\file{0}.txt";  
    FileSplit(genPath, splittedFile, 10);  
    SortSplittedFiles(splittedFiles);  
    TempFileMerging(splittedFiles);  
    }  
     
    public static void Main(String[] args)  
    {  
    var sss = DateTime.Now;  
    OptimizedSort(@"C:\Users\Владислав\Desktop\ss.txt");  
    Console.WriteLine((DateTime.Now - sss));  
      
    Console.ReadLine();  
      
    var arr = Array.ConvertAll(File.ReadAllLines(@"C:\Users\Владислав\Desktop\sss.txt"), Parse);;  
      
    var ss = DateTime.Now;  
    Sort(arr, 0, arr.Length - 1);  
    Console.WriteLine((DateTime.Now - ss).Seconds);  
    }  
   }

Скрин результатов



Висновок

При виконанні даної лабораторної роботи я навчився оптимізувати алгоритми сортування, а також швидкої та ефективної роботи з файлами.

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%.