КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 1(21), 2023
ISSN 2663 - 4023
DOI 10.28925/2663-4023.2023.21.8698
УДК 004.424.5

Трофименко Олена Григорівна
кандидат технічних наук, доцент, доцент кафедри інформаційних технологій,
Національний університет "Одеська юридична академія", м. Одеса, Україна,
ORCID ID 0000-0001-7626-0886
trofymenko@onua.edu.ua

Прокоп Юлія Віталіївна
кандидат історичних наук, доцент, доцент кафедри інженерії програмного забезпечення
Національний університет "Одеська політехніка", м. Одеса, Україна,
ORCID ID 0000-0002-6608-3668
prokop.y.v@op.edu.ua

Чепурна Олена Євгенівна
кандидат фізико-математичних наук, доцент, доцент кафедри кібербезпеки,
Національний університет "Одеська юридична академія", м. Одеса, Україна,
ORCID ID 0000-0002-1432-0799
chepurna@onua.edu.ua

Корнійчук Микола Миколайович
студент факультету кібербезпеки та інформаційних технологій,
Національний університет «Одеська юридична академія», Одеса, Україна,
ORCID ID 0009-0009-4223-9291
chuguystyr@gmail.com

ПОРІВНЯННЯ ШВИДКОДІЇ АЛГОРИТМІВ СОРТУВАННЯ У РІЗНИХ МОВАХ ПРОГРАМУВАННЯ

Анотація. Сортування як один із базових алгоритмів має широкий спектр застосування під час розробки програмного забезпечення. Зі зростанням обсягів даних, що опрацьовуються, значно зростає потреба у засобах швидкого та ефективного впорядкування даних. Існує велика кількість різноманітних алгоритмів сортування та їхніх розширень. Однак серед них неможливо вибрати оптимальний та універсальний. Усі ці алгоритми мають свою специфіку роботи, яка зумовлює сферу ефективного використання. Тому актуальною є проблема вибору оптимального алгоритму для певних специфічних умов. Цей вибір є часто нетривіальною задачею, а невдалий вибір алгоритму може спричинити проблеми зі швидкодією опрацювання даних. Щоб визначити, який саме алгоритм буде найкращим в конкретній ситуації, потрібно проаналізувати усі фактори, які впливають на роботу алгоритмів: розмір та структуру набору даних, діапазон значень його елементів, форму доступа (довільна чи послідовна), ступінь впорядкованості, розмір додаткової пам'яті, яка необхідна для виконання алгоритму, тощо. Крім того, різні алгоритми мають неоднакову швидкодію у різних мовах програмування. У дослідженні проаналізовані переваги та недоліки дев'яти популярних алгоритмів сортування (бульбашкою, вставкою, вибором, Шелла, злиттям, швидке, підрахунком, за розрядами, купою), зумовлені їхньою специфікою, та обмеження щодо можливого використання. Протестована швидкодія зазначених алгоритмів, реалізованих чотирма популярними мовами програмування (C++, C#, Java і JavaScript). Експериментально з'ясовано, що швидкодія алгоритмів сортування має відмінності залежно від мови програмування. Прикладний аспект дослідження полягає в тому, що його висновки і результати дозволять розробникам вибирати найкращий алгоритм для певної мови програмування, залежно від розміру, діапазону, структури тощо набору даних, який треба відсортувати. Врахування цього є важливим, коли виникає потреба перегрупування великих обсягів даних у пошукових системах, наукових та інженерних застосуваннях. Адже ефективність алгоритму сортування суттєво впливає на загальну продуктивність системи.

Keywords: алгоритми сортування, О-нотація, час роботи, продуктивність, сортування.

ВСТУП

Алгоритми сортування широко використовуються розробниками програмного забезпечення для ефективного перегрупування великих обсягів даних, наприклад, у пошукових системах, наукових та інженерних застосуваннях. Вони часто використовуються для підвищення ефективності інших алгоритмів, які потребують на вході відсортованих даних для своїх операцій. У багатьох системах впорядкування є важливою частиною конвеєра оброблення даних, а ефективність алгоритму сортування суттєво впливає на загальну продуктивність цих систем. Зі зростанням обсягів оброблюваних даних питання вибору найефективнішого алгоритму за певних умов стає дедалі більш гострим й актуальним.

Постановка проблеми. Кожен із численної групи різних алгоритмів сортування має свою специфіку і зумовлені особливостями реалізації переваги та недоліки. Вибір оптимального алгоритму за певних даних та умов може стати нетривіальною задачею, а невдалий вибір алгоритму впорядкування може спричинити проблеми зі швидкодією опрацювання даних.

Поняття «Big-O notation» вживається для позначення часової складності, як залежності між часом роботи алгоритму та обсягом вхідних даних або необхідним обсягом пам’яті в найгіршому випадку. О-нотації обґрунтовано визначені для всіх відомих алгоритмів сортування, і вони не залежать від мови програмування, якою реалізується алгоритм [1]. Використання О-нотацій є поширеною практикою серед програмістів для вибору ефективних алгоритмів у широкому колі сфер розробки [2]: від інтернету речей, науки про дані та аналітики даних до машинного навчання та кібербезпеки. Водночас на практиці виникають певні неоднозначності та складнощі під час вибору того чи іншого алгоритму сортування за певних умов та обставин. Адже на цей вибір впливає не лише оцінка О-нотації, а й інші чинники: обсяги необхідної та наявної апаратної пам’яті, розмір, діапазон та структура вхідних даних, ступінь їх впорядкованості, а також мова та середовище програмування.

