МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ  
«ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ

ЛУЦЬКОГО НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ»

Випускова циклова комісія Компʼютерних систем та інформаційних технологій

**Звіт**

**з практичної роботи №5**

з дисципліни “Алгоритми та структури даних”

Виконав: студент **2** курсу, групи **22-ІСТ**

Сииця Назар

Викладач: Володимир Остапюк

Термін здачі: 07-11-2024

Здано:

Луцьк – 2024

**Завдання**

**Теорія та дослідження:**

1. Опишіть принцип роботи алгоритму сортування підрахунком (Counting Sort). Як створюється допоміжний масив для підрахунку кількості входжень елементів? Як відбувається формування відсортованого масиву?
2. Опишіть принцип роботи алгоритму сортування за розрядами (Radix Sort). Який алгоритм сортування використовується для сортування за кожним розрядом? Чому важливий порядок сортування за розрядами (від молодшого до старшого)?
3. Яка часова складність алгоритму Counting Sort? Від чого вона залежить? В яких випадках цей алгоритм є ефективним?
4. Яка часова складність алгоритму Radix Sort? Від чого вона залежить?
5. Порівняйте алгоритми Counting Sort та Radix Sort. В яких ситуаціях доцільно використовувати кожен з них? Які обмеження цих алгоритмів?
6. Чи є ці алгоритми стабільними? Обґрунтуйте свою відповідь.

**Практичні завдання (Python):**

1. Реалізуйте функцію counting\_sort(array\_to\_sort), яка сортує список array\_to\_sort за допомогою алгоритму сортування підрахунком. Вхідний масив array\_to\_sort містить лише невід'ємні цілі числа.
2. Реалізуйте функцію radix\_sort(array\_to\_sort), яка сортує список array\_to\_sort за допомогою алгоритму сортування за розрядами. Вхідний масив array\_to\_sort містить лише невід'ємні цілі числа.
3. Напишіть програму, яка зчитує список чисел з клавіатури та сортує його за допомогою обох алгоритмів. Виведіть відсортований список.
4. Порівняйте ефективність алгоритмів Counting Sort та Radix Sort на списках різної довжини та з різним діапазоном значень. Виміряйте час виконання кожного алгоритму та побудуйте графік залежності часу виконання від розміру та діапазону значень списку.
5. \*[За бажанням] Модифікуйте функцію counting\_sort(array\_to\_sort) для сортування масиву, який містить від'ємні цілі числа.

**Питання для закріплення:**

1. Який з реалізованих вами алгоритмів виявився ефективнішим на практиці? Чи збігається це з теоретичною оцінкою складності?
2. Чому Radix Sort називають "цифровим" сортуванням?
3. В яких випадках використання Counting Sort або Radix Sort може бути недоцільним?

**Висновок**

**Теорія та дослідження:**

1. **Опишіть принцип роботи алгоритму сортування підрахунком (Counting Sort). Як створюється допоміжний масив для підрахунку кількості входжень елементів? Як відбувається формування відсортованого масиву?**

Алгоритм сортування підрахунком (Counting Sort) є ефективним методом сортування, який підходить для масивів цілих чисел, що знаходяться в обмеженому діапазоні. Принцип роботи цього алгоритму полягає в підрахунку кількості входжень кожного елемента у вхідному масиві, а потім у використанні цієї інформації для формування відсортованого масиву.

Спочатку визначають мінімальне і максимальне значення у вхідному масиві, щоб знати діапазон значень елементів. Далі створюють допоміжний масив для підрахунку, де індекси відповідають можливим значенням елементів у вхідному масиві, а значення за кожним індексом відображають кількість входжень елемента. У процесі проходу по вхідному масиву для кожного елемента збільшують значення у відповідному індексі в допоміжному масиві. Наприклад, якщо у масиві є елемент 5, то збільшують значення в осередку, який відповідає числу 5.

Після цього йде процес накопичення сум у допоміжному масиві. Кожен елемент масиву підрахунку додає до себе значення попереднього елемента, що дає змогу визначити індекс розміщення кожного числа у відсортованому масиві. Завдяки цьому, коли ми починаємо заповнювати вихідний масив у зворотному порядку, ми можемо точно розмістити кожен елемент на його позицію у відсортованому масиві та зменшити відповідний лічильник у допоміжному масиві. Після заповнення всіх позицій вихідний масив буде відсортований.

1. **Опишіть принцип роботи алгоритму сортування за розрядами (Radix Sort). Який алгоритм сортування використовується для сортування за кожним розрядом? Чому важливий порядок сортування за розрядами (від молодшого до старшого)?**

Алгоритм сортування за розрядами (Radix Sort) використовує принцип сортування чисел за кожним розрядом (цифрою) окремо, починаючи з найменш значущого розряду і поступово переходячи до найстаршого. Цей підхід дозволяє сортувати числа у правильному порядку за допомогою кількох проходів через масив, де кожен прохід сортує елементи за одним із розрядів.

