

# **Проектування алгоритмів**

---

**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)

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

**Київ 2022**

### ЗМІСТ

1. МЕТА ЛАБОРАТОРНОЇ РОБОТИ	3
2. ЗАВДАННЯ	3
3. ВИКОНАННЯ	4
3.1. ПСЕВДОКОД АЛГОРИТМУ	4
3.2. ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ	8
3.2.1. <i>Вихідний код</i>	8
3.2.2. <i>Результати тестування</i>	19
3.3. АНАЛІЗ АЛГОРИТМУ	22
ВИСНОВОК	28

### 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  $j \neq n-1$

##### 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 i 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 i 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

## Проектування алгоритмів

---

**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  $i$  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**  $level \neq 0$

**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  $i$  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

```
1  import java.io.BufferedReader;
2      import java.io.File;
3      import java.io.FileWriter;
4      import java.io.IOException;
5      import java.nio.charset.StandardCharsets;
6      import java.nio.file.Files;
7      import java.nio.file.Path;
8      import java.util.Arrays;
9      import java.util.Random;
10     import java.util.Scanner;
11
12     import static java.lang.Character.toUpperCase;
13     import static java.lang.Integer.MAX_VALUE;
14
15     public class Main {
16         74 usages
17         public static int n, j, l;
18
19     public static void main(String[] args) {
20         System.out.println("Hello world!");
21         Scanner scan = new Scanner(System.in);
22
23         System.out.print("Enter the name of the initial file: ");
24         String name = scan.nextLine() + ".txt";
25
26         String system = inputSystem();
27
28         long size = inputSize(system);
29
30         double time = System.currentTimeMillis();
31         double resTime;
32         try {
33             long count = generateFile(name, system, size);
34             resTime = System.currentTimeMillis() - time;
35
36             System.out.println("The file named \"" + name + "\" with a size of " + size + system +
37                 " was successfully generated within " + resTime + "ms!");
38
39             System.out.println();
40             n = inputN();
41
42             System.out.print("Do you want to use modified version of the algorithm? (Y/N): ");
43             boolean yn = checkYN();
44
45             int res;
46             time = System.currentTimeMillis();
47             if(yn) {
48                 res = polyPhaseSort_M(count, name, system, size);
49                 resTime = System.currentTimeMillis() - time;
50                 System.out.println("The result of the modified algorithm is in the file T"+res+
51                     ".txt and completed within "+resTime+"ms!");
52             }
53             else {
54                 res = polyPhaseSort(count, name);
```



## Проектування алгоритмів

```
54         resTime = System.currentTimeMillis() - time;
55         System.out.println("The result of the basic algorithm is in the file T"+res+
56             ".txt and completed within "+resTime+"ms!");
57     }
58     test(name, sorted: "T"+res+".txt");
59 } catch (Exception e) {
60     System.err.println(e);
61 }
62 }
63
64 1 usage
65 private static String inputSystem() {
66     Scanner sc = new Scanner(System.in);
67     String system;
68     while (true) {
69         try {
70             System.out.print("In which unit of information the file must be created? (Kb/Mb/Gb): ");
71             system = sc.nextLine().toUpperCase();
72             if(system.equals("GB") || system.equals("MB") || system.equals("KB") || system.equals("B")) break;
73             else System.out.println("Try again!");
74         } catch (Exception e) {
75             System.out.println("Try again!");
76             sc.next();
77         }
78     }
79     return system;
80 }
81
82 1 usage
83 private static long inputSize(String system) {
84     Scanner sc = new Scanner(System.in);
85     long size;
86     while (true) {
87         try {
88             System.out.print("Enter the file size in " + system + ": ");
89             size = sc.nextLong();
90             if(size>0) break;
91             else System.out.println("Size must be positive. Try again!");
92         } catch (Exception e) {
93             System.out.println("Size must be integer. Try again!");
94             sc.next();
95         }
96     }
97     return size;
98 }
99
100 1 usage
101 private static int inputN() {
102     Scanner sc = new Scanner(System.in);
103     int n;
104     System.out.print("How many files should be used for sorting? ");
105     while (true) {
```

