МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В.Н.Каразіна

Факультет математики і інформатики

Кафедра теоретичної та прикладної інформатики

Індивідуальне завдання № 4

з курсу «Алгоритми і структури данних»

(назва дисципліни)

на тему: «Методи сортування»

Виконав: студент 2 курсу групи мф-21

напряму підготовки (спеціальності)

Комп'ютерні науки

122 Комп'ютерні науки

Єлагін І.А.

Прийняв: _____

Національна шкала: _____

Кількість балів: _____

Оцінка: ECTS _____

Харків – 2024

Завдання:

1. Оберіть і програмно реалізуйте один з методів вдосконаленого сортування з обчисленням кількості порівнянь та переміщень.

2. Програмно реалізуйте один з методів зовнішнього сортування з обчисленням кількості порівнянь та переміщень.

3. Протестуйте ці реалізацію в найкращому, найгіршому та середньому випадках для даних розміром 100, 1000, 10000, 100000, 1000000 елементів. Збережіть отримані кількості порівнянь та переміщень у підготовленому середовищі та проаналізуйте отримані графіки. Порівняйте ці методи із вже реалізованими. Зробіть свої висновки та запишіть їх. Перевірте, чи відображують отримані вами результати загальні формули трудомісткості та очікувану поведінку методу для різних початкових послідовностей.

Код:

```
#include <iostream>
#include <vector>
#include <ctime>

using namespace std;

// Лічильники для алгоритмів
int comparisons = 0, movements = 0;

// Сортування злиттям (з попереднього прикладу)
void merge(vector<int>& arr, int left, int mid, int right);
void mergeSort(vector<int>& arr, int left, int right);

// Генерація даних
vector<int> generateData(int size, const string& type) {
   vector<int> data(size);
   if (type == "best") {
      // Найкращий випадок: вже відсортовані дані
      for (int i = 0; i < size; ++i) data[i] = i;
   }
   else if (type == "worst") {
      // Найгірший випадок: дані відсортовані у зворотному порядку
      for (int i = 0; i < size; ++i) data[i] = size - i;
```

```
    }
    else if (type == "average") {
        // Середній випадок: випадкові дані
        srand(static_cast<unsigned>(time(nullptr)));
        for (int i = 0; i < size; ++i) data[i] = rand() % size;
    }
    return data;
}

// Тестування сортування злиттям
void testMergeSort(int size, const string& caseType, FILE* file) {
    vector<int> data = generateData(size, caseType);
    comparisons = movements = 0; // Обнулити лічильники

    clock_t start = clock();
    mergeSort(data, 0, data.size() - 1);
    clock_t end = clock();

    double duration = static_cast<double>(end - start) / CLOCKS_PER_SEC * 1000.0;

    // Запис результатів у файл
    fprintf(file, "%d,%s,%d,%d,%.2f\n", size, caseType.c_str(), comparisons, movements, duration);
    printf("Size: %d, Case: %s, Comparisons: %d, Movements: %d, Time: %.2f ms\n",
        size, caseType.c_str(), comparisons, movements, duration);
}

// Реалізація функцій merge та mergeSort
void merge(vector<int>& arr, int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    vector<int> L(n1), R(n2);
    for (int i = 0; i < n1; ++i) {
        L[i] = arr[left + i];
```

