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-25 Карпов Л. В.* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Головченко М.М.* _____
(прізвище, ім'я, по батькові)

Київ 2023

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

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

2 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

3 ВИКОНАННЯ

3.1 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

3.2 Програмна реалізація алгоритму

[Github]

3.2.1 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%.