Пузырьковая сортировка — это простой алгоритм сортировки, который работает по принципу многократного прохода по массиву, сравнения соседних элементов и обмена их местами, если они находятся в неправильном порядке. Основные характеристики этого алгоритма:

• Сложность:

• Временная сложность в худшем и среднем случае составляет O(n²)

• В лучшем случае (если массив уже отсортирован) O(n)

• Простота реализации: Пузырьковая сортировка является простым для понимания и реализации алгоритмом, однако из-за своей низкой эффективности на больших объемах данных обычно не используется на практике.

• Стабильность: Пузырьковая сортировка является стабильной сортировкой, что означает, что она сохраняет относительный порядок равных элементов.

Исходный массив:

[5, 3, 8, 4, 2]

Шаги сортировки:

1. Первый проход:

- Сравниваем 5 и 3, обмен: [3, 5, 8, 4, 2]

- Сравниваем 5 и 8, ничего не меняем: [3, 5, 8, 4, 2]

- Сравниваем 8 и 4, обмен: [3, 5, 4, 8, 2]

- Сравниваем 8 и 2, обмен: [3, 5, 4, 2, 8]

На первом проходе 8 "всплыл" на последнее место.

2. Второй проход:

- Сравниваем 3 и 5, ничего не меняем: [3, 5, 4, 2, 8]

- Сравниваем 5 и 4, обмен: [3, 4, 5, 2, 8]

- Сравниваем 5 и 2, обмен: [3, 4, 2, 5, 8]

На втором проходе 5 "всплыл" на предпоследнее место.

3. Третий проход:

- Сравниваем 3 и 4, ничего не меняем: [3, 4, 2, 5, 8]

- Сравниваем 4 и 2, обмен: [3, 2, 4, 5, 8]

На третьем проходе 4 занимает свое место.

4. Четвертый проход:

- Сравниваем 3 и 2, обмен: [2, 3, 4, 5, 8]

Теперь массив отсортирован. Пузырьковая сортировка завершает свою работу, когда в очередном проходе не происходит никаких обменов.

**Пирамидальная сортировка** (или heapsort) — это алгоритм сортировки, основанный на структуре данных, называемой кучей (heap). Он относится к классу сортировок с выбором и имеет несколько интересных свойств. Вот основные аспекты теории этого алгоритма:

▎*Основные понятия*

1. Куча (Heap):

• Это специальная структура данных, которая представляет собой бинарное дерево, удовлетворяющее свойству кучи.

• В max-heap (максимальной куче) каждый родительский узел больше или равен своим дочерним узлам. В min-heap (минимальной куче) каждый родитель меньше или равен своим дочерним узлам.

*2. Свойства кучи:*

• Высота кучи составляет O(log n)

• Наиболее важные операции: insert (вставка) и extract-max (извлечение максимального элемента).

▎*Алгоритм heapsort*

Heapsort состоит из двух основных этапов:

1. Построение кучи:

• Исходный массив преобразуется в max-heap. Это делается с помощью процедуры sift-down, которая восстанавливает свойства кучи для каждого узла, начиная с последнего не-листового узла и двигаясь к корню.

• Построение кучи требует O(n) времени.

2. Сортировка:

• После того как max-heap построен, максимальный элемент (корень кучи) будет находиться в начале массива. Он извлекается и помещается в конец массива (или в отсортированную часть).

• Затем выполняется операция sift-down для восстановления свойств кучи для оставшихся элементов.

• Этот процесс повторяется до тех пор, пока не будет отсортирован весь массив. Сортировка занимает O(n log n) времени.

Исходный массив:

[4, 10, 3, 5, 1]

Шаги сортировки:

1. Построение кучи:

- Преобразуем массив в max-кучу. Для нашего массива это будет выглядеть так:

10

/ \

5 4

/ \

3 1

В результате у нас будет:

[10, 5, 4, 3, 1]

2. Сортировка:

- Обмениваем корневой элемент (10) с последним (1):

[1, 5, 4, 3, 10]

- Уменьшаем кучу и восстанавливаем свойство кучи, начиная с корня:

5

/ \

1 4

/ \

3

В результате у нас будет:

[5, 3, 4, 1, 10]

- Повторяем процесс: обменяем корень (5) с (1):

[1, 3, 4, 5, 10]

И восстанавливаем кучу:

4

/ \

3 1

После восстановления:

[4, 3, 1, 5, 10]

- Продолжаем:

[1, 3, 4, 5, 10] → обмен с (3) и восстановление.

Повторяем:

[3, 1, 4, 5, 10]

Наконец, продолжаем до тех пор, пока не получим:

[1, 3, 4, 5, 10]

3. Конечный отсортированный массив:

[1, 3, 4, 5, 10]

**Сортировка вставками** — Он работает по принципу «постепенного построения отсортированной последовательности» из неотсортированных элементов. Давайте рассмотрим основные аспекты этого алгоритма.