## Проектування алгоритмів

```
105     try {
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
119 1 usage
120 private static boolean checkYN() {
121     Scanner scan = new Scanner(System.in);
122     char yn = toUpperCase(scan.nextLine().charAt(0));
123     return (yn=='Y' || yn=='1');
124 }
125
126 1 usage
127 @ private static long generateFile(String name, String system, long size) throws IOException {
128     File file = new File(name);
129     file.createNewFile();
130     FileWriter in = new FileWriter(file);
131     Random rand = new Random();
132     StringBuilder s = new StringBuilder();
133     long count = 0;
134     switch (system) {
135         case "GB":
136             size *= 1024;
137         case "MB":
138             size *= 1024;
139         case "KB":
140             size *= 1024;
141     }
142     while (file.length() < size) {
143         for (int i = 0; i < 10240; ++i) {
144             s.append(rand.nextInt(MAX_VALUE));
145             s.append('\n');
146         }
147         in.write(s.toString());
148         s.setLength(0);
149         ++count;
150     }
151     in.close();
152     return count * 10240;
153 }
154
155 6 usages
156 @ public static void select(int[] a, int[] d) { // алгоритм распределения серий
157     int i,a0;
158     if(d[j]<d[j+1]) ++j;
```

## Проектування алгоритмів

```
156     else {
157         if(d[j]==0) {
158             ++l;
159             a0=a[0];
160             for (i=0;i<n-1;++i) {
161                 d[i]=a0+a[i+1]-a[i];
162                 a[i] = a0 + a[i + 1];
163             }
164         }
165         j=0;
166     }
167     --d[j];
168 }
169
170 1 usage
171 public static int polyPhaseSort(long eof, String name) throws IOException {
172     int i, z, k, x, mx, min, tmpI, tmpD;
173     boolean eor = false; // конец ли серии (end of run)
174     int[] a = new int[n]; // идеальное число серий
175     int[] d = new int[n]; // число пустых серий
176     int[] t = new int[n]; // индексация файлов
177     int[] ta = new int[n-1]; // индексы файлов, участвующих в слиянии
178     File[] f = new File[n];
179     File f0 = new File(name);
180
181     FileWriter[] in = new FileWriter[n];
182     for(i=0;i<n-1;++i) { // создаем и открываем файлы для записи серий
183         f[i] = new File( pathname: "T"+(i+1)+".txt");
184         f[i].createNewFile();
185         in[i]= new FileWriter(f[i]);
186     }
187     f[n-1] = new File( pathname: "T"+n+".txt"); // создаем файл, куда сделаем первое слияние
188     f[n-1].createNewFile();
189
190     for(i=0;i<n-1;++i) {
191         a[i] = 1; d[i] = 1;
192     }
193     l = 1; j = 0; a[n-1] = 0; d[n-1] = 0;
194
195     BufferedReader reader = Files.newBufferedReader(f0.toPath(), StandardCharsets.UTF_8);
196     String firstX; // текущее читаемое значение из начального файла
197     String[] firstY=new String[n]; // текущее читаемое значение из файла распределения
198
199     firstX = reader.readLine();
200     do {
201         select(a,d); // распределяем числа Фибоначчи по уровням
202         do { // запись серии в файл с индексом j; можно было бы выделить в функцию writeRuns()
203             firstY[j]=firstX;
204             in[j].write( str: firstX+'\n');
205             firstX = reader.readLine();
206             if(firstX!=null) {
207                 eor = Integer.parseInt(firstX) < Integer.parseInt(firstY[j]);
208             }

