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 алгоритмів зовнішнього сортування»

Варіант 4

Виконав студент ІП-15, Буяло Дмитро Олександрович
(шифр, прізвище, ім'я, по батькові)

Перевірив Ахаладзе Ілля Елдарійович
(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

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

Індивідуальне завдання

Варіант 4

2 ЗАВДАННЯ

Для алгоритму багатофазного сортування 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 32Гб. 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.

3 ВИКОНАННЯ

3.1 Псевдокод алгоритму

Основна програма

polyPhaseSort()

Початок

level = 0; j = 0; a[n-1] = 0; d[n-1] = 0

//Запис по одній серії в кожен з вхідних файлів

Повторити

select()

writeRuns()

Поки не кінець файлу f0 та j! = n-1

Все повторити

j = 0

//Виконати початковий розподіл серій

Повторити

Поки не кінець файлу f0

select()

Якщо f[j].first <= f0.first

то

writeRuns()

Якщо кінець файлу f0

то

d[j]++

інакше

writeRuns()

Все якщо

```
        інакше
            writeRuns()
        Все якщо
    Все повторити
//Відкрити вхідні послідовності f[i] читання
    Повторити для і від 0 до n-1
        t[i] = i
    Все повторити
    t[n-1] = n-1
//Сортування
    Повторити
        z = a[n-2]; d[n-1] = 0
        Повторити
            k = 0
            Повторити для і від 0 до n-1
                Якщо d[i] > 0
                 то
                    d[i]--
                 інакше
                    ta[k] = t[i]; k++
                Все якщо
            Все повторити
            Якщо k=0
             то
                d[n-1]++
             інакше
                Повторити
                    i = 0; mx = 0; min = f[ta[0]].first
```

```
                Повторити
                Поки i < k
                    i++; x = f[ta[i]].first
                    Якщо x < min
                      то
                        min = x; mx = i
                    Все якщо
                Все повторити
                записати min у вихідну послідовність, індекс якої t[N-1];
                Якщо кінець серії у файлі f[ta[mx]]
                  то
                    k--; ta[mx] = ta[k]
                Все якщо
              Поки k > 0
              Все повторити
            Все якщо
            z--
        Поки z > 0
        Все повторити
        tmpT = t[n-1]; tmpD = d[n-1]; z = a[n-2]
        Повторити для і від n-1 до 0
            t[i] = t[i-1]; d[i] = d[i-1]; a[i] = a[i-1] - z
        Все повторити
        t[0] = tmpT; d[0] = tmpD; a[0] = z
        level--
    Поки level != 0
    Все повторити
Кінець
```

Підпрограми:

select()

Початок

Якщо d[j] < d[j+1]

то

j++

інакше

Якщо d[j] = 0

то

level++; a0 = a[0]

Повторити для і від 0 до n-1

d[i] = a0 + a[i+1] - a[i]

a[i] = a0 + a[i+1]

Все повторити

Все якщо

j = 0

Все якщо

d[j]--

Кінець

writeRuns()

