Оглавление

Методы внутренних сортировок

Сортировка вставками

Сортировка вставками — это простой алгоритм сортировки, который работает аналогично тому, как вы сортируете игральные карты в руках. Массив виртуально разделен на отсортированную и несортированную часть. Значения из несортированной части выбираются и помещаются в правильную позицию в отсортированной части.

Чтобы отсортировать массив размера N в порядке возрастания, выполните итерацию по массиву и сравните текущий элемент (ключ) с его предшественником. Если ключевой элемент меньше своего предшественника, сравните его с предыдущими элементами. Переместите элементы большего размера на одну позицию вверх, чтобы освободить место для замененного элемента.

Алгоритм

Шаг 1: Начальный массив.

Допустим, у нас есть несортированный массив: **[5, 2, 9, 3, 1]**.

Шаг 2. Начало сортировки.

Мы начинаем сортировку со второго элемента (индекс 1), поскольку отсортированным всегда считается отдельный элемент.

1. **Итерация 1:**
   - Текущий массив: **[5, 2, 9, 3, 1]**
   - Ключ (элемент, который нужно вставить): **2**
   - Сравнить **2** с **5**. Так как **2** меньше, то меняем их местами.
   - Обновленный массив: **[2, 5, 9, 3, 1]**
2. **Итерация 2:**
   - Текущий массив: **[2, 5, 9, 3, 1]**
   - Ключ: **9**
   - Сравнить **9** с **5**. Никакого обмена не требуется, так как **9** он больше.
   - Обновленный массив: **[2, 5, 9, 3, 1]**
3. **Итерация 3:**
   - Текущий массив: **[2, 5, 9, 3, 1]**
   - Ключ: **3**
   - Сравните **3** с **9**, затем с **5**. Так как **3** меньше, то меняем их местами.
   - Сравнить **3** с **2**. Так как **3** больше **2,** то оставляем все на своих местах.
   - Обновленный массив: **[2, 3, 5, 9, 1]**

4. **Итерация 4:**
    - Текущий массив: **[2, 3, 5, 9, 1]**
    - Ключ: **1**
    - Сравните **1** с **9**, затем с **5,** после с **3** и **2**. Так как **1** меньше всех этих чисел, то меняем их местами.
    - Обновленный массив: **[1, 2, 3, 5, 9]**

Шаг 3: Окончательный отсортированный массив.

Окончательно отсортированный массив имеет вид **[1, 2, 3, 5, 9]**.

Алгоритм сортировки вставками работает путем итеративного рассмотрения каждого элемента массива и вставки его в правильное положение среди уже отсортированных элементов. Этот процесс продолжается до тех пор, пока весь массив не будет отсортирован.

Анализ сложности сортировки вставками

- Наихудшая временная сложность сортировки вставкой равна O(N^2).
- Средняя временная сложность сортировки вставкой равна O(N^2).
- Временная сложность в лучшем случае равна O(N).

Пространственная сложность сортировки вставками

Сложность вспомогательного пространства при сортировке вставками равна **O(1).**

Характеристики сортировки вставками

- Этот алгоритм является одним из простейших алгоритмов с простой реализацией.
- По сути, сортировка вставкой эффективна для небольших значений данных.
- Сортировка вставками по своей природе является адаптивной, т. е. она подходит для наборов данных, которые уже частично отсортированы.

Источник: https://www.geeksforgeeks.org/insertion-sort/

Сортировка выбором

Сортировка выбором — это простой и эффективный алгоритм сортировки, который работает путем многократного выбора наименьшего (или самого большого) элемента из неотсортированной части списка и перемещения его в отсортированную часть списка.

Алгоритм неоднократно выбирает наименьший (или наибольший) элемент из несортированной части списка и заменяет его первым элементом несортированной части. Этот процесс повторяется для оставшейся неотсортированной части, пока не будет отсортирован весь список.

Алгоритм

1. **Итерация 1:**
    - Текущий массив: **[5, 2, 9, 3, 1]**
    - Найдите наименьший элемент в неотсортированной части. В нашем случае это элемент **1**.
    - Поменяйте его местами с первым элементом неотсортированной части. Элементы **1** и **5** меняются местами.
    - Обновленный массив: **[1, 2, 9, 3, 5]**
2. **Итерация 2:**
    - Текущий массив: **[1, 2, 9, 3, 5]**
    - Найдите наименьший элемент в неотсортированной части. В нашем случае это элемент **2**.
    - Поменяйте его местами с первым элементом неотсортированной части. Элементы **2** и **2** меняются местами.
    - Обновленный массив: **[1, 2, 9, 3, 5]**
3. **Итерация 3:**
    - Текущий массив: **[1, 2, 9, 3, 5]**
    - Найдите наименьший элемент в неотсортированной части. В нашем случае это элемент **3**.
    - Поменяйте его местами с первым элементом неотсортированной части. Элементы **9** и **3** меняются местами.
    - Обновленный массив: [1, 2, 3, 9, 5]
4. **Итерация 4:**
    - Текущий массив: **[1, 2, 3, 9, 5]**

- Найдите наименьший элемент в неотсортированной части. В нашем случае это элемент **5**.
- Поменяйте его местами с первым элементом неотсортированной части. Элементы **9** и **5** меняются местами.
- Обновленный массив: **[1, 2, 3, 5, 9]**

5. **Окончательный отсортированный массив:**

   Окончательный отсортированный массив имеет размер **[1, 2, 3, 5, 9]**.

