**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт**

з лабораторної роботи № 1 з дисципліни

«Проектування алгоритмів»

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

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-25 Карпов Л. В.*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.М.*

Київ 2023

Зміст

[1 Мета лабораторної роботи 3](#_Toc109342184)

[2 ЗаВдання 4](#_Toc109342185)

[3 Виконання 6](#_Toc109342186)

[3.1 Псевдокод алгоритму 6](#_Toc109342187)

[3.2 Програмна реалізація алгоритму 6](#_Toc109342188)

[3.2.1 Вихідний код 6](#_Toc109342189)

[Висновок 7](#_Toc109342190)

[Критерії оцінювання 8](#_Toc109342191)

# Мета лабораторної роботи

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

# Завдання

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

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

Здійснити модифікацію програми і відсортувати випадковим чином згенерований масив цілих чисел, що зберігається у файлі розміром не менше ніж двократний обсяг ОП вашого ПК. Досягти швидкості сортування з розрахунку 1Гб на 3хв. або менше. Достатньо штучно обмежити доступну ОП, для уникнення багатогодинних сортувань (наприклад використовуючи віртуальну машину).

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

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

Таблиця 2.1 – Варіанти алгоритмів

|  |  |
| --- | --- |
| **№** | **Алгоритм сортування** |
| 1 | Пряме злиття |
| 2 | Природне (адаптивне) злиття |
| 3 | Збалансоване багатошляхове злиття |
| 4 | Багатофазне сортування |
| 5 | Пряме злиття |
| 6 | Природне (адаптивне) злиття |
| 7 | Збалансоване багатошляхове злиття |
| 8 | Багатофазне сортування |
| 9 | Пряме злиття |
| 10 | Природне (адаптивне) злиття |
| 11 | Збалансоване багатошляхове злиття |
| 12 | Багатофазне сортування |
| 13 | Пряме злиття |
| 14 | Природне (адаптивне) злиття |
| 15 | Збалансоване багатошляхове злиття |
| 16 | Багатофазне сортування |
| 17 | Пряме злиття |
| 18 | Природне (адаптивне) злиття |
| 19 | Збалансоване багатошляхове злиття |
| 20 | Багатофазне сортування |
| 21 | Пряме злиття |
| 22 | Природне (адаптивне) злиття |
| 23 | Збалансоване багатошляхове злиття |
| 24 | Багатофазне сортування |
| 25 | Пряме злиття |
| 26 | Природне (адаптивне) злиття |
| 27 | Збалансоване багатошляхове злиття |
| 28 | Багатофазне сортування |
| 29 | Пряме злиття |
| 30 | Природне (адаптивне) злиття |
| 31 | Збалансоване багатошляхове злиття |
| 32 | Багатофазне сортування |
| 33 | Пряме злиття |
| 34 | Природне (адаптивне) злиття |
| 35 | Збалансоване багатошляхове злиття |

# Виконання

## Псевдокод алгоритму

Create input file with random numbers

While (!input\_file.eof){

For i=0 to I < fib(n) – seq\_in\_temp\_file{

Read seq from input\_file

Internal sort seq

Write seq to temp file

}

Change targeted temp file

}

While (sum of seq in temp files != 1){

Find empty file

Merge other files to empty one

}

Find file with 1 seq

Rename this file

Delete temp files

## Програмна реалізація алгоритму

[Github](https://github.com/okfok/PA_labs/tree/lab1)

### Вихідний код

#include <iostream>  
#include <fstream>  
#include <chrono>  
#include <vector>  
  
int pow(int x, int y) {  
 int res = 1;  
 for (int i = 0; i < y; ++i) {  
 res \*= x;  
 }  
 return res;  
}  
  
bool dec(int mask, int num) {  
 return (mask % pow(2, num + 1)) / pow(2, num);  
}  
  
int unpow(int x, int y) {  
 int res = 0;  
 while (x != 1) {  
 x /= y;  
 res++;  
 }  
 return res;  
}  
  
const std::string INPUT\_FILE\_NAME = "data.bin";  
const std::string OUTPUT\_FILE\_NAME = "result.bin";  
const int M = 5;  
const int N = pow(2, 17);  
const char \*TEMP\_FILE\_NAMES[] = {"1", "2", "3", "4", "5"};  
int count[M];  
int len[M] = {1, 1, 1, 1, 0};  
  
  
int partition(int arr[], int start, int end) {  
  
 int pivot = arr[start];  
  
 int count = 0;  
 for (int i = start + 1; i <= end; i++) {  
 if (arr[i] <= pivot)  
 count++;  
 }  
  
 int pivotIndex = start + count;  
 std::swap(arr[pivotIndex], arr[start]);  
  
 int i = start, j = end;  
  
 while (i < pivotIndex && j > pivotIndex) {  
  
 while (arr[i] <= pivot) {  
 i++;  
 }  
  
 while (arr[j] > pivot) {  
 j--;  
 }  
  
 if (i < pivotIndex && j > pivotIndex) {  
 std::swap(arr[i++], arr[j--]);  
 }  
 }  
  
 return pivotIndex;  
}  
  
void quickSort(int arr[], int start, int end) {  
 if (start >= end)  
 return;  
  
 int p = partition(arr, start, end);  
 quickSort(arr, start, p - 1);  
 quickSort(arr, p + 1, end);  
}  
  
void create\_unsorted\_file(std::string file\_name) {  
 std::ofstream file(file\_name, std::ios::binary);  
 long long len = pow(2, 17) \* 1297;  
 for (long long i = 0; i < len; ++i) {  
 int num = rand();  
 file.write((char \*) (&num), sizeof(num));  
  
