1. Сортировка выбором (Selection Sort)

Определение: сортировка выбором — это алгоритм сортировки, который на каждом шаге находит минимальный элемент в неотсортированной части массива и меняет его местами с первым элементом неотсортированной части. Процесс повторяется до полной сортировки массива.

Анализ: алгоритм разделяет массив на две части: отсортированную (в начале) и неотсортированную (оставшаяся часть). На каждом шаге алгоритм:

· Находит минимальный элемент в неотсортированной части массива

· Меняет его местами с первым элементом неотсортированной части

· Увеличивает размер отсортированной части на один элемент

Внешний цикл выполняется n-1 раз, так как последний элемент автоматически занимает правильную позицию. Внутренний цикл для i-й итерации выполняет n-i-1 сравнений. Общее количество сравнений составляет n(n-1)/2, что является квадратичной функцией.

Временная сложность:

· Худший случай: O(n²) — массив отсортирован в обратном порядке

· Лучший случай: O(n²) — даже для уже отсортированного массива

· Средний случай: O(n²)

Почему O(n²): алгоритм всегда выполняет примерно n(n-1)/2 сравнений независимо от исходного порядка элементов. Количество операций растет пропорционально квадрату размера массива.

1. Сортировка обменом (пузырьком) (Bubble Sort)

Определение: сортировка обменом (или пузырьковая сортировка) — это простой алгоритм, который многократно проходит по массиву, сравнивая соседние элементы и меняя их местами, если они расположены в неправильном порядке.

Анализ: алгоритм проходит по массиву, сравнивая соседние элементы и меняя их местами, если левый больше правого. Это вызывает "всплывание" самого большого элемента в конец массива на каждой итерации.

· Внешний цикл for выполняется n-1 раз

· Внутренний цикл for в худшем случае выполняется n-1, затем n-2, ..., раз

· На каждой итерации внутреннего цикла выполняется сравнение и возможный обмен

· Оптимизация с флагом swapped позволяет выйти из цикла раньше, если массив уже отсортирован

· После i-й итерации внешнего цикла i последних элементов уже находятся на своих местах

Временная сложность:

· Худший случай: O(n²) — массив отсортирован в обратном порядке

· Лучший случай: O(n) — массив уже отсортирован, срабатывает оптимизация

· Средний случай: O(n²)

Почему O(n²): в худшем случае требует n\*(n-1)/2 сравнений и потенциальных обменов. Количество операций растет пропорционально квадрату размера массива.

1. Сортировка вставками (Insertion Sort)

Определение: сортировка вставками — это алгоритм, который строит отсортированный массив по одному элементу за раз, вставляя каждый новый элемент в правильную позицию среди уже отсортированных элементов.

Анализ: алгоритм обрабатывает элементы слева направо. Для каждого нового элемента (начиная со второго) он:

· Сохраняет текущий элемент как "ключ"

· Сдвигает элементы отсортированной части, которые больше ключа, на одну позицию вправо

· Вставляет ключ в найденную позицию

В лучшем случае (уже отсортированный массив) внутренний цикл выполняется только один раз для каждого элемента. В худшем случае (обратно отсортированный массив) для каждого элемента выполняется максимальное количество сдвигов.

Временная сложность:

· Худший случай: O(n²) — массив отсортирован в обратном порядке

· Лучший случай: O(n) — массив уже отсортирован

· Средний случай: O(n²)

Почему O(n²): в худшем случае для каждого из n элементов выполняется до i сравнений и сдвигов, что дает сумму 1+2+3+...+(n-1) = n(n-1)/2 операций.

1. Сортировка слиянием (Merge Sort)

Определение: сортировка слиянием — это алгоритм "разделяй и властвуй", который рекурсивно делит массив на две половины, сортирует каждую половину, а затем сливает отсортированные половины в один массив.