```

## Проектування алгоритмів

```
209         --eof;
210     } while (!eor && eof!=0);
211 } while (eof!=0 && j!=n-2); // если серий больше чем файлов
212 j=0;
213 while (eof!=0) { // записываем в файлы оставшиеся серии изначального файла
214     select(a,d);
215     // проверяем, получится ли при дозаписи цельная серия
216     if(Integer.parseInt(firstY[j]) <= Integer.parseInt(firstX)) {
217         eor=false;
218         do { // writeRuns()
219             firstY[j]=firstX;
220             in[j].write( str: firstX+'\n');
221             firstX = reader.readLine();
222             if(firstX!=null) {
223                 if (Integer.parseInt(firstX) < Integer.parseInt(firstY[j])) eor = true;
224             }
225             --eof;
226         } while (!eor && eof!=0);
227         if(eof==0) { // если конец файла, то добавляем пустую взамен той, которая слилась
228             ++d[j];
229         }
230         else { // если нет, то просто пишем дальше
231             eor = false;
232             do { // writeRuns()
233                 firstY[j]=firstX;
234                 in[j].write( str: firstX+'\n');
235                 firstX = reader.readLine();
236                 if(firstX!=null) {
237                     if (Integer.parseInt(firstX) < Integer.parseInt(firstY[j])) eor = true;
238                 }
239                 --eof;
240             } while (!eor && eof!=0);
241         }
242     }
243     else { // записываем новую серию
244         eor = false;
245         do { // writeRuns()
246             firstY[j]=firstX;
247             in[j].write( str: firstX+'\n');
248             firstX = reader.readLine();
249             if(firstX!=null) {
250                 if (Integer.parseInt(firstX) < Integer.parseInt(firstY[j])) eor = true;
251             }
252             --eof;
253         } while (!eor && eof!=0);
254     }
255 }
256 reader.close();
257
258 BufferedReader[] readers = new BufferedReader[n]; // открываем файлы на чтение
259 for(i=0;i<n-1;++i) {
260     in[i].close();
261     t[i] = i;
262     readers[i] = Files.newBufferedReader(f[i].toPath(), StandardCharsets.UTF_8);
```

## Проектування алгоритмів

```
263 }
264 t[n-1] = n-1; // выходной файл после прохода уровня
265
266 for(i=0;i<n-1;++i) {
267     firstY[i]=readers[i].readLine();
268 }
269
270 do { // сортируем слиянием
271     in[t[n-1]]= new FileWriter(f[t[n-1]]);
272     z=a[n-2]; d[n-1]=0;
273     do { // z - число сливаемых из всех файлов серий
274         k = 0;
275         for (i = 0; i < n - 1; ++i) {
276             if (d[i] > 0) {
277                 --d[i]; // слияние пустой серии
278             } else {
279                 ta[k] = t[i]; ++k; // файлы, где еще есть серии
280             }
281         }
282         if (k == 0) {
283             ++d[n-1]; // запись пустой серии в выходной файл
284         }
285         else { // слияние реальной серии
286             do {
287                 i = 0; mx = 0; min = Integer.parseInt(firstY[ta[0]]);
288                 while (i < k-1) { // ищем минимальный элемент
289                     ++i; x = Integer.parseInt(firstY[ta[i]]);
290                     if (x < min) {
291                         min = x; mx = i;
292                     }
293                 }
294                 in[t[n-1]].write( str: min+"\n");
295                 firstY[ta[mx]] = readers[ta[mx]].readLine();
296                 if(firstY[ta[mx]]!=null) {
297                     if (min > Integer.parseInt(firstY[ta[mx]])) {
298                         eor = true;
299                     }
300                 }
301                 else {
302                     firstY[ta[mx]] = "2147483647";
303                     eor=true;
304                 }
305                 if (eor) {
306                     ta[mx]=ta[k-1];
307                     --k;
308                     eor=false;
309                 }
310             } while (k > 0);
311         }
312     } while (z>0);
313     in[t[n-1]].close();
314     // новый входной файл
315     readers[t[n-1]] = Files.newBufferedReader(f[t[n-1]].toPath(), StandardCharsets.UTF_8);
```

