ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

Факультет прикладної математики та інформатики

Теорія алгоритмів

ЛАБОРАТОРНА РОБОТА №3

Тема: «Спеціальні сортування»

Виконав:

Ст. *Лук'янчук Денис*

Група ПМІ-23

2025

Тема: «Спеціальні сортування»

Мета роботи: Дослідити та реалізувати алгоритми цифрового сортування, зокрема сортування підрахунком (*Counting Sort*) та порозрядне сортування (*Radix Sort*). Перевірити їхню коректність за допомогою модульного тестування, оцінити їхню ефективність та визначити випадки, у яких вони найбільш доцільні для використання.

Завдання 1:

Дерево рішень — це ієрархічна структура, яка використовується для прийняття рішень на основі послідовності логічних умов.

Основні компоненти дерева рішень:

1. **Вузол (Node) –** містить умову (наприклад, порівняння двох елементів).
2. **Гілки (Edges) –** напрямки від вузла до наступного вузла або листка.
3. **Лист (Leaf) –** кінцевий результат (наприклад, відсортований масив у випадку сортування).

Алгоритм виконання:

1. Знаходимо всі можливі перестановки вхідного масиву.

2. Будуємо дерево рішень, де на кожному кроці порівнюємо елементи масиву.

3. Для кожного порівняння розділяємо перестановки на дві групи (менші та більші).

4. Рекурсивно будуємо піддерева для кожної групи.

5. Виводимо дерево рішень у вигляді текстової діаграми.

6. Проходимо дерево, щоб знайти відсортовану послідовність.

Приклад:

```
Дерево рішень для масиву [3, 1, 2]
                        (arr[0] ? arr[1])
          (arr[0] ? arr[2])                       (arr[0] ? arr[2])
 (arr[1] ? arr[2])   [(2, 3, 1)]        [(2, 1, 3)]        (arr[1] ? arr[2])
[(1, 2, 3)][(1, 3, 2)]                                    [(3, 1, 2)][(3, 2, 1)]

Відсортована послідовність: (1, 2, 3)
PS C:\Users\denys>
```

Завдання 2:

Сортування підрахунком (Counting Sort) – це алгоритм сортування, який працює шляхом підрахунку кількості входжень кожного елемента у вхідному масиві. Він ефективний для сортування чисел або об'єктів, які можна відображати у вигляді цілих чисел у невеликому діапазоні.

Часова складність алгоритму: O(n+k)

Алгоритм виконання:

1. Зчитування вхідних даних – отримуємо масив чисел A.

2. Знаходимо найбільший елемент k у масиві, щоб створити допоміжний масив підрахунку C.

3. Ініціалізація масиву підрахунку C розміром k+1 нулями.

4. Підрахунок частоти входжень кожного елемента масиву A у C.

5. Обчислення накопичених значень у C, щоб визначити позиції елементів у відсортованому масиві.

6. Заповнення вихідного масиву B відповідно до значень у C.

7. Вивід відсортованого масиву.

Приклад:

```
Не посортована послідовність: [4, 2, 2, 8, 3, 3, 1]
Посортована послідовність: [1, 2, 2, 3, 3, 4, 8]
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
PS C:\Теорія алгоритмів\Lab_3>
```

Приклад Unit-тесту(з помилкою):

```
F
======================================================================
FAIL: test_counting_sort (__main__.TestSortingAlgorithms.test_counting_sort)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "c:\Теорія алгоритмів\Lab_3\Lab_3.2.py", line 33, in test_counting_sort
    self.assertEqual(counting_sort([4, 2, 1, 8, 3, 3, 1]), [1, 2, 2, 3, 3, 4, 8])
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
AssertionError: Lists differ: [1, 1, 2, 3, 3, 4, 8] != [1, 2, 2, 3, 3, 4, 8]

First differing element 1:
1
2

- [1, 1, 2, 3, 3, 4, 8]
?     ---

+ [1, 2, 2, 3, 3, 4, 8]
?        +++


----------------------------------------------------------------------
Ran 1 test in 0.001s

FAILED (failures=1)
PS C:\Теорія алгоритмів\Lab_3>
```

Завдання 3:

Цифрове сортування (Radix Sort) – це алгоритм сортування, який працює за принципом розбиття чисел на розряди (цифри) та їхнього послідовного сортування, починаючи з найменш значущого розряду (LSD – Least Significant Digit) або найбільш значущого розряду (MSD – Most Significant Digit).

Часова складність алгоритму: O(d(n+k))

Алгоритм виконання:

1. Зчитування вхідних даних – отримуємо масив чисел.
2. Знаходимо найбільше число в масиві, щоб визначити максимальну кількість цифр d.
3. Сортуємо числа за кожною цифрою окремо – від молодшого до старшого розряду.
4. Для кожного розряду застосовуємо Counting Sort, оскільки він є стабільним.
5. Процес повторюється для кожного розряду, поки не буде відсортовано всі числа.
6. Вивід відсортованого масиву.

Приклад:

```
Не посортована послідовність: [170, 45, 75, 90, 802, 24, 2, 66]
Посортована послідовність: [2, 24, 45, 66, 75, 90, 170, 802]
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
PS C:\Теорія алгоритмів\Lab_3>
```

Приклад Unit-тесту(з помилкою):

```
F
======================================================================
FAIL: test_radix_sort (__main__.TestSortingAlgorithms.test_radix_sort)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "c:\Теорія алгоритмів\Lab_3\Lab_3.3.py", line 45, in test_radix_sort
    self.assertEqual(radix_sort([171, 45, 75, 90, 802, 24, 2, 66]), [2, 24, 45, 66, 75, 90, 170, 802])
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
AssertionError: Lists differ: [2, 24, 45, 66, 75, 90, 171, 802] != [2, 24, 45, 66, 75, 90, 170, 802]

First differing element 6:
171
170

- [2, 24, 45, 66, 75, 90, 171, 802]
?                          ^

+ [2, 24, 45, 66, 75, 90, 170, 802]
?                          ^


----------------------------------------------------------------------
Ran 1 test in 0.001s

FAILED (failures=1)
PS C:\Теорія алгоритмів\Lab_3>
```

Висновок: У ході виконання роботи були розглянуті та реалізовані два алгоритми сортування: Сортування підрахунком (*Counting Sort*) – ефективний алгоритм для сортування чисел у малому діапазоні значень та з повторюваними елементами. Поразрядне сортування (*Radix Sort*) – стабільний алгоритм, який працює швидко для цілих чисел, особливо коли їхній розряд малий. Було проведене модульне тестування, яке підтвердило коректність роботи алгоритмів. Результати показали, що: *Counting Sort* має лінійну складність O(n+k) та добре підходить для малої кількості унікальних значень. *Radix Sort* працює за O(d(n + k)), що є лінійним часом для сталого розряду (*d*). Radix Sort не підтримує від'ємні числа в класичному вигляді, тому було реалізовано модифіковану версію з окремою обробкою від'ємних значень. Таким чином, алгоритми цифрового сортування можуть бути ефективною альтернативою порівняльним методам (*O(n log n)*) за певних умов.