**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)** ІП-12 Бобрик Максим Геннадійович 20.10.2022

(шифр, прізвище, ім’я, по батькові)

**Перевірив**  Сопов Олексій Олександрович 21.10.2022

(прізвище, ім’я, по батькові)

Київ 2022

**ЗМІСТ**

1. **МЕТА ЛАБОРАТОРНОЇ РОБОТИ 9**
2. **ЗАВДАННЯ 9**
3. **ВИКОНАННЯ 9**
   1. Псевдокод алгоритму **9**
   2. Програмна реалізація алгоритму **9**
      1. Вихідний код **9 ВИСНОВОК 9 КРИТЕРІЇ ОЦІНЮВАННЯ 9**
         1. МЕТА ЛАБОРАТОРНОЇ РОБОТИ

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

* + - 1. 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 |

* + - 1. ВИКОНАННЯ

3.1 Псевдокод алгоритму

Створення файлів

binf = **ВІДКРИТИ** ‘numbers.bin’ у режимі зчитування # Файл, що буде сортуватись

input\_files = [] # Масив файлів

input\_file\_runs = []

output\_file\_runs = []

output\_file = **ВІДКРИТИ** ‘file0.bin’ у режимі запису

**ПОВТОРИТИ ВІД** i=1 **ДО** file\_count:

input\_files.append(‘file**{i}**.bin’ у режимі запису)

input\_fileRuns.append([])

**КІНЕЦЬ ПОВТОРИТИ**

Розрахунок розподілу

current\_size = input\_files.length

distributions = [1] \* current\_size

target\_size = binf.size # Кількість чиселу у файлі

**ПОКИ** current\_size < target\_size **ВИКОНАТИ**:

max\_val = -1

max\_i = 0

**ПОВТОРИТИ ВІД** i=0 **ДО** distributions.length**:**

**ЯКЩО** distributions[i] > maxVal **ВИКОНАТИ**:

max\_val = distributions[i]

max\_i = i

**КІНЕЦЬ ПОВТОРИТИ**

**ПОВТОРИТИ ВІД** i=0 **ДО** distributions.length:

**ЯКЩО** i! = maxI **ВИКОНАТИ:**

distributions[i] += max\_val

current\_size += max\_val

**КІНЕЦЬ ПОВТОРИТИ**

**КІНЕЦЬ ПОКИ**

total\_runs = current\_size

Розбиття вхідного файлу у відсортовані серії

write\_file\_pos = [0] \* input\_files.length

write\_to\_files = [0, 1 ... input\_files.length]

distr = distributions.copy()

write\_to\_file = 0

write\_to\_file\_idx = 0

read\_pos = 0

runs\_written\_so\_far = 0

**ПОКИ** readPos < binf.size **ВИКОНАТИ:**

int = binf.get\_int32(readPos, runSize)

input\_files[write\_to\_file].set\_int32(write\_file\_pos[write\_to\_file], int)

input\_file\_runs[write\_to\_file].append(Run(write\_file\_pos[write\_to\_file], 1, input\_files[write\_to\_file]))

distr[write\_to\_file\_idx] -= 1

**ЯКЩО** distr[write\_to\_file\_idx] < 1 **ВИКОНАТИ:**

write\_to\_files.pop(write\_to\_file\_idx)

distr.pop(write\_to\_file\_idx)

write\_to\_file\_idx -= 1

read\_pos += 1

write\_file\_pos[write\_to\_file] += 1

write\_to\_file\_idx = (write\_to\_file\_idx + 1) % len(write\_to\_files)

write\_to\_file = write\_to\_files[write\_to\_file\_idx]

runs\_written += 1

**КІНЕЦЬ ПОКИ**

Додавання пустих серій у файли

**ПОКИ** runsWrittenSoFar < totalRuns **ВИКОНАТИ:**

input\_file\_runs[write\_to\_file].append(Run(write\_file\_pos[write\_to\_file], 0, input\_files[write\_to\_file]))