## Проектування алгоритмів

```
317         firstV[t[n-1]] = readers[t[n-1]].readLine();
318         tmpI = t[n-1]; tmpD = d[n-1]; z = a[n-2];
319         for (i=n-1; i>0; --i) { // обновление чисел Фибоначчи
320             t[i]=t[i-1];
321             d[i]=d[i-1];
322             a[i]=a[i-1]-z;
323         }
324         // новый выходной файл
325         t[0] = tmpI;
326         d[0] = tmpD; a[0] = z;
327         --l;
328     } while (l!=0);
329     in[t[0]].close();
330     return (t[0]+1);
331 }
332
333 1 usage
334 public static int polyPhaseSort_M(long eof, String name, String system, long size) throws IOException {
335     int i, z, k, x, mx, min, tmpI, tmpD, lastInRun;
336     long runLen; // количество чисел в серии
337     boolean eor = false; // конец ли серии (end of run)
338     int[] a = new int[n]; // идеальное число серий
339     int[] d = new int[n]; // число пустых серий
340     int[] t = new int[n]; // индексация файлов
341     int[] ta = new int[n-1]; // индексы файлов, участвующих в слиянии
342     File[] f = new File[n];
343     File f0 = new File(name);
344     FileWriter[] in = new FileWriter[n];
345
346     for(i=0; i<n-1; ++i) { // создаем и открываем файлы для записи серий
347         f[i] = new File( pathname: "T"+(i+1)+".txt");
348         f[i].createNewFile();
349         in[i] = new FileWriter(f[i]);
350     }
351     f[n-1] = new File( pathname: "T"+n+".txt"); // создаем файл, куда сделаем первое слияние
352     f[n-1].createNewFile();
353
354     for(i=0; i<n-1; ++i) {
355         a[i] = 1; d[i] = 1;
356     }
357     l = 1; j = 0;
358     a[n-1] = 0; d[n-1] = 0;
359
360     runLen = (long) (2.25*eof/size);
361     switch (system) {
362         case "B":
363             runLen *= 1024;
364         case "KB":
365             runLen *= 1024;
366         case "MB":
367             runLen *= 1024;
368     }
369 }
```

## Проектування алгоритмів

```
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     }
```

## Проектування алгоритмів

```
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
```



## Проектування алгоритмів

```
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(firstV[ta[0]]);
489         while (i < k-1) { // ищем минимальный элемент
490             ++i; x = Integer.parseInt(firstV[ta[i]]);
491             if (x < min) {
492                 min = x; mx = i;
493             }
494         }
495         in[t[n-1]].write(str: min+"\n");
496         firstV[ta[mx]] = readers[ta[mx]].readLine();
497         if(firstV[ta[mx]]!=null) {
498             if (min > Integer.parseInt(firstV[ta[mx]])) {
499                 eor = true;
500             }
501         }
502         else {
503             firstV[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 firstV[t[n-1]] = readers[t[n-1]].readLine();
519 tmpI = 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] = tmpI;
528 d[0] = tmpD; a[0] = z;
529 --l;
530 } while (l!=0);
531 in[t[0]].close();
```

## Проектування алгоритмів

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

## Проектування алгоритмів

---

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.

## Проектування алгоритмів

---

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. М (Modified) – позначення для модифікованої версії, а N (Non-modified) – для базової.

