що допомагає уникнути повторних обчислень [2]. Так складність може зменшитися з експоненційної до поліноміальної.

5. Жадібні алгоритми та евристики. Жадібні алгоритми приймають локально оптимальний вибір у кожному кроці, сподіваючись, що це приведе до глобально оптимального рішення. Евристичні підходи застосовують наближені методи, щоб швидко отримати достатньо хороший розв'язок, коли екзогенно оптимальний алгоритм є занадто повільним. Такі методи особливо важливі для задач NP-складності, де точний розв'язок може вимагати надто тривалого часу.

Отже, кожен із розглянутих методів оптимізації виконує унікальну функцію та застосовується для різних типів алгоритмічних задач і вибір конкретного методу оптимізації залежить від структури задачі, обмежень ресурсів та вимог до точності. Найвищу ефективність забезпечує комбінація декількох підходів, що дає змогу збалансувати швидкість, складність реалізації й кінцеву якість результату.

Список використаних джерел

1. Креневич А. П. Алгоритми і структури даних: підручник. Київ: ВПЦ «Київський Університет», 2021. 200 с.

2. Introduction to algorithms / T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. 4th ed. Cambridge, MA: The MIT Press, 2022. 1312 p. URL: https://books.google.com.ua/books?id=NLngYyWFl_YC&printsec=copyright&hl=uk#v=onepage&q&f=false (дата звернення: 15.11.2025).

3. Skiena S. S. The algorithm design manual. 3rd ed. Cham: Springer, 2020. 810 p.

УДК 004.422.5

Горчук К. В., здобувач 2 курсу
спеціальності 122 Комп'ютерні науки,
Потапова Н. А., канд. екон. наук, доцент,
доцент кафедри інформаційних технологій

ОСОБЛИВОСТІ ТА ПЕРЕВАГИ ВИКОРИСТАННЯ АЛГОРИТМУ БІНАРНОГО ПОШУКУ

Донецький національний університет імені Василя Стуса, м. Вінниця

Бінарний пошук – це алгоритм пошуку індексу елементу у впорядкованому масиві. На кожній ітерації відбувається поділ масиву на дві частини, через що алгоритм називають методом ділення навпіл. Цей спосіб пошуку даних достатньо простий для розуміння, водночас він дуже ефективний, оскільки на кожному кроці кількість елементів у робочій області масиву зменшується вдвічі [1].

Алгоритм бінарного пошуку полягає у визначенні значення елементу посередині робочої області масиву даних і порівнянні його із шуканим. У разі рівності елементів повертається індекс, а якщо значення елементу посередині масиву більше від шуканого, то пошук продовжується в лівій частині масиву, якщо інакше – в

правій. Далі варто провести перевірку на сходження меж робочої області, і якщо це так, то значення не знайдене, якщо ні – перехід на перший крок [1]. Через те, що область пошуку зменшується експоненціально (поділ навпіл), кількість необхідних кроків k задовольняє $n/2^k = 1 \rightarrow 2^k = n \rightarrow k = \log_2 n$ [4]. Тобто в найгіршому й середньому випадках алгоритм потребує приблизно $\log_2 n$ порівнянь – звідси і складність $O(\log n)$.

Згідно зі стандартним описом алгоритму, бінарний пошук виконується на упорядкованому масиві, і на кожному кроці пошукова область ділиться навпіл – це дає змогу ефективно зменшувати кількість елементів для перевірки. У найгіршому та середньому випадках алгоритм виконує приблизно $\log_2 n$ порівнянь, де n – розмір масиву, що забезпечує часову складність $O(\log n)$. Водночас додаткових витрат пам'яті майже немає (просторова складність $O(1)$), оскільки алгоритм потребує лише кілька змінних для меж та середнього індексу [5].

Бінарний пошук особливо корисний для роботи з великими впорядкованими наборами даних, де перевіряти кожен елемент послідовно було б занадто повільно. Завдяки тому, що на кожному кроці область пошуку скорочується рівно вдвічі, алгоритм дає змогу значно економити час і швидко знаходити потрібний елемент навіть у масивах із десятків чи сотень тисяч значень. У таких випадках класичний лінійний пошук втрачає ефективність, тоді як бінарний демонструє стабільно хорошу продуктивність [2].