distr[write\_to\_file\_idx] -= 1

**ЯКЩО** distr[write\_to\_file\_idx] < 1 **ВИКОНАТИ:**

write\_to\_files.pop(write\_to\_file\_idx)

distr.pop(write\_to\_file\_idx)

write\_to\_file\_idx -= 1

**ЯКЩО** len(write\_to\_files) > 0 **ВИКОНАТИ:**

write\_to\_file\_idx = (write\_to\_file\_idx + 1) % len(write\_to\_files)

write\_to\_file = write\_to\_files[write\_to\_file\_idx]

runs\_written += 1

**КІНЕЦЬ ПОКИ**

Багатофазне злиття

sorting = True

output\_pos = 0

switches = 0

**ПОКИ** sorting **ВИКОНАТИ:**

has\_more = True

output\_start = output\_pos

**ПОКИ** has\_more **ВИКОНАТИ:**

min\_i = -1

min\_val = 9999999

**ПОВТОРИТИ ВІД** i=0 **ДО** len(input\_files):

**ЯКЩО** len(input\_file\_runs[i]) > 0 **ТА** input\_file\_runs[i][0].has\_more **ВИКОНАТИ:**

value = input\_file\_runs[i][0].value

**ЯКЩО** value < min\_val **ВИКОНАТИ**:

min\_val = value

min\_i = i

**КІНЕЦЬ ПОВТОРИТИ**

**ЯКЩО** min\_i > -1 **ВИКОНАТИ**:

output\_file.set\_int32(output\_pos, min\_val)

output\_pos += 1

input\_file\_runs[min\_i][0].next()

has\_more = True

**ІНАКШЕ**:

has\_more = False

**КІНЕЦЬ ПОКИ**

output\_length = output\_pos - output\_start

output\_file\_runs.append(Run(output\_start, output\_length, output\_file))

switch\_to = -1

switch\_to\_count = 0

**ПОВТОРИТИ ВІД** i=0 **ДО** len(input\_files):

**ЯКЩО** len(input\_file\_runs[i]) > 0 **ВИКОНАТИ**:

input\_file\_runs[i].pop(0)

**ЯКЩО** len(input\_file\_runs[i]) == 0 **ВИКОНАТИ**:

switch\_to = i

switch\_to\_count += 1

**КІНЕЦЬ ПОВТОРИТИ**

**ЯКЩО** switch\_to\_count == len(input\_files) **ВИКОНАТИ**:

sorting = False

**ІНАКШЕ ЯКЩО** switch\_to > -1 **ВИКОНАТИ**:

switches += 1

output\_file, input\_files[switch\_to] = input\_files[switch\_to], output\_file

output\_file\_runs, input\_file\_runs[switch\_to] = input\_file\_runs[switch\_to], output\_file\_runs

output\_pos = 0

**КІНЕЦЬ ПОКИ**

**ПОВТОРИТИ ВІД** i=0 **ДО** output\_file\_runs[0].length:

binf.set\_int32(i, output\_file.get\_int32(i))

**КІНЕЦЬ ПОВТОРИТИ**

**ПОВТОРИТИ ДЛЯ** inf **В** input\_files:

inf.delete()

**КІНЕЦЬ ПОВТОРИТИ**

output\_file.delete()