Сортировка выбором работает путем многократного выбора наименьшего элемента из неотсортированной части и замены его первым элементом неотсортированной части, пока не будет отсортирован весь массив.

Анализ сложности сортировки выбором

Временная сложность: временная сложность сортировки выбором равна $O(N^2)$, поскольку существует два вложенных цикла:

- Один цикл для выбора элемента массива один за другим = O (N)
- Еще один цикл для сравнения этого элемента с любым другим элементом массива = O (N)
- Следовательно, общая сложность = O(N) * O(N) = O(N*N) = $O(N^2)$.

Вспомогательное пространство: O(1), поскольку единственная дополнительная память используется для временных переменных при замене двух значений в массиве. Сортировка выбором никогда не производит более O(N) свопов и может быть полезна, когда запись в память требует больших затрат.

Преимущества алгоритма сортировки выбором

- Просто и легко понять.
- Хорошо работает с небольшими наборами данных.

Недостатки алгоритма сортировки выбором

- Сортировка выбором имеет временную сложность O(n^2) в худшем и среднем случае.
- Не очень хорошо работает с большими наборами данных.
- Не сохраняет относительный порядок элементов с одинаковыми ключами, что означает нестабильность.

Источник: https://www.geeksforgeeks.org/selection-sort/?ref=lbp

Сортировка обменом

Обменная сортировка — это алгоритм, используемый для сортировки как по возрастанию, так и по убыванию. Он сравнивает первый элемент с каждым элементом, если какой-либо элемент кажется неправильным, он заменяет его местами.

Алгоритм

1. **Проход 1:**
    - **Итерация 1:**
        - Текущий массив: **[5, 1, 4, 2, 8]**
        - Сортировка по обмену начинается с самых первых элементов, сравнивая их с другими элементами, чтобы проверить, какой из них больше.
        - Сравним элементы **5** и **1**. Элемент **5** больше **1**. Поменяем их местами.
        - Так как 1 меньше любого другого элемента в данном массиве, то никаких других итераций не нужно.
        - Обновленный массив: **[1, 5, 4, 2, 8].**
2. **Проход 2:**
    - **Итерация 1:**
        - Текущий массив: **[1, 5, 4, 2, 8].**
        - Сравним элементы **5** и **4**. Элемент **5** больше **4**, меняем их местами.
        - Обновленный массив: **[1, 4, 5, 2, 8].**
    - **Итерация 2:**
        - Текущий массив: **[1, 4, 5, 2, 8].**
        - Сравним элементы **4** и **2**. Элемент **4** больше **2**, меняем их местами.
        - Обновленный массив: **[1, 2, 5, 4, 8].**
    - **Итерация 3:**
        - Текущий массив: **[1, 2, 5, 4, 8].**
        - Никаких изменений, поскольку здесь нет другого элемента меньше 2.
        - Обновленный массив: **[1, 2, 5, 4, 8].**
3. **Проход 3:**
    - **Итерация 1:**
        - Текущий массив: **[1, 2, 5, 4, 8].**
        - Сравним элементы **5** и **4**. Элемент **5** больше **4**, меняем их местами.
        - Обновленный массив: **[1, 2, 4, 5, 8].**

- **Итерация 2:**
  - Текущий массив: **[1, 2, 4, 5, 8].**
  - Сравним элементы **4** и **8**. Элемент **4** меньше **5**, изменений нет.
  - Обновленный массив: **[1, 2, 4, 5, 8].**

4. **Проход 4:**
   - **Итерация 2:**
     - Текущий массив: **[1, 2, 4, 5, 8].**
     - Сравним элементы **5** и **8**. Элемент **5** меньше **8**, изменений нет.
     - Обновленный массив: **[1, 2, 4, 5, 8].**
5. **Окончательный отсортированный массив:**

   Окончательный отсортированный массив имеет размер **[1, 2, 4, 5, 8].**

Анализ сложности сортировки обменом

Временная сложность: O(N^2)

Вспомогательное пространство: O(1)

Преимущества алгоритма сортировки обменом

Преимущества алгоритма сортировки обменом перед другими методами сортировки:

- В некоторых ситуациях сортировка обменом может быть предпочтительнее других алгоритмов. Например, сортировка обменом может быть полезна при сортировке очень маленьких массивов или при сортировке данных, которые уже в основном отсортированы. В этих случаях накладные расходы на реализацию более сложного алгоритма могут не оправдать потенциального прироста производительности.
- Еще одним преимуществом сортировки обмена является то, что она стабильна, то есть сохраняет относительный порядок равных элементов. Это может быть важно в некоторых приложениях, например, при сортировке записей, содержащих несколько полей.

Источник: https://www.geeksforgeeks.org/introduction-to-exchange-sort-algorithm/

Сортировка пузырьком

Пузырьковая сортировка — это простейший алгоритм сортировки, который работает путем многократной замены соседних элементов, если они расположены в неправильном порядке. Этот алгоритм не подходит для больших наборов данных, поскольку его средняя и наихудшая временная сложность довольно высока.

