Алгоритмы сортировки

Алгоритмы сортировки

- это алгоритм, который упорядочивает набор входных данных в определенном порядке. Основная задача состоит в том, чтобы упорядочить элементы в нужном порядке, чтобы записи были переставлены для упрощения.

В случае, когда элемент в списке имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки.

Сортировка — это перестановка заданного массива или списка элементов в соответствии с оператором сравнения элементов. Он используется для определения их нового порядка в соответствующей структуре данных. Сортировка переупорядочивает все элементы либо по возрастанию, либо по убыванию.

Оптимальные алгоритмы позволяют снизить время выполнения программ и обеспечить предсказуемое поведение системы. Без них приложения будут страдать от затяжных задержек или даже сбоев при работе с большими данными.

Сортировка также служит основой для многих других алгоритмов: от поиска до обработки графов и распознавания образов. При решении таких задач важно учитывать не только количество выполняемых операций, но и объем используемой памяти.

Ключевые понятия

Временная сложность определяет время выполнения алгоритма. Для оценки используется Big O Notation:

- O(n) — линейная сложность, количество операций увеличивается пропорционально размеру входных данных.
- $O(n^2)$ — квадратичная сложность, количество операций возрастает квадратично.
- O(n log n) — логарифмическая сложность, оптимальное время для большинства сортировок, включая Quick Sort и Merge Sort.

Пространственная сложность определяет, сколько дополнительной памяти требует алгоритм помимо самого массива. Чем меньше, тем лучше, особенно при работе с большими объемами данных.

Стабильность. Алгоритм сортировки называется стабильным, если он сохраняет относительный порядок одинаковых элементов. Это особенно важно, если в данных есть несколько ключей для сортировки, например, по фамилии и затем по имени.

Классификация алгоритмов сортировки

Внутренние алгоритмы сортировки - работа выполняется в оперативной памяти. Они отлично справляются с задачами, когда объем данных небольшой и все помещается в ОЗУ. К внутренним алгоритмам относятся:

- пузырьковая сортировка,
- сортировка вставками,
- сортировка выбором,
- быстрая сортировка (Quick Sort),
- сортировка слиянием (Merge Sort).

Внешние алгоритмы сортировки подключают жесткий диск или другие внешние устройства хранения. Используются, когда данных так много, что они не помещаются в оперативной памяти. Например, многопутевая сортировка слиянием разбивает их на несколько файлов, а затем соединяет в один отсортированный массив.

По способу реализации Quick Sort и Merge Sort относят к категории «разделяй и властвуй». Она предполагает, что массив рекурсивно делится на несколько мелких частей, которые затем упорядочиваются и объединяются. При этом каждый алгоритм реализует концепцию по-разному.

Устойчивая сортировка — сортировка, не меняющая относительный порядок сортируемых элементов, имеющих одинаковые ключи, по которым происходит сортировка.

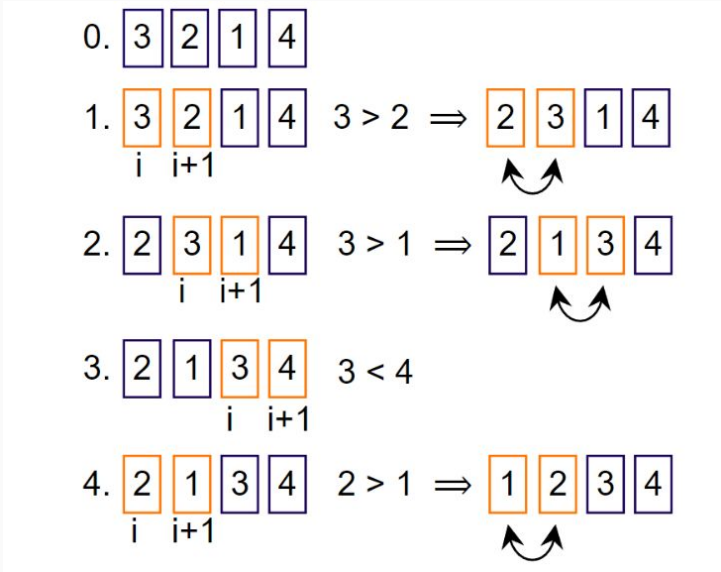
Пузырьковая сортировка

Этот простой алгоритм выполняет итерации по списку, сравнивая элементы попарно и меняя их местами, пока более крупные элементы не «всплывут» в начало списка, а более мелкие не останутся на «дне».

Сначала сравниваются первые два элемента списка. Если первый элемент больше, они меняются местами. Если они уже в нужном порядке, оставляем их как есть. Затем переходим к следующей паре элементов, сравниваем их значения и меняем местами при необходимости. Этот процесс продолжается до последней пары элементов в списке.

