УДК 004.62

ОСОБЛИВОСТІ РОБОТИ АЛГОРИТМІВ СОРТУВАННЯ В PYTHON

Хоменко А. О.
аспірант факультету інформаційних технологій кафедри комп. наук
ORCID https://orcid.org/0009-0003-1445-170X
E-mail: andmayster@gmail.com
Кириченко В. В.
канд. фіз-мат. наук, доцент
ORCID https://orcid.org/0009-0001-0575-8684
E-mail: v.kyrychenko@nubip.edu.ua
Національний університет біоресурсів і природокористування України
Героїв оборони 15, м. Київ, Україна

Анотація. Стаття присвячена аналізу різних алгоритмів сортування з використанням мови програмування Python та за допомогою алгоритмів сортування на мові програмування Cython. В дослідженні порівнюються класичні методи сортування, такі як сортування бульбашкою, вставками та швидке сортування, з метою визначення їх ефективності для великих наборів даних. Особливу увагу приділено проблемам локалізації при сортуванні рядків у неанглійських мовах, зокрема, застосуванню спеціалізованого словника для коректної обробки українського алфавіту. Представлено методики вимірювання продуктивності та візуалізації результатів у вигляді графіків, що дозволяє глибше оцінити масштабування кожного з алгоритмів в залежності від обсягу даних.

Ключові слова: Python, Cython, алгоритми сортування, великі набори даних, локалізація, український алфавіт, продуктивність сортування, візуалізація даних.

Актуальність. Алгоритми сортування відіграють важливу роль у програмуванні та обробці даних, дозволяючи ефективно організовувати та аналізувати інформацію. Однак, сортування даних, що містять кирилицю, може викликати певні труднощі через особливості локалізації та кодування символів. Ця стаття розглядає проблеми та особливості сортування рядкових даних на українській мові в Python, аналізує різні підходи та методи сортування, та пропонує ефективні рішення для оптимізації цього процесу.

Проблематика сортування кирилиці. Сортування текстових даних, які містять символи кирилиці, може створити виклики, пов'язані з локалізацією та упорядкуванням символів. Проблема полягає в тому, що стандартні методи сортування в Python не завжди адекватно обробляють символи кирилиці, особливо коли мова йде про українські символи, такі як 'І', 'Є', які мають особливе місце в алфавіті. Це може призвести до неправильного порядку сортування, коли дані відсортовані за алфавітом.

Метою дослідження є оцінка продуктивності різних алгоритмів сортування під управлінням Python та Cython та надання рекомендацій щодо їх вибору для конкретних обчислювальних завдань. Стаття також спрямована на вирішення проблем, пов'язаних з локалізацією сортування рядків, з акцентом на українську мову, яка представляє певні виклики через відмінності в алфавіті порівняно з англійською мовою.

Аналіз останніх досліджень та публікацій. Сучасний стан досліджень у сфері алгоритмів сортування та їх оптимізації свідчить про значний інтерес науковців та розробників до підвищення ефективності обробки даних. Розробка мови програмування

Python, зокрема її розширення Cython, забезпечує нові можливості для оптимізації вже існуючих алгоритмів та створення нових, більш ефективних методів обробки великих обсягів інформації. Значну увагу приділено також адаптації алгоритмів під особливості конкретних мов, що є важливим для міжнародних програмних продуктів.

У роботах Ван Россума Г. та Дрейка Ф. Л. [1], а також Орлова С. А. [3] наголошується на важливості забезпечення високої продуктивності програм на Python шляхом ефективного використання існуючих алгоритмів сортування та розробки нових методів. Дослідження Бехтерева А. М. [2] підкреслює роль Cython у оптимізації Python коду, зокрема для алгоритмів сортування, демонструючи значне збільшення продуктивності.

Специфічні аспекти локалізації алгоритмів сортування, особливо для мов з нелатинською абеткою, розглянуті у працях Кормена Т. Х. та ін. [5], де аналізуються основні труднощі та пропонуються методи їх вирішення. Важливість візуалізації даних для аналізу ефективності алгоритмів підкреслена у роботах Маккинні У. [7], що вказує на необхідність застосування сучасних інструментів для графічного представлення результатів досліджень.

Загалом, аналіз останніх досліджень та публікацій показує тенденцію до пошуку нових методів оптимізації алгоритмів сортування, їх адаптації під специфіку різних мов та культур, а також розвитку інструментів для аналізу та візуалізації даних. Основними напрямками подальших досліджень у цій сфері є розвиток алгоритмів машинного навчання та штучного інтелекту для обробки великих даних, а також створення нових інструментів для ефективної взаємодії з даними в різних мовних середовищах.