## Проектування алгоритмів

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

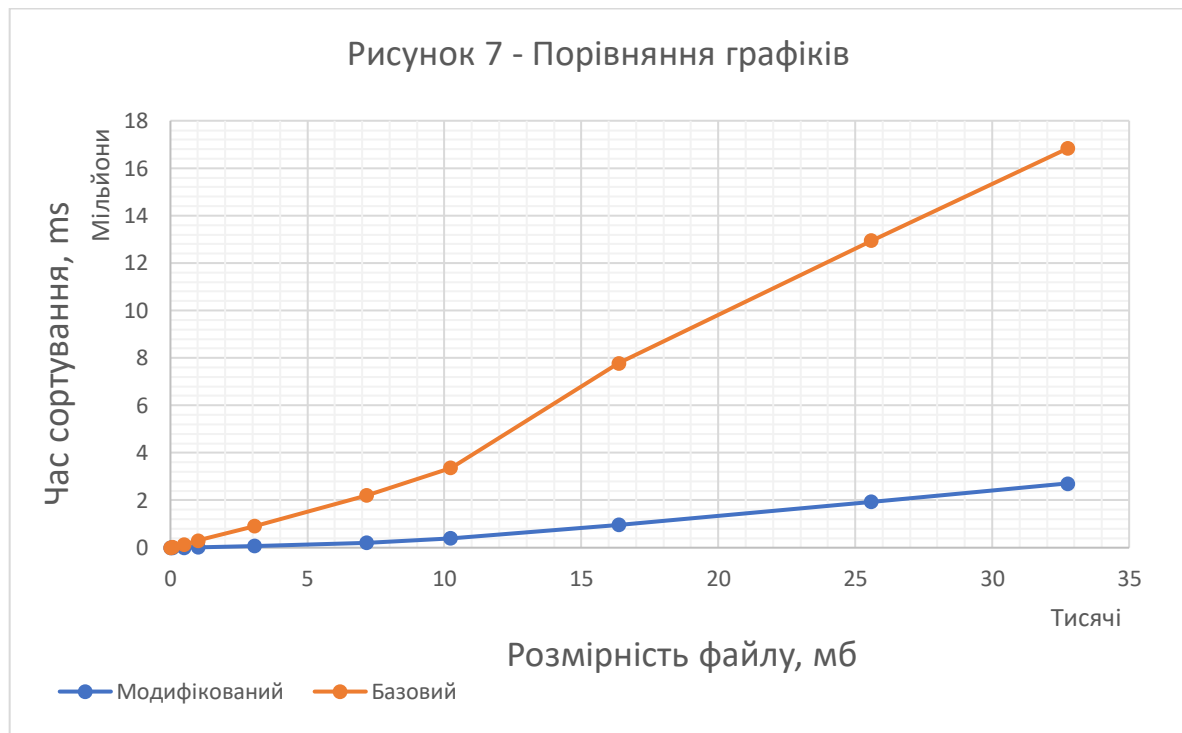
## Проектування алгоритмів

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