Аналіз останніх досліджень і публікацій. Оскільки з потребою впорядкування даних розробники програмного забезпечення стикаються доволі часто, то й актуальність швидкого й ефективного добору алгоритму сортування підтверджується численними дослідженнями у цій сфері. Так, робота [3] зосереджена на вивченні швидкодії п’яти алгоритмів сортування (швидке, купою, злиттям, інтроспективне та за розрядами) для 11 000 елементів, реалізованих засобами мови програмування Python, де робота кожного алгоритму перевірялася до п’яти разів поспіль. У статтях [4] і [5] виконано порівняння ефективності двох, а в статті [6] – трьох алгоритмів сортування засобами мови Java. Слід зауважити, що роботи [3] – [6] обмежуються дослідженням алгоритмів сортування, реалізованих лише однією мовою програмування, і не враховують можливих відмінностей, які залежать від мови. У статті [7] без урахування мови програмування порівнюються три (злиттям, швидке, купою), а в дослідженні [8] – чотири різні алгоритми сортування (злиттям, швидке, вставкою та бульбашкою) на основі різних параметрів, як-от: найкраща і найгірша часові складності, стабільність і просторова складність. У дослідженні [9] виконано асимптотичні вимірювання продуктивності алгоритмів сортування для псевдовипадкових даних у діапазоні від 10 000 до 100 000 засобами трьох мов C++, Python і Java. Його автори дійшли висновків: 1) при реалізації засобами Python сортування злиттям є кращим, аніж швидке сортування; 2) при реалізації Java швидке сортування має швидкодію кращу за інші алгоритми; 3) при реалізації C++ найкращі показники показує теж швидке сортування, при цьому результати, отримані в C++, є кращими, аніж в Java та Python для усіх п’яти розглянутих

алгоритмів сортування. Дослідження [10] присвячене порівнянню часу виконання двох алгоритмів (злиття та бульбашкою) для масивів розміром 500, 5 000, 7 500 і 10 000 при реалізації чотирма мовами: JavaScript, PHP, Python та C. У роботі [11] оцінювали час і витрати пам'яті на виконання трьох алгоритмів сортування (бульбашкою, вставкою, вибором) мовою C++ для масивів розміром до 200 000 елементів. А в статті [12] порівнювали швидкодію п'яти алгоритмів (бульбашкою, вставкою, вибором, злиттям та швидке), а також їх покращених версій, реалізованих мовою C++. З'ясовано, що для 1000 елементів найкращим вибором є покращене швидке сортування, а для невеликих наборів даних добре підходять покращене сортування вибором і швидке. Сортування бульбашкою доцільне тільки для малих, майже впорядкованих масивів, а сортування вставкою непогано справляється з малими, повністю невпорядкованими масивами. У роботі [13] порівнювали швидкодію таких само п'яти алгоритмів, реалізованих мовами C++ і Java, для масивів розміром до 500 000 елементів. В обох мовах для великих масивів найкращі результати на випадковому наборі даних показали сортування злиттям та швидке, причому реалізація мовою C++ було швидшою за Java. У статті [14] досліджується вплив реалізації мовами Java та C на час виконання шести алгоритмів сортування (бульбашкою, вставкою, вибором, купою, злиттям та швидке) псевдовипадкових масивів цілих чисел розміром від 100 000 до 5 000 000. Її автори вважають, що питання про те, яка комбінація алгоритму сортування та мови реалізації є найбільш ефективною для вирішення задач, залишається відкритим.

Отже, численні публікації свідчать, що у науковців є потреба досліджувати швидкодію алгоритмів сортування задля ефективного вибору того чи іншого алгоритму за певних умов та даних. І оптимальне рішення все ще не знайдено. Проте більшість наявних досліджень або обмежуються малою кількістю розглянутих алгоритмів сортування, або досліджують швидкодію цих алгоритмів за умови реалізації малої кількості мов програмування, або взагалі ігнорують можливу залежність здобутих результатів від мови програмування.

Метою статті є порівняльний аналіз дев'яти найбільш популярних алгоритмів сортування (бульбашкою, вставкою, вибором, Шелла, злиттям, швидке, підрахунком, за розрядами, купою), імплементованих мовами програмування, найбільш затребуваними на сьогодні на ринку IT-праці. Прикладний аспект дослідження полягає у виявленні того, як реалізація конкретною мовою програмування впливає на фактичний час виконання алгоритму для масивів псевдовипадкових чисел різних розмірів.

МЕТОДИКА ДОСЛІДЖЕННЯ

1. Визначення переліку мов програмування для дослідження

Вибір мов програмування для цього дослідження зумовлений тією роллю, яку ці мови наразі відіграють у сферах розробки програмного забезпечення, та їх показниками у топі рейтингів мов програмування 2023 року. Були розглянуті індекси популярності мов програмування PYPL [15], TIOBE [16], RedMonk [17], IEEE Spectrum [18] та DOU [19]. Їх аналіз дозволив сформувати для дослідження список із п'яти найпопулярніших мов програмування: Python, C++, C#, Java та JavaScript.

Після низки досліджень було вирішено виключити мову Python із розгляду. Справа в тому, що одним із потужних і швидких обчислювальних засобів Python, який на практиці широко використовуються розробниками для опрацювання даних, є бібліотека NumPy. У її гнучко налаштовуваній функції Sort можна окремим аргументом власноруч задавати алгоритм сортування: 'quicksort', 'mergesort', 'heapsort' або 'timsort' [20].

Усталене значення аргументу, тобто його відсутність задає швидке сортування, яке у версії 1.12.0 було змінено на інтроспективне сортування (introsort). При цьому, якщо сортування не досягає достатнього прогресу, то відбувається перемикання на тип ‘heapsort’. Значення аргументу ‘stable’ автоматично змінює алгоритм сортування і вибирає найкращий продуктивний і стабільний алгоритм сортування для певного типу даних. Тому для розробника на Python немає потреби власноручного добору найкращого за продуктивністю алгоритму сортування.

Через це перелік мов програмування у дослідженні швидкодії алгоритмів сортування було звужено до чотирьох: C++, C#, Java та JavaScript.

2. Аналіз переліку алгоритмів сортування для дослідження

Контраст швидкодії алгоритмів можна продемонструвати, залучаючи до дослідження алгоритми, що належать до різних класів часової складності, і досліджуючи водночас декілька алгоритмів, що належать до одного класу складності, розширюючи перелік алгоритмів так, щоб включити у нього приклади алгоритмів з усіх основних класів: $O(n^2)$, $O(n\log(n))$, $O(nk)$, $O(n+k)$. Саме тому для дослідження було вибрано по декілька популярних представників цих класів часової складності. У табл. 1 зібрано основні характеристики найбільш поширених алгоритмів сортування.