Матеріали і методи дослідження. Для тестування алгоритмів сортування та демонстрації проблеми локалізації важливо мати реалістичний набір тестових даних. Один із способів генерації таких даних - використання бібліотеки Faker, яка дозволяє створювати велику кількість різноманітних даних, включаючи імена, прізвища та інше, для імітації реальних даних.

Для додавання українських 'по-батькові' до наших тестових даних, ми розширимо бібліотеку Faker власним провайдером:

```
from faker import Faker
from faker.providers import BaseProvider
import random
class ProviderUk(BaseProvider):
    def patronymic(self):
        patronymics = ['Олександрович', 'Максимович', 'Іванович', 'Петрович', 'Сергійович']
        return random.choice(patronymics)
fake = Faker('uk_UA')
fake.add_provider(ProviderUk)
```

Використовувати його можна наступним чином:

```
def generate_and_save_data_to_csv(filename, num_records):
    with open(filename, mode='w', newline='', encoding='utf-8') as file:
        writer = csv.writer(file)
        writer.writerow(['Name', 'Patronymic', 'Surname'])
        for _ in range(num_records):
            name = fake.first_name()
            patronymic = fake.patronymic()
            surname = fake.last_name()
            writer.writerow([name, patronymic, surname])
```

В цьому прикладі ми реалізовуємо функцію для створення csv файлу та генерацію даних за допомогою Faker, в тому числі й 'по-батькові' яке відсутнє у стандартній бібліотеці Faker.

Методи сортування. Розглянемо декілька підходів до сортування даних в Python, кожен з яких має свої особливості та області застосування. Це допоможе зрозуміти, як можна оптимізувати процес сортування, особливо при роботі з великими обсягами даних або коли потрібно враховувати специфіку локалізації.

- **Стандартне сортування Python.** Метод sorted() в Python є універсальним інструментом для сортування списків. Він застосовує алгоритм сортування Timsort, який є гібридом між Merge Sort та Insertion Sort. Timsort адаптований до реальних даних, які часто містять вже відсортовані секвенції, забезпечуючи високу продуктивність.
- **Сортування з використанням NumPy.** NumPy використовує різні алгоритми сортування в залежності від конкретного випадку, включно з Quicksort, Mergesort, та Heapsort. Функція np.sort() за замовчуванням використовує Quicksort, який є ефективним загальним алгоритмом сортування. Його ефективність обумовлена складністю O(n log n), яка є середня і залежить від вхідних даних та глибини рекурсії. Це підтверджується в роботі [5], де докладно описується робота і ефективність алгоритму Quicksort.
- **Сортування з використанням Pandas.** Pandas надає функціонал для сортування даних у DataFrame за допомогою методу sort_values(). Це дозволяє сортувати дані за значеннями одного або декількох стовпців, надаючи гнучкі можливості для аналізу даних.
- **Сортування з врахуванням локалізації.** Врахування особливостей мови та локалізації може бути важливим при сортуванні рядкових даних. Використання спеціального словника для визначення порядку символів у алфавіті дозволяє точно сортувати рядки, враховуючи особливості мови, як у випадку з українською кирилицею.

У цьому розділі ми детально зупинимося на реалізації трьох ключових алгоритмів сортування за допомогою Cython: сортування бульбашкою, вставками та швидке сортування (QuickSort). Ми вибрали ці алгоритми через їх різноманітність застосувань та рівнів складності, що дозволяє дослідити ефективність сортування від простих до більш складних структур даних. Кожен з них показує унікальні характеристики при роботі з наборами даних різного розміру та типу, надаючи цінне розуміння того, як оптимізації на рівні мови програмування можуть впливати на загальну продуктивність обробки даних.

Сортування бульбашкою (bubble sort) — це простий алгоритм сортування, який працює шляхом многократного проходження через список, порівнюючи кожну пару сусідніх елементів та обмінюючи їх місцями, якщо вони в неправильному порядку. Процес повторюється до тих пір, поки не буде пройдено весь список без жодного обміну, що є знаком того, що список відсортовано. Це один з найпростіших алгоритмів сортування для розуміння та кодування. Сортування бульбашкою не є найефективнішим алгоритмом для великих списків через його квадратичну складність в гіршому та середньому випадках (O(n^2), де n — кількість елементів у списку). Також це стабільний алгоритм сортування, тому що він не змінює порядок однакових елементів.

Сортування вставками (insertion sort) є простим, але ефективним алгоритмом сортування, який працює набагато краще на невеликих або частково відсортованих списках. Алгоритм розділяє список на відсортовану та не відсортовану частини, і поступово вбудовує кожен елемент з не відсортованої частини в правильну позицію у відсортованій.