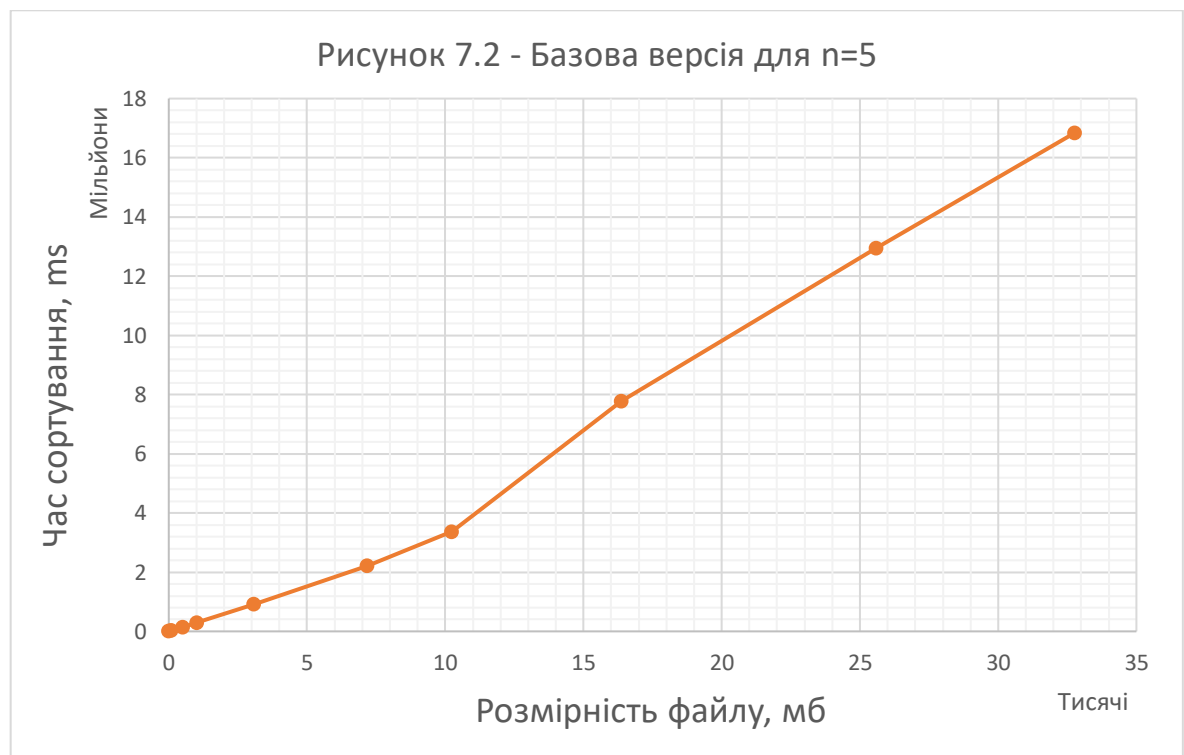
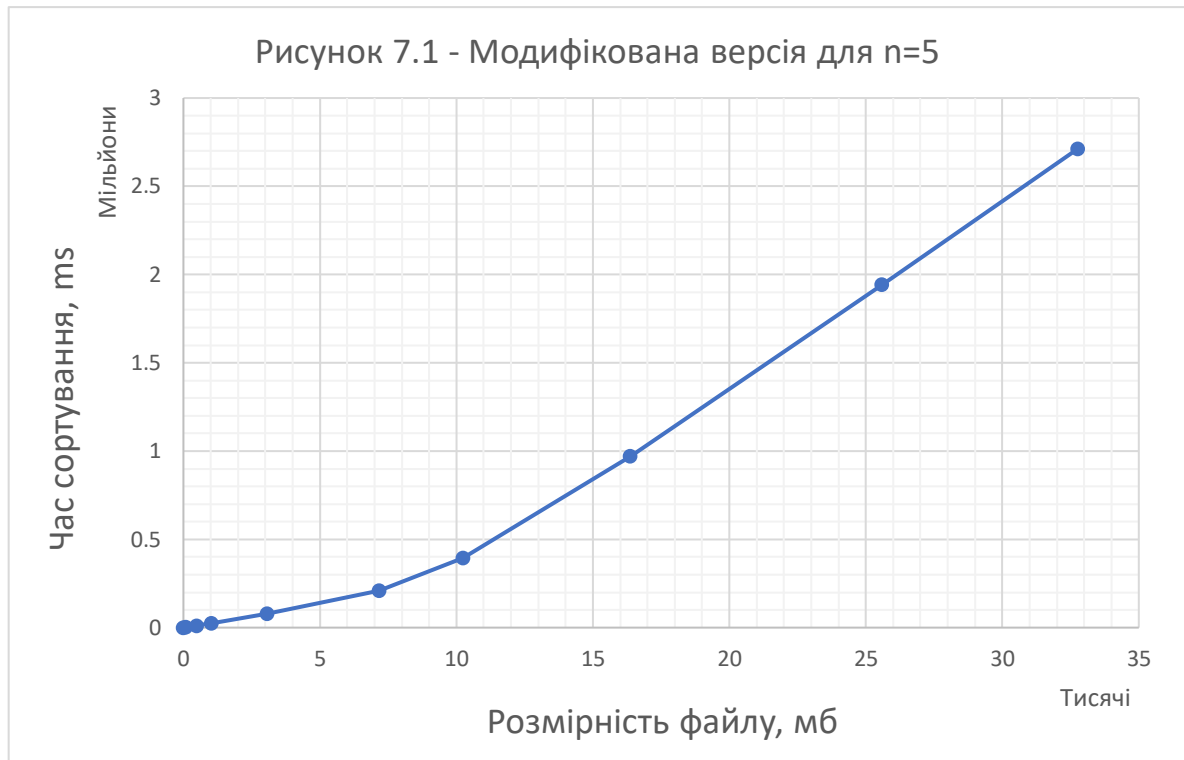
Таблиця 2 – Середні значення

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.





