УДК 004.4+519.2 https://doi.org/10.15407/pp2022.03-04.032

ДВОХКОМПОНЕНТНІ АЛГОРИТМИ СОРТУВАННЯ

Віктор Шинкаренко, Анатолій Дорошенко, Олена Яценко,
Валентин Разносілін, Костянтин Галанін

У роботі досліджувалися можливості покращення часових параметрів сортувань за допомогою попередньої обробки стохастичним сортуванням. Експериментально підтверджено гіпотезу про можливість суттєвого поліпшення часової ефективності двокомпонентного сортування стохастичне + класичне порівняно з таким же класичним однокомпонентним. За класичне прийняті сортування різної обчислювальної складності: шейкерне, з обчислювальною складністю $O(n^2)$, вставками $O(n^2)$, Шелла $O(n\cdot(\log n)^2)$ ... $O(n^{3/2})$, а також швидке з оптимізацією кінцевих ділянок $O(n\cdot\log n)$. Найбільший ефект досягається при виконанні порівнянь стохастичним сортуванням в обсязі 160 % від обсягу масиву. Введені показники ефективності обміну двох елементів та серії порівнянь з обмінами дозволили встановити найбільшу ефективність стохастичного сортування як першого компонента двокомпонентного сортування, коли один елемент для порівняння обирається з першої частини масиву, а інший – з другої. Покращення часової ефективності досягало 70–80 % для алгоритмів з обчислювальною складністю $O(n^2)$. Однак, для сортування Шелла та швидкого попереднє стохастичне сортування не має позитивного ефекту, а навпаки збільшує загальний час сортування, що, вочевидь, пояснюється початковою високою ефективністю даних методів сортування. Гіпотеза про підвищення часової ефективності сортування при трикомпонентному сортуванні швидке + стохастичне + вставками не підтвердилася. Однак, під час експерименту встановлено рекомендований розмір масиву, за якого в двокомпонентному сортуванні швидке + вставками необхідно переходити до другої компоненти – сортуванню вставками. Оптимальна довжина кінцевої ділянки лежить у діапазоні від 60 до 80 елементів. Враховуючи те, що часова ефективність алгоритмів залежить від архітектури комп'ютера, операційної системи, програмного середовища розробки та виконання програми, типів даних, обсягів даних та їхніх значень показники часової ефективності слід уточнювати у кожному конкретному випадку.

Ключові слова: сортування, стохастичне сортування, алгоритм, часова ефективність, експеримент.

The possibilities of improving sorting time parameters through preprocessing by stochastic sorting were investigated. The hypothesis that two-component stochastic + classical sorting outperforms classic one-component sorting in terms of time efficiency was experimentally confirmed. Sorting with different computational complexity is accepted as classical sorting algorithms: shaker sorting with computational complexity $O(n^2)$, insertions $O(n^2)$, Shell $O(n\cdot(\log n)^2)$ ... $O(n^{3/2})$, fast with optimization of ending sequences $O(n\cdot\log n)$. The greatest effect is obtained when performing comparisons using stochastic sorting in the amount of 160 percent of the array's size. Indicators of the efficiency of the exchange of two elements, as well as series of exchanges, are introduced. This allowed to determine the highest efficiency of stochastic sorting (as the first component of two-component sorting), when one element for comparison is chosen from the first part of the array and the other from the second. For algorithms with a computational complexity of $O(n^2)$ the improvement in time efficiency reached 70–80 percent. However, for Shell sort and quick sort, the stochastic presort has no positive effect, but instead increases the total sorting time, which is apparently due to the initial high efficiency of these sorting methods. The hypothesis that three-component sorting fast + stochastic + insertions would increase sorting time efficiency was not confirmed. However, during the experiment, the recommended size of the array was determined, at which point the two-component quick + insertion sort must be switched to the second component – insertion sorting. The optimal length of the ending sequences is between 60 and 80 elements. Given that algorithm time efficiency is affected by computer architecture, operating system, software development and execution environment, data types, data sizes, and their values, time efficiency indicators should be specified in each specific case.

Keywords: sorting, stochastic sorting, algorithm, time efficiency, experiment.

Вступ

У роботі досліджувалися можливості покращення часових параметрів сортувань за допомогою попередньої обробки стохастичним сортуванням.

Стохастичне сортування полягає в циклічному порівнянні випадково обраних двох елементів масиву та за необхідності їх перестановки. Звісно, стохастичне сортування неможливо застосовувати самостійно, через відсутність умови завершення й теоретичне зациклювання. Однак, була висунута гіпотеза про його ефективність як попередньої підготовки масиву для подальшого сортування одним із класичних методів. Стохастичні алгоритми надто корисні для вирішення складних оптимізаційних завдань, наприклад, при пошуку коренів складних рівнянь [1]. Було цікаво, наскільки вони можуть бути корисними у вирішенні нетрадиційних завдань, таких як сортування даних.

З метою підтвердження або спростування висунутої гіпотези експериментально досліджено двокомпонентні сортування: стохастичне → класичне.

За класичні прийняті сортування різної обчислювальної складності [2]: шейкерне, з обчислювальною складністю $O(n^2)$, вставками (відповідно до [3], найшвидше стабільне сортування класу $O(n^2)$), Шелла $O(n\cdot(\log n)^2)$ ... $O(n^{3/2})$, а також швидке з оптимізацією кінцевих ділянок $O(n\cdot\log n)$ [4].

Пов'язані роботи

Багато різних алгоритмів сортування давно відомі [5] та застосовуються для вирішення різних завдань програмування. Хоча вони досить універсальні, але мають свою сферу ефективного застосування. У зв'язку з цим тривають пошуки нових та вдосконалення існуючих алгоритмів сортування.

Наприклад, алгоритм “2 mm” [6] є модифікацією алгоритму Min-Max Bidirectional Parallel Selection Sort (MMBPSS) [7], який у свою чергу модифікує двонаправлене паралельне сортування вибором. У алгоритмі MMBPSS запропоновано використання стека задля скорочення кількості порівнянь, а в [6] – пам’ять порядку $O(n)$.