Алгоритм пузырьковой сортировки

1. Пройти слева направо и сравнить соседние элементы, причем более высокий помещать справа.
2. Таким образом, самый большой элемент сначала перемещается в самый правый конец.
3. Затем этот процесс продолжается, чтобы найти второй по величине и разместить его, и так далее, пока данные не будут отсортированы.

Алгоритм

1. **Проход 1:**
    - **Итерация 1:**
        - Текущий массив: **[5, 2, 9, 3, 1].**
        - Сравним элементы **5** и **2**. Элемент **5** больше **2**, меняем их местами.
        - Обновленный массив: **[2, 5, 9, 3, 1].**
    - **Итерация 2:**
        - Текущий массив: **[2, 5, 9, 3, 1].**
        - Сравним элементы **5** и **9**. Элемент **5** меньше **9**, обмена не требуется.
        - Обновленный массив: **[2, 5, 9, 3, 1].**
    - **Итерация 3:**
        - Текущий массив: **[2, 5, 9, 3, 1].**
        - Сравним элементы **9** и **3**. Элемент **9** больше **3**, меняем их местами.
        - Обновленный массив: **[2, 5, 3, 9, 1].**
    - **Итерация 4:**
        - Текущий массив: **[2, 5, 3, 9, 1]**
        - Сравним элементы **9** и **1**. Элемент **9** больше **1**, меняем их местами.
        - Обновленный массив: **[2, 5, 3, 1, 9].**
2. **Проход 2:**
    - **Итерация 1:**

- Текущий массив: **[2, 5, 3, 1, 9]**
- Сравним элементы **2** и **5**. Элемент **2** меньше **5**, обмена не требуется.
- Обновленный массив: **[2, 5, 3, 1, 9].**

- **Итерация 2:**
  - Текущий массив: **[2, 5, 3, 1, 9].**
  - Сравним элементы **5** и **3**. Элемент **5** больше **3**, меняем их местами.
  - Обновленный массив: **[2, 3, 5, 1, 9].**
- **Итерация 3:**
  - Текущий массив: **[2, 3, 5, 1, 9].**
  - Сравним элементы **5** и **1**. Элемент **5** больше **1**, меняем их местами.
  - Обновленный массив: **[2, 3, 1, 5, 9].**

3. **Проход 3:**
   - **Итерация 1:**
     - Текущий массив: **[2, 3, 1, 5, 9].**
     - Сравним элементы **2** и **3**. Элемент **2** меньше **3**, обмена не требуется.
     - Обновленный массив: **[2, 3, 1, 5, 9].**
   - **Итерация 2:**
     - Текущий массив: **[2, 3, 1, 5, 9].**
     - Сравним элементы **3** и **1**. Элемент **3** больше **1**, меняем их местами.
     - Обновленный массив: **[2, 1, 3, 5, 9].**
4. Проход 4:
   - **Итерация 1:**
     - Текущий массив: **[2, 1, 3, 5, 9].**
     - Сравним элементы **2** и **1**. Элемент **2** больше **1**, меняем их местами.
     - Обновленный массив: **[1, 2, 3, 5, 9].**
5. Окончательный отсортированный массив: окончательный отсортированный массив имеет размер [1, 2, 3, 5, 9].

Анализ сложности пузырьковой сортировки

Временная сложность: O(N 2)

Вспомогательное пространство: O(1)

Преимущества пузырьковой сортировки

- Пузырьковая сортировка проста для понимания и реализации.
- Он не требует дополнительного места в памяти.

- Это стабильный алгоритм сортировки, означающий, что элементы с одинаковым значением ключа сохраняют свой относительный порядок в отсортированном выводе.

Недостатки пузырьковой сортировки

- Пузырьковая сортировка имеет временную сложность O(N 2), что делает ее очень медленной для больших наборов данных.
- Пузырьковая сортировка — это алгоритм сортировки на основе сравнения. Это означает, что для определения относительного порядка элементов во входном наборе данных требуется оператор сравнения. В некоторых случаях это может ограничивать эффективность алгоритма.

Сортировка пузырьком с улучшениями

1. Запоминаем были или не были перестановки в процессе некоторого прохода. Если в последнем проходе перестановок не было, то алгоритм можно заканчивать.
2. Мы запоминаем индекс элемента с последним обменом. Все элементы до этого индекса уже упорядочены, и, следовательно, отсортированную часть можно увеличить не на один элемент, а до элемента с последним обменом.

Грамотная реализация метода пузырька осуществляется с помощью этих улучшений.

Методы внутренних сортировок с улучшениями

Сортировка вставками с поиском методом половинного деления

Бинарная сортировка вставкой — это алгоритм сортировки, аналогичный сортировке вставкой, но вместо использования линейного поиска для поиска места, куда следует вставить элемент, мы используем двоичный поиск . Таким образом, мы уменьшаем сравнительную ценность вставки одного элемента с O (N) до O (log N).

Это гибкий алгоритм, что означает, что он работает быстрее, когда одни и те же элементы уже тщательно отсортированы, т. е. текущее местоположение объекта ближе к его фактическому положению в отсортированном списке.