Кінець

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

polyphase\_sort.py

from classes import Binary, Run  
import time  
  
  
def sort(file\_count):  
 start = time.time()  
 binf = Binary("numbers", "r+b")  
  
 input\_files = []  
 input\_file\_runs = []  
 output\_file = Binary("file0", "w+b")  
 output\_file\_runs = []  
 for i in range(1, file\_count):  
 input\_files.append(Binary("file"+str(i), "w+b"))  
 input\_file\_runs.append([])  
 print("Створено "+str(file\_count)+" тимчасових файлів.")  
 print("-"\*60)  
  
 current\_size = len(input\_files)  
 distributions = [1] \* current\_size  
 target\_size = binf.size  
 total\_switches = 0  
 while current\_size < target\_size:  
 print(distributions)  
 total\_switches += 1  
 max\_val = -1  
 max\_i = 0  
 for i in range(len(distributions)):  
 if distributions[i] > max\_val:  
 max\_val = distributions[i]  
 max\_i = i  
 for i in range(len(distributions)):  
 if i != max\_i:  
 distributions[i] += max\_val  
 current\_size += max\_val  
 print("\nРозподіл: " + str(distributions))  
 print("Пустих серій: " + str(current\_size - target\_size))  
 total\_runs = current\_size  
  
  
 write\_file\_pos = [0] \* len(input\_files)  
 write\_to\_files = [x for x in range(len(input\_files))]  
 distr = [distributions[x] for x in range(len(input\_files))]  
 write\_to\_file = 0  
 write\_to\_file\_idx = 0  
 read\_pos = 0  
 runs\_written = 0  
  
 while read\_pos < binf.size:  
 int = binf.get\_int32(read\_pos)  
 input\_files[write\_to\_file].set\_int32(write\_file\_pos[write\_to\_file], int)  
 input\_file\_runs[write\_to\_file].append(Run(write\_file\_pos[write\_to\_file], 1, input\_files[write\_to\_file]))  
 distr[write\_to\_file\_idx] -= 1  
 if distr[write\_to\_file\_idx] < 1:  
 write\_to\_files.pop(write\_to\_file\_idx)  
 distr.pop(write\_to\_file\_idx)  
 write\_to\_file\_idx -= 1  
  
 read\_pos += 1  
 write\_file\_pos[write\_to\_file] += 1  
 write\_to\_file\_idx = (write\_to\_file\_idx + 1) % len(write\_to\_files)  
 write\_to\_file = write\_to\_files[write\_to\_file\_idx]  
 runs\_written += 1  
  
 while runs\_written < total\_runs:  
 input\_file\_runs[write\_to\_file].append(Run(write\_file\_pos[write\_to\_file], 0, input\_files[write\_to\_file]))  
 distr[write\_to\_file\_idx] -= 1  
 if distr[write\_to\_file\_idx] < 1:  
 write\_to\_files.pop(write\_to\_file\_idx)  
 distr.pop(write\_to\_file\_idx)  
 write\_to\_file\_idx -= 1  
  
 if len(write\_to\_files) > 0:  
 write\_to\_file\_idx = (write\_to\_file\_idx + 1) % len(write\_to\_files)  
 write\_to\_file = write\_to\_files[write\_to\_file\_idx]  
 runs\_written += 1  
  
 sorting = True  
 output\_pos = 0  
 switches = 0  
 while sorting:  
 has\_more = True  
 output\_start = output\_pos  
 while has\_more:  
 min\_i = -1  
 min\_val = 9999999  
 for i in range(len(input\_files)):  
 if len(input\_file\_runs[i]) > 0 and input\_file\_runs[i][0].has\_more:  
 value = input\_file\_runs[i][0].value  
 if value < min\_val:  
 min\_val = value  
 min\_i = i  
 if min\_i > -1:  
 output\_file.set\_int32(output\_pos, min\_val)  
 output\_pos += 1  
 input\_file\_runs[min\_i][0].next()  
 has\_more = True  
 else:  
 has\_more = False  
 output\_length = output\_pos - output\_start  
 output\_file\_runs.append(Run(output\_start, output\_length, output\_file))  
 switch\_to = -1  
 switch\_to\_count = 0  
 for i in range(len(input\_files)):  
 if len(input\_file\_runs[i]) > 0:  
 input\_file\_runs[i].pop(0)  
 if len(input\_file\_runs[i]) == 0:  
 switch\_to = i  
 switch\_to\_count += 1  
  