Окремо відзначимо гібридизацію алгоритмів з метою підвищення їх часової ефективності. У роботі [8] проведено експеримент із розробки гібридної програми сортування на основі використання методу вибору алгоритмів, системи машинного навчання та алгебро-алгоритмічного підходу до проєктування програм [9]. У гібридному алгоритмі виконувався вибір сортування вставками або швидкого сортування залежно від довжини та ступеня відсортованості масиву. Алгоритм побудований на основі дерева рішень, отриманого в результаті аналізу статистичних даних сортування масивів. У [10] розглядається гібридне паралельне сортування, в якому виконується сортування злиттям або вставками залежно від розміру блоку числового масиву даних. Оптимальне значення розміру блоку підбирається за допомогою системи автоматичного налагодження програм.

Як відомо, “ефективні реалізації зазвичай використовують гібридний алгоритм, що поєднує асимптотично ефективний алгоритм для загального сортування із сортуванням вставками для невеликих обсягів у нижній частині рекурсії” [6].

Також тривають роботи з дослідження ефективності алгоритмів сортування. Пропонуються додаткові показники якості. Так, у [11] визначили діапазон асимптотичної поведінки відомих алгоритмів.

Проводяться роботи з автоматизації розробки алгоритмів сортування, зокрема, на основі логіки та комбінаторики [12] тощо.

Експериментальна база

Програмні засоби. Експерименти проводилися в операційній системі Windows 10. Розроблено 32-бітову віконну програму у середовищі Delphi 10.3, що реалізує алгоритми сортування з різними варіантами випадкових перестановок. Серія експериментів запускається у вигляді окремого робочого потоку. Програма відкомпільована в режимі оптимізації коду (release), запускалася поза середовищем розробки, водночас ніякі інші віконні програми не були запущені. Мережа на час проведення експериментів вимикалася.

Технічні засоби. Експерименти виконані на ноутбуці з технічними характеристиками, наведеними у табл. 1.

Таблиця 1. Характеристики технічних засобів

Характеристики центрального процесору	
Поле	Значення
Тип	Mobile QuadCore Intel Core i7-3610QM, 3200 МГц (32 x 100)
Кількість ядер	8
Кеш L1 коду	32 КБ per core
Кеш L1 даних	32 КБ per core
Кеш L2	256 КБ per core (On-Die, ECC, Full-Speed)
Кеш L3	6 МБ (On-Die, ECC, Full-Speed)
Набір інструкцій	x86, x86-64, MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, AVX, AES
Характеристики оперативної пам’яті	
Поле	Значення
Ідентифікатор	SK hynix HMT351S6CFR8C-PB
Розмір модулю	4 ГБ (2 ranks, 8 banks)
Кількість модулів	2 x 4 ГБ

Структури даних. До сортування готувались масиви цілих 4-байтових чисел. Розміри масивів виділялися залежно від типу сортування (менші для менш ефективних сортувань, у яких є сенс їх виконання). Пам’ять під масиви виділялася безпосередньо засобами ОС Windows, викликом VirtualAlloc [13]. Сортований масив заповнювався числами від 0 до $N-1$, де N – кількість елементів масиву x_i. Після цього виконувалося перемішування шляхом багаторазового обміну місцями двох випадково обраних елементів. Кількість таких обмінів бралася $5N$ що цілком достатньо для якісного перемішування вихідного масиву та отримання випадкового розподілу елементів у масиві.

Організація експерименту.

У зв'язку з нестабільністю роботи потоків операційної системи Windows та технічних засобів, час виконання сортування одного й того ж масиву буде випадковим із деякою дисперсією.

Час сортування визначався як середній і медіанний час при *M*-кратному виконанні сортування одного й того ж масиву даних (адреса масиву в пам'яті не змінювалася, дані копіювалися з оригінального невідсортованого масиву перед кожним вимірюванням).

Перед виконанням окремого сортування (паралельного вимірювання) виконувалося очищення кешу. Для цього використовувалися такі прийоми:

– послідовний доступ до комірок заздалегідь виділеної ділянки пам'яті протягом щонайменш 10 проходів;

– виклик функції Sleep на 500 мс.

Попередні експерименти

Для оцінки необхідної кількості паралельних вимірювань (раніше позначено як *M*), яке дасть стійке значення середнього та медіанного часу сортування, був виконаний попередній експеримент з використанням швидкого сортування, яке виконувалося на одному й тому ж масиві даних 500 разів.

Попередній експеримент (див. рис. 1 і 2) показав, що для адекватної оцінки середнього та медіанного часу сортування достатньо близько 100 паралельних вимірювань для одного й того ж масиву даних. Разом з тим час сортування буде швидше завищений, ніж занижений, і відхилявся від значення для 500 паралельних вимірів не більше ніж на 0,5 %. Всі дані, які далі представлені в цій роботі, були отримані для 100 паралельних вимірювань (для кожного виду сортування, обсягу даних тощо).

