налювати системи та методи використання ШІ в навчальному процесі для максимальної відповідності потребам студентів та викладачів.

Отже, використання штучного інтелекту в процесах університетської освіти має великий потенціал для покращення якості освіти та підвищення академічної успішності студентів. Дослідження показують, що персоналізоване навчання, адаптивність системи, розробка інноваційних методів навчання й підвищення мотивації до навчання – це лише деякі з переваг використання ШІ в освіті. Однак успішне впровадження цих технологій вимагає не лише технічної експертизи, а й уваги до етичних аспектів, навчання персоналу, партнерства з промисловістю та забезпечення доступності для всіх категорій студентів. Очікується, що розвиток штучного інтелекту в освіті продовжуватиме зростати, відкриваючи нові можливості для створення більш ефективних та інноваційних методів навчання. Важливо продовжувати дослідження в цій галузі і розвивати передовий досвід для забезпечення якісної освіти та навчання для майбутніх поколінь.

Abstract. This article discusses the potential opportunities and prospects for the use of artificial intelligence (AI) in the educational process of higher education institutions. The importance of integrating modern technologies into the field of education and their impact on the quality and efficiency of the educational process is emphasized. The authors consider the benefits of using AI to individualize learning, automate administrative processes, and improve the quality of education. Special attention is paid to the ethical and social aspects of using AI in the learning environment. The challenges faced by higher education institutions in implementing and integrating AI are explored, and ways to overcome them are suggested. The general conclusion is that the use of AI can significantly improve the educational process in higher education institutions, but requires careful planning, staff training, and careful consideration of ethical and legal aspects.

Keywords: information technology, artificial intelligence, higher education, learning.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Crompton H., Song D. The Potential of Artificial Intelligence in Higher Education. *Teaching & Learning Faculty Publications*. 2021. Vol. 62. P. 1–4.

2. Chen L., Chen P., Lin Z. Artificial Intelligence in Education: A Review. *IEEE Access*. 2020. Vol. 8. P. 75264–75278.

3. Dhawan S., Batra G. Artificial Intelligence in Higher Education: Promises, Perils, and Perspective. *An international journal of research in management*. 2020. Vol. 1. P. 11–22.

4. Ігрові технології навчання: корисні прийоми, як зробити навчання цікавим. URL: https://bdut.co.ua/pronas/igrovi-tekhnologiyi-navchannya/ (дата звернення: 01.03.2024).

УДК 004.422

QUICKSORT: СТАТИСТИЧНЕ ПОРІВНЯННЯ ЧАСУ РОБОТИ МЕТОДІВ ОБЧИСЛЕННЯ ОПОРНОГО ЕЛЕМЕНТУ

М. В. Шевцов, Н. А. Потапова

Анотація. Досліджено алгоритм сортування QuickSort, який є одним із найефективніших методів сортування. У роботі зосереджено увагу на статистичному порівнянні часу роботи алгоритму в різних сценаріях вибору опорного елементу. Розглянуто найгірший, середній та найкращий випадки, а також використання рандомізованого та детермінованого підходів під час вибору опорного елементу. Виконано практичний експеримент, де порівняні реалізації алгоритму з використанням рандомізованого та детермінованого алгоритмів вибору опорного елементу.

Ключові слова: QuickSort, опорний елемент, рандомізований вибір, детермінований вибір, сортування, аналіз алгоритмів, статистичне порівняння.

Вступ. Класичними алгоритмами під час побудови програмних модулів є алгоритми сортування, основним призначенням яких визначається впорядкування множини даних з метою подальшого поліпшення операцій пошуку. QuickSort, або сортування Хоара, є одним з найефективніших алгоритмів сортування, побудованих на використанні стратегії «розділяй і володарюй». Сутність цієї стратегії передбачає використання рекурсивних обчислень, що застосовуються базовим алгоритмом на відокремленій частині даних із подальшим використанням

на наступних наборах, отриманих наступним діленням та повторенням алгоритму до отримання кінцевого результату.

