Міністерство освіти і науки
України Львівський національний університет імені Івана Франка
Факультет електроніки та комп'ютерних технологій

Звіт

про виконання лабораторної роботи №3

“Сортування за лінійний час”

Виконав:

студент 2 курсу

групи ФЕП-23с

Чепара Станіслав

Викладач:

доц. Середницька Христина Ігорівна

Львів - 2025

Мета роботи

Вивчити принципи сортування за лінійний час на прикладі алгоритмів

Сортування підрахунком (Counting Sort) та Порозрядного сортування (Radix Sort). Здобути практичні навички реалізації цих алгоритмів мовою C++. Дослідити важливість таких властивостей, як стабільність сортування, та її вплив на коректність роботи складних алгоритмів.

Теоретичні відомості:

Алгоритми сортування, розглянуті в цій роботі, відрізняються від методів, заснованих на порівняннях (як Quick Sort). Вони здатні досягати **лінійної часової складності** за певних умов, оскільки їхня логіка базується не на порівнянні елементів між собою, а на властивостях самих даних.

1. Сортування підрахунком (Counting Sort)

Сортування підрахунком — це алгоритм, що ефективно застосовується, коли кількість унікальних значень (ключів) у масиві є невеликою. Час його роботи лінійно залежить як від загальної кількості елементів у масиві (n), так і від величини максимального елемента (k).

Ідея алгоритму полягає в наступному:

1. Спочатку підрахувати, скільки разів кожен елемент зустрічається у вхідному масиві.
2. На основі цих даних можна точно визначити кінцеву позицію кожного елемента у відсортованому масиві.
3. Після цього за один прохід розставити всі елементи на свої місця.

Стабільна версія алгоритму (реалізована в лабораторній): Для того, щоб сортування було **стабільним** (тобто не змінювало відносний порядок однакових елементів), використовується більш складний підхід:

1. **Підрахунок частот:** Створюється допоміжний масив `C`, де `C[i]` зберігає кількість входжень елемента `i` у вхідний масив `A`. 2. **Обчислення позицій:** Масив `C` модифікується таким чином, що тепер кожен елемент `C[i]` містить суму кількостей всіх елементів, менших або рівних `i`. Тепер `C[i]` вказує на кінцеву позицію для елемента `i` у відсортованому масиві.
3. **Розстановка елементів:** Створюється вихідний масив `B`. Прохід по вхідному масиву `A` здійснюється **з кінця в початок**. Кожен елемент `A[j]` розміщується у масиві `B` на позицію, вказану в `C[A[j]]`, після чого значення в `C[A[j]]` зменшується. Прохід з кінця гарантує стабільність алгоритму.

2. Сортування за розрядами (Radix Sort)

Сортування за розрядами — це швидкий та

стабільний алгоритм, який використовується для впорядкування елементів, що можуть бути представлені як послідовності, наприклад, рядки або цілі числа.

Ідея алгоритму полягає в тому, щоб сортувати масив по частинах — розряд за розрядом.

1. Спочатку всі елементи впорядковуються за **молодшим розрядом** (наприклад, за одиницями).
2. Потім результат стабільно сортується за наступним розрядом (за десятками).
3. Процес повторюється для всіх розрядів аж до найстаршого.

Ключова вимога: Для коректної роботи алгоритму внутрішня сортировка, що застосовується на кожному кроці для кожного розряду,

обов'язково має бути стабільною. Це означає, що якщо два числа мають однакову цифру в поточному розряді, їхній відносний порядок, встановлений на попередніх кроках, не повинен змінитися. Саме тому Сортування підрахунком ідеально підходить в якості допоміжного алгоритму для Поразрядного сортування.

Зміст роботи

Частина 1. Сортування підрахунком.

1. У бібліотеці Sort (створеній раніше), згідно описаного в теоретичній частині алгоритму, створити функцію CountingSort(…) для реалізації сортування методом підрахунку.

2. Створити новий проект Lab_03_01 та підключити до нього бібліотеку Sort. У функції main() проекту реалізувати можливість введення одномірного масиву цілих чисел з обмеженнями та відображення результатів сортування.

3. Відкомпілювати проект та продемонструвати його роботу викладачеві. У функції CountingSort(…) змінити порядок проходження останнього циклу (зі спадання на зростання). Пояснити отриманий результат.

Частина 2. Сортування за розрядами.

1. У бібліотеці Sort, згідно описаного в теоретичній частині алгоритму, створити функцію RadixSort(…) для реалізації процедури сортування за розрядами. В якості алгоритму сортування окремого розряду обрати метод сортування підрахунками. Сортування здійснювати в циклі для кожного розряду окремо. Щоб отримати i-ий розряд n-розрядного числа (1 i n) потрібно цілочисельно поділити його на 10i-1 та взяти остачу від ділення на 10.

2. Створити новий проект Lab_03_02 та підключити до нього бібліотеку Sort. У функції main() проекту реалізувати можливість введення

одномірного масиву n-розрядних цілих чисел та відображення результатів сортування.