Для сортування елементів на кожному окремому розряді Radix Sort часто використовує алгоритм сортування підрахунком (Counting Sort), який є стабільним — тобто зберігає порядок елементів з однаковим значенням. Стабільність у цьому випадку є важливою, оскільки на кожному кроці ми сортуємо за наступним розрядом, і лише завдяки стабільності попередній порядок чисел зберігається.

Сортування від молодшого до старшого розряду є ключовим для Radix Sort. Якщо почати сортування зі старшого розряду, кінцевий порядок буде порушено. Наприклад, якщо число 345 сортувати спершу за сотнями, то воно стане у певну позицію, не враховуючи десятки та одиниці, що призведе до неправильного результату. Проте, коли починаємо сортувати з молодшого розряду, кожен наступний прохід уточнює позицію числа у масиві, забезпечуючи правильний порядок від молодшого до старшого розряду.

Таким чином, сортування за розрядами використовує стабільне сортування для кожного розряду, рухаючись від найменшого до найбільш значущого розряду, щоб забезпечити точний результат сортування для чисел у вхідному масиві.

1. **Яка часова складність алгоритму Counting Sort? Від чого вона залежить? В яких випадках цей алгоритм є ефективним?**

Часова складність алгоритму Counting Sort становить O(n+k) де n — кількість елементів у списку, а k — діапазон значень, які можуть приймати елементи (різниця між найбільшим і найменшим значенням у списку). Часова складність залежить від розміру вхідного списку та діапазону чисел, оскільки алгоритм створює додатковий масив для підрахунку кількості входжень кожного можливого значення.

Counting Sort є ефективним, коли діапазон значень k порівняно невеликий і не перевищує значно кількість елементів n, оскільки за великого діапазону обчислювальні витрати на додатковий масив стають невиправдано високими. Цей алгоритм найкраще підходить для сортування цілих чисел, коли відомо, що вони належать до обмеженого діапазону.

1. **Яка часова складність алгоритму Radix Sort? Від чого вона залежить?**

Часова складність алгоритму Radix Sort залежить від кількості елементів у списку n та кількості цифр або розрядів d у максимальному числі в цьому списку. Радикс сортування має складність O(d⋅(n+b)), де b — основа числової системи, яка зазвичай дорівнює 10 для десяткової системи. При використанні ефективних внутрішніх сортувань, таких як Counting Sort для кожного розряду, ця складність зменшується до O(d⋅n) за умови, що b є постійним числом. Тому часова складність алгоритму Radix Sort переважно залежить від довжини списку n та кількості розрядів d.

1. **Порівняйте алгоритми Counting Sort та Radix Sort. В яких ситуаціях доцільно використовувати кожен з них? Які обмеження цих алгоритмів?**

Counting Sort та Radix Sort — це обидва алгоритми сортування, що не базуються на порівняннях.

Counting Sort є ефективним, коли елементи списку є цілими числами з обмеженим діапазоном значень k, причому k незначно перевищує n. Він має часову складність O(n+k) оскільки створює масив для підрахунку кожного можливого значення. Проте алгоритм споживає значні обсяги пам'яті при великому діапазоні значень, і його не можна застосовувати для сортування даних з плаваючою точкою або рядків. Counting Sort підходить для цілочисельних значень у невеликому діапазоні та у випадках, коли стабільність сортування є важливою.

Radix Sort, на відміну від Counting Sort, застосовується для сортування чисел за розрядами (наприклад, у десятковій або двійковій системах). Він послідовно сортує числа за кожним розрядом, починаючи з найменшого, використовуючи внутрішній алгоритм сортування, як-от Counting Sort, для кожного розряду. Часова складність Radix Sort становить O(d⋅(n+b)), де d — кількість розрядів у числах, а b — основа числової системи. Radix Sort доцільно використовувати для сортування великих наборів цілих чисел або рядків однакової довжини, особливо якщо діапазон значень великий, але довжина чисел (кількість розрядів) є обмеженою.

Обмеження Counting Sort полягають у значному споживанні пам’яті при великому діапазоні значень, а також у неможливості сортувати типи даних, відмінні від цілих чисел з обмеженим діапазоном. Радикс сортування обмежується типами даних, де легко розділити на розряди (наприклад, цілі числа та рядки), і менш ефективне для коротких чисел з великим діапазоном значень.

Загалом, Counting Sort краще підходить для чисел із невеликим діапазоном, а Radix Sort — для даних, де діапазон великий, але кількість розрядів у числах обмежена.