Это стабильный алгоритм фильтрации — элементы с одинаковыми значениями появляются в той же последовательности в последнем порядке, что и в первом списке.

Применение сортировки двоичной вставкой:

- Двоичная сортировка вставкой работает лучше всего, когда в массиве меньше элементов.
- При быстрой сортировке или сортировке слиянием, когда размер подмассива становится меньше (скажем, <= 25 элементов), лучше всего использовать двоичную сортировку вставкой.
- Этот алгоритм также работает, когда стоимость сравнений между ключами достаточно высока. Например, если мы хотим отфильтровать несколько строк, производительность сравнения двух строк будет выше.

Как работает двоичная сортировка вставками?

- В режиме двоичной сортировки вставкой мы разделяем одни и те же элементы на два подмассива – фильтрованный и нефильтрованный. Первый элемент из тех же членов находится в организованном подмассиве, а все остальные элементы являются незапланированными.
- Затем мы итерируем от второго элемента к последнему. При повторении i-го мы делаем текущий объект нашим «ключом». Этот ключ — это функция, которую мы должны добавить в наш существующий список ниже.
- Для этого мы сначала используем двоичный поиск в отсортированном подмассиве ниже, чтобы найти местоположение элемента, большего, чем наш ключ. Назовем эту позицию «поз.» Затем мы сдвигаем вправо все элементы с pos на 1 и создаем Array[pos] = key.

- Заметим, что при каждом i-м умножении левая часть массива до (i – 1) уже отсортирована.

Алгоритм

1. **Начальный массив:** Давайте рассмотрим несортированный массив: **[8, 6, 1, 5, 3]**.
2. **Процесс сортировки:** алгоритм поддерживает отсортированный подмассив и неоднократно берет элементы из неотсортированной части, находит их правильную позицию с помощью двоичного поиска и вставляет их в отсортированный подмассив.
    - **Итерация 1:**
        - Текущий массив: **[8, 6, 1, 5, 3]**.
        - Предполагаем, что первый элемент уже отсортирован – 8.
        - Берем второй элемент – 6. Сохраняем в переменной ключе.
        - Теперь мы используем двоичный поиск, чтобы найти элемент слева от текущего элемента, который просто больше его.
        - В данном случае у нас есть только один элемент, 8, и он больше 6. Итак, мы сдвигаем 8 на один индекс вправо и помещаем 6 на его позицию.
        - Обновленный массив: **[6, 8, 1, 5, 3].**
    - **Итерация 2:**
        - Текущий массив: **[8, 6, 1, 5, 3]**.
        - Теперь мы берем третий элемент, 1. Обратите внимание, что все элементы до текущего элемента отсортированы.
        - Мы сохраняем 1 в ключе и находим в отсортированной части элемент чуть больше 1, используя двоичный поиск.
        - Здесь требуемый элемент — 6. Итак, мы сдвигаем 6 и 8 на один индекс вправо и перед сдвигом помещаем 1 на позицию 6.
        - Обновленный массив: **[1, 6, 8, 5, 3]**
    - **Итерация 3:**
        - Текущий массив: **[1, 6, 8, 5, 3]**
        - Теперь мы берем четвертый элемент, 5, и сохраняем его в ключе.
        - Теперь мы берем четвертый элемент, 5, и сохраняем его в ключе.
        - Теперь мы берем четвертый элемент, 5, и сохраняем его в ключе.
        - Обновленный массив: **[1, 5, 6, 8, 3]**
    - **Итерация 4:**
        - Текущий массив: **[1, 5, 6, 8, 3]**

- Теперь мы берем последний (5-й) элемент, равный 3, и находим элемент, который чуть больше его в отсортированной части.
- Требуемый элемент — 5. Мы сдвигаем 5, 6 и 8 на один индекс вправо и перед сдвигом помещаем 3 на позицию 5.
- Обновленный массив: **[1, 3, 5, 6, 8]**

3. **Окончательный отсортированный массив:** окончательный отсортированный массив имеет размер **[1, 3, 5, 6, 8]**

Двоичная сортировка вставками — это улучшение по сравнению с обычной сортировкой вставками, особенно для больших наборов данных, поскольку она уменьшает количество сравнений, необходимых для поиска правильной позиции для каждого элемента.

Источник: https://www.geeksforgeeks.org/binary-insertion-sort/

Шейкерная сортировка

. Коктейльная сортировка — это разновидность пузырьковой сортировки . Алгоритм пузырьковой сортировки всегда перемещает элементы слева и перемещает самый большой элемент в правильное положение на первой итерации, второй по величине на второй итерации и так далее. Коктейльная сортировка поочередно проходит через заданный массив в обоих направлениях. Коктейльная сортировка не требует ненужных итераций, что делает ее эффективной для больших массивов.

Коктейльная сортировка разрушает барьеры, которые ограничивают эффективность пузырьковой сортировки на больших массивах, не позволяя ей проходить ненужные итерации в одном конкретном регионе (или кластере) перед переходом к другому разделу массива.