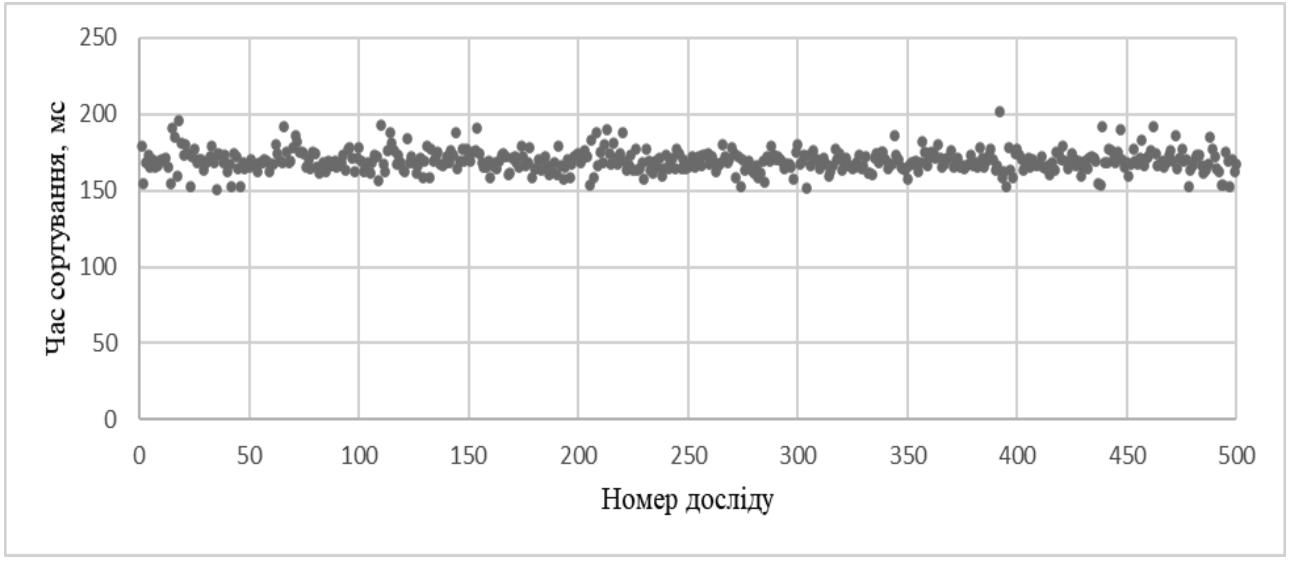

Рис. 1. Розкид результатів при сортуванні

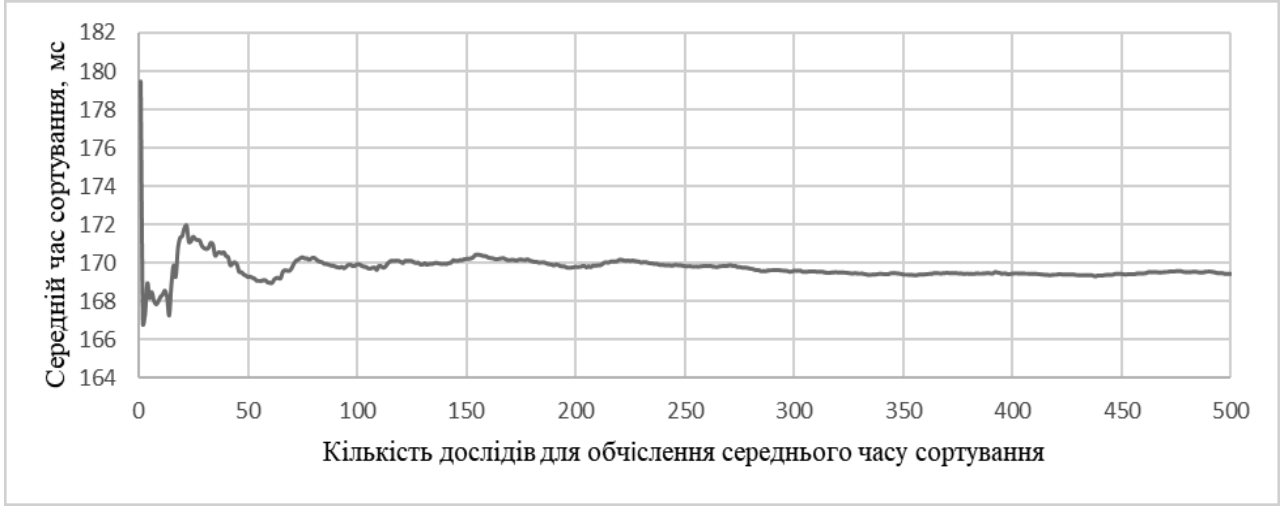

Рис. 2. Середній час сортування, розрахований наростаючим підсумком

Дослідження стохастичного сортування

Перед проведенням основної частини експериментів було досліджено ефективність упорядкування випадково обраних елементів за ступенем наближення масиву до впорядкованого. Часові показники серії обмінів на цьому етапі не визначались.

Досліджувалися два способи вибору випадкових елементів для порівняння (з можливим подальшим обміном).

Перший спосіб полягає в тому, що викидаються два рівномірно розподілені числа в діапазоні від 0 до $N - 1$. Пара елементів, які будуть порівнюватися, може бути в будь-якій частині масиву.

У *другому* перший елемент у парі обирається за рівномірним законом розподілу в діапазоні від 0 до $N / 2 - 1$, а другий від $N / 2$ до $N - 1$. Таким чином, формується пара елементів, які знаходяться в різних половинах масиву.

Для проведення експерименту був підготовлений масив з 10^5 цілих чисел від 0 до 99999, які були довільно перемішані. Елементи для порівняння вибиралися серіями по 1000 пар, послідовно виконувалося 200 серій, що склало $2 \cdot 10^5$ або 2 N порівнянь. Для кожної серії порівнянь фіксувалися такі показники:

– кількість обмінів у серії;

– середня ефективність обмінів у серії;

– невпорядкованість масиву після закінчення серії порівнянь.

На рис. 3 показано кількість перестановок для першого та другого способів. Також показано лінію тренда (за експоненційним законом). По осі X наростаючим результатом показано загальну кількість порівнянь. По осі Y показано число виконаних перестановок у серії обмінів із 1000 порівнянь.