1. **Чи є ці алгоритми стабільними? Обґрунтуйте свою відповідь.**

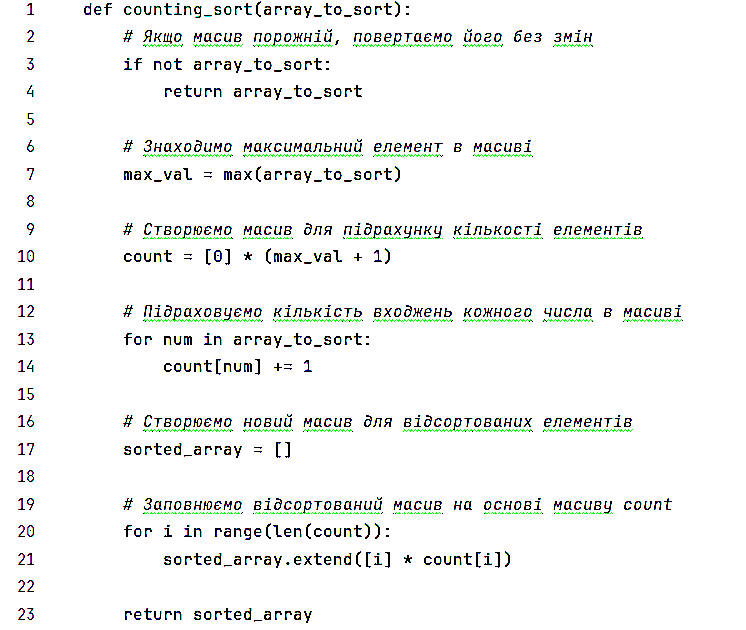
**Counting Sort** є стабільним алгоритмом, оскільки при сортуванні елементів з однаковими значеннями їхній порядок у вихідному списку зберігається в результаті сортування. Це досягається завдяки тому, що Counting Sort використовує підрахунок кількості входжень кожного значення і потім розміщує елементи в результаті на їхніх правильних позиціях. Алгоритм додає елементи в новий масив у порядку їхнього знаходження у вихідному масиві, що забезпечує стабільність сортування.

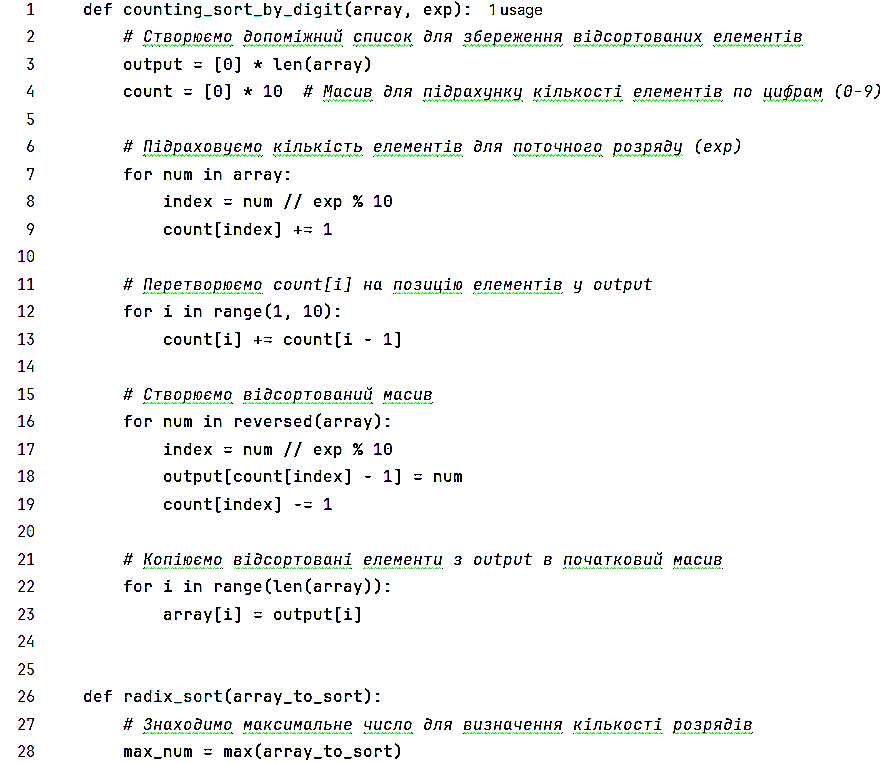
**Radix Sort** також є стабільним, оскільки під час сортування за кожним розрядом використовується стабільний алгоритм сортування, як-от Counting Sort, для сортування елементів по конкретному розряду. Завдяки тому, що порядок елементів з однаковими значеннями розряду зберігається на кожному етапі сортування, кінцевий порядок елементів з однаковими значеннями в результаті залишається таким самим, як у вихідному списку.