Алгоритм. Каждая итерация алгоритма разбита на 2 этапа:

1. На первом этапе массив обрабатывается слева направо, как и при сортировке пузырьком. Во время цикла сравниваются соседние элементы, и если значение слева больше значения справа, значения меняются местами. В конце первой итерации наибольшее число будет находиться в конце массива.
2. Второй этап проходит по массиву в противоположном направлении — начиная с элемента непосредственно перед последним отсортированным элементом и возвращаясь к началу массива. Здесь также сравниваются соседние элементы и при необходимости меняются местами.

Случай	Сложность
Лучший случай	O(n)
Средний случай	O(n2)
Худший случай	O(n2)
Максимальное количество сравнений	O(n2)

Коктейльная сортировка, также известная как сортировка шейкером или двунаправленная пузырьковая сортировка, представляет собой разновидность алгоритма пузырьковой сортировки. Как и алгоритм пузырьковой сортировки, коктейльная сортировка сортирует массив элементов, неоднократно меняя местами соседние элементы, если они расположены в неправильном порядке. Однако коктейльная сортировка также перемещается в противоположном направлении после каждого прохода по массиву, что в некоторых случаях делает ее более эффективной.

Основная идея коктейля заключается в следующем

1. Начните с начала массива и сравните каждую соседнюю пару элементов. Если они расположены в неправильном порядке, поменяйте их местами.
2. Продолжайте перебирать массив таким же образом, пока не дойдете до конца массива.
3. Затем двигайтесь в обратном направлении от конца массива к началу, сравнивая каждую соседнюю пару элементов и меняя их местами при необходимости.
4. Продолжайте перебирать массив таким же образом, пока не дойдете до начала массива.
5. Повторяйте шаги 1–4, пока массив не будет полностью отсортирован.

Алгоритм

1. **Начальный массив:** Давайте рассмотрим несортированный массив: **(5 1 4 2 8 0 2)**
2. **Процесс сортировки:** Алгоритм состоит из двунаправленных проходов по массиву. На каждом проходе алгоритм сравнивает и меняет местами соседние элементы.
   - **Проход 1 (вперед):**

     (5 1 4 2 8 0 2) ? (1 5 4 2 8 0 2), 5 > 1, поменять местами

     (1 5 4 2 8 0 2) ? (1 4 5 2 8 0 2), 5 > 4, поменять местами

     (1 4 5 2 8 0 2) ? (1 4 2 5 8 0 2), 5 > 2, поменять местами

     (1 4 2 5 8 0 2) ? (1 4 2 5 8 0 2)

(1 4 2 5 8 0 2) ? (1 4 2 5 0 8 2), 8 > 0, поменять местами

(1 4 2 5 0 8 2) ? (1 4 2 5 0 2 8), 8 > 2, поменять местами

После первого прямого прохода самый большой элемент массива будет присутствовать в последнем индексе массива.

- **Проход 1 (назад):**

  (1 4 2 5 0 2 8) ? (1 4 2 5 0 2 8)

  (1 4 2 5 0 2 8) ? (1 4 2 0 5 2 8), 5 > 0, поменять местами

  (1 4 2 0 5 2 8) ? (1 4 0 2 5 2 8), 2 > 0, поменять местами

  (1 4 0 2 5 2 8) ? (1 0 4 2 5 2 8), 4 > 0, поменять местами

  (1 0 4 2 5 2 8) ? (0 1 4 2 5 2 8), 1 > 0, поменять местами

  После первого обратного прохода наименьший элемент массива будет присутствовать в первом индексе массива.

- **Проход 2 (вперед):**

  (0 1 4 2 5 2 8) ? (0 1 4 2 5 2 8)

  (0 1 4 2 5 2 8) ? (0 1 2 4 5 2 8), 4 > 2, поменять местами

  (0 1 2 4 5 2 8) ? (0 1 2 4 5 2 8)

  (0 1 2 4 5 2 8) ? (0 1 2 4 2 5 8), 5 > 2, поменять местами

- **Проход 2 (назад):**

  (0 1 2 4 2 5 8) ? (0 1 2 2 4 5 8), 4 > 2, поменять местами

  (0 1 2 4 2 5 8) ? (0 1 2 2 4 5 8)

  (0 1 2 4 2 5 8) ? (0 1 2 2 4 5 8)

- Теперь массив уже отсортирован, но наш алгоритм не знает, завершен ли он. Алгоритму необходимо выполнить весь этот проход без какой-либо замены, чтобы знать, что он отсортирован.

  (0 1 2 2 4 5 8) ? (0 1 2 2 4 5 8)

- Окончательный отсортированный массив: окончательный отсортированный массив имеет размер [0, 1, 2, 2, 4, 5, 8].

Преимущества

1. В некоторых случаях коктейльная сортировка может быть более эффективной, чем пузырьковая сортировка, особенно когда сортируемый массив имеет небольшое количество несортированных элементов ближе к концу.
2. Коктейльная сортировка — это простой для понимания и реализации алгоритм, что делает его хорошим выбором для образовательных целей или для сортировки небольших наборов данных.