 if switch\_to\_count == len(input\_files):  
 sorting = False  
 elif switch\_to > -1:  
 switches += 1  
 print("["+str(switches)+"/"+str(total\_switches)+"]")  
 output\_file, input\_files[switch\_to] = input\_files[switch\_to], output\_file  
 output\_file\_runs, input\_file\_runs[switch\_to] = input\_file\_runs[switch\_to], output\_file\_runs  
 output\_pos = 0  
  
 for i in range(output\_file\_runs[0].length):  
 binf.set\_int32(i, output\_file.get\_int32(i))  
 for inf in input\_files:  
 inf.delete()  
 output\_file.delete()  
  
 print(f"Час виконання: {str(time.time() - start)}")  
  
  
def main():  
 file\_count = int(input("Кількість файлів: "))  
 sort(file\_count)  
  
  
main()

classes.py

import os  
  
  
class Binary:  
 def \_\_init\_\_(self, name, mode="r+b"):  
 try:  
 my\_file = open(name, mode)  
 self.my\_file = my\_file  
 self.name = name  
 my\_file.seek(0, 2)  
 self.size = my\_file.tell() // 4  
 self.ok = True  
 self.closed = False  
 except IOError:  
 self.ok = False  
  
 def get\_int32(self, pos):  
 self.my\_file.seek(pos\*4)  
 arr = self.my\_file.read(4)  
 return ((arr[3]\*256+arr[2])\*256+arr[1])\*256+arr[0]  
  
 def get\_ints32(self, pos, amount):  
 self.my\_file.seek(pos\*4)  
 arr = self.my\_file.read(amount\*4)  
 outputs = []  
 for i in range(0, len(arr), 4):  
 outputs.append(((arr[i+3]\*256+arr[i+2])\*256+arr[i+1])\*256+arr[i])  
 return outputs  
  
 def set\_int32(self, pos, value):  
 self.my\_file.seek(pos\*4)  
 arr = []  
 for i in range(4):  
 arr.append(value % 256)  
 value //= 256  
 self.my\_file.write(bytearray(arr))  
 if pos+1 > self.size:  
 self.size = pos+1  
  
 def set\_ints32(self, pos, values):  
 self.my\_file.seek(pos\*4)  
 arr = []  
 for j in range(len(values)):  
 value = values[j]  
 for i in range(4):  
 arr.append(value % 256)  
 value //= 256  
 self.my\_file.write(bytearray(arr))  
  
 def close(self):  
 if not self.closed:  
 self.my\_file.close()  
 self.closed = True  
  
 def delete(self):  
 if not self.closed:  
 self.my\_file.close()  
 self.closed = True  
 os.remove(self.name)  
  
  
class Run:  
 def \_\_init\_\_(self, start, length, bin\_file):  
 self.start = start  
 self.length = length  
 self.pos = 0  
 self.bin\_file = bin\_file  
 if length > 0:  
 self.has\_more = True  
 self.value = bin\_file.get\_int32(start)  
 else:  
 self.has\_more = False  
 self.value = 0  
  
 def next(self):  
 self.pos += 1  
 if self.pos < self.length:  
 self.value = self.bin\_file.get\_int32(self.start + self.pos)  
 else:  
 self.has\_more = False

modified\_polyphase\_sort.py

from classes import Binary  
import math  
import time  
  
cache\_size = 4096  
  
class Run:  
 def \_\_init\_\_(self, start, length, bin\_file):  
 self.start = start  
 self.length = length  
 self.pos = 0  
 self.bin\_file = bin\_file  
  
 def prepare(self):  
 if self.length > 0:  
 self.has\_more = True  
 self.cache = self.bin\_file.get\_ints32(self.start, min(cache\_size, self.length))  
 self.cache\_pos = 0  
 self.value = self.cache[0]  
 else:  
 self.has\_more = False  
 self.cache = []  
 self.cache\_pos = 0  
 self.value = 0  
  