При достижении конца списка процесс повторяется заново для каждого элемента. Это крайне неэффективно, если в массиве нужно сделать, например, только один обмен. Алгоритм повторяется n^2 раз, даже если список уже отсортирован.

Для оптимизации алгоритма нужно знать, когда его остановить, то есть когда список отсортирован.



Пузырьковая сортировка

Сложность:

- O(n) - лучший случай
- $O(n^2)$ - средний и худший случаи

Является устойчивым

Преимущества и недостатки:

+ Простота реализации

- Медленный

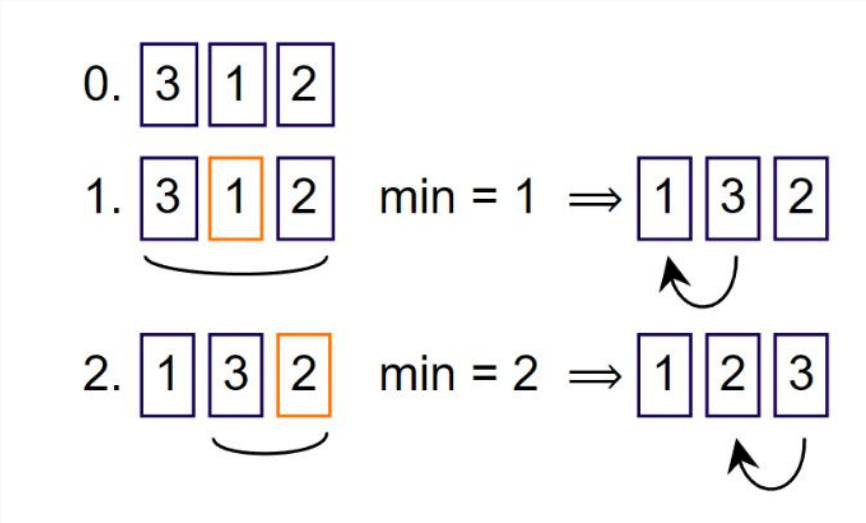
Сортировка выбором (Selection Sort)

Сортировка выбором — также простой алгоритм, но более эффективный по сравнению с пузырьковой сортировкой. В большинстве случаев сортировка выбором будет более удачным выбором из двух.

Этот алгоритм сегментирует список на две части: отсортированную и неотсортированную. Наименьший элемент удаляется из второго списка и добавляется в первый.

На практике не нужно создавать новый список для отсортированных элементов. В качестве него используется крайняя левая часть списка. Находится наименьший элемент и меняется с первым местами.

Теперь, когда нам известно, что первый элемент списка отсортирован, находим наименьший элемент из оставшихся и меняем местами со вторым. Повторяем это до тех пор, пока не останется последний элемент в списке.



Сортировка выбором (Selection Sort)

Сложность:

- $O(n^2)$ - во всех случаях

Не является устойчивым

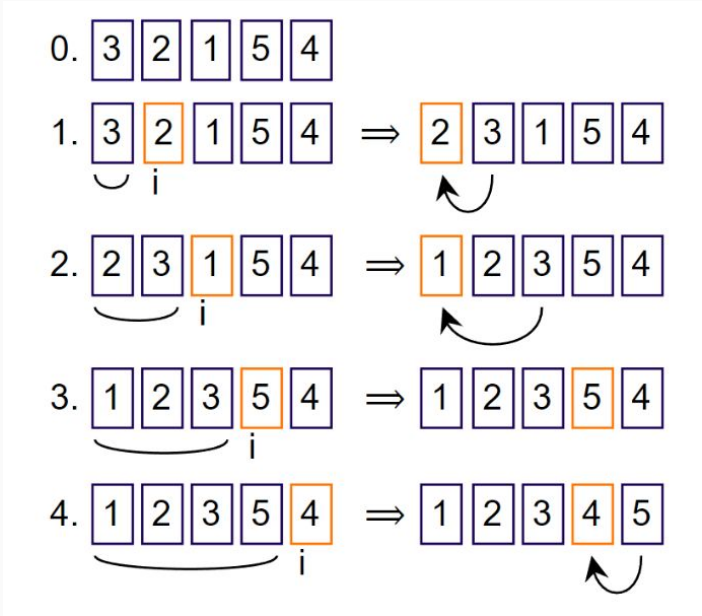
Преимущества и недостатки:
+ Простота реализации
- Медленный

Сортировка вставками (Insertion Sort)

Сортировка вставками быстрее и проще двух предыдущих. Именно так большинство людей тасует карты любой игре. На каждой итерации программа берет один из элементов и подыскивает для него место в уже отсортированном списке. Так происходит до тех пор, пока не останется ни одного неиспользованного элемента.