Основна ідея методу QuickSort полягає у виборі опорного елементу, внаслідок чого отримується найбільш ефективне розбиття початкового масиву на дві частини: в одну входитимуть усі елементи, менші за опорний, а в другий – більші за нього. Після поділу початкового масиву алгоритм застосовується повторно на підмасивах доти, поки початковий масив довжини N не розіб'ється на N масивів довжини 1. Масиви, в яких є всього один елемент, є відсортованими, тому залишається розставити одиничні масиви у правильному порядку у початковий масив. Зрозумілим є те, що основними операціями будуть вважатися вибір опорного елементу та подальше розбиття масиву. Процедура розбиття (partition) проходить всередині основного масиву, що не вимагає додаткової пам'яті для виділення нового масиву. Тобто змінюється порядок елементів підмасивів без залучення додаткової пам'яті [1, 2]. Такий підхід, який називається in-place-сортування (сортування на місці), є однією з головних особливостей цього алгоритму і відрізняє швидке сортування від інших методів, наприклад, сортування злиттям.

Основний розділ. Ми з'ясували, що основними операціями швидкого сортування є пошук опорного елементу та розбиття. У перших реалізаціях опорним обирався перший елемент масиву, що знижувало ефективність під час сортування вже відсортованих масивів. Для поліпшення ефективності може використовуватися середній, випадковий елемент чи медіана масиву [3]. Приділимо більше уваги асимптотичній складності алгоритму (у Θ-нотації) у різних ситуаціях вибору опорного елементу:

1. У найгіршому випадку вибір опорного елементу може призвести до розбиття масиву на два підмасиви довжиною 1 і n-1. Якщо врахувати, що для розділення масиву необхідно $\Theta(n)$ часу, то кінцевий час роботи у випадку вже відсортованого масиву становитиме $\Theta(n^2)$ [2].

2. Середній випадок обчислюється з урахуванням усіх можливих варіантів розділення масиву та усереднює результати. Повне викладення розрахунку асимптотики середнього випадку наведено Томасом Корменом у [2]. Зупинимось на обрахунку математичного сподівання рекурентною формулою. Припустимо, що масив розбивався у фіксованому відношенні, наприклад, 4 : 3. Тоді математичне сподівання можна описати формулою:

$$T(n) = T\left({}^{3n}/_{4}\right) + T\left({}^{n}/_{4}\right) + \Theta(n), \quad (1)$$

де n – кількість елементів масиву.

У цьому випадку глибина дерева рекурсії дорівнює $\log_{{}^{4}/_{3}} n = \Theta(\log n)$. Аналіз показує, що при будь-якому фіксованому відношенні довжин підмасивів на кожному кроці середня глибина рекурсії залишається логарифмічною, а час роботи квазілінійним [2].

3. У найкращому випадку масив розбивається навпіл, тобто опорним обирається середній елемент. При n таких розбиттів отримуємо логарифмічну складність, а час роботи алгоритму становить $\Theta(n \log n)$.

Сутність відмінності між середнім і найкращім випадками реалізації алгоритму, незважаючи на однакове математичне сподівання, полягає у способі розділення масиву даних. У найкращому випадку масив постійно розбивається навпіл, а за середнього випадку, як ми вже з'ясували, розбиття відбувається завжди по-різному.

Отже, розглянувши операцію пошуку опорного елементу, зупинимось на другій, а саме – на операції розбиття масиву. Є 2 загальні алгоритми цієї операції: схема Хоара та розбиття Ломуто.

Перший принцип знаходить якимось способом опорний елемент (для простоти це буде центральний елемент) та фіксує на початку та в кінці масиву індекси. Потім індекс початку інкрементується, а кінцевої точки – декрементується, доки не знайдеться така пара елементів, для якої один елемент більший за опорний і розташований у підмасиві, де всі елементи менші за опорний, а інший – менший за опорний і розташований у підмасиві, де всі елементи більші за опорний [2]. Такий обмін триватиме доти, поки індекси не дорівнюватимуть один одному.

Розбиття Ломуто працює по-іншому: алгоритм на початку визначає певну змінну $i = 0$, яка буде змінюватись завжди, коли знаходиться елемент, не більший за опорний [3]. Опорним

же в цьому варіанті розбиття обирається останній елемент масиву. Якщо було знайдено елемент, не більший за нього, то цей елемент стане перед опорним. Після останнього кроку розбиття опорний елемент буде мати індекс i, тобто буде знаходитись на межі двох підмножин: тієї, яка менша за опорний, і тієї, яка менша. Схема Хоара ефективніша за схему Ломуто, оскільки в середньому вона потребує втричі менше обмінів елементів, і розбиття виходить ефективнішим, навіть коли всі елементи рівні. Візуалізацію процедури розбиття можна побачити на рис. 1.