Основними перевагами бінарного пошуку є висока швидкодія, низькі витрати пам'яті та простота реалізації в системах. Він використовується для визначення індексу елементу в масиві. За умови правильного розуміння обчислення насправді доволі прості. Для цього методу оптимальними є великі набори даних. Для величезного розміру набору даних він є найбільш ефективним [3].

Варто зауважити, що існують модифікації алгоритму (наприклад, під час обчислення середнього індексу з округленням вгору), які можуть бути корисні під час роботи з масивами, що містять дублікати – тоді алгоритм може повертати, наприклад, правий або лівий з еквівалентних індексів. Водночас бінарний пошук – не універсальний: якщо дані часто модифікуються (вставки, видалення) або потрібен швидкий довільний доступ без упорядкування, доцільно використовувати інші структури (наприклад, хеш-таблиці) [5].

Отже, бінарний пошук забезпечує логарифмічну часову складність завдяки поділу масиву навпіл на кожному кроці. Алгоритм ефективний для великих впорядкованих наборів даних, має низькі витрати пам'яті та просту реалізацію. Основним обмеженням є необхідність попереднього сортування та менш ефективна робота в структурах із частими змінами. Алгоритм залишається базовим інструментом для оптимізації пошуку в сучасних обчислювальних системах.

Список використаних джерел

1. C#.Net: Бінарний пошук. URL: https://programm.top/uk/c-sharp/algorithm/search/binary-search/ (дата звернення: 28.11.2025).

2. Linear Search vs Binary Search. URL: https://www.geeksforgeeks.org/dsa/linear-search-vs-binary-search/ (дата звернення: 28.11.2025).

3. What are the advantages and disadvantages of binary search? *Quora*. URL: https://www.quora.com/What-are-the-advantages-and-disadvantages-of-binary-search (дата звернення: 28.11.2025).

4. Algorithms – How come the time complexity of Binary Search is log *n*. URL: https://math.stackexchange.com/questions/3224473/how-come-the-time-complexity-of-binary-search-is-log-n (дата звернення: 28.11.2025).

5. Binary search algorithm. URL: https://upload.wikimedia.org/wikiversity/en/e/ea/Binary_search_algorithm.pdf (дата звернення: 28.11.2025).

УДК 004.422.5

Коваль В. С., здобувач 2 курсу спеціальності
122 Комп'ютерні науки,
Фриз І. В., канд. фіз.-мат. наук,
старший викладач кафедри інформаційних технологій

РОЛЬ АЛГОРИТМІВ ТА СТРУКТУР ДАНИХ У СУЧАСНИХ ІТ-СИСТЕМАХ

Донецький національний університет імені Василя Стуса, м. Вінниця

Алгоритми та структури даних становлять основу будь-якого програмного забезпечення і визначають ефективність роботи інформаційних систем. Алгоритм є формальним описом послідовності дій, необхідних для розв'язання певної задачі, тоді як структура даних визначає спосіб організації, зберігання та обробки інформації. Саме від поєднання цих двох компонентів залежить продуктивність, масштабованість і надійність програмних рішень.

Сучасні структури даних поділяють на базові та абстрактні. До базових належать числа, логічні значення, символи, а до абстрактних – масиви, списки, стеки, черги, дерева, хеш-таблиці, графи. Масиви забезпечують швидкий доступ за індексом та часто використовуються у задачах сортування. Зв'язні списки ефективні під час частих вставок та видалень. Стек застосовується для зберігання історії викликів, реалізації рекурсії та механізмів відміни дій, тоді як черги актуальні у системах планування процесів та потокової обробки даних. Дерева та графи є основою алгоритмів пошуку, оптимізації маршрутів, роботи компіляторів та навігаційних систем.

Алгоритми, що працюють із різними структурами даних, оцінюються за допомогою нотації Big O, яка дає змогу визначити час виконання та обсяг використаної пам'яті залежно від розміру вхідних даних. Наприклад, алгоритм сортування злиттям має складність $O(n \log n)$, тоді як наївний пошук у масиві – $O(n)$. Порівняння таких характеристик дає змогу обирати найбільш ефективні методи для вирішення конкретних задач.

Важливою частиною аналізу алгоритмів є визначення компромісу між швидкодією та ресурсами пам'яті. Деякі структури даних забезпечують високу швид-