Початок

Повторити

Записати серію поелементно в файл f[j] з файлу f0

Поки не кінець серії

Все повторити

Кінець

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

Main.java

```java
import java.io.BufferedReader;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.Arrays;
import java.util.Random;
import java.util.Scanner;

import static java.lang.Character.toUpperCase;
import static java.lang.Integer.MAX_VALUE;

public class Main {
    74 usages
    public static int n, j, l;

    public static void main(String[] args) {
        System.out.println("Hello world!");
        Scanner scan = new Scanner(System.in);

        System.out.print("Enter the name of the initial file: ");
        String name = scan.nextLine() + ".txt";

        String system = inputSystem();

        long size = inputSize(system);

        double time = System.currentTimeMillis();
        double resTime;
        try {
            long count = generateFile(name, system, size);
            resTime = System.currentTimeMillis() - time;

            System.out.println("The file named \"" + name + "\" with a size of " + size + system +
                    " was successfully generated within " + resTime + "ms!");

            System.out.println();
            n = inputN();

            System.out.print("Do you want to use modified version of the algorithm? (Y/N): ");
            boolean yn = checkYN();

            int res;
            time = System.currentTimeMillis();
            if(yn) {
                res = polyPhaseSort_M(count, name, system, size);
                resTime = System.currentTimeMillis() - time;
                System.out.println("The result of the modified algorithm is in the file T"+res+
                        ".txt and completed within "+resTime+"ms!");
            }
            else {
                res = polyPhaseSort(count, name);
```

```
                resTime = System.currentTimeMillis() - time;
                System.out.println("The result of the basic algorithm is in the file T"+res+
                        ".txt and completed within "+resTime+"ms!");
            }
            test(name, sorted: "T"+res+".txt");
        } catch (Exception e) {
            System.err.println(e);
        }
    }

    1 usage
    private static String inputSystem() {
        Scanner sc = new Scanner(System.in);
        String system;
        while (true) {
            try {
                System.out.print("In which unit of information the file must be created? (Kb/Mb/Gb): ");
                system = sc.nextLine().toUpperCase();
                if(system.equals("GB") || system.equals("MB")||system.equals("KB") || system.equals("B")) break;
                else System.out.println("Try again!");
            }
            catch (Exception e) {
                System.out.println("Try again!");
                sc.next();
            }
        }
        return system;
    }

    1 usage
    private static long inputSize(String system) {
        Scanner sc = new Scanner(System.in);
        long size;
        while (true) {
            try {
                System.out.print("Enter the file size in " + system + ": ");
                size = sc.nextLong();
                if(size>0) break;
                else System.out.println("Size must be positive. Try again!");
            }
            catch (Exception e) {
                System.out.println("Size must be integer. Try again!");
                sc.next();
            }
        }
        return size;
    }

    1 usage
    private static int inputN() {
        Scanner sc = new Scanner(System.in);
        int n;
        System.out.print("How many files should be used for sorting? ");
        while (true) {
```

```
105             try {                                                                    ⚠ 12
106                 System.out.print("N = ");
107                 n = sc.nextInt();
108                 if(n>2) break;
109                 else System.out.println("N must be greater than 2. Try again!");
110             }
111             catch (Exception e) {
112                 System.out.println("N must be integer. Try again!");
113                 sc.next();
114             }
115         }
116         return n;
117     }
118
            1 usage
119     private static boolean checkYN() {
120         Scanner scan = new Scanner(System.in);
121         char yn = toUpperCase(scan.nextLine().charAt(0));
122         return (yn=='Y' || yn=='1');
123     }
124
            1 usage
125 @   private static long generateFile(String name, String system, long size) throws IOException {
126         File file = new File(name);
127         file.createNewFile();
128         FileWriter in = new FileWriter(file);
129         Random rand = new Random();
130         StringBuilder s = new StringBuilder();                                       ⚠ 12
131         long count = 0;
132         switch (system) {
133             case "GB":
134                 size *= 1024;
135             case "MB":
136                 size *= 1024;
137             case "KB":
138                 size *= 1024;
139         }
140         while (file.length() < size) {
141             for (int i = 0; i < 10240; ++i) {
142                 s.append(rand.nextInt(MAX_VALUE));
143                 s.append('\n');
144             }
145             in.write(s.toString());
146             s.setLength(0);
147             ++count;
148         }
149         in.close();
150         return count * 10240;
151     }
152
            6 usages
153 @   public static void select(int[] a, int[] d) { // алгоритм распределения серий
154         int i,a0;
155         if(d[j]<d[j+1]) ++j;
```

```
    else {
        if(d[j]==0) {
            ++l;
            a0=a[0];
            for (i=0;i<n-1;++i) {
                d[i]=a0+a[i+1]-a[i];
                a[i] = a0 + a[i + 1];
            }
        }
        j=0;
    }
    --d[j];
}

1 usage
public static int polyPhaseSort(long eof, String name) throws IOException {
    int i, z, k, x, mx, min, tmpT, tmpD;
    boolean eor = false;  // конец ли серии (end of run)
    int[] a = new int[n]; // идеальное число серий
    int[] d = new int[n]; // число пустых серий
    int[] t = new int[n]; // индексация файлов
    int[] ta = new int[n-1]; // индексы файлов, участвующих в слиянии
    File[] f = new File[n];
    File f0 = new File(name);

    FileWriter[] in = new FileWriter[n];

    for(i=0;i<n-1;++i) { // создаем и открываем файлы для записи серий
        f[i] = new File( pathname: "T"+(i+1)+".txt");
        f[i].createNewFile();
        in[i]= new FileWriter(f[i]);
    }
    f[n-1] = new File( pathname: "T"+n+".txt"); // создаем файл, куда сделаем первое слияние
    f[n-1].createNewFile();

    for(i=0;i<n-1;++i) {
        a[i] = 1; d[i] = 1;
    }
    l = 1; j = 0; a[n-1] = 0; d[n-1] = 0;

    BufferedReader reader = Files.newBufferedReader(f0.toPath(), StandardCharsets.UTF_8);