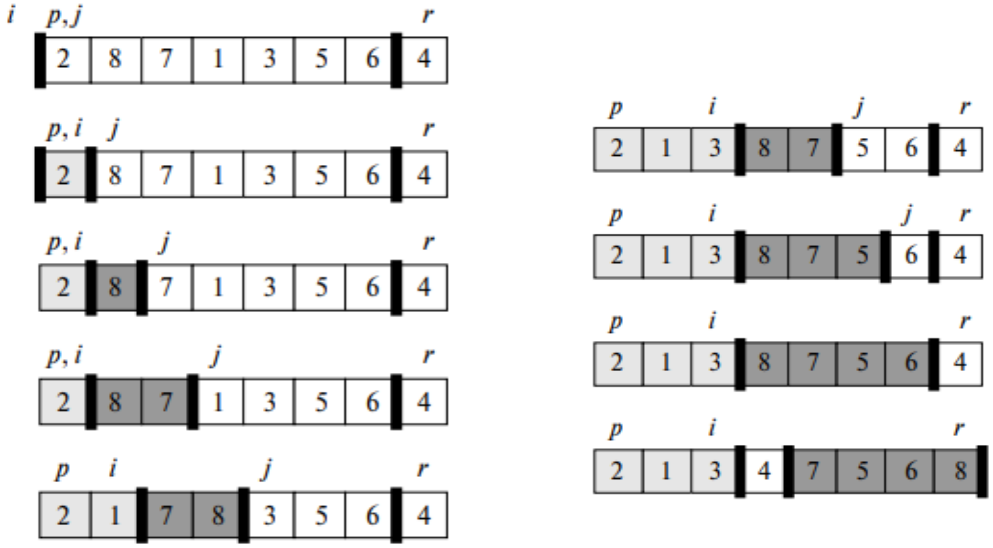

Рис. 1. Приклад роботи процедури розбиття [2]

Одним із підходів, що забезпечують збалансованість розбиття масиву, є використання рандомізації. Рандомізація під час вибору опорного елементу забезпечує включення випадковості у процес розподілу масиву. В якості першого підходу можна використати алгоритм на основі сучасного методу генерації псевдовипадкових чисел під назвою «Вихор Мерсенна», який допоможе визначити випадково індекс опорного елементу (в середньому цей вибір буде успішним) [2]. У заголовковому файлі <random> «Вихор Мерсенна» та інші схожі алгоритмі називаються рушіями [5]. Попередньо нам треба встановити певний закон розподілу ймовірностей, що описує значення випадкової величини та ймовірність її появи. У коді нами задано «універсальний» розподіл випадкових чисел від 0 до розміру початкового масиву, а також так зване зерно, від якого буде відштовхуватись генератор під час обрахунку наступного випадкового числа. У цьому випадку зерно генерується за допомогою функціоналу файла <chrono>. Такий спосіб генерації зерна допоможе уникнути повторення генерованих чисел після кожної ітерації. Реалізація детермінованого алгоритму дещо простіша, і її ідея в нашому випадку полягає у знаходженні опорного елементу, найбільш близького до медіани масиву. Оптимальним опорним елементом є медіана всієї послідовності, але її знаходження занадто трудомістке, тому використано алгоритм пошуку медіани п’яти елементів: першого, останнього, центрального всього масиву та центральних правого і лівого підмасивів.

Практична оцінка складності алгоритму досліджена на основі проведення комп’ютерного експерименту. Оцінка розглянутих вище реалізацій алгоритму QuickSort дасть змогу визначити найбільш ефективну та встановити фактори, від яких залежить ця ефективність. У якості наборів даних нами використано цілі додатні числа, згенеровані за допомогою алгоритму «Вихор Мерсенна» з заданим «універсальним» розподілом. Під час викладення результатів експерименту будемо називати QuickSort на основі рандомізовного вибору опорного елементу RQS (RandomizedQuickSort), а на основі детермінованого – DQS (DeterministicQuickSort). Зауважимо, що алгоритм розбиття розроблено за схемою Хоара. Деталі реалізації обох версій можна знайти у репозиторії GitHub [6]. Час роботи обраховано у мікросекундах. Результати експерименту зведено у табл. 1.

Таблиця 1

Результати експерименту

Алгоритм	Кількість елементів для сортування			
	5’000	30’000	100’000	1’000’000
RQS	5’668 мкс.	30’145 мкс.	72’618 мкс.	810’070 мкс.
DQS	4’952 мкс.	28’909 мкс.	84’607 мкс.	912’483 мкс.