Предполагается, что первый элемент списка отсортирован. Переходим к следующему элементу, обозначим его x. Если x больше первого, оставляем его на своём месте. Если он меньше, копируем его на вторую позицию, а x устанавливаем как первый элемент.

Переходя к другим элементам несортированного сегмента, перемещаем более крупные элементы в отсортированном сегменте вверх по списку, пока не встретим элемент меньше x или не дойдём до конца списка. В первом случае x помещается на правильную позицию.



Сортировка вставками (Insertion Sort)

Сложность:

- O(n) - лучший случай
- $O(n^2)$ - средний и худший случаи

Является устойчивым

Преимущества и недостатки:

+ Простота реализации
+ Быстрая на частично упорядоченных последовательностях
- Медленная в общем случае

Сортировка слиянием (Merge Sort)

Сортировка слиянием — элегантный пример использования подхода «Разделяй и властвуй». Он состоит из двух этапов:

1. Несортированный список последовательно делится на N списков, где каждый включает один «несортированный» элемент, а N — это число элементов в оригинальном массиве.
2. Списки последовательно сливаются группами по два, создавая новые отсортированные списки до тех пор, пока не появится один финальный отсортированный список.

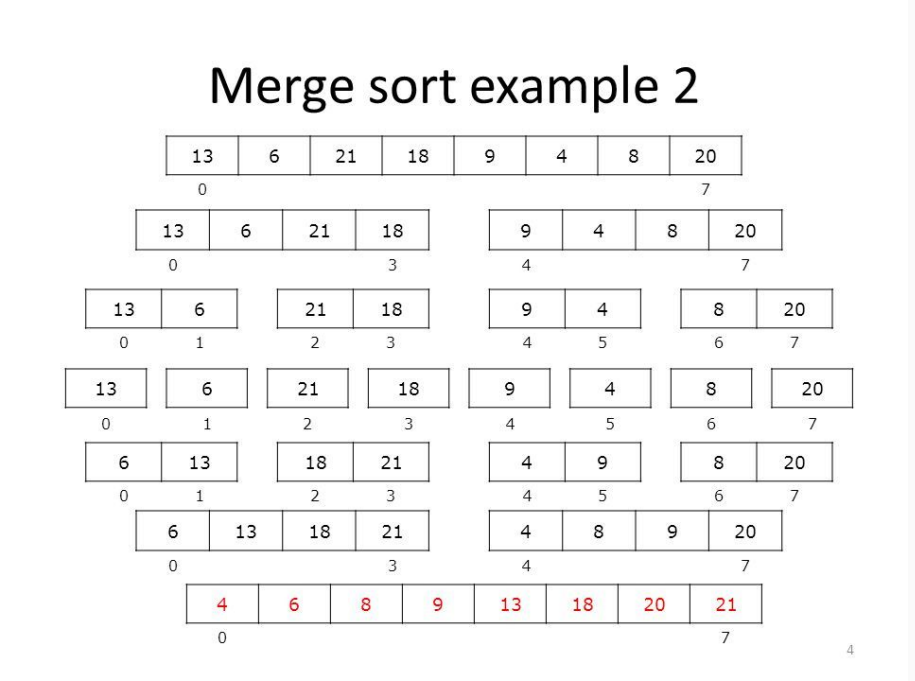
В отличие от предыдущих алгоритмов, рассматриваемый алгоритм возвращает новый список, а не сортирует существующий. Поэтому такая сортировка требует больше памяти для создания нового списка того же размера, что и входной список.

В среднем время сортировки слиянием составляет O(n log n)

Сортировка слиянием (Merge Sort)

Список рекурсивно разделяется пополам, пока в итоге не получатся списки размером в один элемент. Массив из одного элемента считается упорядоченным. Соседние элементы сравниваются и соединяются вместе. Это происходит до тех пор, пока не получится полный отсортированный список.

Сортировка осуществляется путём сравнения наименьших элементов каждого подмассива. Первые элементы каждого подмассива сравниваются первыми. Наименьший элемент перемещается в результирующий массив. Счётчики результирующего массива и подмассива, откуда был взят элемент, увеличиваются на 1.



Быстрая сортировка (Quick Sort)

Этот алгоритм также относится к алгоритмам «разделяй и властвуй». Его используют чаще других алгоритмов. При правильной конфигурации он чрезвычайно эффективен и не требует дополнительной памяти, в отличие от сортировки слиянием. Массив разделяется на две части по разные стороны от опорного элемента. В процессе сортировки элементы меньше опорного помещаются перед ним, а равные или большие — позади.