3. Відкомпілювати проект та продемонструвати його роботу викладачеві.

3.1.У процедурі сортування підрахунками змінити порядок проходження останнього циклу (зі спадання на зростання). Пояснити отриманий результат.

3.2.У функції RadixSort(…) змінити порядок проходження циклу вибору розряду (зі зростання на спадання). Пояснити отриманий результат.

main.cpp

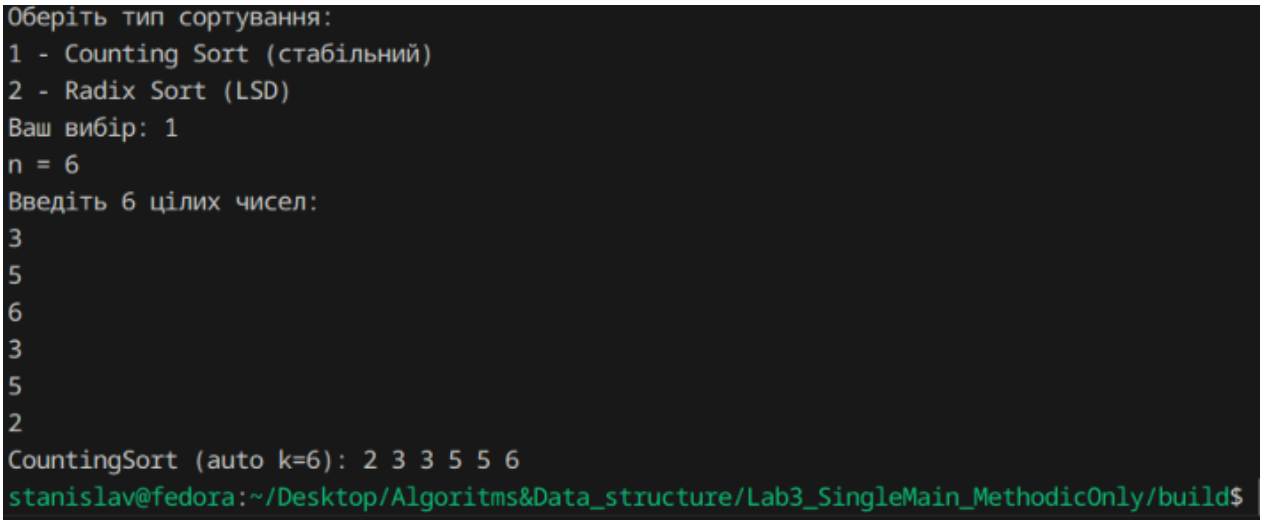
CountingSort.cpp

```cpp
#include "CountingSort.h"

void CountingSort(int* A, int* B, int n, int k)
{
    int* C = new int[k + 1];

    for (int i = 0; i <= k; i++)
    {
        C[i] = 0;
    }

    for (int j = 0; j < n; j++)
    {
        C[A[j]] = C[A[j]] + 1;
    }

    for (int i = 1; i <= k; i++)
    {
        C[i] = C[i] + C[i - 1];
    }

    for (int j = n - 1; j >= 0; j--)
    {
        B[C[A[j]]] = A[j];
        C[A[j]] = C[A[j]] - 1;
    }

    delete[] C;
}
```

RadixSort.cpp

```
static void CountingSortByDigit(int* A, int* B, int n, int exp)
{
    int C[10];
    for (int i = 0; i <= 9; i++)
    {
        C[i] = 0;
    }

    for (int j = 0; j < n; j++)
    {
        int d = (A[j] / exp) % 10;
        C[d] = C[d] + 1;
    }

    for (int i = 1; i <= 9; i++)
    {
        C[i] = C[i] + C[i - 1];
    }

    for (int j = n - 1; j >= 0; j--)
    {
        int d = (A[j] / exp) % 10;
        B[C[d]] = A[j];
        C[d] = C[d] - 1;
    }

    for (int i = 1; i <= n; i++)
    {
        A[i - 1] = B[i];
    }
}

void RadixSort(int* A, int n)
{
    if (n <= 0) return;

    int mx = A[0];
    for (int i = 1; i < n; i++)
    {
        if (A[i] > mx) mx = A[i];
    }

    int* B = new int[n + 1];

    for (int exp = 1; mx / exp > 0; exp = exp * 10)
    {
        CountingSortByDigit(A, B, n, exp);
    }

    delete[] B;
}
```

Результати



Висновок

В ході виконання лабораторної роботи було успішно реалізовано два алгоритми сортування за лінійний час: Сортування підрахунком та Поразрядне сортування. Було засвоєно їхню ключову відмінність від порівняльних сортировок — здатність досягати лінійної часової складності за умови, що вхідні дані є цілими числами в певному діапазоні.

На практиці було досліджено поняття **стабільності** сортування. Було встановлено, що зворотний порядок проходження фінального циклу в Сортуванні підрахунком є критичним для збереження цієї властивості, тоді як прямий прохід хоч і сортує числа правильно, але робить алгоритм нестабільним .