Недостатки

1. Коктейльная сортировка имеет наихудшую временную сложность O(n^2), что означает, что она может быть медленной для больших наборов данных или наборов данных, которые уже частично отсортированы.
2. Коктейльная сортировка требует дополнительного учета для отслеживания начальных и конечных индексов сортируемых на каждом проходе подмассивов, что может затруднить
3. Алгоритм менее эффективен с точки зрения использования памяти, чем другие алгоритмы сортировки.
4. Существуют более эффективные алгоритмы сортировки, такие как сортировка слиянием и быстрая сортировка, которые имеют лучшую временную сложность в среднем и худшем случае, чем коктейльная сортировка.

Улучшенные методы сортировок

Метод Шелла

Сортировка Шелла — это, главным образом, разновидность сортировки вставками. При сортировке вставками мы перемещаем элементы только на одну позицию вперед. Когда элемент необходимо переместить далеко вперед, требуется множество движений. Идея ShellSort заключается в том, чтобы обеспечить возможность обмена удаленными элементами. При сортировке Шелла мы отсортируем массив по h для большого значения h. Мы продолжаем уменьшать значение h, пока оно не станет равным 1. Говорят, что массив отсортирован по h, если все подсписки каждого h-го элемента отсортированы.

Алгоритм

- Шаг 1 — Начало.
- Шаг 2 — Инициализируйте значение размера зазора. Пример: h
- Шаг 3 — Разделите список на меньшие части. Каждый из них должен иметь равные интервалы для h.
- Шаг 4. Отсортируйте эти подсписки с помощью сортировки вставками.
- Шаг 5. Повторяйте этот шаг 2, пока список не будет отсортирован.
- Шаг 6 – Распечатайте отсортированный список.
- Шаг 7 – Стоп.

Алгоритм сортировки Шелла

1. **Начальный массив:** Давайте рассмотрим несортированный массив:
   [33, 31, 40, 8, 12, 17, 25, 42]
2. **Процесс сортировки:** алгоритм начинается с сортировки элементов, находящихся далеко друг от друга, и постепенно уменьшает разрыв между элементами.
   - **Шаг 1. Определите пробелы (h):**
     - Начните с большого пробела (часто в половину длины массива) и уменьшайте его, пока он не станет равным 1.
     - Давайте использовать последовательность **h = [4, 2, 1]** для этого примера.
   - **Шаг 2: Процесс сортировки:**
     - **Итерация 1 (h=4):**
       - Текущий массив: **[33, 31, 40, 8, 12, 17, 25, 42]**

- Подмассивы (сортируем каждый подмассив методом вставок):
  - [33, 12] - Сравните и поменяйте местами: [12, 33]
  - [31, 17] - Сравните и поменяйте местами: [17, 31]
  - [40, 25] - Сравните и поменяйте местами: [25, 40]
  - [8, 42] – Изменения не требуются
- Обновленный массив: [12, 17, 25, 8, 31, 40, 33, 42]

- **Итерация 2 (h=2):**
  - Текущий массив: **[12, 17, 25, 8, 31, 40, 33, 42]**
  - Подмассивы (сортируем каждый подмассив методом вставок)
    - [12, 25, 31, 33] - Изменения не требуются
    - [17, 8, 40, 42] - Сравнить и поменять местами: [8, 17, 40, 42]
  - Обновленный массив: [8, 12, 17, 25, 31, 33, 40, 42]
- **Итерация 3 (h=1):**
  - Текущий подмассив: [8, 12, 17, 25, 31, 33, 40, 42]
  - На этом этапе весь массив рассматривается как один подмассив и сортируется с использованием стандартной сортировки вставкой.
  - Подмассивы (стандартная сортировка вставками):
    - [8, 12, 17, 25, 31, 33, 40, 42] - Изменения не требуются
  - Обновленный массив: [8, 12, 17, 25, 31, 33, 40, 42]

3. **Окончательный отсортированный массив:** окончательный отсортированный массив имеет размер **[1, 2, 3, 5, 9]**.

Сортировка Шелла достигает своей эффективности за счет быстрого сокращения количества инверсий в массиве, приближая его к отсортированному состоянию перед последним проходом с пробелом, равным 1, что эквивалентно обычной сортировке вставками. Выбор последовательности пропусков может повлиять на производительность алгоритма. В приведенном здесь примере использовалась простая последовательность, но на практике существуют более сложные способы определения последовательности пробелов.

Временная сложность

Временная сложность приведенной выше реализации сортировки Шелла равна $O(n^2)$. В приведенной выше реализации разрыв сокращается вдвое на каждой

итерации. Есть много других способов уменьшить пробелы, что приводит к повышению временной сложности. Смотрите это для более подробной информации.

Сложность наихудшего случая. Сложность сортировки оболочки наихудшего случая равна $O(n^2)$.

Сложность наилучшего случая. Когда данный список массивов уже отсортирован, общее количество сравнений каждого интервала равно размеру данного массива. Таким образом, в лучшем случае сложность равна $\Omega(n \log(n))$

Средняя сложность случая. Сортировка оболочки по средней сложности зависит от интервала, выбранного программистом. $\theta(n \log(n)2)$.Средняя сложность случая: $O(n*\log n) \sim O(n^{1,25})$.

Пространственная сложность. Пространственная сложность сортировки оболочки равна *O(1)* .