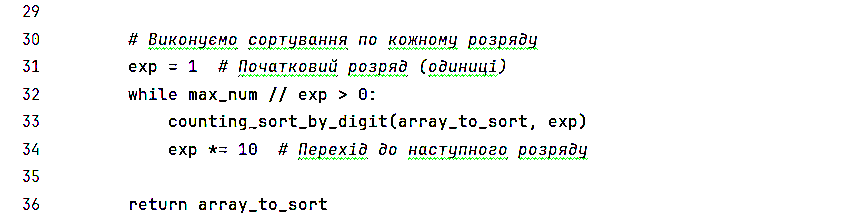
Таким чином, і **Counting Sort**, і **Radix Sort** є стабільними алгоритмами, що означає, що для елементів з однаковими значеннями їхній порядок не змінюється після сортування.

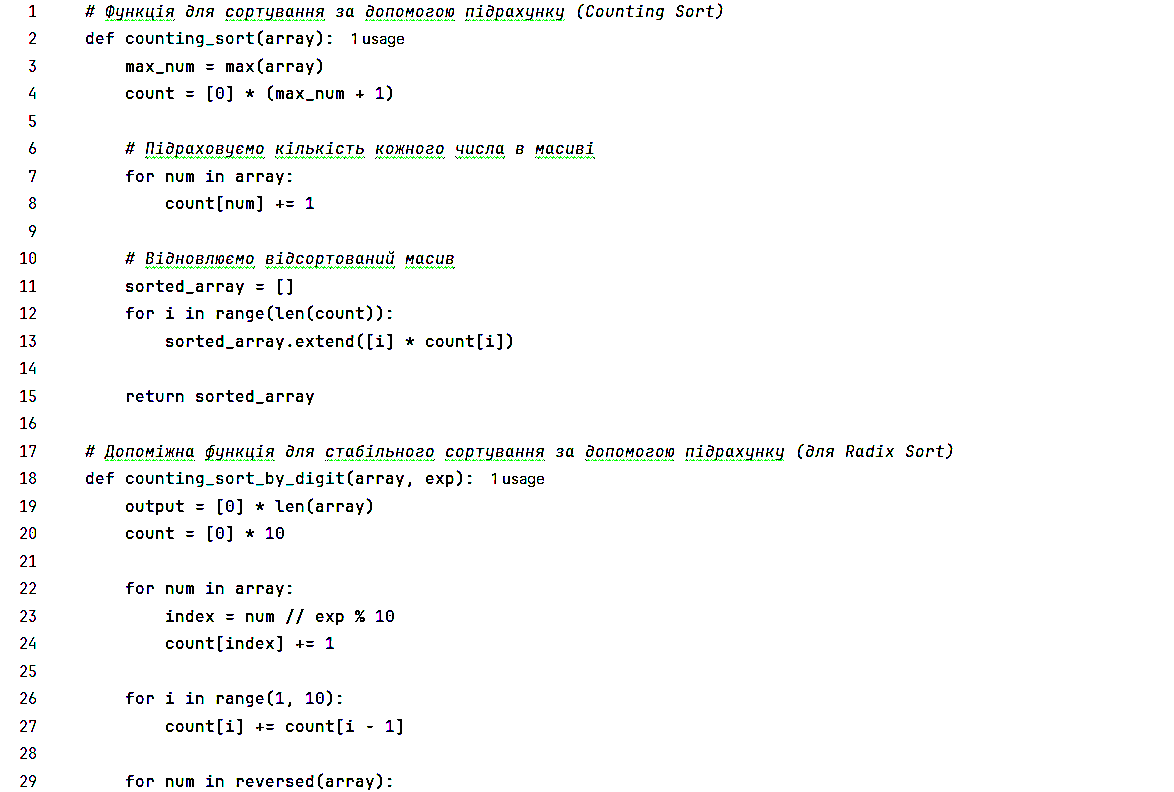
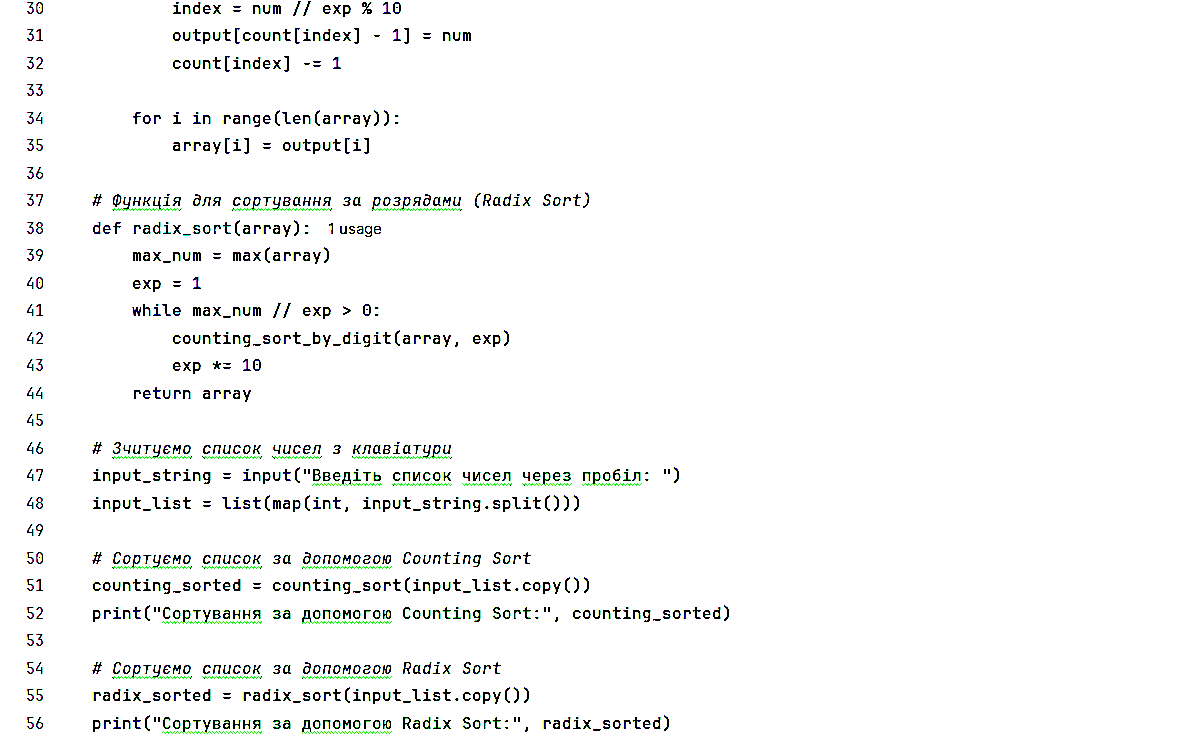
**Практичні завдання (Python):**