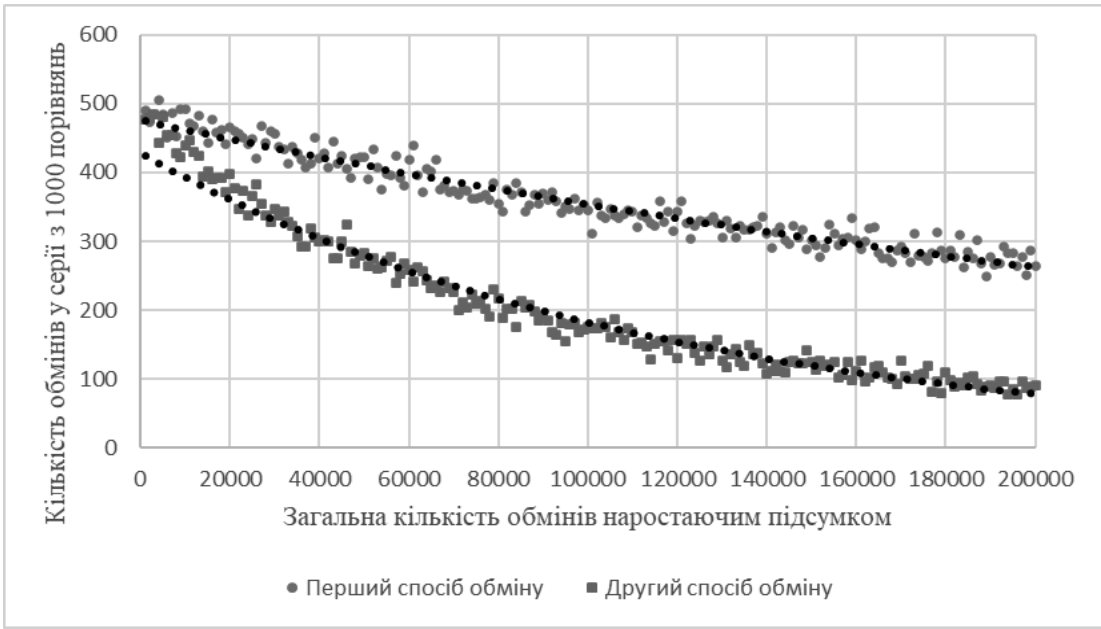

Рис. 3. Порівняння кількості обмінів в одній серії (1000 пар)

Відзначимо, що при першому способі частіше виконується обмін пари елементів. Однак, у цьому випадку необхідно виконати додаткове порівняння для визначення порядку обраних індексів i та j ($i > j$, $i < j$ або $i = j$). Для другого способу вибору індексів гарантується, що $i < j$.

Ефективність перестановки пари елементів визначається як відносна зміна положення елементів до та після перестановки до їх кінцевого положення (у відсортованому масиві):

$$e(i, j, N) = \frac{|x_i - i| - |x_i - j|}{\max(x_i, N - x_i)},$$

де i, j – індекси елементів у масиві x; N – кількість елементів у масиві. Врахуємо, що значення елемента масиву одночасно є його індексом в упорядкованому масиві (через запропонований вище спосіб формування невпорядкованого масиву).

Показник $e(i, j, N)$ визначає, наскільки елемент з індексом i стане ближчим до своєї кінцевої позиції в упорядкованому масиві x, якщо він буде переміщений у позицію з індексом j.

Слід зауважити, що ця функція дає однаковий діапазон значень незалежно від кількості елементів N у масиві x. Найгірше значення $e(0,0,N)=-1$ при x_i, а найкраще – $e(0,N,N)$при $x_i = N$.

Показник ефективності обміну $E(i, j, N)$ усереднює $e(i, j, N)$ та $e(j, i, N)$ для пари елементів, що переставляються:

$$E(i, j, N) = \frac{e(i, j, N) + e(j, i, N)}{2} * 100\%.$$

Ефективність обміну для кожного елемента пари може бути як позитивною, так і негативною. Найгірший варіант – якщо в результаті обміну обидва елементи погіршили свої позиції, найкращий варіант – обидва елементи покращили свої позиції.

На рис. 4 показана середня ефективність обмінів для першого та другого способів вибору елементів для порівняння. Також показана лінія тренду (побудована за експоненційним законом).

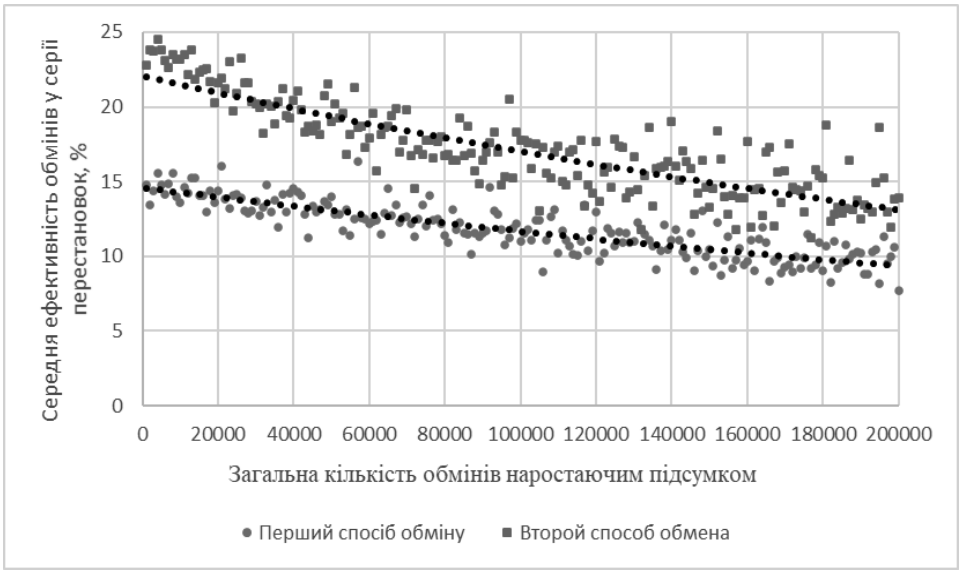

Рис. 4. Порівняння середньої ефективності обмінів
в одній серії перестановок (1000 пар) для різних способів перестановок

За показником ефективності обмінів другий спосіб перевищує перший приблизно в півтора рази.

Останній показник ефективності перестановок визначає невпорядкованість масиву після закінчення серії порівнянь:

$$U_0 = \frac{1}{N}\left(\sum_{i=0}^{N}\frac{|x_i - i|}{\max(i, N-i)}\right)*100\%.$$

На рис. 5 показана середня невпорядкованість елементів масиву для обох способів. По осі *X* наростаючим підсумком показано загальну кількість порівнянь. По осі *Y* показана середня невпорядкованість елементів масиву (у відсотках) після завершення чергової серії порівнянь (по 1000 пар).

Другий спосіб краще за ефективністю обмінів при кількості попередніх обмінів понад 160 % від загальної кількості елементів масиву.

Нижче представлені результати експериментів, у яких використовується другий спосіб обміну, за якого значення індексів обираються з першої і другої половини масиву, що сортується. Саме цей спосіб практично є найбільш вигідним, оскільки при значній кількості попередніх обмінів ($2N$ – $3N$) у нього нижчі накладні витрати за рахунок меншої кількості операторів умовного переходу.