Алгоритм состоит из трех этапов:

1. Выбирается один опорный элемент.
2. Все элементы меньше опорного перемешаются слева от него, остальные — направо. Это называется операцией разбиения.
3. Рекурсивно повторяются 2 предыдущих шага к каждому новому списку, где новые опорные элементы будут меньше и больше оригинального соответственно.

В среднем время выполнения быстрой сортировки составляет $O(n \log n)$.
Обратите внимание, что алгоритм быстрой сортировки будет работать медленно, если опорный элемент равен наименьшему или наибольшему элементам списка. При таких условиях, в отличие от сортировок кучей и слиянием, обе из которых имеют в худшем случае время сортировки $O(n \log n)$, быстрая сортировка в худшем случае будет выполняться $O(n^2)$.

Быстрая сортировка (Quick Sort)

Пирамидальная сортировка

Также известна как сортировка кучей. Этот популярный алгоритм, как и сортировки вставками или выборкой, сегментирует список на две части: отсортированную и неотсортированную. Алгоритм преобразует второй сегмент списка в структуру данных «куча» (heap), чтобы можно было эффективно определить самый большой элемент.

Сначала преобразуем список в **Max Heap** — бинарное дерево, где самый большой элемент является вершиной дерева. Затем помещаем этот элемент в конец списка. После перестраиваем **Max Heap** и снова помещаем новый наибольший элемент уже перед последним элементом в списке.

Этот процесс построения кучи повторяется, пока все вершины дерева не будут удалены.

В среднем время сортировки кучей составляет $O(n \log n)$, что уже значительно быстрее предыдущих алгоритмов

Пирамидальная сортировка

Сравнение скоростей сортировок

Раз	Пузырьковая	Выборкой	Вставками	Куча	Слиянием	Быстрая
1	5.5318861007	1.2315289974	1.6035542488	0.0400667190	0.0261991024	0.0163919925
2	4.9217622280	1.2472858428	1.5910329818	0.0399959087	0.0258429050	0.0166139602
3	4.9164218902	1.2244019508	1.5936298370	0.0440728664	0.0286228656	0.0164628028
4	5.1547043323	1.2505383491	1.6346361637	0.04128289222	0.0288281440	0.0186078548
5	4.9552288055	1.2898740768	1.6175961494	0.0451571941	0.0331487655	0.0188508033
6	5.0490729808	1.2546651363	1.6251549720	0.0425729751	0.0259521007	0.0162870883
7	5.0559189319	1.2491188049	1.6198101043	0.0402898788	0.0273351669	0.0176029205
8	5.0879919528	1.2580881118	1.6260371208	0.0426468849	0.0263381004	0.0170559883
9	5.0328917503	1.2491509914	1.6144649982	0.0430219173	0.0329370498	0.0176239013
10	5.1429288387	1.2202110290	1.5727391242	0.0396611690	0.0257260799	0.0160610675
Ср. зн.	5.0848807811	1.2474863290	1.6098655700	0.0418768405	0.0280930280	0.0171558380

Сравнение скоростей сортировок

Пузырьковая сортировка — самый медленный из всех алгоритмов. Возможно, он будет полезен как введение в тему алгоритмов сортировки, но не подходит для практического использования.
Быстрая сортировка хорошо оправдывает своё название, почти в два раза быстрее, чем **сортировка слиянием**, и не требуется дополнительное место для результирующего массива.
Сортировка вставками выполняет меньше сравнений, чем **сортировка выборкой** и в реальности должна быть производительнее, но в данном эксперименте она выполняется немного медленней. Сортировка вставками делает гораздо больше обменов элементами. Если эти обмены занимают намного больше времени, чем сравнение самих элементов, то такой результат вполне закономерен.

Встроенные функции сортировки на Python

```
>>> apples_eaten_a_day = [2, 1, 1, 3, 1, 2, 2]
>>> apples_eaten_a_day.sort()
>>> apples_eaten_a_day
[1, 1, 1, 2, 2, 2, 3]
```

```
>>> apples_eaten_a_day_2 = [2, 1, 1, 3, 1, 2, 2]
>>> sorted_apples = sorted(apples_eaten_a_day_2)
>>> sorted_apples
[1, 1, 1, 2, 2, 2, 3]
```

```
# Обратная сортировка списка на месте
>>> apples_eaten_a_day.sort(reverse=True)
>>> apples_eaten_a_day
[3, 2, 2, 2, 1, 1, 1]

# Обратная сортировка, чтобы получить новый список
>>> sorted_apples_desc = sorted(apples_eaten_a_day_2,
reverse=True) >>> sorted_apples_desc
[3, 2, 2, 2, 1, 1, 1]
```

Дополнительно

https://youtu.be/rQtereWDc24