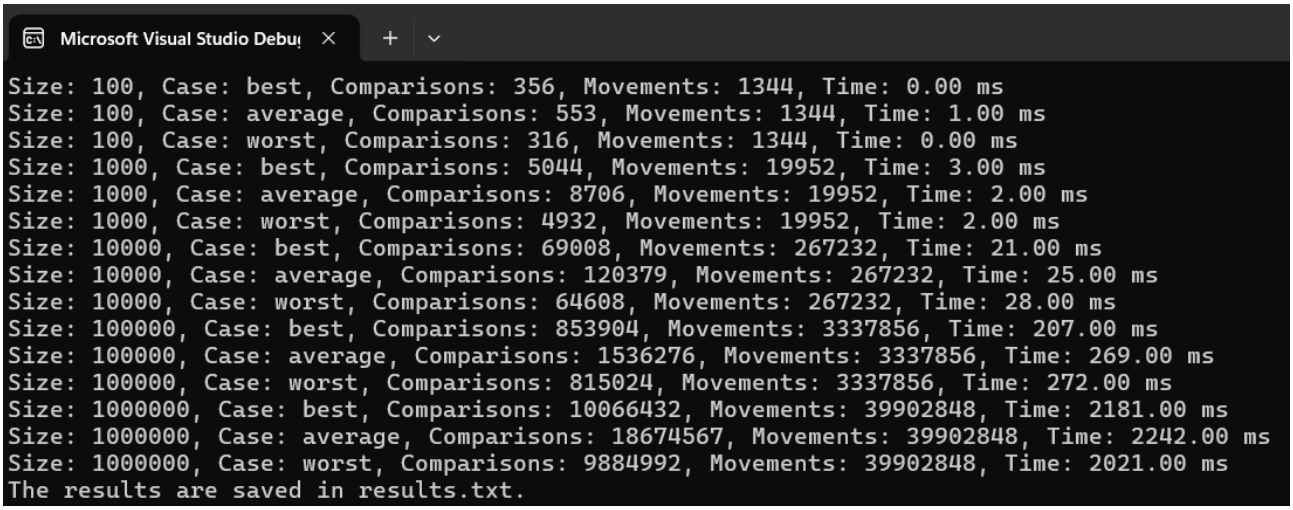
```
    movements++;
}
for (int i = 0; i < n2; ++i) {
    R[i] = arr[mid + 1 + i];
    movements++;
}

int i = 0, j = 0, k = left;
while (i < n1 && j < n2) {
    comparisons++;
    if (L[i] <= R[j]) {
        arr[k] = L[i];
        i++;
    }
    else {
        arr[k] = R[j];
        j++;
    }
    movements++;
    k++;
}

while (i < n1) {
    arr[k] = L[i];
    i++;
    k++;
    movements++;
}

while (j < n2) {
    arr[k] = R[j];
    j++;
    k++;
    movements++;
```

```
    }
}

void mergeSort(vector<int>& arr, int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

int main() {
    vector<int> sizes = { 100, 1000, 10000, 100000, 1000000 };
    vector<string> cases = { "best", "average", "worst" };

    FILE* file;
    errno_t err = fopen_s(&file, "results.txt", "w");
    if (err != 0) {
        cerr << "Не вдалося відкрити файл для запису!" << endl;
        return 1;
    }

    fprintf(file, "Size,Case,Comparisons,Movements,Time(ms)\n"); // Заголовок CSV

    for (int size : sizes) {
        for (const string& caseType : cases) {
            testMergeSort(size, caseType, file);
        }
    }

    fclose(file);
    cout << " The results are saved in results.txt." << endl;
```

```
    return 0;
}
```

Вхідні данні: Рандомні в кількості 100, 1000, 10000, 100000, 1000000 елементів.

Результат роботи коду:



Пояснення коду та операцій:

1. Merge:

Ця функція виконує злиття двох частин масиву, які вже впорядковані. Це ключова частина алгоритму сортування злиттям.

- **1.1.** Вхідні параметри:
  - **1.1.1.** vector<int>& arr – посилання на масив, що сортується.
  - **1.1.2.** int left – початкова позиція лівого підмасиву.
  - **1.1.3.** int mid – межа між лівим і правим підмасивами.
  - **1.1.4.** int right – кінцева позиція правого підмасиву.
- **1.2.** Етапи роботи:
  - **1.2.1.** Розділення масиву на два підмасиви:
    - **1.2.1.1.** Лівий підмасив: arr[left..mid].
    - **1.2.1.2.** Правий підмасив: arr[mid+1..right].

    Для збереження цих підмасивів створюються два нові масиви L та R.

    int n1 = mid - left + 1; // Розмір лівого підмасиву

    int n2 = right - mid; // Розмір правого підмасиву

  - **1.2.2.** Злиття підмасивів у вихідний масив:
    - **1.2.2.1.** Індекси:
      - **1.2.2.1.1.** i – для лівого підмасиву (L).
      - **1.2.2.1.2.** j – для правого підмасиву (R).

1.2.2.1.3. k – для злитого масиву (arr).