Таблиця 1

Основні характеристики базових алгоритмів сортування

Алгоритми сортування	Часова складність			Найгірша просторова складність
	найгірша	середня	краща	
Сортування вибором	$O(n^2)$	$\Theta(n^2)$	$\Omega(n^2)$	1
Сортування бульбашкою	$O(n^2)$	$\Theta(n^2)$	$\Omega(n)$	1
Сортування вставкою	$O(n^2)$	$\Theta(n^2)$	$\Omega(n)$	1
Сортування Шелла	$O(n^2)$	$\Theta(n(\log(n))\text{^}2)$	$\Omega(n\log(n))$	1
Швидке сортування	$O(n^2)$	$\Theta(n\log(n))$	$\Omega(n\log(n))$	$\log(n)$
Сортування злиттям	$O(n\log(n))$	$\Theta(n\log(n))$	$\Omega(n\log(n))$	n
Сортування купою	$O(n\log(n))$	$\Theta(n\log(n))$	$\Omega(n\log(n))$	1
Сортування за розрядами	$O(nk)$	$\Theta(nk)$	$\Omega(nk)$	$n+k$
Сортування підрахунком	$O(n+k)$	$\Theta(n+k)$	$\Omega(n+k)$	$n+k$

При розгляді часової складності алгоритмів беруть до уваги найкращий $\Omega(n)$, середній $\Theta(n)$ і найгірший $O(n)$ сценарії. Для звичайних алгоритмів послідовного сортування хороша поведінка дорівнює $O(n\log(n))$, а погана – $O(n^2)$. Використання пам’яті для алгоритмів сортування «на місці» потребує однієї комірки пам’яті для обміну даних.

Не існує ідеального та універсального алгоритму сортування. Усі зібрані в табл. 1 алгоритми доволі різні й мають свою специфіку роботи. Так, сортування вибором та вставкою краще працюють для невеликих масивів, оскільки часова складність сортування цих алгоритмів становить $O(n^2)$, що робить їх неефективним для великих наборів даних. Однак на малих масивах різниця між повільними та більш швидкими алгоритмами не дуже відчутна.

Сортування вставкою є стабільним алгоритмом (тобто рівні за значенням елементи не переміщуються) і швидко працює, коли список є майже відсортований, адже тоді його часова складність становить $O(n)$. Сортування бульбашкою відповідно до специфіки його роботи є найшвидшим алгоритмом для малого або майже відсортованого набору даних. Він добре працює і з великими наборами даних, де елементи майже відсортовані,

оскільки тоді потрібна лише одна ітерація, щоб визначити, відсортовано список чи ні. Але, якщо список значною мірою не відсортований, тоді цей алгоритм має сенс використовувати лише для невеликих наборів даних або списків. Для алгоритмів сортування вибором, бульбашкою та вставкою використання пам’яті є мінімальним, оскільки його просторова складність становить O(1).

На відміну від сортування вставкою, сортування Шелла не є стабільним сортуванням і виконується швидше, коли вхідні дані частково відсортовані.

Сортування купою (Heapsort, пірамідальне сортування) використовує структуру даних купа, яка є бінарним сортувальним деревом. Оптимізована продуктивність, ефективність і точність є одними з найкращих якостей цього алгоритму. Алгоритм сортування купою не залежить від стану впорядкованості масиву. Для роботи йому не потрібен додатковий простір пам’яті, на відміну від сортування злиттям або рекурсивного швидкого сортування. До недоліків сортування купою можна віднести його нестабільність, високу вартість і неефективність при роботі зі складними даними. Крім того, алгоритм не розпаралелюється і може бути реалізований тільки на структурах із прямим доступом до елементів за індексом. На зв’язних списках такий алгоритм реалізувати не вдасться. Сортування купою буде ефективнішим на великих обсягах даних. Якщо даних небагато, сортування Шелла його обжене.

Швидке сортування і сортування злиттям використовують стратегію DAC (divide and conquer – розділяй і володарюй). Обидва мають середню часову складність $\Theta(n \log(n))$. Сортування злиттям зазвичай використовується для надто великого набору даних для зберігання чи оброблення у внутрішній пам’яті [4].

Хоча у найгіршому випадку швидке сортування має часову складність $O(n^2)$, на практиці впровадження цього алгоритму часто призводить до швидшого впорядкування, ніж алгоритми сортування зі складністю $O(n \log(n))$. Швидке сортування працює краще для частково відсортованих масивів, аніж для заповнених випадково. Для великих наборів даних швидке сортування виявляється неефективним через рекурсивне використання пам’яті. Зазвичай використовується для наборів даних з довільним доступом (масивів).

Сортування злиттям (Merge Sort) показує гарні результати для зв'язних списків, оскільки звертається до даних послідовно і потреба у довільному доступі є низькою. Цей алгоритм широко використовується для зовнішнього сортування, де довільний доступ може бути дуже і дуже дорогим порівняно з послідовним доступом.

Всі алгоритми порівняльного сортування вимагають $O(n \log(n))$ [5]. При великих n їх ефективність швидко знижується і змінити цю ситуацію неможливо. Продуктивність алгоритмів, які не зосереджені на порівняннях, часто краща. Сортування за розрядами (Radix sort) є алгоритмом без порівняння з лінійною часовою складністю. Цей алгоритм є швидшим, коли діапазон елементів масиву відносно вузький, але кількість елементів для сортування велика. Його обчислювальна складність є $O(nk)$, де k кількість проходів масивом. Якщо n доволі велике, а k навпаки дуже мале, то даний алгоритм виграє у $O(n \log(n))$. Він також набагато швидший, коли алгоритм виконується одночасно на паралельних машинах. З іншого боку, Radix sort потребує простору для допоміжного масиву, який воно використовує для сортування. Просторова складність сортування за розрядами суттєво більша, порівняно зі швидким сортуванням, яке виконує сортування на місці. Крім того, реалізація сортування за розрядами відрізняється для різних типів даних, і тому цей алгоритм є менш гнучким, ніж інші алгоритми сортування.

Сортування підрахунком (Counting sort) теж не є алгоритмом на основі порівняння, він гешує значення в тимчасовому масиві підрахунків і використовує їх для сортування. Підрахункове сортування є ефективним, якщо діапазон значень вхідних даних є меншим

за кількість об’єктів, які потрібно сортувати, а також за умови, що набір даних містить багато однакових елементів.

Щоб вирішити, який з алгоритмів сортування буде найкращим у конкретній ситуації, потрібно проаналізувати усі фактори, які впливають на їх роботу: кількість елементів масиву, діапазон та розподіл значень елементів, формат, довжина і складність структури даних (масив, список), ступінь впорядкованості (наскільки елементи вже відсортовані), вимоги до додаткової пам’яті тощо. У табл. 2 зібрано та систематизовано орієнтовні обмеження можливих сфер використання, зумовлені специфікою відповідних алгоритмів сортування.