 def next(self):  
 self.pos += 1  
 self.cache\_pos += 1  
 if self.pos < self.length:  
 if self.cache\_pos >= cache\_size:  
 self.cache = self.bin\_file.get\_ints32(self.start + self.pos, min(cache\_size, self.length))  
 self.cache\_pos = 0  
 self.value = self.cache[self.cache\_pos]  
 return self.value  
 else:  
 self.has\_more = False  
 return 0  
  
  
def sort(file\_count, run\_size):  
 binf = Binary("numbers", "r+b")  
 start = time.time()  
  
 input\_files = []  
 input\_file\_runs = []  
 output\_file = Binary("file0", "w+b")  
 output\_file\_runs = []  
 for i in range(1, file\_count):  
 input\_files.append(Binary("file"+str(i), "w+b"))  
 input\_file\_runs.append([])  
  
 distributions = [1] \* len(input\_files)  
 current\_size = len(input\_files)  
 target\_size = math.ceil(binf.size / run\_size)  
 total\_switches = 0  
 while current\_size < target\_size:  
 print(distributions)  
 total\_switches += 1  
 max\_val = -1  
 max\_i = 0  
 for i in range(len(distributions)):  
 if distributions[i] > max\_val:  
 max\_val = distributions[i]  
 max\_i = i  
 for i in range(len(distributions)):  
 if i != max\_i:  
 distributions[i] += max\_val  
 current\_size += max\_val  
 print("\nРозподіл: "+str(distributions))  
 print("Пустих серій: "+str(current\_size - target\_size))  
  
 write\_file\_pos = [0] \* len(input\_files)  
 write\_to\_files = [x for x in range(len(input\_files))]  
 distr = [distributions[x] for x in range(len(input\_files))]  
 write\_to\_file = 0  
 write\_to\_file\_idx = 0  
 read\_pos = 0  
 runs\_written = 0  
  
 while read\_pos < binf.size:  
 ints = binf.get\_ints32(read\_pos, run\_size)  
 ints.sort()  
 input\_files[write\_to\_file].set\_ints32(write\_file\_pos[write\_to\_file], ints)  
 input\_file\_runs[write\_to\_file].append(Run(write\_file\_pos[write\_to\_file], len(ints), input\_files[write\_to\_file]))  
 distr[write\_to\_file\_idx] -= 1  
 if distr[write\_to\_file\_idx] < 1:  
 write\_to\_files.pop(write\_to\_file\_idx)  
 distr.pop(write\_to\_file\_idx)  
 write\_to\_file\_idx -= 1  
 read\_pos += run\_size  
 write\_file\_pos[write\_to\_file] += run\_size  
 if len(write\_to\_files) > 0:  
 write\_to\_file\_idx = (write\_to\_file\_idx + 1) % len(write\_to\_files)  
 write\_to\_file = write\_to\_files[write\_to\_file\_idx]  
 runs\_written += 1  
  
 for i in range(current\_size - target\_size):  
 input\_file\_runs[write\_to\_file].append(Run(write\_file\_pos[write\_to\_file], 0, input\_files[write\_to\_file]))  
 distr[write\_to\_file\_idx] -= 1  
 if distr[write\_to\_file\_idx] < 1:  
 write\_to\_files.pop(write\_to\_file\_idx)  
 distr.pop(write\_to\_file\_idx)  
 write\_to\_file\_idx -= 1  
 if len(write\_to\_files) > 0:  
 write\_to\_file\_idx = (write\_to\_file\_idx + 1) % len(write\_to\_files)  
 write\_to\_file = write\_to\_files[write\_to\_file\_idx]  
 runs\_written += 1  
  