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

## Проектування алгоритмів

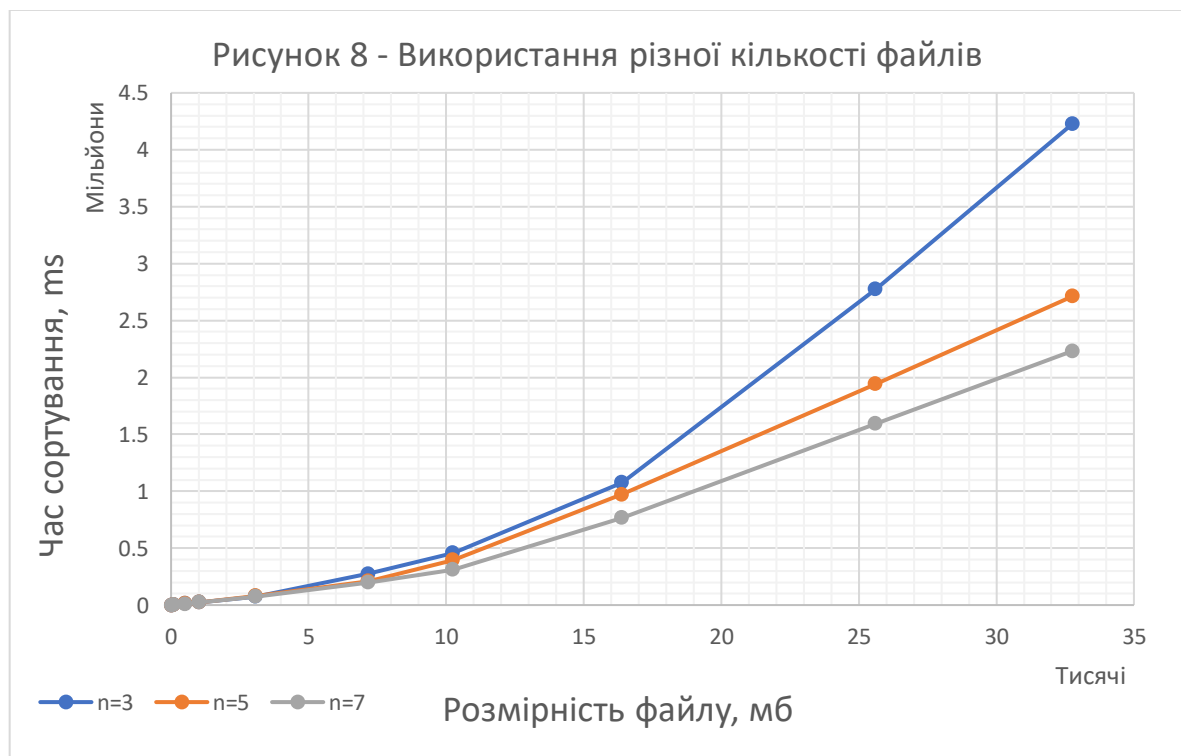
Порівняємо для різної кількості допоміжних файлів графіки залежності часу сортування від розмірності файлу. У таблиці 3 наведено дані, отримані експериментально для  $n=3$ ,  $n=5$  та  $n=7$ . Було використано модифікований алгоритм.

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	2685118	1943051	1594771
32768	4335147	2712773	2306099

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).



## Проектування алгоритмів

---

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  $(N\text{-тий})$  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.