Таблиця 2

Сфери застосування базових алгоритмів сортування

Специфіка та обмеження використання	**Алгоритми сортування**								
	бульбашкою	**вибором**	**вставкою**	**Шелла**	**злиттям**	**швидке**	**купою**	**за розрядами**	**підрахунком**
Невеликі масиви	+	+	+	+	+	+	+	+	+
Великі масиви	–	–	–	+	+	–	+	+	+
Повністю невідсортовані масиви	–	–	–	+	+	–	+	+	+
Частково або майже відсортовані масиви	+	+	+	+	+	+	+	+	+
Обмежена пам’ять	+	+	+	+	–	–	+	–	–
Стабільність (не переміщуються однакові елементи)	+	+	+	–	+	–	–	+	+
Складні структуровані дані	+	+	+	+	+	+	–	+	+
Набори даних з послідовним доступом	+	+	+	+	+	–	–	+	+
Діапазон елементів масиву відносно вузький, невелика кількість різних елементів (ключів) у масиві	+	+	+	+	+	+	+	+	+
Діапазон елементів масиву відносно широкий	+	+	+	+	–	+	+	–	–

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

У практичні частині дослідження ми порівнювали фактичний час роботи кожного алгоритму для наборів псевдовипадкових чисел від 100 до 100 000 елементів для кожної з вибраних мов програмування: C++, Java, JavaScript та C#. Числові 64-бітні цілі псевдовипадкові значення були згенеровані вбудованими засобами кожної конкретної мови. Розміри масивів для всіх мов однакові: {100, 300, 500, 1000, 1500, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000, 15000, 20000, 25000, 30000, 35000, 40000, 50000, 60000, 70000, 80000, 90000, 100000}. Апаратно-програмна складова дослідження: ноутбук на базі процесора Ryzen 5 5600H; Java – JDK 20; C++ – LLVM GCC Compiler 16.0; C# – .NET 7.0; JS – Node.js 19.8.1. Часом вважали середнє арифметичне значення п’яти вимірювань часу роботи алгоритму, оскільки цей підхід демонструє свою ефективність у багатьох дослідженнях [3] – [11]. Одиниці виміру – мілісекунди (у випадку, якщо вбудована функція повертала результат в інших одиницях, він підлягав конвертації).

Функції та типи результатів функцій, використаних для визначення часу в різних мовах:

– JS – process.hrtime.bigint() – повертає системний час у високій роздільній здатності у наносекундах у BigInt;
– Java – Sysytem.TimeNano() – повертає час у наносекундах у long;

– C++ – auto startTime = std::chrono::high_resolution_clock::now() – метод бібліотеки chrono, що повертає час у високій роздільній здатності, обрахування у потрібних одиницях відбувається автоматично із застосуванням одиниці вимірювання;
– C# – методи класу StopWatch, який відноситься до System.Diagnostics, повертає тип TimeSpan і в подальшому конвертується у рядок.

Аналіз результатів програмної реалізації алгоритмів сортування масиву псевдовипадкових чисел мовою Java показує, що найповільнішим, незалежно від кількості елементів, є сортування бульбашкою, а найшвидшим – залежно від розміру масиву або швидке сортування (для масиву до 2 тис. елементів включно), або сортування підрахунком для масивів від 3 тис. елементів (табл. 2, найкращі результати позначено напівжирним шрифтом), при цьому часова різниця між алгоритмами є суттєвою і зростає стрімко зі зростанням кількості елементів.

Таблиця 2

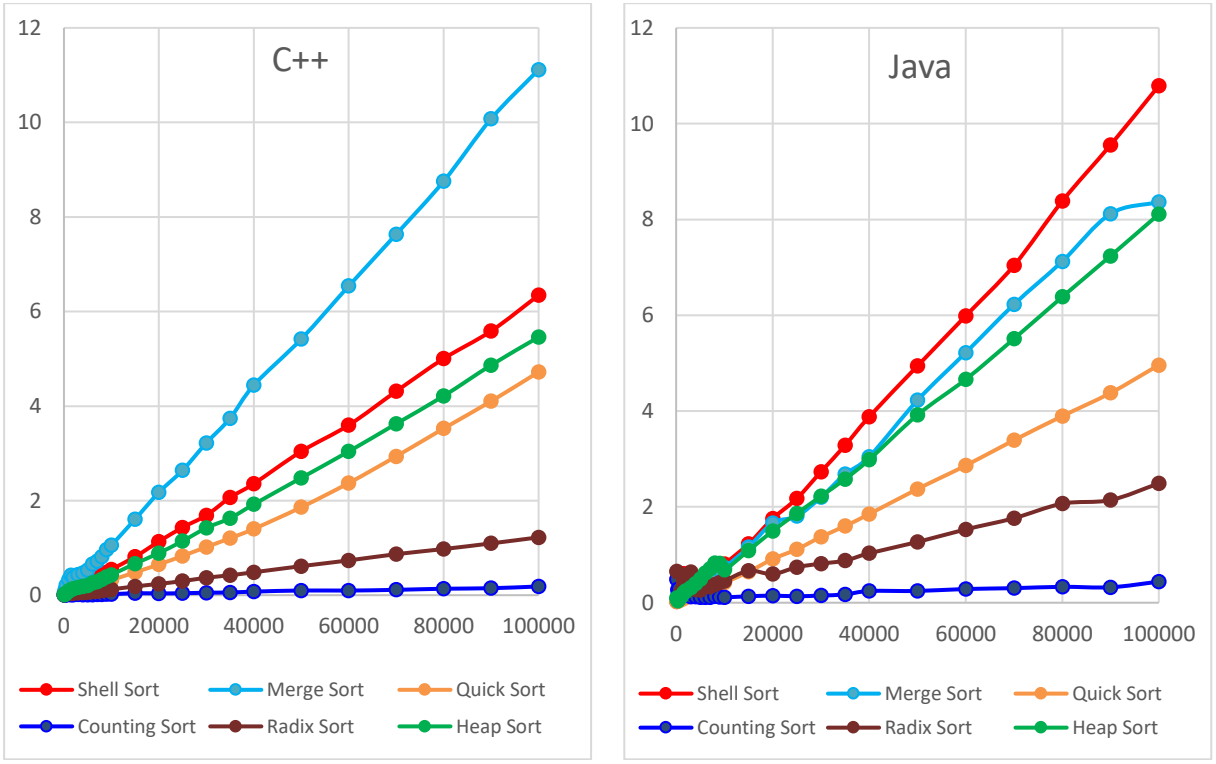
Швидкодія алгоритмів сортування при реалізації засобами мови Java