    String firstX; // текущее читаемое значение из начального файла
    String[] firstY=new String[n]; // текущее читаемое значение из файла распределения

    firstX = reader.readLine();
    do {
        select(a,d); // распределяем числа Фибоначчи по уровням
        do { // запись серии в файл с индексом j; можно было бы выделить в функцию writeRuns()
            firstY[j]=firstX;
            in[j].write( str: firstX+'\n');
            firstX = reader.readLine();
            if(firstX!=null) {
                eor = Integer.parseInt(firstX) < Integer.parseInt(firstY[j]);
            }
```

```
            --eof;
        } while (!eor && eof!=0);
    } while (eof!=0 && j!=n-2); // если серий больше чем файлов
    j=0;
    while (eof!=0) { // записываем в файлы оставшиеся серии изначального файла
        select(a,d);
        // проверяем, получится ли при дозаписи цельная серия
        if(Integer.parseInt(firstY[j]) <= Integer.parseInt(firstX)) {
            eor=false;
            do { // writeRuns()
                firstY[j]=firstX;
                in[j].write( str: firstX+'\n');
                firstX = reader.readLine();
                if(firstX!=null) {
                    if (Integer.parseInt(firstX) < Integer.parseInt(firstY[j])) eor = true;
                }
                --eof;
            } while (!eor && eof!=0);
            if(eof==0) { // если конец файла, то добавляем пустую взамен той, которая слилась
                ++d[j];
            }
            else { // если нет, то просто пишем дальше
                eor = false;
                do { // writeRuns()
                    firstY[j]=firstX;
                    in[j].write( str: firstX+'\n');
                    firstX = reader.readLine();
                    if(firstX!=null) {
                        if (Integer.parseInt(firstX) < Integer.parseInt(firstY[j])) eor = true;
                    }
                    --eof;
                } while (!eor && eof!=0);
            }
        }
        else { // записываем новую серию
            eor = false;
            do { // writeRuns()
                firstY[j]=firstX;
                in[j].write( str: firstX+'\n');
                firstX = reader.readLine();
                if(firstX!=null) {
                    if (Integer.parseInt(firstX) < Integer.parseInt(firstY[j])) eor = true;
                }
                --eof;
            } while (!eor && eof!=0);
        }
    }
    reader.close();

    BufferedReader[] readers = new BufferedReader[n]; // открываем файлы на чтение
    for(i=0;i<n-1;++i) {
        in[i].close();
        t[i] = i;
        readers[i] = Files.newBufferedReader(f[i].toPath(), StandardCharsets.UTF_8);
```

```
    }
    t[n-1] = n-1; // выходной файл после прохода уровня

    for(i=0;i<n-1;++i) {
        firstY[i]=readers[i].readLine();
    }

    do { // сортируем слиянием
        in[t[n-1]]= new FileWriter(f[t[n-1]]);
        z=a[n-2]; d[n-1]=0;
        do { // z - число сливаемых из всех файлов серий
            k = 0;
            for (i = 0; i < n - 1; ++i) {
                if (d[i] > 0) {
                    --d[i]; // слияние пустой серии
                } else {
                    ta[k] = t[i]; ++k; // файлы, где еще есть серии
                }
            }
            if (k == 0) {
                ++d[n-1]; // запись пустой серии в выходной файл
            }
            else { // слияние реальной серии
                do {
                    i = 0; mx = 0; min = Integer.parseInt(firstY[ta[0]]);
                    while (i < k-1) { // ищем минимальный элемент
                        ++i; x = Integer.parseInt(firstY[ta[i]]);
                        if (x < min) {
                            min = x; mx = i;
                        }
                    }
                    in[t[n-1]].write( str: min+"\n");
                    firstY[ta[mx]]= readers[ta[mx]].readLine();
                    if(firstY[ta[mx]]!=null) {
                        if (min > Integer.parseInt(firstY[ta[mx]])) {
                            eor = true;
                        }
                    }
                    else {
                        firstY[ta[mx]]= "2147483647";
                        eor=true;
                    }
                    if (eor) {
                        ta[mx]=ta[k-1];
                        --k;
                        eor=false;
                    }
                } while (k > 0);
            }
            z;
        }while (z>0);
        in[t[n-1]].close();
        // новый входной файл
        readers[t[n-1]] = Files.newBufferedReader(f[t[n-1]].toPath(), StandardCharsets.UTF_8);
```

```
            firstY[t[n-1]] = readers[t[n-1]].readLine();
            tmpT = t[n-1]; tmpD = d[n-1]; z = a[n-2];
            for (i=n-1;i>0;--i) { // обновление чисел Фибоначчи
                t[i]=t[i-1];
                d[i]=d[i-1];
                a[i]=a[i-1]-z;
            }
            // новый выходной файл
            t[0] = tmpT;
            d[0] = tmpD; a[0] = z;
            --l;
        } while (l!=0);
        in[t[0]].close();
        return (t[0]+1);
    }

    1 usage
    public static int polyPhaseSort_M(long eof, String name, String system, long size) throws IOException {
        int i, z, k, x, mx, min, tmpT, tmpD, lastInRun;
        long runLen; // количество чисел в серии
        boolean eor = false;  // конец ли серии (end of run)
        int[] a = new int[n]; // идеальное число серий
        int[] d = new int[n]; // число пустых серий
        int[] t = new int[n]; // индексация файлов
        int[] ta = new int[n-1]; // индексы файлов, участвующих в слиянии
        File[] f = new File[n];
        File f0 = new File(name);

        FileWriter[] in = new FileWriter[n];

        for(i=0; i<n-1; ++i) { // создаем и открываем файлы для записи серий
            f[i] = new File( pathname: "T"+(i+1)+".txt");
            f[i].createNewFile();
            in[i] = new FileWriter(f[i]);
        }
        f[n-1] = new File( pathname: "T"+n+".txt"); // создаем файл, куда сделаем первое слияние
        f[n-1].createNewFile();

        for(i=0; i<n-1; ++i) {
            a[i] = 1; d[i] = 1;
        }
        l = 1; j = 0;
        a[n-1] = 0; d[n-1] = 0;

        runLen = (long) (2.25*eof/size);
        switch (system) {
            case "B":
                runLen *= 1024;
            case "KB":
                runLen *= 1024;
            case "MB":
                runLen *= 1024;
        }
```

```
370     int runsCount = (int) (eof/ runLen);  // количество серий длиной 2,25 GB
371     int lastRunLen = (int) (eof% runLen); // остаток
372
373     if(runsCount == 0) {
374         runLen = lastRunLen;
375         lastRunLen = 0;
376         runsCount = 1;
377     }
378
379     int[] runs = new int[(int) runLen]; // для записи серии
380     int[] lastRun = new int[lastRunLen]; // для записи последней серии
381     StringBuilder s = new StringBuilder();