QuickSort є одним з найшвидших загальних алгоритмів сортування для великих наборів даних. Алгоритм використовує стратегію 'розділяй та володарюй', обираючи один елемент як опорний (pivot) і розділяючи список на дві частини: елементи менші за опорний і елементи більші за опорний. Цей процес рекурсивно застосовується до кожної частини.

Вирішення проблеми локалізації. Один з підходів до вирішення проблеми локалізації при сортуванні рядків, що містять символи не з базового латинського алфавіту, полягає в використанні словника для визначення порядку літер у специфічному алфавіті. Цей метод дозволяє визначити точний порядок сортування для символів, що виходять за рамки стандартного ASCII набору, забезпечуючи правильне впорядкування рядків відповідно до правил конкретної мови або локалі.

Спочатку реалізовується словник(dictionary), з індексами у ролі ключа та літерами у ролі значень.

```
ALPHABET_DICTIONARY = {
   char: index for index, char in
enumerate("АаБбВвГгҐґДдЕеЄєЖжЗзИиІіЇїЙйКкЛлМмНнОоПпРрСсТтУуФфХхЦцЧчШшЩщЬьЮюЯя")
}
```

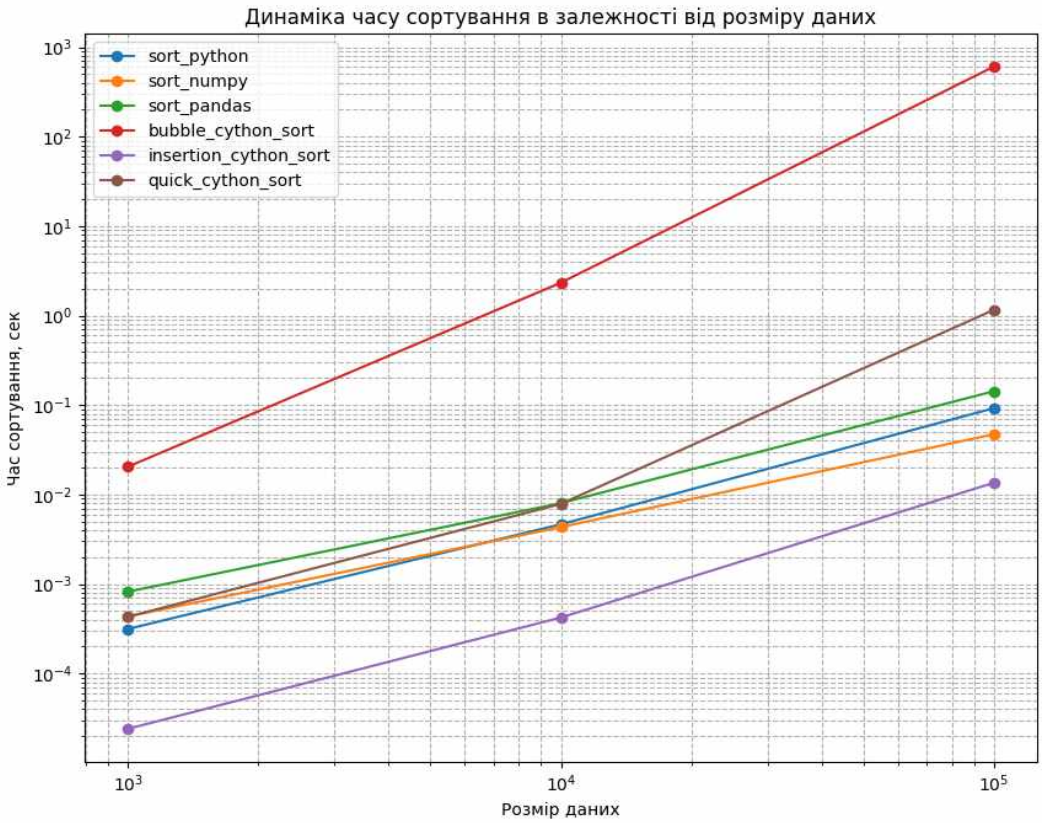
Код функції sorter, яка використовує цей метод:

```
def sorter(compare_str: str) -> int:
      return [ALPHABET_DICTIONARY.get(char, ord(char)) for char in compare_str]
```

І тепер використовуючи стандартний метод для сортування у Python, ми можемо вказати key, надаючи спеціальну функцію ключа, для налаштування порядку сортування.
Нижче наведено такий приклад:

```
actual_word_list = sorted(
["", "і", "ї", "інтернаціональний", "міжнародний", "євгеній", "єлизавета", "абрикос"], key=sorter
)
```