Розмір масиву	Сортування								
	бульбашкою	**вибором**	**вставкою**	**Шелла**	**злиттям**	**швидке**	**підра-хунком**	**за роз-рядами**	**купою**
100	0,140	0,061	0,047	0,023	0,059	**0,026**	0,487	0,656	0,098
300	1,157	0,492	0,372	0,089	0,159	**0,089**	0,275	0,102	0,035
500	0,867	0,584	0,867	0,158	0,064	**0,035**	0,284	0,123	0,064
1000	1,005	0,791	0,540	0,365	0,105	**0,063**	0,218	0,229	0,117
1500	2,765	2,075	0,413	0,606	0,148	**0,087**	0,186	0,365	0,199
2000	4,590	2,856	0,305	0,564	0,193	**0,112**	0,155	0,441	0,248
3000	7,520	2,596	0,438	0,302	0,227	0,140	**0,126**	0,641	0,312
4000	7,154	3,876	0,894	0,470	0,336	0,181	**0,127**	0,270	0,397
5000	11,161	5,244	1,242	0,507	0,434	0,215	**0,112**	0,263	0,495
6000	16,272	7,394	1,714	0,487	0,446	0,257	**0,113**	0,357	0,619
7000	23,911	9,698	2,307	0,545	0,524	0,303	**0,111**	0,335	0,707
8000	35,988	13,028	3,003	0,748	0,592	0,344	**0,131**	0,379	0,824
9000	47,987	16,088	3,732	0,733	0,689	0,385	**0,116**	0,419	0,821
10000	61,794	18,493	4,410	0,805	0,733	0,418	**0,111**	0,454	0,689
15000	164,331	41,731	9,764	1,228	1,159	0,644	**0,134**	0,665	1,088
20000	325,920	72,732	17,811	1,753	1,664	0,911	**0,147**	0,598	1,497
25000	550,950	112,381	27,372	2,175	1,807	1,115	**0,135**	0,741	1,864
30000	834,038	163,113	39,402	2,733	2,202	1,373	**0,150**	0,815	2,222
35000	1183,037	220,517	53,129	3,284	2,682	1,601	**0,173**	0,879	2,581
40000	1575,846	286,796	69,567	3,882	3,046	1,850	**0,244**	1,032	2,987
50000	2531,741	449,920	109,792	4,942	4,226	2,373	**0,245**	1,267	3,920
60000	3710,014	645,044	158,911	5,988	5,220	2,864	**0,286**	1,531	4,662
70000	5111,171	875,461	217,096	7,040	6,227	3,393	**0,303**	1,764	5,510
80000	6760,528	1146,487	283,628	8,382	7,125	3,895	**0,332**	2,068	6,384
90000	8649,009	1448,465	374,694	9,555	8,118	4,379	**0,322**	2,143	7,239
100000	10770,493	1813,723	471,963	10,792	8,364	4,960	**0,438**	2,489	8,111

На рис. 1,а та 1,б наведено порівняння швидкодії алгоритмів сортування при реалізації мовами C++ та Java для кількості елементів в інтервалі від 100 та 100 000 елементів. Тут і далі з візуалізації виключені три найповільніші алгоритми сортування бульбашкою, вибором та вставкою, оскільки вони мають доволі великі показники в усіх мовах і на їх фоні важко спостерігати й аналізувати тенденції поведінки інших шести алгоритмів. Про порядок відповідних значень можна судити по даних, наведених у

ISSN 2663 - 4023

табл. 2. Із рис. 1,а та 1,б видно, що найшвидшими для великих масивів даних є (у порядку спадання ефективності) сортування підрахунком, за розрядами, швидке і купою. Ретельний аналіз виявив певні відмінності поведінки часової складності алгоритмів сортування злиттям і Шелла. Так, при реалізації мовою Java алгоритм сортування злиттям виявився швидшим за сортування Шелла. При реалізації засобами C++ алгоритм сортування Шелла навпаки працює повільніше за алгоритм сортування злиттям.



а) б)

Рисунок 1 – Порівняння швидкодії алгоритмів сортування мовами C++ та Java

На рис. 2 наведено результати роботи шести алгоритмів сортування мовою C#. Встановлено, що найшвидшими є сортування підрахунком та за розрядами. Далі за спаданням швидкодії йдуть сортування Шелла, злиттям, купою і швидке.

