**Обзор и принцип работы основных алгоритмов сортировки и поиска**

**Вводная часть:**

Алгоритмы сортировки и поиска являются важными инструментами в компьютерной науке, используемыми для организации и поиска данных в массивах. Алгоритм сортировки упорядочивает элементы массива в определенном порядке (например, по возрастанию или убыванию), а алгоритм поиска находит конкретный элемент в отсортированном массиве.

**Алгоритмы сортировки:**

Существует множество различных алгоритмов сортировки с различными уровнями сложности и эффективности.

Некоторые распространенные алгоритмы сортировки включают:

* Пузырьковая сортировка: Неэффективный алгоритм, который многократно проходит по массиву, сравнивая попарно соседние элементы и меняя их местами, если они находятся в неправильном порядке.
* Сортировка выбором: Алгоритм, который ищет наименьший элемент в массиве и помещает его в начало. Затем процесс повторяется с оставшимся массивом до тех пор, пока весь массив не будет отсортирован.
* Сортировка вставками: Алгоритм, который работает путем постепенной сортировки массива. Берет каждый элемент массива и вставляет его на правильное место в уже отсортированном фрагменте массива.
* Сортировка слиянием: Эффективный алгоритм, который использует стратегию "разделяй и властвуй". Разделяет массив на меньшие массивы, сортирует их и затем сливает в один отсортированный массив.
* Быстрая сортировка: Еще один эффективный алгоритм, который также использует стратегию "разделяй и властвуй". Выбирает опорный элемент, разделяет массив на элементы меньше и больше опорного, и рекурсивно сортирует обе части.

**Пузырьковая сортировка.**

Представим, что у нас есть массив целых чисел:

1-20219-7a560c.jpg

Во время первого прохода по массиву сравниваются значения 3 и 7. Так как семь больше, всё остаётся в первоначальном виде. Далее сравниваются 7 и 4. Т. к. четыре меньше, цифры меняются местами:

2-20219-2d35ce.jpg

В общем, процесс повторяется, пока 7 не дойдёт практически до конца. Почему практически? Потому что, как вы уже наверняка догадались, последний элемент — это 8, который больше семи, поэтому обмен не происходит. Всё чрезвычайно просто:

3-20219-664e30.jpg

Однако пока обмен происходит, сортировка продолжается, в результате чего перемещается 6:

4-20219-d810d3.jpg

При очередном проходе обмен не выполняется, т. к. все значения массива расположены по порядку. В итоге алгоритм сортировки пузырьком заканчивает свою работу.

Сортировка выбором:

## **Сортировка данных выбором**

Сортировка выбором — некий гибрид между сортировкой вставками и пузырьковой сортировкой. Давайте посмотрим, как работает эта сортировка на нашем массиве:

9-20219-46eae6.jpg

Во время первого же прохода алгоритм посредством метода FindIndexOfSmallestFromIndex пробует найти самое меньшее значение для перемещения его в начало массива.

Так как в нашем примере массив небольшой, мы легко скажем, что это «три», да и вообще, эта цифра уже находится там, где надо. После второго прохода мы получим следующее:

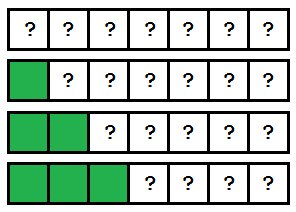
10-20219-87f8d8.jpg

И ещё после 2-х проходов алгоритм сортировки завершит работу:

11-20219-673c99.jpg

**Сортировка вставками:**

Эта сортировка работает путём прохождения по массиву и перемещения нужного значения в его начало. Важно помнить, что сортировка обрабатывает элементы массива по порядку. Т. к. алгоритм работает слева направо, становится очевидно, что всё, что находится слева от текущего индекса, — отсортировано. Давайте посмотрим на увеличение отсортированной части массива с каждым последующим проходом:



По ходу работы отсортированная часть массива растёт, таким образом, в конечном итоге, массив становится упорядоченным.

Приведём пример. Вот неотсортированный массив:

6-20219-f9b31e.jpg

Алгоритм сортировки начнёт работать с индекса «ноль» и значения «три». Т. к. это 1-й индекс, массив до него включительно будет считаться уже отсортированным.

Потом перейдём к семёрке. Семь больше любого значения в отсортированной части, значит, осуществляется переход к последующему элементу. Отметим, что на прошедшем этапе были отсортированы компоненты с индексами 0..1, про компоненты с индексами 2..n мы пока ничего не знаем.

Теперь алгоритм проверяет четвёрку. Четыре меньше, чем 7, поэтому переносится на другую, более правильную позицию, которая находится в отсортированной части массива. Позиция определяется методом FindInsertionIndex. Метод сравнивает переданное значение (в нашем случае это 4) с каждым значением из отсортированной части и так до тех пор, пока не будет найдено место для вставки.