1.2.2.2. Логіка злиття:

1.2.2.2.1. Порівнюються елементи з L[i] та R[j].

1.2.2.2.2. Менший елемент додається до arr[k].

1.2.2.2.3. Відповідний індекс (i, j) збільшується.

1.2.2.2.4. Кількість порівнянь (comparisons) та переміщень (movements) збільшується.

Коли один із підмасивів закінчився, залишок іншого копіюється у вихідний масив.

1.2.3. Результат:
Впорядкований підмасив у діапазоні arr[left..right].

2. mergeSort:

Ця функція реалізує рекурсивну логіку сортування злиттям.

1.1. Вхідні параметри:

1.1.1. vector<int>& arr – посилання на масив, що сортується.

1.1.2. int left – початковий індекс підмасиву.

1.1.3. int right – кінцевий індекс підмасиву.

1.2. Етапи роботи:

1.2.1. Умова завершення рекурсії: Якщо left >= right, це означає, що підмасив має один або нуль елементів і не потребує сортування.

1.2.2. Розділення масиву: Масив розділяється на дві частини, визначаючи середній індекс.

1.3. Рекурсивні виклики:

1.3.1. Викликається mergeSort для лівого підмасиву (arr[left..mid]).

1.3.2. Викликається mergeSort для правого підмасиву (arr[mid+1..right]).

1.4. Злиття результатів: Після сортування підмасивів викликається функція merge, щоб об'єднати їх у впорядкований масив.

3. generateData:

Ця функція генерує тестові дані для сортування.

3.1. Вхідні параметри:

3.1.1. int size – кількість елементів у масиві.

3.1.2. const string& type – тип даних:

3.1.2.1. "best": відсортовані в порядку зростання (найкращий випадок).

3.1.2.2. "worst": відсортовані в порядку спадання (найгірший випадок).

3.1.2.3. "average": випадкові числа (середній випадок).

3.2. Логіка роботи:

3.2.1. Для найкращого випадку: Генерується масив з послідовними числами від 0 до size-1.

3.2.2. Для найгіршого випадку: Генерується масив у зворотному порядку: від size до 1.

3.2.3. Для середнього випадку: Генерується масив випадкових чисел за допомогою rand().

4. testMergeSort:

Ця функція тестує сортування злиттям для заданого розміру масиву та типу даних.

4.1. Вхідні параметри:
- **4.1.1.** int size – кількість елементів.
- **4.1.2.** const string& caseType – тип даних (best, average, worst).
- **4.1.3.** FILE* file – дескриптор файлу для запису результатів.

4.2. Етапи роботи:
- **4.2.1.** Генерація тестових даних: Викликається generateData.
- **4.2.2.** Вимірювання часу:
  - **4.2.2.1.** Запам'ятовується початковий час за допомогою clock().
  - **4.2.2.2.** Виконується mergeSort.
  - **4.2.2.3.** Фіксується кінцевий час.
- **4.2.3.** Обчислення тривалості: Час виконання в мілісекундах.
- **4.2.4.** Запис результатів: Результати тестування записуються у файл.

5. Main:

Це головна функція програми, яка координує весь процес тестування.

Логіка роботи:

5.1. Визначення розмірів масивів та типів тестових випадків:

```
vector<int> sizes = {100, 1000, 10000, 100000, 1000000};

vector<string> cases = {"best", "average", "worst"};
```

5.2. Відкриття файлу для запису: За допомогою fopen_s створюється файл results.txt, де зберігаються результати.

5.3. Запуск тестів: Для кожного розміру масиву та кожного випадку даних викликається testMergeSort.

5.4. Закриття файлу: Після завершення тестів файл закривається.

Висновок

Алгоритм сортування злиттям є ефективним і надійним методом для великих наборів даних, особливо коли потрібна стабільність сортування. Його результати відповідають теоретичним оцінкам складності, і він демонструє однакову продуктивність для будь-якого початкового розташування елементів.

Для практичного використання сортування злиттям доцільне у випадках, коли:

1. Розмір масиву великий.
2. Необхідно забезпечити стабільність сортування.
3. Обмеження за додатковою пам'яттю не є критичними.