Анализ: алгоритм состоит из трех основных этапов:

· Разделение: массив рекурсивно делится пополам до тех пор, пока не останутся подмассивы размером 0 или 1

· Сортировка: каждый подмассив тривиально отсортирован (размер 0 или 1)

· Слияние: отсортированные подмассивы сливаются в правильном порядке

Процесс слияния требует дополнительной памяти для хранения временных результатов. Глубина рекурсии составляет log₂n уровней.

Временная сложность:

· Худший случай: O(n log n)

· Лучший случай: O(n log n)

· Средний случай: O(n log n)

Почему O(n log n): на каждом из log₂n уровней рекурсии выполняется O(n) операций сравнения и копирования при слиянии. Таким образом, общая сложность составляет O(n log n).

1. Сортировка Шелла (Shell Sort)

Определение: сортировка Шелла — это алгоритм сортировки, являющийся усовершенствованной версией сортировки вставками. Он сортирует элементы, находящиеся на определенном расстоянии друг от друга, постепенно уменьшая это расстояние.

Анализ: алгоритм работает путем сравнения и обмена элементов, находящихся на определенном расстоянии (gap). Начинается с большого расстояния и постепенно уменьшает его до 1.

· Использует последовательность промежутков (gaps), начиная с n/2 и уменьшая вдвое

· На каждом шаге сортирует элементы, находящиеся на расстоянии gap друг от друга

· Когда gap становится равным 1, алгоритм превращается в обычную сортировку вставками

· Внутренние циклы выполняют сортировку вставками для подмассивов с шагом gap

Временная сложность:

· Худший случай: O(n²) — зависит от выбранной последовательности gaps

· Лучший случай: O(n log n) — для оптимальных последовательностей

· Средний случай: O(n^(3/2)) — для последовательности Кнута

Почему O(n²): в худшем случае, при неудачной последовательности gaps, алгоритм может вырождаться до квадратичной сложности. Однако на практике работает значительно быстрее сортировки вставками.

1. Быстрая сортировка (Quick Sort)

Определение: быстрая сортировка — это алгоритм "разделяй и властвуй", который выбирает опорный элемент и разделяет массив на две части: элементы меньше опорного и элементы больше опорного, затем рекурсивно сортирует обе части.

Анализ: алгоритм состоит из трех этапов:

· Выбор опорного элемента (pivot)

· Разделение массива относительно опорного элемента

· Рекурсивная сортировка левой и правой частей

· В лучшем случае опорный элемент делит массив пополам

· В худшем случае опорный элемент оказывается минимальным или максимальным

· Алгоритм использует рекурсию и требует стековой памяти

Временная сложность:

· Худший случай: O(n²) — когда массив уже отсортирован или обратно отсортирован

· Лучший случай: O(n log n) — когда опорный элемент делит массив пополам

· Средний случай: O(n log n)

Почему O(n log n): в среднем случае глубина рекурсии составляет O(log n), и на каждом уровне выполняется O(n) операций сравнения и перестановки.

1. Пирамидальная сортировка (Heap Sort)

Определение: пирамидальная сортировка — это алгоритм сортировки, который использует структуру данных "куча" (heap) для сортировки элементов. Алгоритм строит max-heap из массива, затем извлекает максимальные элементы.

Анализ: алгоритм состоит из двух основных этапов:

· Построение max-heap из исходного массива

· Последовательное извлечение максимального элемента из кучи и его перемещение в конец массива

· Функция heapify поддерживает свойство кучи для поддерева

· Построение кучи выполняется за O(n) операций

· Каждое извлечение максимума требует O(log n) операций

Временная сложность:

· Худший случай: O(n log n)

· Лучший случай: O(n log n)

· Средний случай: O(n log n)

Почему O(n log n): построение кучи занимает O(n) времени, а каждое из n извлечений максимума требует O(log n) операций, что в сумме дает O(n log n).

1. Последовательный поиск (Linear Search)

Определение: последовательный поиск — это простейший алгоритм поиска, который проверяет каждый элемент множества по порядку до тех пор, пока не найдет искомый элемент или не достигнет конца множества.