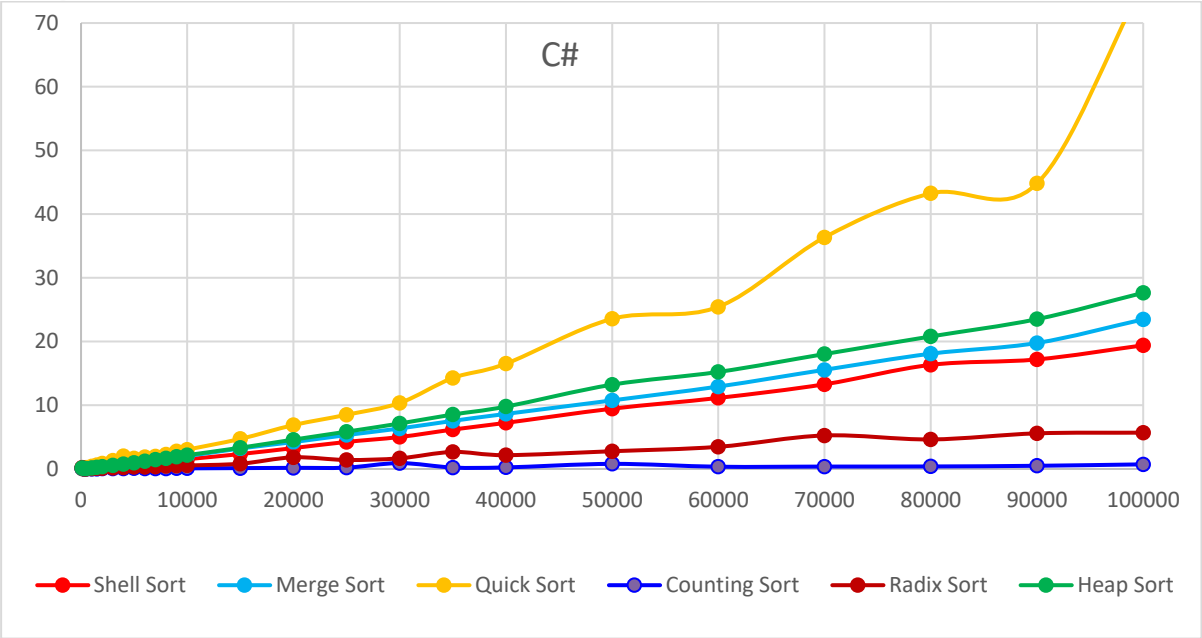


Рисунок 2 – Порівняння швидкодії алгоритмів сортування мовою C#

На рис. 3 наведено результати роботи шести алгоритмів сортування мовою JavaScript. Для малих масивів найкращі результати показали алгоритми сортування купою і Шелла. Для наборів даних понад 60 тис. елементів найкращим виявився алгоритм сортування підрахунком, випереджаючи сортування купою і Шелла, які мають майже однакову ефективність. Швидке сортування, злиттям та розрядами посідають відповідно з четвертого по шосте місця.

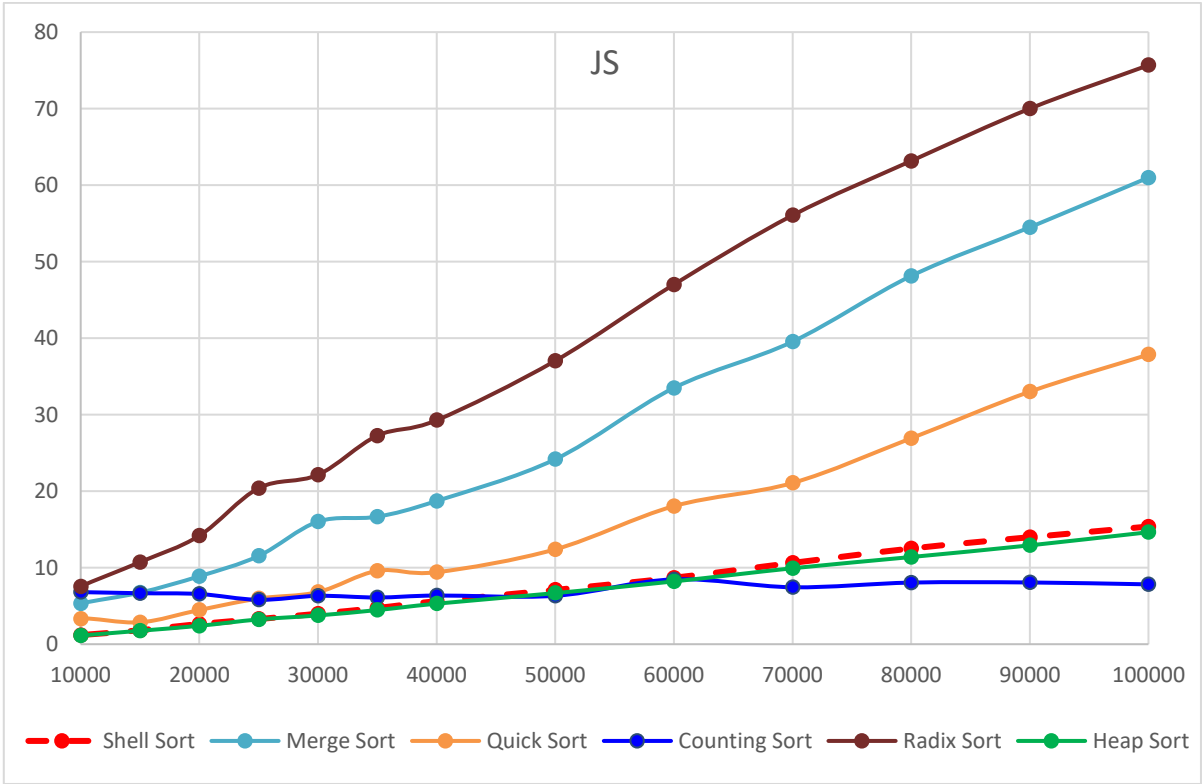


Рисунок 3 – Швидкодія алгоритмів сортування мовою JavaScript

Проведене дослідження дозволило виявити певні відмінності між ефективністю та швидкодією алгоритмів сортування при реалізації різними мовами програмування, які варто враховувати під час вибору того чи іншого алгоритму. Отже, при виборі алгоритму недостатньо керуватись лише О-нотацією.

ВИСНОВКИ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

Алгоритми сортування є фундаментальною частиною завдань при розробці програмного забезпечення для е-комерції, обслуговування, маркетингу, аналізу даних тощо. Повсюдне використання алгоритмів сортування під час розробки програмного забезпечення зумовлює потребу розуміння програмістами механізмів роботи того чи іншого алгоритму. Адже вибір оптимального алгоритму за певних даних та умов використання може стати нетривіальною задачею, а невдалий вибір алгоритму впорядкування даних може спричинити проблеми зі швидкодією програмного опрацювання даних.

З наведеного вище аналізу було зроблено висновок, що мови програмування впливають на швидкодію алгоритмів сортування. Крім того, на вибір алгоритму також впливає кількість та діапазон значень елементів у наборі даних і його структура (зокрема, довільний або послідовний доступ). Хороший вибір алгоритму для певної мови програмування особливо важливий при сортуванні великих наборів даних.

Виявлено, що в розглянутих мовах програмування найкращу швидкість для великих масивів показує алгоритм сортування підрахунком. Але він має певні обмеження у застосуванні. Так, якщо діапазон випадкових значень елементів більшій, ніж розмір набору даних, і кількість однакових елементів є малою, то ефективність цього алгоритму є низькою. Прояви такої поведінки зафіксовано на малих масивах, особливо при реалізації мовами JavaScript і Java, де сортування підрахунком поступається деяким іншім алгоритмам. Тому, якщо використання сортування підрахунком є небажаним через відповідні обмеження, алгоритмом вибору стає другий за ефективністю алгоритм у певній мові програмування. Якщо і його використовувати не можна через обмеження, то треба використовувати третій тощо.