 }  
  
 file.close();  
}  
  
void merge(int in\_files[], int out\_index) {  
 std::fstream files[M - 1];  
  
 for (int i = 0; i < M - 1; ++i) {  
 files[i].open(TEMP\_FILE\_NAMES[in\_files[i]], std::fstream::in | std::ios::binary);  
 }  
  
 std::ofstream out(TEMP\_FILE\_NAMES[out\_index], std::ios::binary);  
  
  
 int last = 0, nums[M - 1], used[M - 1], ef\_mask = 0;  
  
 int seq\_count = INT\_MAX;  
 for (int i = 0; i < M - 1; ++i) {  
 if (seq\_count > count[in\_files[i]] && count[in\_files[i]] != 0)  
 seq\_count = count[in\_files[i]];  
 }  
  
 std::cout << "seq\_count: " << seq\_count << '\n';  
  
 for (int i = 0; i < M - 1; ++i) {  
 files[i].read((char \*) &nums[i], sizeof(int));  
 used[i] = 1;  
 }  
  
 for (int iter = 0; iter < M - 2; ++iter) {  
 bool cont = false;  
  
 for (int i = 0; i < M - 1; ++i) {  
 if (!dec(ef\_mask, i) && files[i].eof()) {  
 ef\_mask += pow(2, i);  
 cont = true;  
 break;  
 }  
 }  
  
 if (cont)  
 continue;  
  
  
 while (true) {  
  
 int ibest = (dec(ef\_mask, 0)) ? (((dec(ef\_mask, 1)) ? 2 : 1)) : 0;  
 int imin = (dec(ef\_mask, 0)) ? (((dec(ef\_mask, 1)) ? 2 : 1)) : 0;  
  
 for (int i = 0; i < M - 1; i++) {  
 if (!dec(ef\_mask, i)) {  
 if (last <= nums[i] && nums[i] < nums[ibest])  
 ibest = i;  
 if (nums[i] < nums[imin])  
 imin = i;  
 }  
 }  
  
 if (last <= nums[ibest]) {  
 out.write((char \*) (&nums[ibest]), sizeof(int));  
 last = nums[ibest];  
 if (used[ibest] < seq\_count \* len[in\_files[ibest]] \* N) {  
 files[ibest].read((char \*) &nums[ibest], sizeof(int));  
 used[ibest]++;  
 } else {  
 ef\_mask += pow(2, ibest);  
 break;  
 }  
 } else {  
 out.write((char \*) (&nums[imin]), sizeof(int));  
 last = nums[imin];  
 if (used[imin] < seq\_count \* len[in\_files[imin]] \* N) {  
 files[imin].read((char \*) &nums[imin], sizeof(int));  
 used[imin]++;  
 } else {  
 ef\_mask += pow(2, imin);  
 break;  
 }  
 }  
  
 }  
 }  
  
 int left = unpow(15 - ef\_mask, 2);  
  
 if (used[left] < seq\_count \* len[in\_files[left]] \* N) {  
 for (long long i = 0; i <= seq\_count \* len[in\_files[left]] \* N - used[left]; ++i) {  
 out.write((char \*) (&nums[left]), sizeof(int));  
 files[left].read((char \*) &nums[left], sizeof(int));  
  
 }  
 }  
  
 count[out\_index] = seq\_count;  
 int sum = 0;  
 for (int i = 0; i < M - 1; ++i) {  
 sum += len[in\_files[i]];  
 }  
 len[out\_index] = sum;  
  
  
 std::vector<int> empty, to\_replace;  
 for (int i = 0; i < M - 1; ++i) {  
 if (count[in\_files[i]] == seq\_count || count[in\_files[i]] == 0) {  
 empty.push\_back(i);  
 count[in\_files[i]] = 0;  
 len[in\_files[i]] = 0;  
  
 }  
 if (count[in\_files[i]] > seq\_count)  
 to\_replace.push\_back(i);  
 }  
  
 while (!to\_replace.empty()) {  
 int i = to\_replace[to\_replace.size() - 1];  
 to\_replace.pop\_back();  
 int j = empty[empty.size() - 1];  
 empty.pop\_back();  
  
 files[j].close();  
 files[j].open(TEMP\_FILE\_NAMES[in\_files[j]], std::fstream::out | std::ios::binary);  
 files[i].seekp((seq\_count \* len[in\_files[i]] \* N) \* sizeof(int), std::ios::beg);  
 while (true) {  
 int seq[N];  
 files[i].read((char \*) &seq, sizeof(seq));  
 if (files[i].eof())  
 break;  
 files[j].write((char \*) (&seq), sizeof(seq));  
  