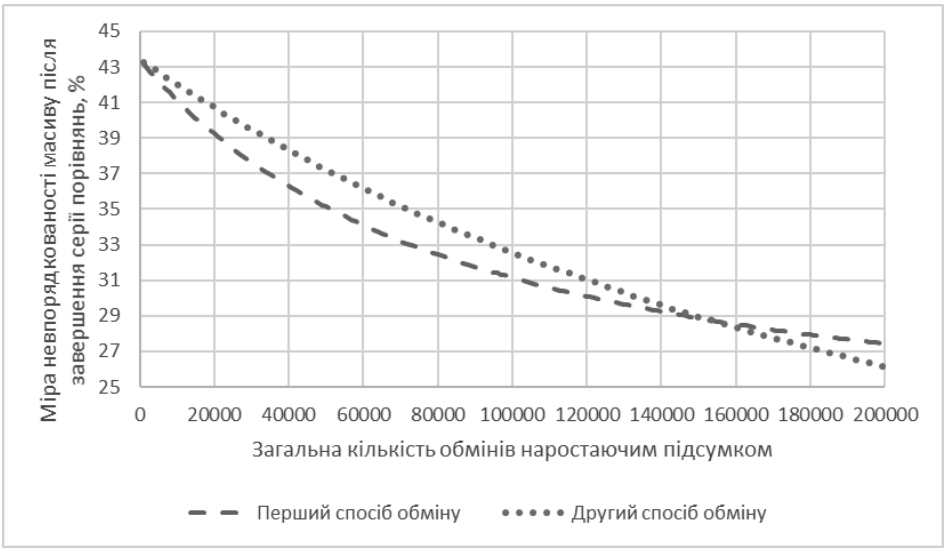

Рис. 5. Порівняння міри невпорядкованості масиву після завершення серії
порівнянь (1000 пар) для різних способів перестановок

Двохкомпонентне сортування: стохастичне → класичне

Для перевірки гіпотези щодо ефективності стохастичного сортування як попередньої обробки масиву для подальшого застосування класичного алгоритму сортування було проведено низку експериментів. Спочатку розглядалися сортування з алгоритмічною складністю $O(n^2)$, а саме, сортування вставками та шейкерне сортування.

Розмір сортованого масиву складав почергово 10^3, 10^4 та 10^5 елементів. Кількість паралельних вимірювань прийнято 100, час сортування усереднювався. Кількість порівнянь при стохастичному сортуванні варіювалася від $0{,}5N$ до $5N$ з кроком $0{,}5\ N$.

На рис. 6 та 7 показані результати проведених експериментів. По осі X показане число порівнянь у процентах від вихідного розміру сортованого масиву. По осі Y показана ефективність (%) двокомпонентного сортування порівняно з однокомпонентним. Ефективність визначалася як

$$Ef = \left(\frac{t_{клас}}{t_{стох} + t_{клас2}} - 1 \right) * 100\%.$$

де $t_{клас}$ – час однокомпонентного класичного сортування; $t_{стох}$ – час стохастичного; $t_{клас2}$ – такого ж класичного, але після стохастичного.

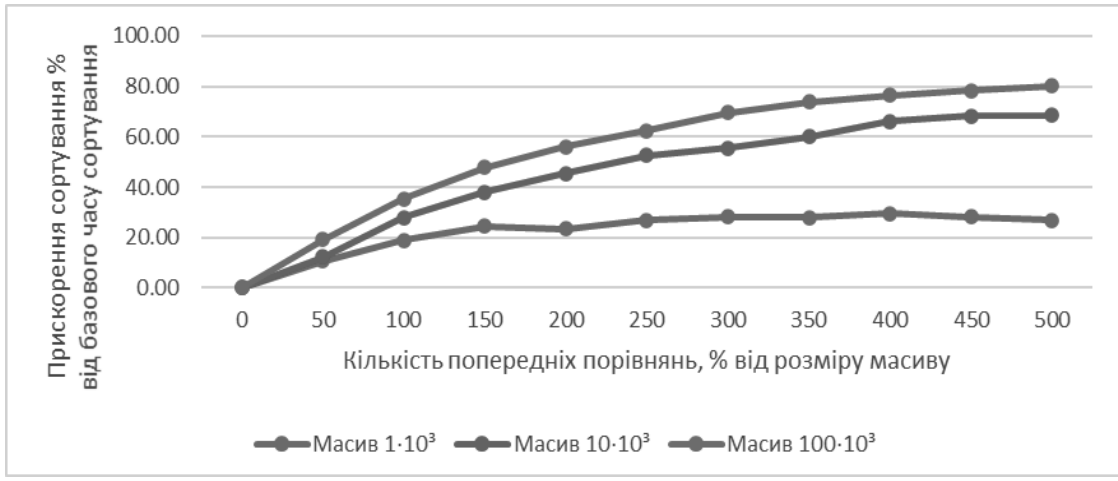

Рис. 6. Ефективність для шейкерного сортування

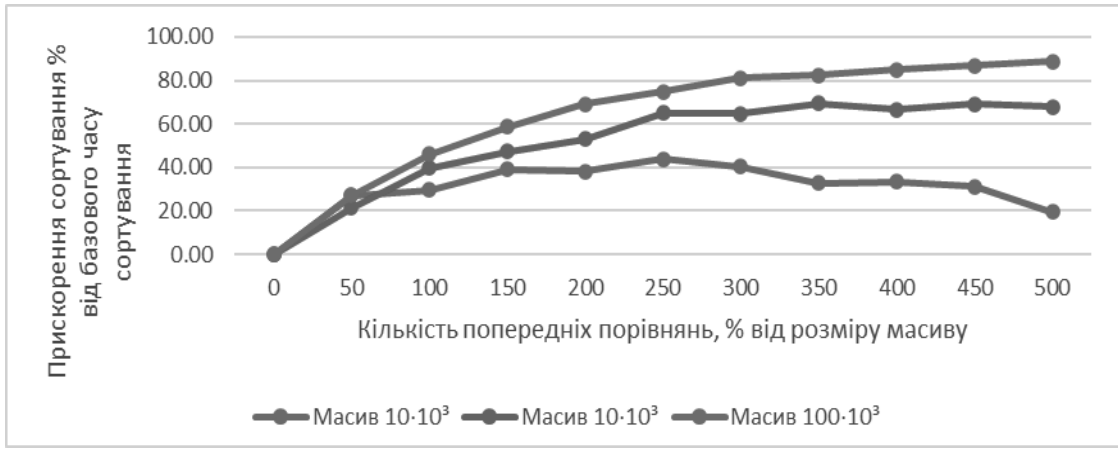

Рис. 7. Ефективність для сортування вставками

Для сортувань з алгоритмічною складністю $O(n^2)$ запропонований метод виявив високу ефективність, тим більшу, чим більший обсяг сортованого масиву.