Источник: https://www.geeksforgeeks.org/shellsort/

Быстрая сортировка

Быстрая сортировка — это алгоритм сортировки, основанный на алгоритме «Разделяй и властвуй», который выбирает элемент в качестве опорного элемента и разбивает заданный массив вокруг выбранного опорного элемента, помещая опорный элемент в правильное положение в отсортированном массиве.

Алгоритм

1. Из массива выбирается некоторый опорный элемент a[i],
2. Запускается процедура разделения массива, которая перемещает все ключи, меньшие, либо равные a[i], влево от него, а все ключи, большие, либо равные a[i] – вправо.
3. Теперь массив состоит из двух подмножеств, причем левое меньше, либо равно правого,

≤ a[i]	a[i]	≥ a[i]

4. Для обоих подмассивов: если в подмассиве более двух элементов, рекурсивно запускаем для него ту же процедуру.

Временная сложность

Наилучший случай : Ω (N log (N)). Наилучший сценарий быстрой сортировки возникает, когда точка поворота, выбранная на каждом шаге, делит массив примерно на равные половины.

В этом случае алгоритм создаст сбалансированные разделы, что приведет к эффективной сортировке.

Средний случай: θ (N log (N)) Производительность быстрой сортировки в среднем случае на практике обычно очень хороша, что делает его одним из самых быстрых алгоритмов сортировки.

Худший случай: O(N2). Наихудший сценарий для быстрой сортировки возникает, когда поворот на каждом этапе последовательно приводит к сильно несбалансированным разделам. Когда массив уже отсортирован и в качестве опорной точки всегда выбирается наименьший или наибольший элемент. Чтобы смягчить наихудший сценарий, используются различные методы, такие как выбор хорошей опорной точки (например, медианы из трех) и использование рандомизированного алгоритма (рандомизированная быстрая сортировка) для перетасовки элемента перед сортировкой.

Вспомогательное пространство: O(1), если не учитывать рекурсивное стековое пространство. Если мы рассмотрим рекурсивное пространство стека, то в худшем случае быстрая сортировка может составить $O(N)$.

Преимущества быстрой сортировки:

- Это алгоритм «разделяй и властвуй», который упрощает решение проблем.
- Он эффективен для больших наборов данных.
- Он имеет низкие накладные расходы, поскольку для его работы требуется лишь небольшой объем памяти.

Недостатки быстрой сортировки:

- Его временная сложность в наихудшем случае равна O(N^2), что происходит, когда опорная точка выбрана неудачно.
- Это не лучший выбор для небольших наборов данных.
- Это не стабильная сортировка. Это означает, что, если два элемента имеют один и тот же ключ, их относительный порядок не будет сохранен в отсортированном выводе в случае быстрой сортировки, поскольку здесь мы меняем местами элементы в соответствии с положением опорной точки (без учета их исходного значения). должности).

Источник: https://www.geeksforgeeks.org/quick-sort/

Внешние сортировки

Внешняя сортировка — это термин, обозначающий класс алгоритмов сортировки, которые могут обрабатывать огромные объемы данных. Внешняя сортировка требуется, когда сортируемые данные не помещаются в основную память вычислительного устройства (обычно ОЗУ) и вместо этого должны находиться в более медленной внешней памяти (обычно на жестком диске).

Основным понятием при использовании внешней сортировки является понятие серии. **Серия (упорядоченный отрезок)** – это последовательность элементов, которая упорядочена по ключу. Максимальное количество серий в файле *N* (все элементы не упорядочены). Минимальное количество серий одна (все элементы упорядочены).

В основе большинства методов внешних сортировок лежит процедура слияния и процедура распределения. **Слияние** – это процесс объединения двух (или более) упорядоченных серий в одну упорядоченную последовательность при помощи циклического выбора элементов, доступных в данный момент. **Распределение** – это процесс разделения упорядоченных серий на два и несколько вспомогательных файла.

Фаза – это действия по однократной обработке всей последовательности элементов. **Двухфазная сортировка** – это сортировка, в которой отдельно реализуется две фазы: распределение и слияние. **Однофазная сортировка** – это сортировка, в которой объединены фазы распределения и слияния в одну.

Двухпутевым слиянием называется сортировка, в которой данные распределяются на два вспомогательных файла. **Многопутевым слиянием** называется сортировка, в которой данные распределяются на *N (N > 2)* вспомогательных файлов.

Временная сложность

Временная сложность: O(N * log N).

- Время, затраченное на сортировку слиянием, равно O(runs * run_size * log run_size), что равно O(N log run_size).
- Для объединения отсортированных массивов временная сложность равна O(N * log).
- Таким образом, общая временная сложность равна O(N * log run_size + N * log run).
- Поскольку log run_size + log run = log run_size*runs = log N, временная сложность результата будет равна O(N * log N).

Вспомогательное пространство:

O(run_size). **run_size** — это пространство, необходимое для хранения массива.

Источник: https://www.geeksforgeeks.org/external-sorting/

Однофазная сортировка

В данном алгоритме длина серий фиксируется на каждом шаге. В исходном файле все серии имеют длину *1*, после первого шага она равна *2*, после второго – *4*, после третьего – *8*, после *k-го* шага – *2^k*.