▎*Основная идея*

Алгоритм сортировки вставками проходит по массиву и на каждом шаге берет один элемент (называемый "ключом") и вставляет его в уже отсортированную часть массива. Этот процесс повторяется, пока все элементы не будут отсортированы.

▎Пошаговое описание алгоритма

1. Инициализация: Начнем с первого элемента массива, который считается отсортированным.

2. Выбор ключа: На каждой итерации алгоритм выбирает следующий элемент (ключ), который будет вставлен в отсортированную часть массива.

3. Сравнение и перемещение:

• Сравниваем ключ с элементами отсортированной части (элементы слева от ключа).

• Если элемент отсортированной части больше ключа, мы сдвигаем его на одну позицию вправо, чтобы освободить место для вставки ключа.

4. Вставка ключа: Когда мы находим правильное место для ключа (или доходим до начала массива), мы вставляем ключ на это место.

5. Повторение: Повторяем процесс для всех элементов массива, пока не обработаем весь массив.

Рассмотрим массив [5, 2, 9, 1, 5, 6]:

• Начинаем с 5 (первый элемент) — он уже отсортирован.

• Берем 2 — сравниваем с 5, перемещаем 5 вправо, вставляем 2 на его место. Массив: [2, 5, 9, 1, 5, 6].

• Берем 9 — он больше 5, оставляем на месте. Массив: [2, 5, 9, 1, 5, 6].

• Берем 1 — сравниваем с 9, 5, и 2, перемещаем все вправо и вставляем 1. Массив: [1, 2, 5, 9, 5, 6].

• Берем 5 — сравниваем с 9, перемещаем 9 вправо и оставляем 5 на месте. Массив: [1, 2, 5, 5, 9, 6].

• Берем 6 — сравниваем с 9, перемещаем 9 вправо и вставляем 6. Массив: [1, 2, 5, 5, 6, 9].

▎Сложность

• Лучший случай: O(n) —

• Средний и худший случай: O(n²)

**Сортировка слиянием (Merge Sort)** — это один из самых популярных алгоритмов сортировки, который использует метод "разделяй и властвуй".

▎Основные идеи алгоритма

1. Разделяй и властвуй:

• Алгоритм делит массив на две половины.

• Рекурсивно сортирует каждую половину.

• Объединяет (сливает) отсортированные половины в один отсортированный массив.

2. Рекурсия:

• Алгоритм рекурсивно вызывает себя для сортировки подмассивов, пока не достигнет базового случая, когда массив содержит один элемент (который по определению отсортирован).

3. Слияние:

• После сортировки двух половин алгоритм сливает их, сравнивая элементы и помещая их в новый массив в отсортированном порядке.▎Временная сложность

O(n log n)

Исходный массив:

[38, 27, 43, 3, 9, 82, 10]

Шаги сортировки:

1. Разделение:

- Делим массив на две части:

[38, 27, 43] и [3, 9, 82, 10]

- Продолжаем делить каждую половину:

[38] и [27, 43]

[27] и [43]

[3, 9] и [82, 10]

[3] и [9]

[82] и [10]

В итоге мы получаем следующие массивы:

[38], [27], [43], [3], [9], [82], [10]

2. Слияние:

- Начинаем слияние подмассивов:

[27] и [43] → [27, 43]

- Теперь у нас:

[38], [27, 43]

- Сливаем их:

[38] и [27, 43] → [27, 38, 43]

- Продолжаем с [3] и [9]:

[3] и [9] → [3, 9]

- Теперь сливаем [3, 9] и [82, 10]:

[82] и [10] → [10, 82]

- Сливаем:

[3, 9] и [10, 82] → [3, 9, 10, 82]

3. Финальное слияние:

- Теперь у нас есть:

[27, 38, 43] и [3, 9, 10, 82]

- Мы сливаем их:

[27, 38, 43] и [3] → [3, 27, 38, 43]

[3, 27, 38, 43] и [9] → [3, 9, 27, 38, 43]

[3, 9, 27, 38, 43] и [10] → [3, 9, 10, 27, 38, 43]

[3, 9, 10, 27, 38, 43] и [82] → [3, 9, 10, 27, 38, 43, 82]

4. Отсортированный массив:

[3, 9, 10, 27, 38, 43, 82]

**Быстрая сортировка (Quick Sort)** —Основная идея этого алгоритма заключается в использовании метода "разделяй и властвуй", что позволяет быстро сортировать массивы с помощью рекурсивного разбиения.

▎Основные идеи алгоритма

1. Разделяй и властвуй:

• Алгоритм выбирает опорный элемент (pivot) из массива.

• Разделяет массив на две части: элементы, меньшие или равные опорному, и элементы, большие опорного.

• Рекурсивно сортирует обе части.