 sorting = True  
 output\_pos = 0  
 switches = 0  
 while sorting:  
 output\_start = output\_pos  
 cache = []  
 cache\_pos = output\_pos  
 files = []  
 values = []  
 for i in range(len(input\_files)):  
 if len(input\_file\_runs[i]) > 0:  
 input\_file\_runs[i][0].prepare()  
 if input\_file\_runs[i][0].has\_more:  
 files.append(input\_file\_runs[i][0])  
 values.append(input\_file\_runs[i][0].value)  
 while len(values) > 1:  
 min\_val = min(values)  
 min\_i = values.index(min\_val)  
 cache.append(min\_val)  
 if len(cache) == cache\_size:  
 output\_file.set\_ints32(cache\_pos, cache)  
 cache = []  
 cache\_pos = cache\_pos + cache\_size  
 run = files[min\_i]  
 run.pos += 1  
 run.cache\_pos += 1  
 if run.pos < run.length:  
 if run.cache\_pos >= cache\_size:  
 run.cache = run.bin\_file.get\_ints32(run.start + run.pos, min(cache\_size, run.length - run.pos))  
 run.cache\_pos = 0  
 values[min\_i] = run.value = run.cache[run.cache\_pos]  
 else:  
 values.pop(min\_i)  
 files.pop(min\_i)  
  
 if len(cache) > 0:  
 output\_file.set\_ints32(cache\_pos, cache)  
 cache\_pos += len(cache)  
  
 if len(values) == 1:  
 run = files[0]  
 run.pos += 1  
 while run.length - run.pos > 0:  
 ints = run.bin\_file.get\_ints32(run.start + run.pos, min(cache\_size, run.length - run.pos))  
 run.pos += cache\_size  
 output\_file.set\_ints32(cache\_pos, ints)  
 cache\_pos += len(ints)  
 output\_pos = cache\_pos  
  
 output\_length = output\_pos - output\_start  
 output\_file\_runs.append(Run(output\_start, output\_length, output\_file))  
 switch\_to = -1  
 switch\_to\_count = 0  
 for i in range(len(input\_files)):  
 if len(input\_file\_runs[i]) > 0:  
 input\_file\_runs[i].pop(0)  
 if len(input\_file\_runs[i]) == 0:  
 switch\_to = i  
 switch\_to\_count += 1  
  
 if switch\_to\_count == len(input\_files):  
 sorting = False  
 elif switch\_to > -1:  
 switches += 1  
 print("["+str(switches)+"/"+str(total\_switches)+"]")  
 output\_file, input\_files[switch\_to] = input\_files[switch\_to], output\_file  
 output\_file\_runs, input\_file\_runs[switch\_to] = input\_file\_runs[switch\_to], output\_file\_runs  
 output\_pos = 0  
  
  
 for i in range(output\_file\_runs[0].length):  
 binf.set\_int32(i, output\_file.get\_int32(i))  
 for inf in input\_files:  
 inf.delete()  
 output\_file.delete()  
  
 print(f"Час виконання: {str(time.time() - start)}")  
  
  
def main():  
 global cache\_size  
 file\_count = int(input("Кількість файлів: "))  
 sort(file\_count, 100)  
  
  
main()

* + - 1. ВИСНОВОК

Під час виконання лабораторної роботи ознайомився з базовими алгоритмами зовнішнього сортування, було розроблено та написано алгоритм багатофазного сортування на мові Python. Для підвищення швидкості роботи була створена модифікація алгоритму, яка полягала в попередньому сортуванню серій за допомогою внутрішнього сортування(стандартні функції Python). Було проведено заміри швидкості алгоритмів, що показали вагомий приріст у швидкості модифікованого алгоритму, у ~4 рази швидше.

|  |  |  |  |
| --- | --- | --- | --- |
| Розмір файлу | Кількість файлів | Базовий | Модифікований |
| 1MB | 3 | 24 с. | 4 с. |
| 1MB | 6 | 16 с. | 3 с. |
| 10MB | 3 | 20 хв. 5с. | 41 с. |
| 100MB | 6 | Дуже багато ☹ | 5 хв. 20 c. |