382     BufferedReader reader = Files.newBufferedReader(f0.toPath(), StandardCharsets.UTF_8);
383     do {
384         select(a,d); // распределяем числа Фибоначчи по уровням
385         for(i=0; i<runLen; ++i) {
386             runs[i] = Integer.parseInt(reader.readLine());
387         }
388         Arrays.parallelSort(runs);
389         for(i=0; i<runLen; ++i) {
390             s.append(runs[i]).append('\n');
391             if(i%1024==0) {
392                 in[j].write(s.toString());
393                 s.setLength(0);
394             }
395         }
396         in[j].write(s.toString());
397         s.setLength(0);
398         --runsCount;
399     } while (runsCount!=0 && j!=n-2); // если серий больше чем файлов
400     if(runsCount==0 && lastRunLen!=0) {
401         select(a,d); // распределяем числа Фибоначчи по уровням
402         for(i=0; i<lastRunLen; ++i) {
403             lastRun[i] = Integer.parseInt(reader.readLine());
404         }
405         Arrays.parallelSort(lastRun);
406         for(i=0; i<lastRunLen; ++i) {
407             s.append(lastRun[i]).append('\n');
408             if(i%1024==0) {
409                 in[j].write(s.toString());
410                 s.setLength(0);
411             }
412         }
413         in[j].write(s.toString());
414         s.setLength(0);
415         lastRunLen = 0;
416     }
417
418     j=0;
419     while (runsCount!=0) { // записываем в файлы оставшиеся серии изначального файла
420         select(a,d);
421         for(i=0; i<runLen; ++i) {
422             runs[i] = Integer.parseInt(reader.readLine());
423         }
```

```
424             Arrays.parallelSort(runs);
425             for(i=0; i<runLen; ++i) {
426                 s.append(runs[i]).append('\n');
427                 if(i%1024==0) {
428                     in[j].write(s.toString());
429                     s.setLength(0);
430                 }
431             }
432             in[j].write(s.toString());
433             s.setLength(0);
434             --runsCount;
435         }
436         lastInRun = runs[(int) (runLen-1)];
437         if(lastRunLen!=0) {
438             select(a, d); // распределяем числа Фибоначчи по уровням
439             for (i = 0; i < lastRunLen; ++i) {
440                 lastRun[i] = Integer.parseInt(reader.readLine());
441             }
442             Arrays.parallelSort(lastRun);
443             for(i=0; i<lastRunLen; ++i) {
444                 s.append(lastRun[i]).append('\n');
445                 if(i%1024==0) {
446                     in[j].write(s.toString());
447                     s.setLength(0);
448                 }
449             }
450             in[j].write(s.toString());
451             s.setLength(0);
452             // если конец файла, то добавляем пустую взамен той, которая слилась
453             if (lastInRun <= lastRun[0]) ++d[j];
454         }
455         reader.close();
456
457         BufferedReader[] readers = new BufferedReader[n]; // открываем файлы на чтение
458         for(i=0;i<n-1;++i) {
459             in[i].close();
460             t[i] = i;
461             readers[i] = Files.newBufferedReader(f[i].toPath(), StandardCharsets.UTF_8);
462         }
463         t[n-1] = n-1; // выходной файл после прохода уровня
464
465         String[] firstY=new String[n]; // текущее читаемое значение из файла распределения
466         for(i=0;i<n-1;++i) {
467             firstY[i]=readers[i].readLine();
468         }
469
470         do { // сортируем слиянием
471             in[t[n-1]]= new FileWriter(f[t[n-1]]);
472             z=a[n-2]; d[n-1]=0;
473             do { // z - число сливаемых из всех файлов серий
474                 k = 0;
475                 for (i = 0; i < n - 1; ++i) {
476                     if (d[i] > 0) {
477                         --d[i]; // слияние пустой серии
```

```
478                 } else {
479                     ta[k] = t[i]; // файлы, где еще есть серии
480                     ++k;
481                 }
482             }
483             if (k == 0) {
484                 ++d[n-1]; // запись пустой серии в выходной файл
485             }
486             else { // слияние реальной серии
487                 do {
488                     i = 0; mx = 0; min = Integer.parseInt(firstY[ta[0]]);
489                     while (i < k-1) { // ищем минимальный элемент
490                         ++i; x = Integer.parseInt(firstY[ta[i]]);
491                         if (x < min) {
492                             min = x; mx = i;
493                         }
494                     }
495                     in[t[n-1]].write( str: min+"\n");
496                     firstY[ta[mx]] = readers[ta[mx]].readLine();
497                     if(firstY[ta[mx]]!=null) {
498                         if (min > Integer.parseInt(firstY[ta[mx]])) {
499                             eor = true;
500                         }
501                     }
502                     else {
503                         firstY[ta[mx]] = "2147483647";
504                         eor = true;
505                     }
506                     if (eor) {
507                         --k;
508                         ta[mx] = ta[k];
509                         eor = false;
510                     }
511                 } while (k > 0);
512             }
513             --z;
514         }while (z > 0);
515         in[t[n-1]].close();
516         // новый входной файл
517         readers[t[n-1]] = Files.newBufferedReader(f[t[n-1]].toPath(), StandardCharsets.UTF_8);
518         firstY[t[n-1]] = readers[t[n-1]].readLine();
519         tmpT =t[n-1];tmpD =d[n-1];
520         z=a[n-2];
521         for (i=n-1;i>0;--i) { // обновление чисел Фибоначчи
522             t[i]=t[i-1];
523             d[i]=d[i-1];
524             a[i]=a[i-1]-z;
525         }
526         // новый выходной файл
527         t[0]= tmpT;
528         d[0]= tmpD; a[0]=z;
529         --l;
530     } while (l!=0);
531     in[t[0]].close();
```

```
532         return (t[0]+1);                                                    ⚠ 12
533     }
534
            1 usage
535     public static void test(String initial, String sorted) throws IOException {
536         BufferedReader reader = Files.newBufferedReader(Path.of(sorted), StandardCharsets.UTF_8);
537         String first, second;
538         boolean eor = true;
539         long c = 1;
540         first = reader.readLine();
541         while ((second = reader.readLine())!=null) {
542             if(Integer.parseInt(first)>Integer.parseInt(second)) {
543                 eor=false;
544             }
545             first=second;
546             ++c;
547         }
548         reader.close();
549
550         long cInit = 0;
551         reader = Files.newBufferedReader(Path.of(initial), StandardCharsets.UTF_8);
552         while ((first = reader.readLine())!=null) {
553             ++cInit;
554         }
555         if(c==cInit && eor) System.out.println("File sorted correctly!");
556     }
557 }
```