1.Реалізуйте функцію counting\_sort(array\_to\_sort), яка сортує список array\_to\_sort за допомогою алгоритму сортування підрахунком. Вхідний масив array\_to\_sort містить лише невід'ємні цілі числа.

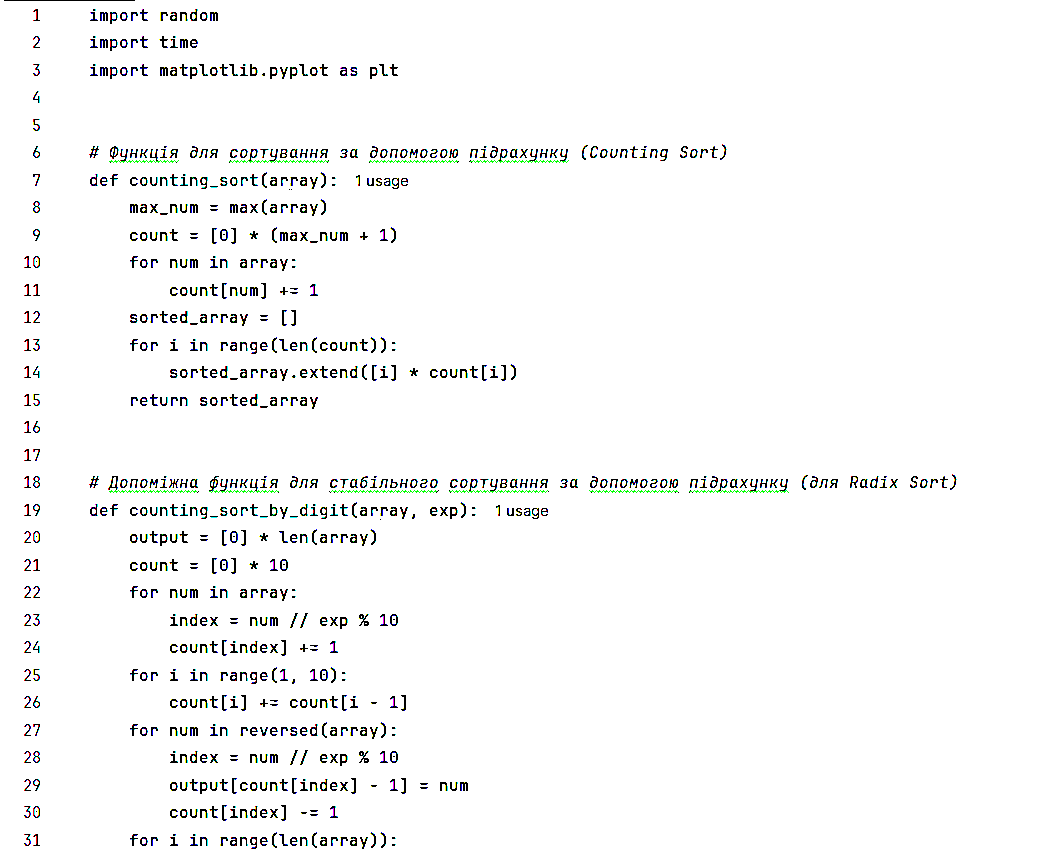


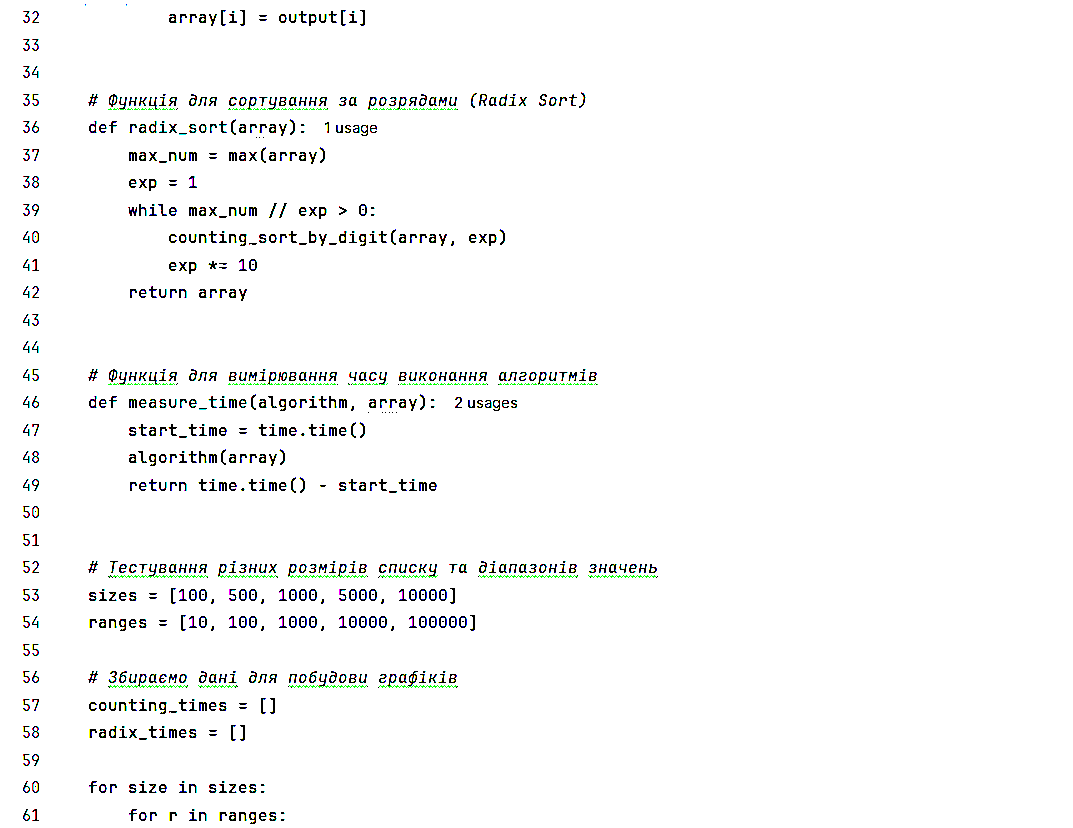
2.Реалізуйте функцію radix\_sort(array\_to\_sort), яка сортує список array\_to\_sort за допомогою алгоритму сортування за розрядами. Вхідний масив array\_to\_sort містить лише невід'ємні цілі числа.