Подальші дослідження можуть стосуватися вивчення поведінки та швидкодії алгоритмів сортування для різних типів та структур даних, різного ступеню впорядкованості елементів при реалізації різними мовами програмування та для одночасного виконання на паралельних машинах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Big-O Complexity Chart. https://www.bigocheatsheet.com/

2 Корнійчук, М.М., Трофименко, О.Г. (2023) Оцінка часової складності деяких алгоритмів сортування у популярних мовах програмування. *Інформаційне суспільство: проблеми та перспективи* : матер. VIII всеукр. наук.-практ. конф., 72-74, https://doi.org/10.32837/11300.25263.

3 Marcellino, M., Pratama, D. W., Suntiarko, S. S., Margi, K. (2021). Comparative of Advanced Sorting Algorithms (Quick Sort, Heap Sort, Merge Sort, Intro Sort, Radix Sort) Based on Time and Memory Usage. *1st International Conference on Computer Science and Artificial Intelligence (ICCSAI)*, 1, 154-160. https://doi.org/10.1109/ICCSAI53272.2021.9609715.

4 Ali, I., Lashari, H., Keerio, I., Maitlo, A., Chhajro, M., Malook, M. (2018). Performance Comparison between Merge and Quick Sort Algorithms in Data Structure. *International Journal of Advanced Computer Science and Applications*, 9, 192-195. https://doi.org/10.14569/IJACSA.2018.091127.

5 Rabiu, A., Garba, E., Baha, B., Mukhtar, M. (2021). Comparative Analysis between Selection Sort and Merge Sort Algorithms. *Nigerian Journal of Basic and Applied Sciences*, 29, 43-48. https://doi.org/10.4314/njbas.v29i1.5.

6 Rabiu, A., Garba, E., Baha, B., Malgwi, Y., Dauda, M. (2022). Performance Comparison of three Sorting Algorithms Using Shared Data and Concurrency Mechanisms in Java. *Arid-zone Journal of Basic & Applied Research (AJBAR)*, 1, 55-64. https://doi.org/10.55639/607fox.

7 Hermehar, B., Amandeep, K. (2022). Comparative Study of Different Sorting Algorithms used in Data Structures. *International Journal of Research Culture Society*, 6, 114-119. https://doi.org/10.2017/IJRCS/202203021.

8 Manan, C., Anurag, D. (2022). Establishing Pertinence between Sorting Algorithms Prevailing in $nlog(n)$ time. *Journal of Robotics and Automation Research*, 3(2), 220-226. https://doi.org/10.33140/JRAR.03.02.09.

9 Durrani, O. K. (2023). Asymptotic performances of popular programming languages for popular sorting algorithms. Semiconductor Optoelectronics, 42, 149-169. https://www.researchgate.net/publication/369196272_asymptotic_performances_of_popular_programming_languages_for_popular_sorting_algorithms.

10 Butt, Sh., Aqib, S., Nawaz, H. (2021). Analysis of Merge Sort and Bubble Sort Python, PHP, JavaScript, and C language. *International Journal of Advanced Trends in Computer Science and Engineering*, 10, 680-686. https://doi.org/10.30534/ijatcse/2021/311022021.

11 Fagbote, O.O. (2022). Performance evaluation of sorting techniques. *Global Scientific Journal*, 10(1), 1-12. https://www.globalscientificjournal.com/researchpaper/performance_evaluation_of_sorting_techniques.pdf.

12 Khan, M., Shaheen, S., Qureshi, F. (2014). Comparative Analysis of five Sorting Algorithms on the basis of Best Case, Average Case, and Worst Case. *International Journal of Information Technology and Electrical Engineering*, 3(1), 1-10. https://www.researchgate.net/publication/310953713_Comparative_Analysis_of_five_Sorting_Algorithms_on_the_basis_of_Best_Case_Average_Case_and_Worst_Case.

13 Durrani, O. K. (2022). Performance Measurement of Popular Sorting Algorithm Implemented using Java & C. *International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, https://doi.org/10.1109/ICECCME55909.2022.9988424.

14 Fagbola, M., Surendra, T. (2019). Investigating the Effect of Implementation Languages and Large Problem Sizes on the Tractability and Efficiency of Sorting Algorithms. *International Journal of Engineering Research and Technology*, 12, 196-203. https://www.researchgate.net/publication/331408485_Investigating_the_Effect_of_Implementation_Languages_and_Large_Problem_Sizes_on_the_Tractability_and_Efficiency_of_Sorting_Algorithms.

15 PYPL PopularitY of Programming Language. https://pypl.github.io/PYPL.html.

16 TIOBE Index for June 2023. URL: https://www.tiobe.com/tiobe-index/

17 The RedMonk Programming Language Rankings: January 2023. https://redmonk.com/sogrady/2023/05/16/language-rankings-1-23/

18 IEEE Spectrum. Top Programming Languages 2022. https://spectrum.ieee.org/top-programming-languages-2022

19 DOU. Рейтинг мов програмування 2023. https://dou.ua/lenta/articles/language-rating-2023/

20 numpy.sort. https://numpy.org/doc/stable/reference/generated/numpy.sort.html.

КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 1 (21), 2023
ISSN 2663 - 4023

Trofymenko Olena
PhD, Associate Professor, Associate Professor at the Department of Information Technologies
of the National University "Odessa Law Academy", Odesa, Ukraine,
ORCID ID: 0000-0001-7626-0886
trofymenko@onua.edu.ua

Prokop Yuliia
PhD, Associate Professor, Associate Professor at the Department of Software engineering
of the Odessa Polytechnic National University, Odesa, Ukraine,
ORCID ID 0000-0002-6608-3668
prokop.y.v@op.edu.ua

Chepurna Olena
PhD, Associate Professor, Associate Professor of the Department of Cybersecurity
of the National University "Odessa Law Academy", Odesa, Ukraine,
ORCID ID 0000-0002-1432-0799
chepurna@onua.edu.ua

Korniichuk Mykola
Student of Faculty of Cybersecurity and Information Technologies
of the National University “Odessa Academy of Law”, Odessa, Ukraine,
ORCID ID 0009-0009-4223-9291
chuguystyr@gmail.com