За умови обсягу масиву від 10^4 елементів і вище кількість пар для упорядкування стохастичним методом доцільно обирати у діапазоні $3N$ – $4N$. Для розглянутих методів сортування час, що витрачається, може бути зменшено приблизно на 70–80 %.

Для масивів невеликого обсягу (менше 10^4 елементів) максимальна ефективність дорівнює приблизно 25–40 % і досягається після стохастичного впорядкування приблизно $2N$ – $2{,}5N$ пар. Далі ефективність виходить на плато (шейкерне сортування) або починає зменшуватися (сортування вставками).

Аналогічна серія експериментів проведена для сортування Шелла з обчислювальною складністю $O(n \cdot (log\ n)^2)$ ... $O(n^{3/2})$ та швидкого сортування – $O(n \cdot log\ n)$. Однак, для цих сортувань попереднє впорядкування довільно обраних пар сортованих елементів масиву вже не показує позитивну ефектив-

ність, а навпаки збільшує загальний час сортування, що, вочевидь, пояснюється початковою високою ефективністю даних методів сортування.

Двохкомпонентне сортування: швидке + вставкою

У подальших експериментах реалізація швидкого сортування передбачає, що кінцеві ділянки понад певну довжину будуть відсортовані сортуванням вставками. Таке сортування краще за швидке за умови малих обсягів даних [14]. Інші модифікації алгоритму швидкого сортування, наприклад, з двома опорними точками [15], SMS алгоритм [16] і гібридні як [17] та інші [18], нами не досліджувалися.

Викликає інтерес ефективність переходу до трикомпонентного сортування: швидке + стохастичне + вставками.

Для визначення оптимального значення, за якого має відбуватися перехід від рекурсивних викликів до сортування вставками, було здійснено ряд експериментів. Масив із $2 \cdot 10^6$ елементів сортувався швидким сортуванням без оптимізації кінцевої ділянки, а також з оптимізацією кінцевих ділянок з кількістю від 16 до 160 елементів з кроком 2. У кожному випадку проводилося 100 паралельних вимірів, результати яких усереднювалися. Порівнювався середній час, витрачений на сортування. Час швидкого сортування без оптимізації становив 212,3 мс. Витрати часу на сортування з оптимізацією при різних розмірах кінцевої ділянки показано на рис. 8. Оптимальна довжина кінцевої ділянки лежить у діапазоні від 60 до 80 елементів. У подальших експериментах даний параметр швидкого сортування брався рівним 70. Двокомпонентне сортування швидке + вставками зменшує час швидкого сортування приблизно на 20 %.

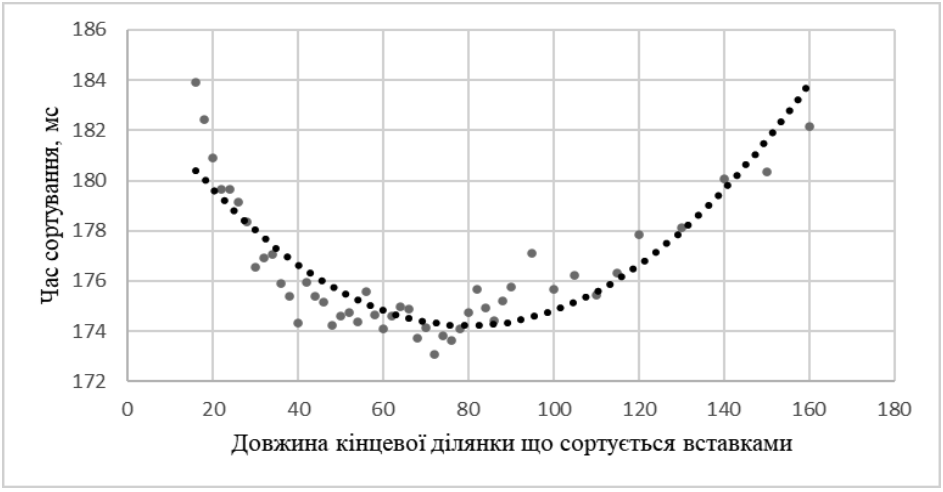

Рис. 8. Ефективність швидкого сортування при різних довжинах кінцевої ділянки, яка сортується вставками

Під час подальших експериментів було зроблено спробу додаткового збільшення ефективності швидкого сортування шляхом застосування стохастичного сортування перед сортуванням вставками на кінцевих ділянках. Для цього попередньо була проведена оцінка розподілу реальних довжин кінцевих ділянок; як максимальну довжину кінцевої ділянки вибрано 70 елементів. На рис. 9 показано розподіл обсягів масивів, що сортуються на кінцевих ділянках. Приблизно 80 % становлять масиви обсягом від 2 до 30 елементів, решта 20 % становлять масиви обсягом від 31 до 70 елементів.