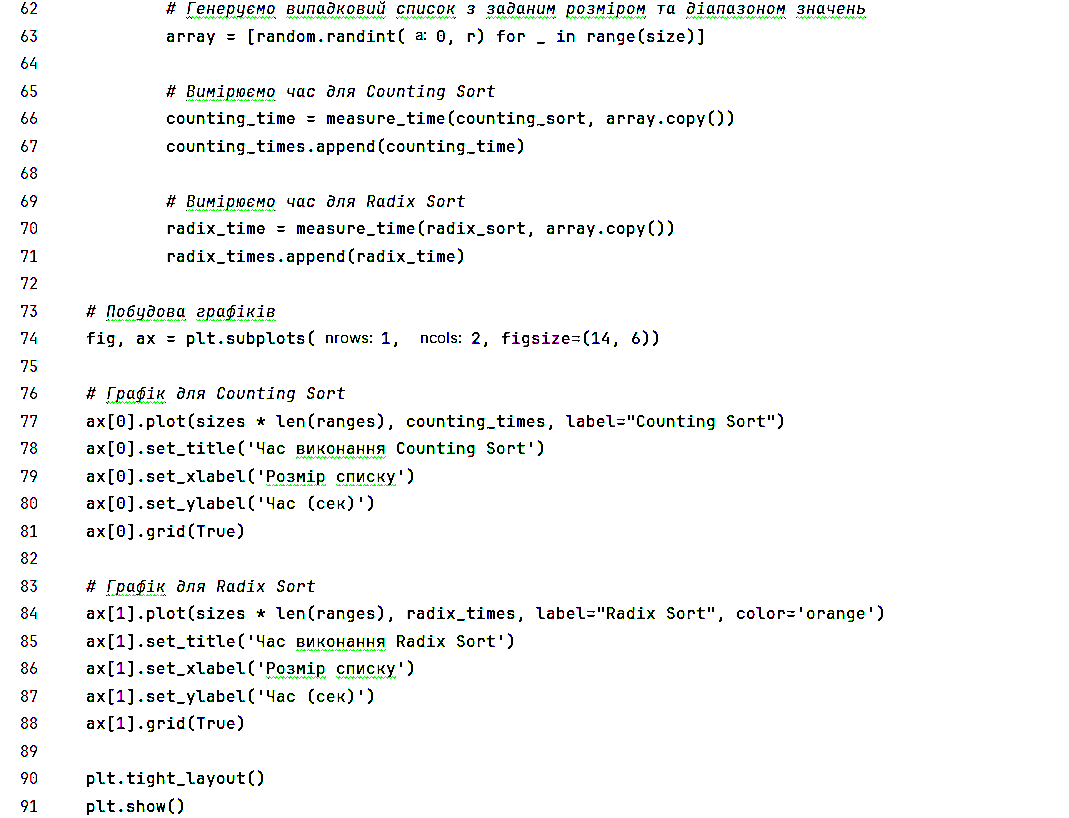


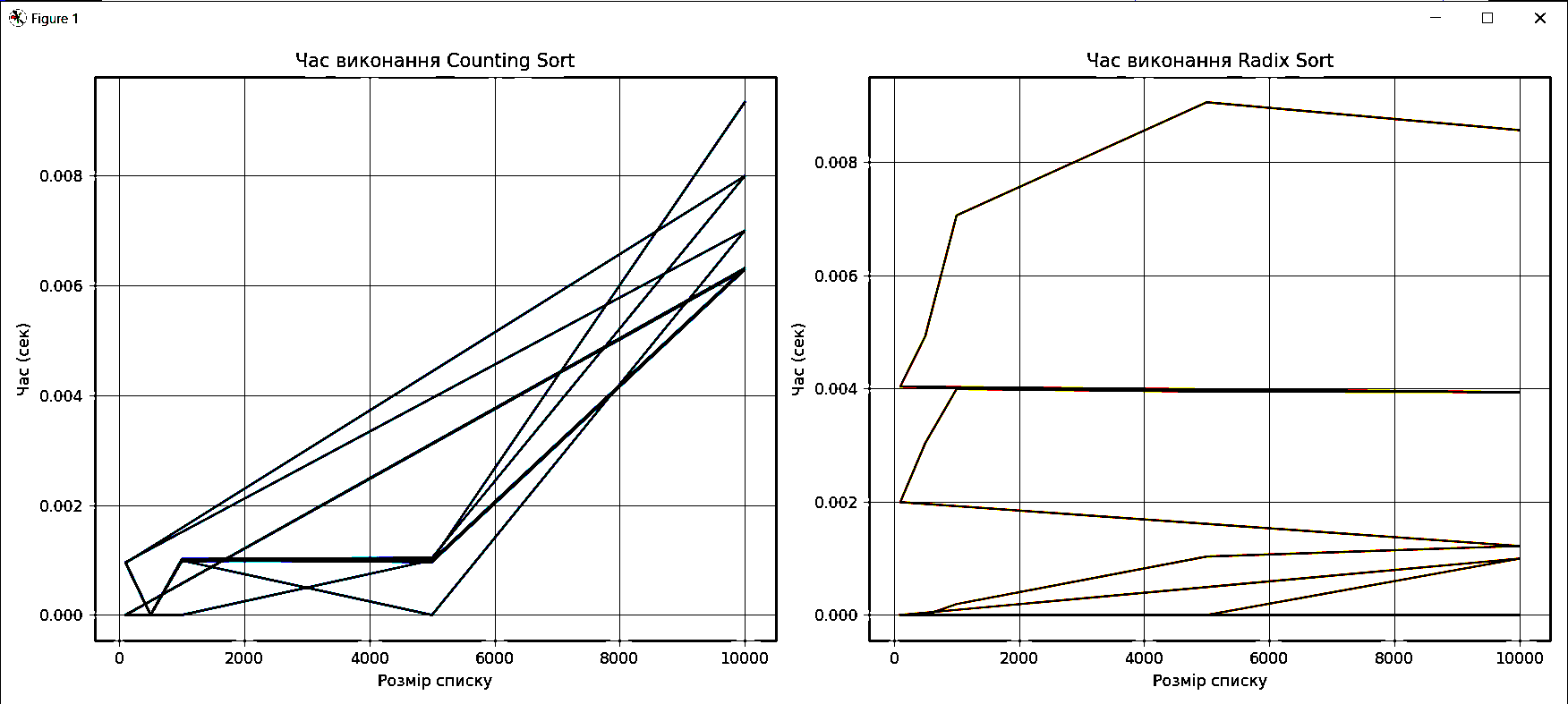
3.Напишіть програму, яка зчитує список чисел з клавіатури та сортує його за допомогою обох алгоритмів. Виведіть відсортований список.