Алгоритм

1. Исходный файл *f* разбивается на два вспомогательных файла *f1* и *f2*.
2. Вспомогательные файлы *f1* и *f2* сливаются в файл *f*, при этом одиночные элементы образуют упорядоченные пары.
3. Полученный файл *f* вновь обрабатывается, как указано в шагах 1 и 2. При этом упорядоченные пары переходят в упорядоченные четверки.
4. Повторяя шаги, сливаем четверки в восьмерки и т.д., каждый раз удваивая длину слитых последовательностей до тех пор, пока не будет упорядочен целиком весь файл.

После выполнения *i* проходов получаем два файла, состоящих из серий длины *2^i*. Окончание процесса происходит при выполнении условия *2^i >= n*. Следовательно, процесс сортировки простым слиянием требует порядка *O(log(n))* проходов по данным.

Замечание: При использовании метода прямого слияния не принимается во внимание то, что исходный файл может быть частично отсортированным, т.е. может содержать упорядоченные подпоследовательности записей.

Исходный файл f: **5 7 3 2 8 4 1**

	Распределение	***Слияние***
1 проход	*f1*: 5 3 8 1 *f2*: 7 2 4	*f*: 5 7 2 3 4 8 1
2 проход	*f1*: 5 7 4 8 *f2*: 2 3 1	*f*: 2 3 5 7 1 4 8
3 проход	*f1*: 2 3 5 7 *f2*: 1 4 8	*f*: 1 2 3 4 5 7 8

Исходный и вспомогательные файлы будут ***O(log(n))*** раз прочитаны и столько же раз записаны.

Источник: https://intuit.ru/studies/courses/648/504/lecture/11473

Двухфазная сортировка

Естественная сортировка

В сравнении с методом прямого слияния, сортировка обладает некоторым преимуществом: учитывается тот факт, что могут содержаться упорядоченные подпоследовательности. То есть, ***длинна серии*** не ограничивается, а определяется количеством элементов в уже упорядоченных подпоследовательностях, выделяемых на каждом проходе.

Алгоритм

1. Исходный файл *f* разбивается на два вспомогательных файла *f1* и *f2*. Распределение происходит следующим образом: поочередно считываются записи *ai* исходной последовательности (неупорядоченной) таким образом, что если значения ключей соседних записей удовлетворяют условию *f(ai) <= f(ai+1)*, то они записываются в первый вспомогательный файл *f1*. Как только встречаются *f(ai) > f(ai+1)*, то записи *ai+1* копируются во второй вспомогательный файл *f2*. Процедура повторяется до тех пор, пока все записи исходной последовательности не будут распределены по файлам.

2. Вспомогательные файлы *f1* и *f2* сливаются в файл *f*, при этом серии образуют упорядоченные последовательности.
3. Полученный файл *f* вновь обрабатывается, как указано в шагах *1* и *2*.
4. Повторяя шаги, сливаем упорядоченные серии до тех пор, пока не будет упорядочен целиком весь файл.

Естественное слияние, у которого после фазы распределения количество серий во вспомогательных файлах отличается друг от друга не более чем на единицу, называется ***сбалансированным слиянием***, в противном случае – ***несбалансированное слияние***.

Варианты

1) [17, 3, 0, 12, 6, 9, 19, 1]

2) [5, 10, 18, 0, 2, 7, 11, 14]

3) [20, 8, 4, 16, 1, 13, 6, 3]

4) [9, 19, 15, 2, 10, 7, 0, 4]

5) [14, 6, 12, 18, 5, 1, 11, 3]

6) [0, 8, 2, 17, 9, 13, 7, 16]

7) [3, 19, 10, 1, 15, 4, 11, 20]

8) [12, 6, 0, 14, 5, 18, 2, 8]

9) [7, 17, 9, 3, 13, 1, 16, 10]

10) [4, 0, 15, 6, 12, 2, 14, 8]

11) [16, 9, 5, 18, 1, 11, 7, 3]

12) [2, 10, 19, 0, 6, 13, 8, 4]

13) [11, 7, 14, 20, 5, 1, 15, 9]

14) [0, 3, 17, 8 ,2 ,12 ,6 ,19]

15) [13 ,9 ,4 ,16 ,1 ,11 ,5 ,20]

16) [7 ,15 ,10 ,2 ,18 ,0 ,14 ,6]

17) [1 ,8 ,19 ,3 ,12 ,16 ,9 ,5]

18) [20 ,11 ,7 ,4 ,17 ,2 ,13 ,0]

19) [6 ,14 ,9 ,1 ,15 ,8 ,5 ,19]

20) [3 ,10, 18, 0, 7, 12, 2, 16]

21) [4, 11, 5, 20, 1, 15, 8, 3]

22) [17, 0, 6, 12, 2, 9, 4, 19]

23) [8, 16, 1, 13, 7, 3, 20, 10]

24) [5, 14, 2, 18, 0, 6, 12, 9]

25) [11, 1, 15, 8, 4, 20, 10, 3]

26) [14, 11, 5, 8, 4, 2, 15, 3]

27) [4, 13, 10, 8, 1, 7, 10, 5]

28) [5, 2, 15, 20, 4, 8, 3, 10]

29) [17, 15 10, 4, 8, 11, 13, 3]