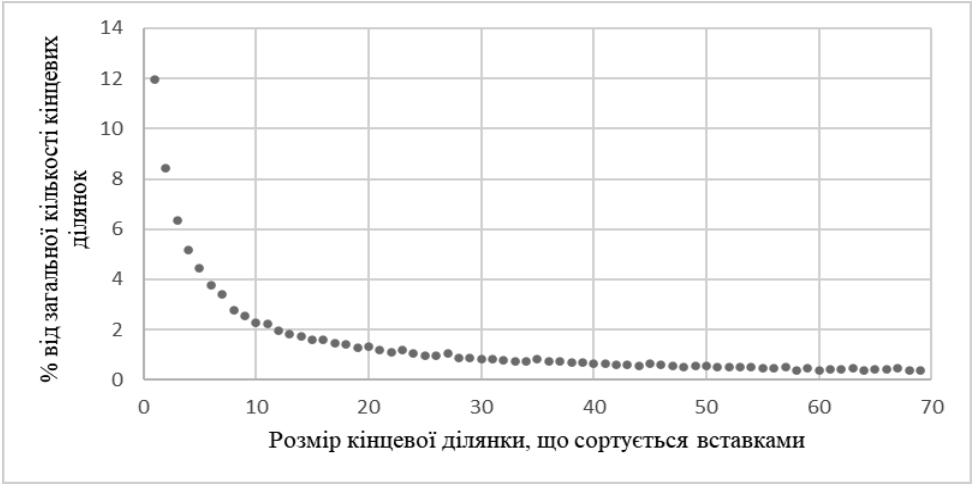

Рис. 9. Розподіл довжин масивів, що сортуються на кінцевих ділянках

Експерименти показали, що використання випадкових перестановок при сортуванні кінцевих ділянок не дає позитивного ефекту. Це пояснюється дуже короткими довжинами кінцевих ділянок (2–30 елементів у 80 % випадків). Як показано раніше, ефективність стохастичного сортування найбільше проявляється на масивах значного обсягу, тобто від 10^4 елементів та вище.

Висновки

В результаті виконаних експериментів підтверджено гіпотезу про можливість суттєвого поліпшення часової ефективності двокомпонентного сортування стохастичне + класичне з часовою ефективністю $O(n^2)$ порівняно з таким же класичним однокомпонентним.

Гіпотеза про підвищення часової ефективності сортування при трикомпонентному сортуванні швидке + стохастичне + вставками не підтвердилася. Однак, в ході експерименту встановлено рекомендований розмір масиву, за якого в двокомпонентному сортуванні швидке + вставками необхідно переходити до другої компоненти – сортуванню вставками.

Враховуючи те, що часова ефективність алгоритмів залежить від архітектури комп'ютера, операційної системи, програмного середовища розробки та виконання програми, типів даних, обсягів даних та їх значень, показники часової ефективності слід уточнювати у кожному конкретному випадку.

Література

1. Shynkarenko V., Ilchenko P., Zabula H. Tools of investigation of time and functional efficiency of bionic algorithms for function optimization problems. Proc. 11th Int. Conf. of Programming (UkrPROG 2018). 2018. CEUR Workshop Proceedings. Vol. 2139. P. 270–280.
2. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to algorithms (3rd ed.). Cambridge, MA: The MIT Press. 2009. 1292 p.
3. Yadav R., Yadav R., Gupta S. B. Comparative study of various stable and unstable sorting algorithms. Artificial Intelligence and Speech Technology. CRC Press, 2021. P. 463–477.
4. Sorting algorithm. URL: https://en.wikipedia.org/wiki/Sorting_algorithm (дата звернення 11.07.2022)
5. Knuth D. The Art of Programming, Vol. 3 Sorting and searching. 1973. ↑800 p.
6. Mubaraka A., Iqbalb S., Naeemc T., Hussaind S. 2 mm: a new technique for sorting data. Theoretical Computer Science. 2022. Vol. 910. P. 68–90.
7. Thabit K., Bawazir A. A novel approach of selection sort algorithm with parallel computing and dynamic programing concepts. JKAU: Comp. IT. 2013. Vol. 2. P. 27–44.
8. Yatsenko O. On application of machine learning for development of adaptive sorting programs in algebra of algorithms. Proc. Int. Workshop "Concurrency: Specification and Programming" (CS&P'2011). 2011. P. 577–588.
9. Doroshenko A., Yatsenko O. Formal and adaptive methods for automation of parallel programs construction: emerging research and opportunities. Hershey: IGI Global, 2021. 279 p.
10. A mixed method of parallel software auto-tuning using statistical modeling and machine learning / Doroshenko A. et al. Communications in Computer and Information Science. Information and Communication Technologies in Education, Research, and Industrial Applications. 2019. Vol. 1007. P. 102–123.
11. Durrani O. K., Shreelakshmi V., Shetty S., Vinutha, D. C. Analysis and determination of asymptotic behavior range for popular sorting algorithms. In Special Issue of International Journal of Computer Science & Informatics. 2018. Vol. 2, No. 1. P. 149–154.
12. Drămnesc I., Jebelean T., Stratulat S. Mechanical synthesis of sorting algorithms for binary trees by logic and combinatorial techniques. Journal of Symbolic Computation. 2019. Vol. 90. P. 3–41.
13. VirtualAlloc function. URL: https://docs.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc (дата звернення 11.07.2022)
14. Comparison study of sorting techniques in static data structure. / Frak A. N. et al. International Journal of Integrated Engineering: Special Issue Data Information Engineering. 2018. Vol. 10, No. 6. P. 106–112.
15. Kushagra S., López-Ortiz A., Qiao A., Munro J. I. Multi-pivot quicksort: theory and experiments. Proc. Sixteenth Workshop on Algorithm Engineering and Experiments (ALENEX). 2014. P. 47–60.
16. Mansi R. Enhanced quicksort algorithm. Int. Arab J. Inf. Technol. 2010. Vol. 7, No. 2. P. 161–166.
17. Review on performance of quick sort algorithm / Aftab A. et al. International Journal of Computer Science and Information Security. 2021. Vol. 19, No. 2. P. 114–120.
18. Preprocessing large data sets by the use of quick sort algorithm. / Woźniak M. et al. Springer: Knowledge, Information and Creativity Support Systems: Recent Trends, Advances and Solutions. 2016. P. 111–121.