У наведеному вихідному коді функція inputSystem відповідає за валідацію вводу розмірності файлу, inputSize – за валідацію вводу розміру файлу, inputN – за валідацію вводу кількості файлів, що потрібні для сортування, checkYN – для визначення потрібно використовувати модифікований алгоритм чи базовий. Функція generateFile генерує початковий файл для сортування. Функція select потрібна для визначення, куди записувати серію. Функції polyPhaseSort та polyPhaseSort_M – це функції сортування базовим та модифікованим алгоритмом відповідно. Функція test викликається, щоб перевірити чи правильно був відсортований початковий файл.

3.2.2 Результати тестування

На рисунку 1 показано приклад роботи базового алгоритму на файлі, розмірністю 10 мегабайт. Файл було відсортовано п’ятифазним сортуванням за 2,6с.

```
Hello world!
Enter the name of the initial file: f10mb
In which unit of information the file must be created? (Kb/Mb/Gb): mb
Enter the file size in MB: 10
The file named "f10mb.txt" with a size of 10MB was successfully generated within 73.0ms!

How many files should be used for sorting? N = 5
Do you want to use modified version of the algorithm? (Y/N): n
The result of the basic algorithm is in the file T2.txt and completed within 2643.0ms!
File sorted correctly!

Process finished with exit code 0
```

Рисунок 1 – Базовий алгоритм на 10Мб

```
Hello world!
Enter the name of the initial file: f1
In which unit of information the file must be created? (Kb/Mb/Gb): gb
Enter the file size in GB: 1
The file named "f1.txt" with a size of 1GB was successfully generated within 5188.0ms!

How many files should be used for sorting? N = 5
Do you want to use modified version of the algorithm? (Y/N): n
The result of the basic algorithm is in the file T5.txt and completed within 286990.0ms!
File sorted correctly!

Process finished with exit code 0
```

Рисунок 2 – Базовий алгоритм на 1Гб

На рисунку 2 показано приклад роботи базового алгоритму на файлі, розмірністю 1 гігабайт. Файл було відсортовано п’ятифазним сортуванням за 4,8хв.

Порівняємо час сортування з модифікованим алгоритмом для таких самих розмірностей файлів.

На рисунку 3 показано приклад роботи модифікованого алгоритму на файлі, розмірністю 10 мегабайт. Файл було відсортовано п’ятифазним сортуванням за 0,4с.

```
Hello world!
Enter the name of the initial file: f10mb
In which unit of information the file must be created? (Kb/Mb/Gb): mb
Enter the file size in MB: 10
The file named "f10mb.txt" with a size of 10MB was successfully generated within 67.0ms!

How many files should be used for sorting? N = 5
Do you want to use modified version of the algorithm? (Y/N): y
The result of the modified algorithm is in the file T5.txt and completed within 432.0ms!
File sorted correctly!

Process finished with exit code 0
```

Рисунок 3 – Модифікований алгоритм на 10Мб

```
Hello world!
Enter the name of the initial file: f1
In which unit of information the file must be created? (Kb/Mb/Gb): gb
Enter the file size in GB: 1
The file named "f1.txt" with a size of 1GB was successfully generated within 4883.0ms!

How many files should be used for sorting? N = 5
Do you want to use modified version of the algorithm? (Y/N): y
The result of the modified algorithm is in the file T5.txt and completed within 22470.0ms!
File sorted correctly!

Process finished with exit code 0
```

Рисунок 4 – Модифікований алгоритм на 1Гб

На рисунку 4 показано приклад роботи модифікованого алгоритму на файлі, розмірністю 1 гігабайт. Файл було відсортовано п’ятифазним сортуванням за 22,5с.