 }  
 count[in\_files[j]] = count[in\_files[i]] - seq\_count;  
 len[in\_files[j]] = len[in\_files[i]];  
 count[in\_files[i]] = 0;  
 len[in\_files[i]] = 0;  
  
 files[i].close();  
 files[i].open(TEMP\_FILE\_NAMES[in\_files[i]], std::fstream::out | std::ios::binary | std::ios::trunc);  
  
 empty.push\_back(i);  
  
 }  
  
 while (!empty.empty()) {  
 int j = empty[empty.size() - 1];  
 empty.pop\_back();  
 files[j].close();  
 files[j].open(TEMP\_FILE\_NAMES[in\_files[j]], std::fstream::out | std::ios::binary | std::ios::trunc);  
 }  
  
  
 for (auto &file: files) {  
 file.close();  
 }  
  
 out.close();  
}  
  
void task(std::string file\_name) {  
 std::ofstream f;  
 for (auto &file\_name: TEMP\_FILE\_NAMES) {  
 f.open(file\_name, std::fstream::out | std::ios::binary | std::ios::trunc);  
 f.close();  
 }  
 std::ifstream input(file\_name, std::ios::binary);  
 std::ofstream temp\_files[M - 1];  
 for (int i = 0; i < M - 1; ++i) {  
 temp\_files[i].open(TEMP\_FILE\_NAMES[i], std::ios::binary);  
 count[i] = 0;  
 }  
  
 int dist[M - 1] = {0, 0, 0, 1};  
  
 while (!input.eof()) {  
  
 int tmp = dist[3];  
 dist[3] = dist[2] + tmp;  
 dist[2] = dist[1] + tmp;  
 dist[1] = dist[0] + tmp;  
 dist[0] = tmp;  
  
 for (int j = 0; j < M - 1; ++j) {  
  
 long long n = dist[j] - count[j % (M - 1)];  
 for (long long i = 0; i < n; ++i) {  
  
  
 int nums[N];  
 input.read((char \*) &nums, sizeof(nums));  
 if (input.eof())  
 break;  
  
 quickSort(nums, 0, N - 1);  
  
  
 temp\_files[j % (M - 1)].write((char \*) (&nums), sizeof(nums));  
 count[j % (M - 1)]++;  
  
  
 }  
 }  
  
 }  
  
  
 for (auto &file: temp\_files) {  
 file.close();  
 }  
 input.close();  
  
  
 while (count[0] + count[1] + count[2] + count[3] + count[4] != 1) {  
 if (count[0] == 0) {  
 int a[4] = {4, 1, 2, 3};  
 merge(a, 0);  
 } else if (count[1] == 0) {  
 int a[4] = {0, 4, 2, 3};  
 merge(a, 1);  
 } else if (count[2] == 0) {  
 int a[4] = {0, 1, 4, 3};  
 merge(a, 2);  
 } else if (count[3] == 0) {  
 int a[4] = {0, 1, 2, 4};  
 merge(a, 3);  
 } else if (count[4] == 0) {  
 int a[4] = {0, 1, 2, 3};  
 merge(a, 4);  
  
 } else  
 throw std::invalid\_argument("2");  
 }  
  
 int res\_ind;  
 for (int i = 0; i < M; ++i) {  
 if (count[i] == 1) {  
 res\_ind = i;  
 break;  
 }  
 }  
  
 rename(TEMP\_FILE\_NAMES[res\_ind], OUTPUT\_FILE\_NAME.c\_str());  
 for (int i = 0; i < M; ++i) {  
 if (i != res\_ind)  
 remove(TEMP\_FILE\_NAMES[i]);  
 }  
  
}  
  
  
int main() {  
 remove(OUTPUT\_FILE\_NAME.c\_str());  
 create\_unsorted\_file(INPUT\_FILE\_NAME);  
  
 auto start = std::chrono::system\_clock::now();  
 task(INPUT\_FILE\_NAME);  
 auto end = std::chrono::system\_clock::now();  
  
  
 std::chrono::duration<double> elapsed\_seconds = end - start;  
 std::cout << elapsed\_seconds.count() << '\n';  
  
  
}

Висновок

При виконанні даної лабораторної роботи я навчився оброблювати великі масиви даних, які перевищують місткість оперативної пам’яті, та оптимізовувати роботу подібних алгоритмів за допомогою матиматичних підходів реалізації задачі

Критерії оцінювання

У випадку здачі лабораторної роботи до 08.10.2022 включно максимальний бал дорівнює – 5. Після 08.10.2022 максимальний бал дорівнює – 4,5.

Критерії оцінювання у відсотках від максимального балу:

* псевдокод алгоритму – 15%;
* програмна реалізація алгоритму – 20%;
* програмна реалізація модифікацій – 20%;
* робота з git – 40%;
* висновок – 5%.