References

1. SHYNKARENKO, V., ILCHENKO, P. & ZABULA, H. (2018) Tools of investigation of time and functional efficiency of bionic algorithms for function optimization problems. In *International Conference of Programming*. Kyiv, Tuesday 22th to Thursday 24th May 2018. CEUR Workshop Proceedings. 2139. p. 270-280.
2. CORMEN, T. H. et al. (2009) Introduction To Algorithms (3rd ed.). Cambridge, MA: The MIT Press, 1292 p.
3. YADAV, R., YADAV, R. & GUPTA, S. B. (2021). Comparative study of various stable and unstable sorting algorithms. *Artificial Intelligence and Speech Technology*. CRC Press.
4. WIKIPEDIA. (2022) *Sorting algorithm.* [Online] July 2022. Available from: URL: https://en.wikipedia.org/wiki/Sorting_algorithm [Accessed: 11th July 2022].
5. KNUTH, D. (1973) The Art of Programming, Vol. 3 Sorting and Searching.
6. MUBARAKA, A., IQBALB, S., NAEEMC, T. & HUSSAIND S. (2022) 2 mm: a new technique for sorting data. *Theoretical Computer Science*. 910. p. 68-90.
7. THABIT, K. & BAWAZIR, A. A (2013) Novel approach of selection sort algorithm with parallel computing and dynamic programing concepts. *Journal of King Abdulaziz University Computing and Information Technology Sciences*. 2. p. 27-44.
8. YATSENKO, O. (2011) On application of machine learning for development of adaptive sorting programs in algebra of algorithms. *Proc. Int. Workshop "Concurrency: Specification and Programming"* (*CS&P'2011*). Pułtusk, Poland, 28-30 September 2011. Bialystok: Bialystok University of Technology. p. 577-588.

9. DOROSHENKO, A. & YATSENKO, O. (2021) Formal and Adaptive Methods for Automation of Parallel Programs Construction: Emerging Research and Opportunities. Hershey: IGI Global.
10. DOROSHENKO, A. et al. (2019) A mixed method of parallel software auto-tuning using statistical modeling and machine learning. *Communications in Computer and Information Science. Information and Communication Technologies in Education, Research, and Industrial Applications*. 1007. p. 102-123.
11. DURRANI, O. K., SHREELAKSHMI, V., SHETTY, S. & VINUTHA, D. C. (2018) Analysis and determination of asymptotic behavior range for popular sorting algorithms. *Special Issue of International Journal of Computer Science & Informatics*. 2(1). p. 149-154.
12. DRĂMNESC, I., JEBELEAN, T. & STRATULAT, S. (2019). Mechanical synthesis of sorting algorithms for binary trees by logic and combinatorial techniques. *Journal of Symbolic Computation*. 90. p. 3-41.
13. MICROSOFT DOCUMENTS. (2022) *VirtualAlloc function*. [Online] May 2022. Available from: https://docs.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc [Accessed: 11th July 2022].
14. FRAK, A. N. et al. Comparison study of sorting techniques in static data structure. *International Journal of Integrated Engineering: Special Issue Data Information Engineering.* 10(6). p. 106-112.
15. KUSHAGRA, S., LÓPEZ-ORTIZ, A., QIAO, A. & MUNRO, J. I. (2014) Multi-pivot quicksort: theory and experiments. In *Workshop on Algorithm Engineering and Experiments*. Portland, Oregon, USA, Sunday 5th January 2014, p. 47-60.
16. MANSI R. (2010) Enhanced quicksort algorithm. *International Arab Journal of Information Technology*. 7(2). p. 161-166.
17. AFTAB, A. et al. (2021) Review on performance of quick sort algorithm. *International Journal of Computer Science and Information Security*. 19(2). p. 114-120.
18. WOŹNIAK, M. et al. (2016) Preprocessing large data sets by the use of quick sort algorithm. In *Knowledge, Information and Creativity Support Systems: Recent Trends, Advances and Solutions*. Springer.

Одеожано 19.07.2022

***Про авторів*:**

[1]*Шинкаренко Віктор Іванович*,
доктор технічних наук, професор.
Кількість наукових публікацій в українських виданнях – понад 250.
Кількість наукових публікацій в зарубіжних виданнях – 15.
Індекс Хірша – 5.
http://orcid.org/ 0000-0001-8738-7225,

[2]*Дорошенко Анатолій Юхимович*,
доктор фізико-математичних наук, професор, завідувач відділу.
Кількість наукових публікацій в українських виданнях – понад 180.
Кількість наукових публікацій в зарубіжних виданнях – понад 60.
Індекс Хірша – 6.
http://orcid.org/0000-0002-8435-1451,

[2]*Яценко Олена Анатоліївна,*
кандидат фізико-математичних наук, старший науковий співробітник.
Кількість наукових публікацій в українських виданнях – 49.
Кількість наукових публікацій в іноземних виданнях – 19.
Індекс Хірша – 3.
http://orcid.org/0000-0002-4700-6704,

[1]*Разносілін Валентин В'ячеславович,*
асистент.
Кількість наукових публікацій в українських виданнях – 10.
http://orcid.org/0000-0002-4463-4588,

[1]*Галанін Костянтин Костянтинович,*
магістр.
https://orcid.org/0000-0001-9296-4808.

Місце роботи авторів:

[1]Український державний університет науки і технологій
49010, м. Дніпро, вул. академіка Лазаряна, 2
Тел.: (056)-373-15-35
E-mail: shinkarenko_vi@ua.fm,
valentin.raznosilin@gmail.com,
konsta.home@gmail.com

[2]Інститут програмних систем НАН України,
03187, м. Київ, проспект Академіка Глушкова, 40.
Тел.: (044) 526 3559.
E-mail: doroshenkoanatoliy2@gmail.com,
oayat@ukr.net.

Прізвища та ініціали авторів доповіді українською мовою:
Шинкаренко В. І., Дорошенко А. Ю., Яценко О. А., Разносілін В. В., Галанін К. К.
Двохкомпонентні алгоритми сортування

Прізвища та ініціали авторів доповіді англійською мовою:
Shynkarenko V. I., Doroshenko A. Yu., Yatsenko O. A., Raznosilin V. V., Halanin K. K.
Bicomponent sorting algorithms

Відповідальний виконавець: Шинкаренко Віктор Іванович
Тел. 063 489 49 15 E-mail: shinkarenko_vi@ua.fm