Тепер проведемо випробування для модифікованого алгоритму на файлі, розмірність якого у два рази більша за операційну пам’ять комп’ютера.

```
Hello world!
Enter the name of the initial file: f32
In which unit of information the file must be created? (Kb/Mb/Gb): gb
Enter the file size in GB: 32
The file named "f32.txt" with a size of 32GB was successfully generated within 210736.0ms!

How many files should be used for sorting? N = 5
Do you want to use modified version of the algorithm? (Y/N): y
The result of the modified algorithm is in the file T2.txt and completed within 2871560.0ms!
File sorted correctly!

Process finished with exit code 0
```

Рисунок 5 – Модифікований алгоритм на 32Гб

На рисунку 5 наведено результат сортування 32-х гігібайтного файлу за допомогою п’ятифазного модифікованого алгоритму.

Файл було відсортовано за 47,9хв. Тобто **зі швидкістю 1Гб за 1,5хв**.

3.3 Аналіз алгоритму

Порівняємо роботу модифікованого та базового алгоритмів на одному файлі (Рисунок 6).

```
Hello world!
Enter the name of the initial file: f32
In which unit of information to create a file? (Kb/Mb/Gb): gb
Enter file size in GB: 32
A file named "f32.txt" with a size of 32GB was successfully generated in 207682.0ms!

How many files to use for sorting? N = 5
The result of the modified algorithm is in the file T2.txt and completed in 2585924.0ms!
File sorted correctly!
The result of the basic algorithm is in the file T4.txt and completed in 1.6653683E7ms!
File sorted correctly!

Process finished with exit code 0
```

Рисунок 6 – Обидва алгоритми на 32Гб

Модифікований алгоритм закінчив роботу за 43хв, а базовий – за 4,6год. Тобто виграш у часі за допомогою модифікованого алгоритму суттєвий – майже у **6,5 разів**!

Щоб повноцінно порівняти базову та модифіковану версію, проведемо ряд досліджень.

На таблиці 1 зображені всі проведені випробування для п’ятифазного алгоритму. Для кожного файлу певної розмірності було проведено по три тести та підраховано середнє значення. M (Modified) – позначення для модифікованої версії, а N (Non-modified) – для базової.

Count	Size	Time, ms		Speed, min/gb	
		M	N	M	N
1	10mb	431	2539	0,735573	4,333227
2	10mb	444	2773	0,75776	4,732587
3	10mb	427	2724	0,728747	4,64896
Average	10mb	434	2678,66667	0,740693	4,571591
1	100mb	2884	26079	0,492203	4,450816
2	100mb	2842	29302	0,485035	5,000875
3	100mb	2874	27834	0,490496	4,750336
Average	100mb	2866,667	27738,3333	0,489244	4,734009
1	500mb	12454	131852	0,425097	4,500548
2	500mb	13756	132059	0,469538	4,507614
3	500mb	13716	127952	0,468173	4,367428
Average	500mb	13308,67	130621	0,454269	4,45853
1	1gb	27255	279207	0,45425	4,65345
2	1gb	25739	325508	0,428983	5,425133
3	1gb	24681	302642	0,41135	5,044033
Average	1gb	25891,67	302452,333	0,431528	5,040872
1	3gb	81539	926597	0,452994	5,147761
2	3gb	81869	910478	0,454828	5,058211
3	3gb	75935	896822	0,421861	4,982344
Average	3gb	79781	911299	0,443228	5,062772
1	7gb	209063	2226851	0,497769	5,302026
2	7gb	218626	2253295	0,520538	5,364988
3	7gb	203313	2160460	0,484079	5,143952
Average	7gb	210334	2213535,33	0,500795	5,270322
1	10gb	377036	3418162	0,628393	5,696937
2	10gb	351571	3358130	0,585952	5,596883
3	10gb	457248	3324789	0,76208	5,541315
Average	10gb	395285	3367027	0,658808	5,611712
1	16gb	1011917	8054689	1,05408	8,390301
2	16gb	958988	7462681	0,998946	7,773626
3	16gb	945921	7837033	0,985334	8,163576
Average	16gb	972275,3	7784801	1,012787	8,109168
1	25gb	1948455	12911929	1,29897	8,607953
2	25gb	1882979	12899079	1,255319	8,599386
3	25gb	1997719	13026086	1,331813	8,684057
Average	25gb	1943051	12945698	1,295367	8,630465
1	32gb	2585924	16653683	1,346835	8,673793
2	32gb	2792544	17011364	1,45445	8,860085
3	32gb	2759851	16857943	1,437422	8,780179
Average	32gb	2712773	16840996,7	1,412903	8,771352