2. Выбор опорного элемента:

• Опорный элемент можно выбирать разными способами: первым элементом, последним, случайным элементом или медианой.

• Правильный выбор опорного элемента критически важен для производительности алгоритма.

3. Разбиение (Partitioning):

• Процесс разбиения включает перемещение элементов так, чтобы все элементы меньше опорного были слева от него, а все элемент▎Временная сложность

• Лучший случай: O(n log n) — когда массив разбивается на две равные части на каждом шаге.

• Средний случай: O(n log n) — при случайном выборе опорного элемента.

• Худший случай: O(n²) — когда массив уже отсортирован или все элементы равны (например, если всегда выбирается первый или последний элемент в качестве опорного).

Исходный массив:

[10, 7, 8, 9, 1, 5]

Шаги сортировки:

1. Выбор опорного элемента:

- Допустим, выбираем последний элемент 5 в качестве опорного.

2. Разбиение:

- Сравниваем все элементы с опорным (5):

[1, 7, 8, 9, 10] ← (элементы больше 5)

[5] ← (опорный элемент)

[10, 7, 8, 9] ← (элементы меньше 5)

- После разбиения у нас массив выглядит следующим образом:

[1, 5, 7, 8, 9, 10]

3. Рекурсивная сортировка:

- Теперь рекурсивно применяем быструю сортировку к подмассивам:

- Сортируем подмассив меньших элементов:

[1]

- Сортируем подмассив больших элементов:

[7, 8, 9, 10]

- Опять выбираем опорный элемент, например 10. Все элементы меньше 10, имя, и ничего не меняем.

[7, 8, 9, 10]

- Применяем быструю сортировку к подмассиву [7, 8, 9], выбирая, например, 9 как опорный.

[7, 8, 9]

4. Финальное слияние:

- Объединяем отсортированные подмассивы:

[1, 5, 7, 8, 9, 10]

**Поразрядная сортировка (Radix Sort)** — это алгоритм сортировки, который сортирует числа поразрядно, начиная с наименее значимого разряда (Least Significant Digit, LSD) и двигаясь к наиболее значимому (Most Significant Digit, MSD). Этот метод особенно эффективен для сортировки целых чисел и строк фиксированной длины.

▎Основные идеи алгоритма

1. Разбиение по разрядам: Алгоритм сортирует числа по отдельным разрядам, используя другой алгоритм сортировки (чаще всего стабильный, например, сортировку слиянием или подсчетом) для упорядочивания элементов в каждой "корзине".

2. Корзины (buckets): Для каждого разряда создаются корзины, в которые помещаются числа в зависимости от значения текущего разряда. В случае десятичной системы счисления используются 10 корзин (0-9).

3. Многократные проходы: Алгоритм выполняет несколько проходов по массиву — один для каждого разряда, начиная с наименее значимого.

▎Пример работы алгоритма

Рассмотрим массив целых чисел, который мы хотим отсортировать:

arr = [170, 45, 75, 90, 802, 24, 2, 66]

▎Проходы по разрядам

1. Первый проход (единицы):

• Корзины:

• 0: []

• 1: []

• 2: [2]

• 3: []

• 4: [24]

• 5: [45, 75]

• 6: [66]

• 7: [170]

• 8: [802]

• 9: [90]

• Объединение:

arr = [2, 24, 45, 75, 66, 170, 802, 90]

2. Второй проход (десятки):

• Корзины:

• 0: [2]

• 1: [170]

• 2: [24]

• 3: []

• 4: [45]

• 5: [75]

• 6: [66]

• 7: []

• 8: [802]

• 9: [90]

• Объединение:

arr = [2, 24, 45, 66, 75, 90, 170, 802]

3. Третий проход (сотни):

• Корзины:

• 0: [2, 24, 45, 66, 75, 90]

• 1: [170]

• 2: []

• 3: []

• 4: []

• 5: []

• 6: []

• 7: []

• 8: [802]

• 9: []

• Объединение:

arr = [2, 24, 45, 66, 75, 90, 170, 802]

**Сортировка с использованием красно-черного дерева** — это алгоритм сортировки, который использует структуру данных красно-черное дерево (RB-tree) для хранения элементов. — это сбалансированные бинарные деревья поиска, которые гарантируют, что операции вставки, удаления и поиска выполняются за \(O(\log n)\) времени.

Основные понятия:

- Красно-черное дерево: Это специальный тип бинарного дерева поиска, у которого каждый узел имеет два дополнительных свойства:

1. Цвет узла: Узлы могут быть красными или черными.

2. Правила:

- Корень дерева всегда черный.

- Все листья (пустые узлы) являются черными.

- Красный узел не может иметь красного родителя (т.е. два красных узла не могут идти подряд).