A PERFORMANCE COMPARISON OF SORTING ALGORITHMS IN DIFFERENT PROGRAMMING LANGUAGES

Abstract. Sorting, as one of the basic algorithms, has a wide range of applications in software development. As the amount of processed data grows, the need for fast and efficient data sorting increases significantly. There are many sorting algorithms and their extensions. However, choosing the best and most versatile among them is impossible. All these algorithms have their specifics, which determine the scope of their effective use. Therefore, the problem of deciding the optimal algorithm for certain specific conditions is relevant. This choice is often a non-trivial task, and an unsuccessful choice of algorithm can cause difficulties with data processing performance. To determine which algorithm will be the best in a particular situation, you need to analyse all the factors that affect the operation of algorithms: the size and structure of the data set, the range of element values, the form of access (random or sequential), the orderliness, the amount of additional memory required to execute the algorithm, etc. In addition, different algorithms have different performance in different programming languages. The study analyses the advantages and disadvantages of nine popular sorting algorithms (Bubble, Insertion, Selection, Shell, Merge, Quick, Counting, Radix, and Heap) due to their specifics and limitations on their possible use. The performance of these algorithms implemented in four popular programming languages (C++, C#, Java and JavaScript) is tested. We experimentally discovered that the performance of sorting algorithms differs depending on the programming language. The applied aspect of the study is that its conclusions and results will allow developers to choose the best algorithm for a particular programming language, depending on the size, range, structure, etc. of the data set to be sorted. Considering this is significant when we have to sort large amounts of data in search engines, scientific and engineering applications. After all, the sorting algorithm's efficiency significantly affects the system's overall performance.

Keywords: sorting algorithms, big-O notation, running time, performance, sorting.

REFERENCES (TRANSLATED AND TRANSLITERATED)

1 Big-O Complexity Chart. https://www.bigocheatsheet.com/
2 Korniichuk, M., Trofymenko, O. (2023) Evaluation of the some sorting algorithms speed in popular programming languages. *Information society: problems and prospects*: VIII All-Ukrainian Science and Practice Conference, 72-74. https://doi.org/10.32837/11300.25263.
3 Marcellino, M., Pratama, D. W., Suntiarko, S. S., Margi, K. (2021). Comparative of Advanced Sorting

Algorithms (Quick Sort, Heap Sort, Merge Sort, Intro Sort, Radix Sort) Based on Time and Memory Usage. *1st International Conference on Computer Science and Artificial Intelligence (ICCSAI)*, 1, 154-160. https://doi.org/10.1109/ICCSAI53272.2021.9609715.

4 Ali, I., Lashari, H., Keerio, I., Maitlo, A., Chhajro, M., Malook, M. (2018). Performance Comparison between Merge and Quick Sort Algorithms in Data Structure. *International Journal of Advanced Computer Science and Applications*, 9, 192-195. https://doi.org/10.14569/IJACSA.2018.091127.

5 Rabiu, A., Garba, E., Baha, B., Mukhtar, M. (2021). Comparative Analysis between Selection Sort and Merge Sort Algorithms. *Nigerian Journal of Basic and Applied Sciences*, 29, 43-48. https://doi.org/10.4314/njbas.v29i1.5.

6 Rabiu, A., Garba, E., Baha, B., Malgwi, Y., Dauda, M. (2022). Performance Comparison of three Sorting Algorithms Using Shared Data and Concurrency Mechanisms in Java. *Arid-zone Journal of Basic & Applied Research (AJBAR)*, 1, 55-64. https://doi.org/10.55639/607fox.

7 Hermehar, B., Amandeep, K. (2022). Comparative Study of Different Sorting Algorithms used in Data Structures. *International Journal of Research Culture Society*, 6, 114-119. https://doi.org/10.2017/IJRCS/202203021.

8 Manan, C., Anurag, D. (2022). Establishing Pertinence between Sorting Algorithms Prevailing in $nlog(n)$ time. *Journal of Robotics and Automation Research*, 3(2), 220-226. https://doi.org/10.33140/JRAR.03.02.09.

9 Durrani, O. K. (2023). Asymptotic performances of popular programming languages for popular sorting algorithms. Semiconductor Optoelectronics, 42, 149-169. https://www.researchgate.net/publication/369196272_asymptotic_performances_of_popular_programming_languages_for_popular_sorting_algorithms.

10 Butt, Sh., Aqib, S., Nawaz, H. (2021). Analysis of Merge Sort and Bubble Sort Python, PHP, JavaScript, and C language. *International Journal of Advanced Trends in Computer Science and Engineering*, 10, 680-686. https://doi.org/10.30534/ijatcse/2021/311022021.

11 Fagbote, O.O. (2022). Performance evaluation of sorting techniques. *Global Scientific Journal*, 10(1), 1-12. https://www.globalscientificjournal.com/researchpaper/performance_evaluation_of_sorting_techniques.pdf.

12 Khan, M., Shaheen, S., Qureshi, F. (2014). Comparative Analysis of five Sorting Algorithms on the basis of Best Case, Average Case, and Worst Case. *International Journal of Information Technology and Electrical Engineering*, 3(1), 1-10. https://www.researchgate.net/publication/310953713_Comparative_Analysis_of_five_Sorting_Algorithms_on_the_basis_of_Best_Case_Average_Case_and_Worst_Case.

13 Durrani, O. (2022). Performance Measurement of Popular Sorting Algorithm Implemented using Java & C. *International Conference on Electrical, Computer, Communications and Mechatronics Engineering*, https://doi.org/10.1109/ICECCME55909.2022.9988424.

14 Fagbola, M., Surendra, T. (2019). Investigating the Effect of Implementation Languages and Large Problem Sizes on the Tractability and Efficiency of Sorting Algorithms. *International Journal of Engineering Research and Technology*, 12, 196-203.

15 PYPL PopularitY of Programming Language. https://pypl.github.io/PYPL.html.

16 TIOBE Index for June 2023. URL: https://www.tiobe.com/tiobe-index/

17 The RedMonk Programming Language Rankings: January 2023. https://redmonk.com/sogrady/2023/05/16/language-rankings-1-23/

18 IEEE Spectrum. Top Programming Languages 2022. https://spectrum.ieee.org/top-programming-languages-2022.

19 DOU. Rating of programming languages 2023. https://dou.ua/lenta/articles/language-rating-2023/

20 numpy.sort. https://numpy.org/doc/stable/reference/generated/numpy.sort.html.