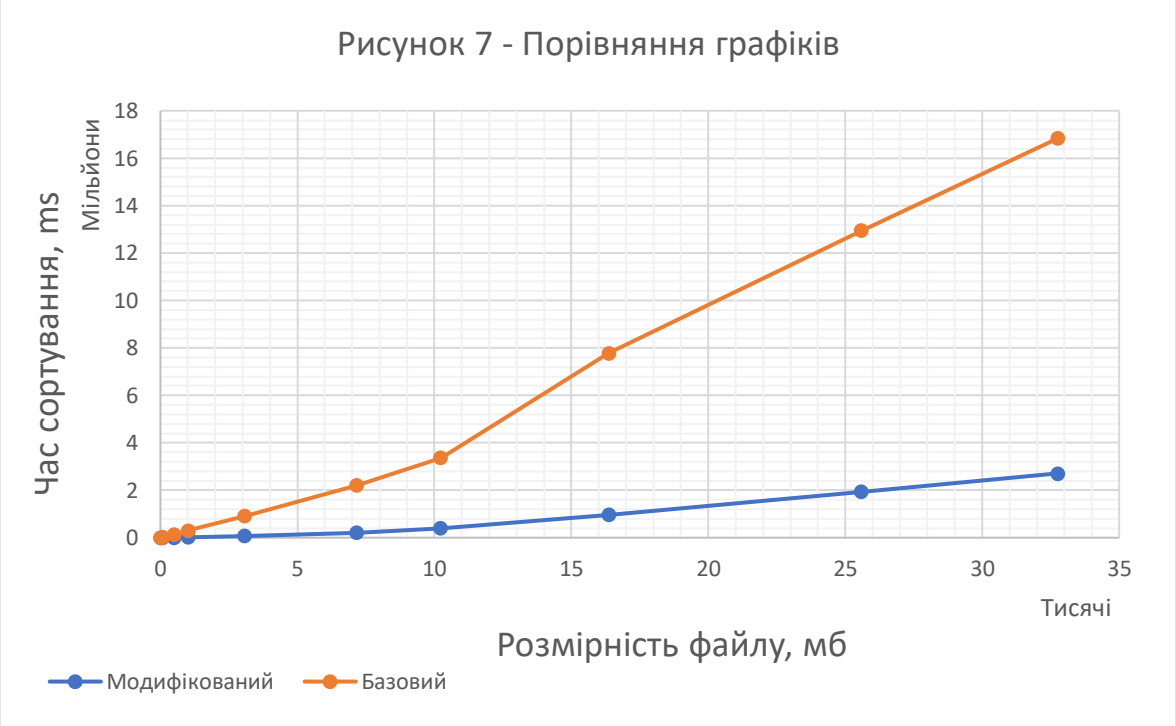
Таблиця 1 – Випробування алгоритмів

З таблиці 1 винесемо отримані середні значення у таблицю 2.

Size, mb	Time, ms		Efficiency (N/M)
	M	N	
10	434	2679	6,172811
100	2867	27738	9,674922
500	13309	130621	9,814486
1024	25892	302452	11,68129
3072	79781	911299	11,42251
7168	210334	2213535	10,5239
10240	395285	3367027	8,517973
16384	972275	7784801	8,006789
25600	1943051	12945698	6,662562
32768	2712773	16840997	6,208038

Таблиця 2 – Середні значення

Тепер побудуємо графік для порівняння ефективності (Рисунок 7).



З графіка видно, що модифікована версія набагато ефективніша за базову. Також можемо помітити, що відбувається значний приріст часу виконання при виході розмірності файлу за межі об'єму операційної пам'яті, хоча це може бути похибка.

Рисунок 7.1 - Модифікована версія для n=5

Рисунок 7.2 - Базова версія для n=5

З рисунків 7.1 та 7.2 можемо помітити, що графіки залежності часу виконання від розмірності файлу нагадують складність O(nlogn). Хоча оцінити складність достовірно неможливо, адже на пристрої завжди виконуються сторонні процеси.