4.Порівняйте ефективність алгоритмів Counting Sort та Radix Sort на списках різної довжини та з різним діапазоном значень. Виміряйте час виконання кожного алгоритму та побудуйте графік залежності часу виконання від розміру та діапазону значень списку.









**Питання для закріплення:**

1. **Який з реалізованих вами алгоритмів виявився ефективнішим на практиці? Чи збігається це з теоретичною оцінкою складності?**

На практиці, **Counting Sort** показує кращі результати, коли діапазон значень невеликий. Наприклад, при сортуванні масивів з елементами, які лежать в межах від 0 до 100, **Counting Sort** може бути значно швидшим, ніж **Radix Sort**. Однак, якщо діапазон значень великий, а самі числа мають невелику кількість розрядів, **Radix Sort** може працювати ефективніше, оскільки він не залежить безпосередньо від величини самого числа, а тільки від кількості його розрядів.

Отже, на практиці **Counting Sort** є більш ефективним при невеликих діапазонах значень, що підтверджує теоретичну оцінку складності. Для великих діапазонів значень з фіксованим числом розрядів **Radix Sort** може бути швидшим, хоча теоретично час його виконання може зрости із збільшенням діапазону. В результаті, теоретична оцінка складності цілком відповідає практичним результатам для різних типів даних.

1. **Чому Radix Sort називають "цифровим" сортуванням?**  
    **Radix Sort** називають "цифровим" сортуванням, оскільки алгоритм сортує числа, орієнтуючись на їх окремі цифри (розряди). Спочатку елементи сортуються за найменш значущим розрядом, наприклад, одиничним, потім переходять до більш значущих розрядів, таких як десятковий, сотневий і так далі. Кожен розряд числа розглядається як "цифра", і алгоритм здійснює сортування саме по цих цифрах. Для кожного розряду використовується стабільне сортування, наприклад, **Counting Sort**, яке допомагає упорядкувати елементи без зміни їх відносного порядку для інших розрядів. Оскільки основний принцип алгоритму полягає в сортуванні чисел по їх цифрових значеннях, **Radix Sort** отримав назву "цифрового" сортування. Це підкреслює, що алгоритм працює з окремими цифрами, а не з цілими числами, як це роблять традиційні порівняльні алгоритми
2. **В яких випадках використання Counting Sort або Radix Sort може бути недоцільним?**

**Counting Sort** та **Radix Sort** можуть бути недоцільними, коли їхні характеристики не відповідають вимогам задачі. **Counting Sort** може бути неефективним, якщо діапазон значень елементів дуже великий, наприклад, коли максимальне значення має кілька мільйонів. Це вимагає значної кількості пам'яті для масиву підрахунку, що може призвести до великих витрат пам'яті. Також, якщо елементи мають великі значення з багатьма розрядами, **Counting Sort** може стати неефективним через необхідність створення дуже великого масиву для підрахунку кожного значення.

**Radix Sort** може бути недоцільним, якщо масив містить великі числа з великою кількістю розрядів, оскільки його час виконання залежить від кількості розрядів, і якщо їх дуже багато, алгоритм може стати дуже повільним. Крім того, **Radix Sort** може бути неефективним для маленьких масивів або якщо діапазон чисел не великий, оскільки порівняльні алгоритми, такі як **Quick Sort**, можуть працювати швидше в таких випадках, не маючи додаткових витрат на обробку розрядів. У таких ситуаціях більш загальні порівняльні алгоритми можуть виявитися кращими.

**Висновок**

У результаті дослідження алгоритмів Counting Sort і Radix Sort можна зробити висновок, що кожен з них має свої переваги в залежності від характеристик вхідних даних. Counting Sort є ефективним при невеликому діапазоні значень, оскільки його складність лінійно залежить від кількості елементів і діапазону значень. Однак його продуктивність значно знижується, коли діапазон значень великий. Radix Sort, в свою чергу, добре працює з великими наборами чисел, особливо коли кількість розрядів невелика. Він забезпечує високу продуктивність при сортуванні великих масивів, навіть якщо діапазон значень великий, але може втратити ефективність при роботі з числами, що мають велику кількість розрядів. Обидва алгоритми стабільні, що робить їх корисними для роботи з масивами, що містять елементи з однаковими значеннями. Вибір між ними залежить від конкретних особливостей вхідних даних, таких як розмір масиву, діапазон значень і кількість розрядів чисел.