Джерело: авторське дослідження

Треба зауважити, що отримані результати, очевидно, будуть змінюватись від запуску до запуску алгоритму через різні згенеровані дані, тому був обраний середній час роботи за вибіркою з 5 замірів.

Для наочності побудуємо графік зміни часу виконання на різних наборах даних (рис. 2):

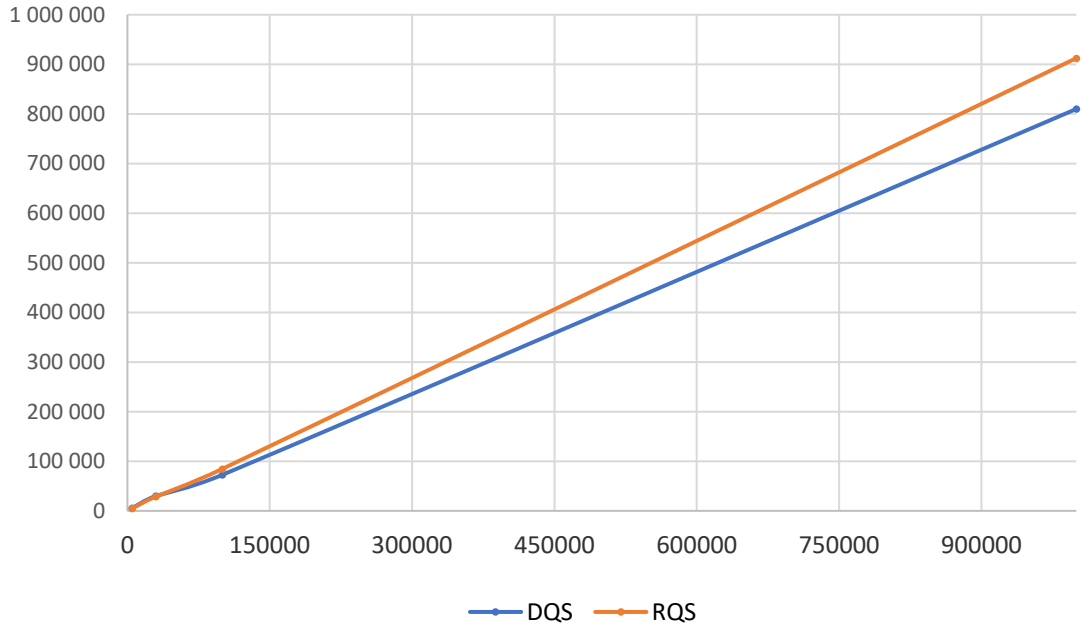

Рис. 2. Візуалізація результатів експерименту

Висновки. Порівняльна оцінка двох різних підходів реалізації алгоритму QuickSort (рандомізованого та детермінованого) дала змогу встановити основні їх відмінності та спільні характеристики обчислювальних процедур. Проведений експеримент показав, що на великих наборах даних алгоритм, побудований на детермінованому виборі опорного елементу, є більш витратним. Це пов’язано з особливостями реалізації, які вимагають складних обчислень, проте на невеликих наборах він працює швидше. Рандомізований алгоритм виявися ефективнішим на наборах даних, більших за 50 тисяч елементів. Тобто можна стверджувати, що вибір тієї чи іншої реалізації залежить від особливостей вхідних даних та конкретних вимог проєкту.

Abstract. Explored the QuickSort sorting algorithm, which is one of the most efficient sorting methods. The focus of the research is on the statistical comparison of the algorithm’s performance in different scenarios of choosing a pivot element. The worst-case, average-case, and best-case scenarios are considered, as well as the use of randomized and deterministic approaches in selecting the pivot element. A practical experiment was conducted, comparing implementations of the algorithm using randomized and deterministic pivot selection algorithms.

Keywords: QuickSort, pivot element, randomized selection, deterministic selection, sorting, algorithm analysis, statistical comparison.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Roughgarden T. Algorithms Illuminated. Part 1: The Basics. *Soundlikeyourself Publishing*, 2017. P. 226.
2. Introduction to Algorithms 3rd Edition / T. H. Cormen, C. E. Leiserson, R L. Rivest, C. Stein. MIT PRESS, 2009. 1292 p.
3. Hoare C. A. R. Quicksort. *The Computer Journal.* 1962. Vol. 5. № 1. P. 10–16.

4. Sedgewick R. Algorithms In C: Fundamentals, Data Structures, Sorting, Searching, Parts 1–4. Addison-Wesley Professional, 1997. 752 p.