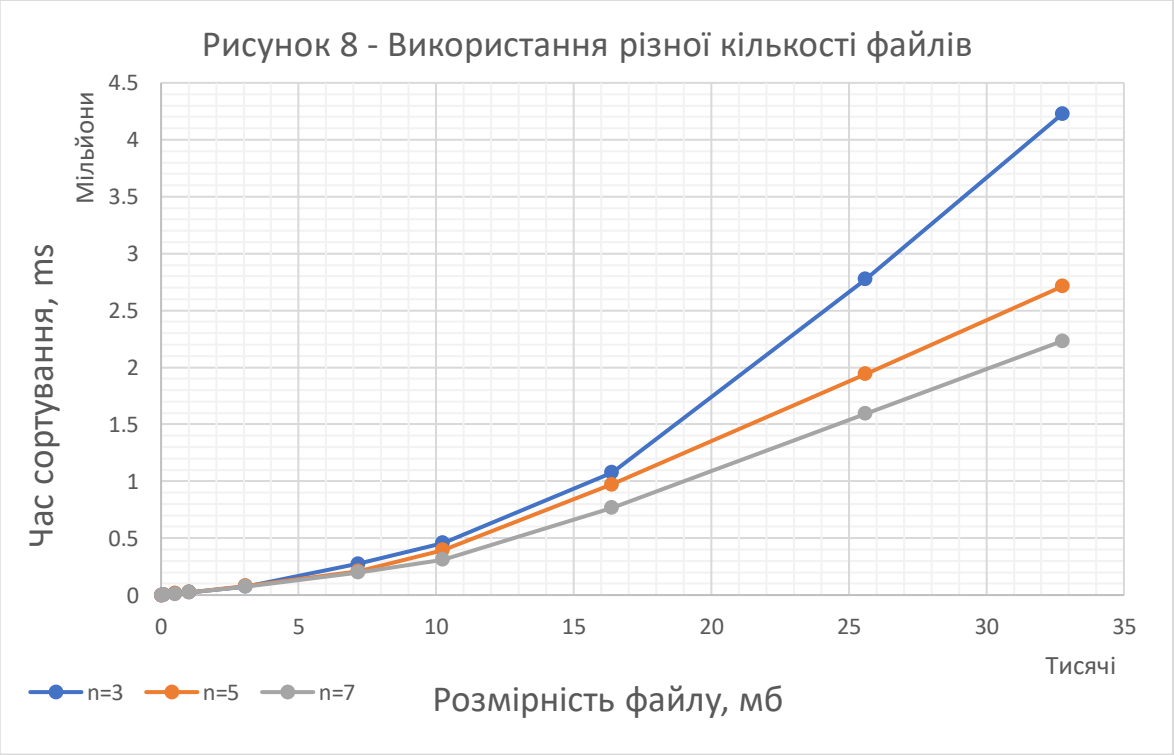
Порівняємо для різної кількості допоміжних файлів графіки залежності часу сортування від розмірності файлу. У таблиці 3 наведено дані, отримані експериментально для n=3, n=5 та n=7. Було використано модифікований алгоритм.

Size, mb	Time, ms		
	N = 3	**N = 5**	**N = 7**
10	452	434	444
100	2664	2867	2738
500	12092	13309	11800
1024	24947	25892	24824
3072	74592	79781	76054
7168	274368	210334	198191
10240	458515	395285	310646
16384	1077668	972275	766189
25600	2775780	1943051	1594771
32768	4228479	2712773	2232601

Таблиця 3 – Порівняння різної кількості допоміжних файлів

Всі дані для таблиці 3 були отримані як середнє значення з трьох випробувань. Проілюструємо ці дані на графіку (Рисунок 8).

Рисунок 8 - Використання різної кількості файлів



Як видно з графіку, чим більша кількість допоміжних файлів, тим швидше працює алгоритм. Це може бути пов’язано з кількістю серій у файлах, бо на малих розмірностях показники приблизно однакові. Чим менше серій у файлі, тим менше буде рівнів для роботи алгоритму (ідеальний випадок, коли в кожному файлі лише по одній серії, бо це перший рівень розподілу Фібоначчі), тому і витраченого часу буде, відповідно, менше. Проте якщо порівняти приріст ефективності між алгоритмами з використанням 3-5 файлів та 5-7 файлів, то можна помітити, що ефективність збільшується “повільніше”. Тому можна зробити припущення, що при використанні більше п’яти допоміжних файлів, ефективність буде зростати не так помітно.

ВИСНОВОК

У першій лабораторній роботі ми спроектували алгоритм багатофазного сортування та його модифіковану версію. Дізналися основні переваги та недоліки багатофазного сортування (порівняно з збалансованим багатошляховим):

- допускає використання довільного числа N допоміжних файлів;
- при злитті використовується (N-1) допоміжний файл, при цьому результат записується у вільний (N-тий) файл;
- злиття проводиться поки всі файли не порожні (стратегія зливати до спустошення - merge-until-empty);
- потребує нерівномірного розподілу серій по допоміжним файлам.

В основі сортування послідовності є багатофазним злиттям. лежить розподіл початкових серій відповідно до числа Фібоначчі.

Дослідили часові характеристики базової версії алгоритму на прикладі п’ятифазного сортування. Середній розрахунок вийшов **8,8хв на 1Гб** при сортуванні файлу, розмір якого складає 32Гб, що у двічі більше за операційну пам’ять.

Щоб покращити ефективність алгоритму, ми його модифікували за допомогою використання операційної пам’яті комп’ютера. Так як звертання-запис до диску це дуже довгі операції, ми почали записувати у файли серії не з кількох елементів, а одразу попередньо відсортувавши їх у операційній пам’яті. Серії було обрано обсягом 2,25Гб, що визначено експериментально. Такий підхід покращив швидкодію алгоритму **у 6,5 разів**! Щоб провести порівняння базового та модифікованого алгоритмів, провели купу випробувань та побудували графіки, спираючись на отримані значення. Та отримали задовільний результат - середній розрахунок вийшов **1,4хв на 1Гб** при сортуванні файлу, розмір якого складає 32Гб, що у двічі більше за операційну пам’ять.

Також дослідили при якій кількості допоміжних файлів алгоритм працює ефективніше. І дійшли до висновку, що чим більше допоміжних файлів ми використовуємо, тим швидше працює алгоритм, але при кількості більше п'яти, швидкодія стає все менш помітною.