**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"**

**ІКНІ**
Кафедра **ПЗ**


**ЗВІТ**
До лабораторної роботи №5
**на тему:** “ Метод сортування підрахунком.”
**з дисципліни:** "Алгоритми і структури даних”

**Лектор:**
доцент кафедри ПЗ
Коротєєва Т. О.

**Виконав:**
студент групи ПЗ-24
Губик А. С.

**Прийняв:**
асистент кафедри ПЗ
Вишневський К. О.

Львів – 2023

## Тема роботи

Метод сортування підрахунком.

## Мета роботи

Вивчити алгоритм сортування підрахунком. Здійснити програмну реалізацію алгоритму сортування підрахунком. Дослідити швидкодію алгоритму сортування підрахунком.

## Індивідуальне завдання

До кожного елемента застосувати $x^2$

## Теоретичні відомості

Сортування підрахунком (англійською «Counting Sort») — алгоритм впорядкування, що застосовується при малій кількості різних елементів (ключів) у масиві даних. Час його роботи лінійно залежить як від загальної кількості елементів у масиві так і від кількості різних елементів.

Ідея алгоритму полягає в наступному: спочатку підрахувати скільки разів кожен елемент (ключ) зустрічається в вихідному масиві. Спираючись на ці дані можна одразу вирахувати на якому місці має стояти кожен елемент, а потім за один прохід поставити всі елементи на свої місця.

В алгоритмі присутні тільки прості цикли довжини N (довжина масиву), та один цикл довжини K (величина діапазону). Отже, обчислювальна складність роботи алгоритму становить O(N + K).

В алгоритмі використовується додатковий масив. Тому алгоритм потребує E(K) додаткової пам'яті.

В такій реалізації алгоритм є стабільним. Саме ця його властивість дозволяє використовувати його як частину інших алгоритмів сортування (наприклад, сортування за розрядами).

Використання даного алгоритму є доцільним тільки у випадку малих K.

## Покроковий опис

1. **Функція:** До кожного елемента застосовуємо $x^2$
2. **Ініціалізація:** По-перше, ініціалізуємо два додаткові масиви - один для підрахунку кількості входжень кожного елемента та інший для зберігання відсортованих елементів.
3. **Знаходження діапазону:** Знаходимо мінімальний та максимальний елементи у вихідному масиві, щоб визначити діапазон значень.
4. **Підрахунок кількості входжень:** Створюємо масив count, де індекси відповідають унікальним елементам в діапазоні, і для кожного елемента підраховуємо кількість його входжень в вхідний масив. Тобто, count[i] буде містити кількість елементів зі значенням i.
5. **Суми кількостей:** За допомогою масиву count розраховуємо кількості елементів, які менші або рівні кожному унікальному значенню. Це визначає позицію, на яку буде поміщено кожний елемент у відсортованому масиві.

6. **Побудова відсортованого масиву:** Проходимо вхідний масив зліва направо і визначаємо позицію кожного елемента у відсортованому масиві, використовуючи масив count. Після цього зменшуємо значення count для відповідного елемента.

7. **Отримання відсортованого масиву:** На цьому етапі маємо відсортований масив зі збереженими елементами в правильному порядку.

## Вихідний код

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <ctime>
#include <algorithm>
#include <chrono>

// Counting Sort function
void countingSort(std::vector<int>& arr) {
    int maxElement = *std::max_element(arr.begin(), arr.end());
    int minElement = *std::min_element(arr.begin(), arr.end());
    int range = maxElement - minElement + 1;

    std::vector<int> count(range);
    std::vector<int> output(arr.size());

    for (int i = 0; i < arr.size(); i++) {
        count[arr[i] - minElement]++;
    }

    for (int i = 0; i < range; i++) {
        std::cout << count[i];
    }
    std::cout << '\n';

    for (int i = 1; i < range; i++) {
        count[i] += count[i - 1];
    }

    for (int i = arr.size() - 1; i >= 0; i--) {
        output[count[arr[i] - minElement] - 1] = arr[i];
        count[arr[i] - minElement]--;
    }

    for (int i = 0; i < range; i++) {
        std::cout << count[i];
    }
    std::cout << '\n';

    for (int i = 0; i < arr.size(); i++) {
        arr[i] = output[i];
    }
}
```

```
int main() {
    // Seed the random number generator
    std::srand(static_cast<unsigned>(std::time(nullptr)));

    // Get the size of the vector from the user
    int vectorSize;
    std::cout << "Enter the size of the vector: ";
    std::cin >> vectorSize;

    // Generate a vector of random integers within a specified range
    std::vector<int> numbers;
    int minRange = -100;  // Minimum range value
    int maxRange = 100;   // Maximum range value

    for (int i = 0; i < vectorSize; i++) {
        int randomInt = std::rand() % (maxRange - minRange + 1) + minRange;
        numbers.push_back(randomInt);
    }

    // Measure sorting time
    auto startTime = std::chrono::high_resolution_clock::now();

    // Counting Sort the vector
    countingSort(numbers);

    auto endTime = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::nanoseconds>(endTime -

    // Print the sorted vector
    std::cout << "Sorted Vector:" << std::endl;
    for (const int& num : numbers) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    std::cout << "Sorting time: " << duration.count() << " nanoseconds" << std::end

    return 0;
}
```

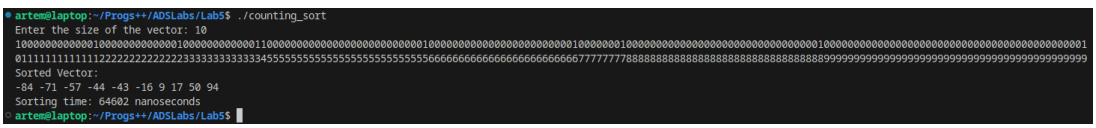


Рис. 1:

## Висновок

Сортування підрахунком є ефективним, коли діапазон можливих значень досить обмежений, але воно не вимагає порівнянь між елементами, що робить його швидким алгоритмом сортування.