5. Programming Languages – C++. Standart (ISO/IEC JTC1 SC22 WG21 N4860). URL: https://isocpp.org/files/papers/N4860.pdf (дата звернення: 20.02.2024).

6. Офісійний сайт Github. URL: https://github.com/shevmee/algorithms (дата звернення: 24.02.2024).

7. Денисюк В. О., Потапова Н. А., Зелінська О. В., Тарасюк М. Б. Програмна реалізація та дослідження алгоритмів паралельного швидкого сортування. *Вісник Хмельницького національного університету. Технічні науки.* 2023. № 4. С. 95–105. URL: http://journals.khnu.km.ua/vestnik/wp-content/uploads/2023/09/323-95-105.pdf (дата звернення: 01.03.2024).

УДК 517.9

ЗАСТОСУВАННЯ МОДЕЛЕЙ ЛАНЧЕСТЕРА ДЛЯ АНАЛІЗУ РОСІЙСЬКО-УКРАЇНСЬКОЇ ВІЙНИ

В. О. Яропуд, О. С. Вєтров

Анотація. Метою роботи є детальний аналіз бойових моделей Ланчестера, їх застосування в різних воєнних конфліктах та визначення ефективності цих моделей в Україні. Однією з ключових переваг моделей Ланчестера є їх здатність до врахування різних факторів, як-от чисельність військ, обмундирування, ефективність зброї і тактичні параметри. Це дає змогу розробляти комплексні стратегії та аналізувати різні сценарії розвитку конфлікту.

Ключові слова: аналіз бойових дій, математичне моделювання, модель Ланчестера.

Вступ. Спираючись на аналіз наукової літератури, можна умовно зазначити чотири загальні класи математичних моделей воєнних дій: описові моделі, імітаційні моделі, оптимізаційні моделі та моделі ухвалення рішень. Кожен із цих класів (можливі й інші підстави класифікації) охоплює значну кількість підкласів, що розрізняються математичним апаратом, який використовується.

Основний розділ. Під час Першої світової війни Ф. У. Ланчестер, англійський інженер і математик, створив математичні моделі повітряних боїв. Ці моделі були застосовані до різних ситуацій – від бойових дій регулярних військ і партизанських з'єднань до комбінованих сценаріїв. Розглядається три моделі [1]. Уявімо, що в бойових діях беруть участь дві сторони – x і y. Чисельний склад цих сторін у момент часу t (вимірюваний у днях, починаючи з першого дня бойових дій) позначається як $x(t)$ і $y(t)$ відповідно. Чисельність сторін у цих моделях є важливим фактором, але врахування інших чинників, як-от бойова готовність, озброєння, досвід командирів, моральний настрій і багато інших, є практично неможливим завданням.

Припустимо, що чисельність сторін $x(t)$ і $y(t)$ змінюється неперервно і є диференційованою функцією часу. Це спрощення, оскільки насправді $x(t)$ і $y(t)$ є цілими числами. Однак у разі достатньо великої чисельності кожної сторони збільшення чисельності на кілька осіб має практично незначний вплив, порівняно з наявною чисельністю. Тому можна припустити, що протягом коротких проміжків часу чисельність змінюється незначно (не цілими кількостями). Це припущення недостатнє для виведення конкретних формул для $x(t)$ і $y(t)$ як функцій від t.

Можна вказати кілька факторів, що впливають на швидкість зміни чисельності сторін. Позначимо OLR як швидкість, з якою сторона x зазнає втрат від хвороб і інших факторів, що не пов'язані з бойовими діями.

Далі припустимо, що CLR представляє швидкість втрат стороною x внаслідок прямих зіткнень зі стороною y під час бойових дій. Позначимо RR як швидкість прибуття підкріплень до сил сторони x. Тоді рівняння для швидкості зміни $x(t)$ можна записати так:

$$\frac{dx(t)}{dt} = -(OLR + CLR) + RR.$$

Аналогічне рівняння буде використовуватись для $y(t)$. Тепер завдання полягає в тому, щоб визначити формули для OLR, CLR і RR, а потім дослідити отримані диференціальні рівняння. Отримані висновки допоможуть відповісти на питання про можливого переможця.

Для сторони $y(t)$ також буде аналогічне рівняння. Задача полягає у визначенні формул для OLR, CLR та RR, а потім дослідженні отриманих диференціальних рівнянь. Отримані висновки дадуть змогу відповісти на питання про потенційного переможця.