- Для любого узла, входящего в дерево, все пути от этого узла до всех его предков содержат одинаковое количество черных узлов. Это правило определяет "черную высоту".

Алгоритм сортировки с использованием красно-черного дерева:

1. Вставка элементов в красно-черное дерево:

- Сначала создается пустое красно-черное дерево.

- Каждый элемент исходного массива поочередно вставляется в дерево, при этом соблюдаются правила красно-черного дерева.

- После каждой вставки выполняется корректировка дерева для поддержания его свойств (например, переопределение цветов на узлах и выполнение вращений).

2. Обход дерева для получения отсортированных данных:

- После того как все элементы были вставлены, выполняется симметричный обход дерева (inorder traversal). При этом элементы извлекаются в порядке возрастания их значений:

- Сначала обходится левое поддерево.

- Затем извлекается значение узла.

- И, наконец, обходится правое поддерево.

- Этот обход гарантирует, что элементы будут возвращаться в отсортированном порядке.

Пример:

Рассмотрим пример сортировки с использованием красно-черного дерева для массива:

[10, 20, 30, 15, 25]

1. Вставка элементов:

- Вставка 10: дерево будет содержать только этот элемент, корень = 10 (черный).

- Вставка 20: добавляется как правый потомок 10 (красный узел).

- Вставка 30: добавляется как правый потомок 20. Здесь мы должны выполнить корректировку, чтобы соблюсти правила. 20 становится черным, 10 остается черным, 30 красным.

- Вставка 15: добавляется как левый потомок 20. Здесь также может потребоваться корректировка. Например, если 10 является красным, его цвет может быть изменен, возможно, для избежания нарушения правил.

- Вставка 25: добавляется как левый потомок 30. Как и прежде, мы следим за балансом.

После всех вставок структура дерева выглядит сбалансированной благодаря правилам красно-черного дерева.

2. Обход для получения отсортированных элементов:

- Выполняя симметричный обход, мы получаем следующий порядок: 10, 15, 20, 25, 30.

Таким образом, массив [10, 20, 30, 15, 25] будет отсортирован в [10, 15, 20, 25, 30] с помощью красно-черного дерева.

**Сортировка с использованием дерева поиска (Binary Search Tree, BST)** основана на концепции бинарного дерева, где каждый узел дерева содержит данные и имеет два подузла: левый и правый. Это дерево построено так, что для любого узла:

- Все значения в левом поддереве меньше значения узла.

- Все значения в правом поддереве больше или равны значению узла.

Основная идея алгоритма:

Алгоритм сортировки с помощью дерева поиска включает в себя следующие ключевые шаги:

1. Вставка элементов:

- Все элементы исходного массива поочередно вставляются в бинарное дерево поиска. При вставке нового элемента дерево корректируется, чтобы сохранить свои свойства.

- В процессе вставки при каждом сравнении с текущим узлом решается, переместиться в левое или правое поддерево.

2. Обход дерева:

- После всех вставок выполняется обход дерева в симметричном порядке (inorder traversal). При таком обходе значения извлекаются в возрастающем порядке:

- Сначала обрабатывается левое поддерево.

- Затем сам узел.

- После этого обрабатывается правое поддерево.

Пример:

Рассмотрим массив чисел:

[15, 10, 20, 8, 12, 17, 25]

#### 1. Вставка элементов:

- Вставка 15:

- Дерево пусто, 15 становится корнем.

15

- Вставка 10:

- 10 меньше 15, вставляем как левый дочерний узел.

15

/

10

- Вставка 20:

- 20 больше 15, вставляем как правый дочерний узел.

15

/ \

10 20

- Вставка 8:

- 8 меньше 15 и меньше 10, становится левым потомком 10.

15

/ \

10 20

/

8

- Вставка 12:

- 12 меньше 15, больше 10, становится правым потомком 10.

15

/ \

10 20

/ \

8 12

- Вставка 17:

- 17 больше 15, меньше 20, становится левым потомком 20.

15

/ \

10 20

/ \ /

8 12 17

- Вставка 25:

- 25 больше 15 и больше 20, становится правым потомком 20.

15

/ \

10 20

/ \ / \

8 12 17 25

#### 2. Обход дерева (inorder):

Теперь выполните симметричный обход, чтобы получить отсортированный массив:

1. Начинаем с корня (15), идем к левому поддереву:

- Идем к 10, идем к его левому поддереву (8), добавляем 8.

2. Вернемся к 10, добавляем 10.

3. Идем к правому поддереву 10 (12), добавляем 12.

4. Вернемся к 15, добавляем 15.

5. Идем к правому поддереву (20), идем к левому поддереву (17), добавляем 17.

6. Вернемся к 20, добавляем 20.

7. Идем к правому поддереву (25), добавляем 25.

Таким образом, после выполнения обхода мы получаем отсортированный массив:

[8, 10, 12, 15, 17, 20, 25]