Аналіз продуктивності алгоритмів сортування. Для порівняння продуктивності різних алгоритмів сортування використовується методика вимірювання часу, яка потребує фіксації часу перед початком сортування та після його завершення. Різниця між цими двома показниками вказує на тривалість виконання алгоритму. Використовуючи різні обсяги даних, можна визначити, як масштабується час виконання в залежності від розміру набору даних. Для візуалізації результатів можна створити графіки, які показують час виконання кожного алгоритму сортування в залежності від розміру набору даних. Це дозволяє візуально оцінити продуктивність алгоритмів та їхню поведінку при зміні обсягів даних. На скріншоті, представленому в дослідженні, ми бачимо, як різні алгоритми сортування масштабуються при збільшенні розмірів даних. На представленому графіку на рисунку 1 відображена динаміка часу сортування в



залежності від розміру даних для різних алгоритмів сортування.

Рисунок 1 - Динаміка часу сортування в залежності від розміру даних

Видно, що **стандартне сортування в Python** (`sort_python`), яке оптимізоване для загальних випадків, має лінійно-логарифмічну складність і показує найкращі результати.

Сортування в NumPy (`sort_numpy`) та **Pandas** (`sort_pandas`) також мають схожу продуктивність, що пояснюється використанням високооптимізованих бібліотек C/C++ під капотом цих пакетів.

Алгоритми сортування, реалізовані на Cython (`bubble_cython_sort`, `insertion_cython_sort` і `quick_cython_sort`), демонструють різну продуктивність.

Сортування бульбашкою на Cython (`bubble_cython_sort`) є найменш ефективним через свою квадратичну складність і рекомендується до використання тільки на невеликих наборах даних або коли дані майже відсортовані.

Сортування вставками на Cython (`insertion_cython_sort`) показує кращу продуктивність на невеликих обсягах даних або коли дані частково відсортовані, але його ефективність різко падає зі збільшенням розміру даних.

QuickSort на Cython (`quick_cython_sort`) показує дуже добру продуктивність на великих наборах даних, маючи середню складність O(n log n). Цей алгоритм варто розглянути для більш складних випадків сортування, де необхідно оптимальне співвідношення між швидкістю та використанням пам'яті, особливо коли працюєте з великою кількістю даних.

Висновок. Проведене порівняльне дослідження алгоритмів сортування підтвердило, що ефективність сортування значно залежить від обсягу та характеру даних. Для невеликих та частково відсортованих датасетів алгоритми сортування вставками виявляються достатньо ефективними, завдяки низькій складності та відмінній продуктивності при обмеженій кількості даних. Проте, як показує дослідження, із збільшенням обсягів даних перевагу слід віддавати більш складним алгоритмам, таким як QuickSort, що, будучи оптимізовані в Cython, забезпечують значне зниження часу виконання завдяки вищій ефективності на великих масштабах. Аналіз продуктивності також розкрив важливість адаптації алгоритмів під специфіку мов та локалей, що є критичним для глобалізованих застосунків та систем. В цьому контексті введення спеціалізованого словника для сортування рядків стало ключовою оптимізацією для підтримки української мови. Враховуючи ці фактори, вибір алгоритму сортування повинен відбуватися з огляду на конкретні потреби проекту, природу даних та вимоги до швидкодії, що дозволяє досягти оптимального балансу між швидкістю обробки та використанням ресурсів.

Список використаних джерел

1. Ван Россум Г., Дрейк Ф. Л. Python. Керівництво для розробників: монографія. Київ: Діалог, 2019. 720 с.
2. Бехтерев А. М. Cython: оптимізація Python програм: монографія. Львів: Львівський технологічний, 2018. 300 с.
3. Орлов С. А. Python для комплексних задач: наука про дані та машинне навчання: монографія. Харків: Факт, 2020. 400 с.
4. Седжвік Р., Уейн К. Алгоритми: сортування, пошук, основні структури даних: підручник. Харків: Плеяди, 2017. 992 с.
5. Кормен Т. Х., Лейзерсон Ч. Е., Рівест Р. Л., Штайн К. Алгоритми: побудова та аналіз: підручник. 3-тє вид. Київ: Видавничий дім 'Вільямс', 2019. 1328 с.
6. Маккинни У. Python для аналізу даних: монографія. 2-ге вид. Київ: Основи, 2018. 540 с.
7. Cython: великий путівник для оптимізації Python коду: електрон. наук. фахове вид. 2022. URL: https://cython.readthedocs.io (дата звернення: 31.03.2024).
8. Гроссман С. А., МакКінни В. Ефективний Python: 59 специфічних способів написання кращого коду: монографія. Київ: Діафільм, 2016. 256 с.
9. Лутц М. Програмування на Python. Том 1: вступ до програмування: підручник. Київ: Діафільм, 2021. 640 с.