Анализ: алгоритм начинает с первого элемента массива и последовательно сравнивает каждый элемент с искомым значением.

· Проходит по всем элементам массива от начала до конца

· Сравнивает текущий элемент с целевым значением

· Если элементы равны, возвращает индекс найденного элемента

· Если достигнут конец массива и элемент не найден, возвращает -1

· Не требует предварительной сортировки данных

· Работает с любыми типами массивов (отсортированными и неотсортированными)

Временная сложность:

· Худший случай: O(n) — элемент находится в конце массива или отсутствует

· Лучший случай: O(1) — элемент находится в начале массива

· Средний случай: O(n)

Почему O(n): в худшем случае алгоритму требуется проверить все n элементов массива. Среднее количество проверок составляет n/2.

1. Бинарный поиск (Binary Search)

Определение: бинарный поиск — это эффективный алгоритм поиска в отсортированном массиве, который повторно делит область поиска пополам, сравнивая средний элемент с искомым значением.

Анализ: алгоритм работает только с отсортированными массивами и использует стратегию "разделяй и властвуй":

· Определяет левую и правую границы поиска

· Вычисляет средний элемент между границами

· Сравнивает средний элемент с целевым значением

· Если значения равны, поиск завершен

· Если целевое значение меньше, поиск продолжается в левой половине

· Если целевое значение больше, поиск продолжается в правой половине

· Процесс повторяется до нахождения элемента или сужения области поиска до нуля

Временная сложность:

· Худший случай: O(log n) — элемент отсутствует или находится после максимального количества делений

· Лучший случай: O(1) — элемент находится в середине массива

· Средний случай: O(log n)

Почему O(log n): на каждом шаге алгоритм уменьшает область поиска вдвое. Максимальное количество шагов равно log₂n.

1. Интерполирующий поиск (Interpolation Search)

Определение: интерполирующий поиск — это алгоритм поиска в отсортированном массиве, который предсказывает позицию искомого элемента на основе значений элементов и использует линейную интерполяцию.

Анализ: алгоритм является улучшением бинарного поиска и работает особенно эффективно на равномерно распределенных данных:

· Вычисляет приблизительную позицию элемента по формуле: pos = lo + ((x - arr[lo]) \* (hi - lo)) / (arr[hi] - arr[lo])

· Сравнивает элемент в вычисленной позиции с целевым значением

· Если значения равны, поиск завершен

· Если целевое значение меньше, поиск продолжается в левой части

· Если целевое значение больше, поиск продолжается в правой части

· Требует равномерного распределения данных для максимальной эффективности

Временная сложность:

· Худший случай: O(n) — при неравномерном распределении данных

· Лучший случай: O(1) — точное предсказание позиции

· Средний случай: O(log log n) — при равномерном распределении

Почему O(log log n): на равномерно распределенных данных алгоритм уменьшает область поиска экспоненциально быстрее, чем бинарный поиск.

1. Поиск Фибоначчи (Fibonacci Search)

Определение: поиск Фибоначчи — это алгоритм поиска в отсортированном массиве, который использует числа Фибоначчи для определения позиций сравнения и деления области поиска.

Анализ: алгоритм аналогичен бинарному поиску, но использует числа Фибоначчи для деления массива:

· Находит наименьшее число Фибоначчи, большее или равное размеру массива

· Использует три последовательных числа Фибоначчи для определения позиций сравнения

· На каждом шаге сравнивает элемент с целевым значением и соответствующим образом изменяет область поиска

· Деление массива происходит в золотом сечении (приблизительно 1:1.618)

· Требует только сложения и вычитания, без операций деления

Временная сложность:

· Худший случай: O(log n) — элемент отсутствует или находится после максимального количества сравнений

· Лучший случай: O(1) — элемент находится в первой проверяемой позиции

· Средний случай: O(log n)

Почему O(log n): алгоритм уменьшает размер области поиска в отношении, близком к золотому сечению, что дает логарифмическую сложность, аналогичную бинарному поиску.