Так для вставки был определён индекс 1. Вставка осуществляется методом Insert. Вставляемое значение удаляется из массива, все остальные цифры сдвигаются вправо, начиная с индекса для вставки. Вот как стал выглядеть массив после сортировки:

7-20219-c78b95.jpg

Итог работы алгоритма сортировки вставками очевиден:

8-20219-363db1.jpg

## **Сортировка данных слиянием**

Предыдущее алгоритмы — линейные (имея квадратичную сложность, они используют мало памяти). Сортировка слиянием соответствует принципу «Divide and conquer» («разделяй и властвуй»). Она работает путём разделения крупной задачи на более мелкие, которые решаются проще.

Представьте, что вы работаете на крыше или стройплощадке, и у вас повредился электрокабель, от которого запитывается важный инструмент. Строительные и кровельные кабели очень длинные и часто достигают 100 и более метров. Вам нужно срочно окончить работу, но у вас нет средств диагностики, чтобы починить провод. Неопытные работники просто прекращают все действия, доложив начальству. Мастера-сдельщики режут кабель пополам, получая в 99 % случаев 50 метров работающего провода. Если нужно, они режут пополам и неработающую часть, что позволяет либо достаточно быстро найти место внешнего повреждения, внимательно изучив четверть кабеля (25 м), либо получить в итоге 75 метров, которых хватит для выполнения большинства строительных задач. Всё, что потребуется, — нож и моток изоленты.

Пример достаточно отдалённый, но всё же помогает понять, что такое сортировка слиянием. При выполнении этого алгоритма массив делится пополам до тех пор, пока каждый участок не будет иметь длину в один элемент. Далее участки сливаются (возвращаются на место), но уже в правильном порядке.

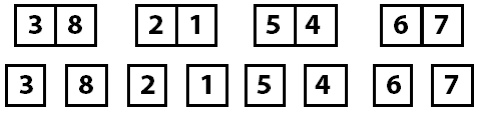
Итак, наш массив:

12-20219-d1ff2e.jpg

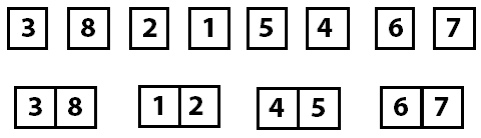
Он делится наполовину:

13-20219-449466.jpg

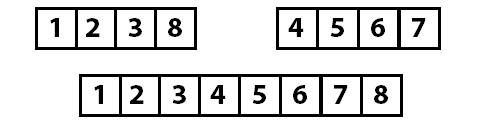
И потом опять, и опять наполовину:



Потом сливание/соединение в верном порядке:



И результат:



Алгоритм работает путём реализации следующих операций: 1. Рекурсивное разделение массива на группы с помощью метода Sort. 2. Слияние в верном порядке через метод Merge.

Сортировка слиянием делит и склеивает массив вне зависимости от того, был ли он изначально отсортирован либо нет. Это значит, что данный алгоритм — не самое оптимальное решение, если наш массив уже частично упорядочен и отсортирован (производительность сортировки слиянием может быть ниже, чем у линейного алгоритма).

## **Быстрая сортировка данных**

Быстрая сортировка тоже представляет собой алгоритм, отвечающий типу «разделяй и властвуй». Сортировка данных происходит рекурсивно и поэтапно: 1. Выбирается ключевой индекс, массив делится по нему на 2 части. Выбор осуществляется разными способами, мы же возьмём случайное число. 2. Все элементы, которые больше ключевого, перемещаются в правую часть массива, которые меньше — в левую. Теперь ключевой элемент больше любого значения слева и меньше любого значения справа. 3. Первые два шага повторяются до полной сортировки массива.

Смотрим на работу алгоритма:

17-20219-f18cc8.jpg

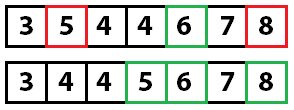
Выбираем ключевой элемент случайным образом:

18-20219-fd215f.jpg

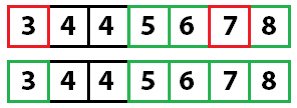
И переносим значения справа от ключевого индекса, располагая их в верном положении (используем метод partition).

19-20219-c4d883.jpg

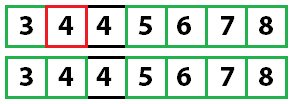
Потом повторяем этот процесс и для левой части. Рекурсивно вызываем метод quicksort для каждой из частей. Ключевым элементом слева становится пятерка — она меняет свой индекс при перемещении значений. Важно не забывать, что нас интересует в первую очередь именно ключевое значение, а не его индекс.



И снова быстрая сортировка:



И опять:



**Алгоритмы поиска:**

**Алгоритмы поиска данных** — это способы нахождения нужного элемента или информации в наборе данных.

Существует несколько основных алгоритмов поиска данных:

1. **Линейный поиск**. Последовательно проверяет каждый элемент в наборе данных, пока не будет найден нужный элемент или пока не будут просмотрены все элементы.
2. **Бинарный поиск**. Делит набор данных на две части и сравнивает искомый элемент с элементом в середине набора данных.
3. **Интерполяционный поиск**. Улучшенная версия бинарного поиска, которая использует интерполяцию для приближённого определения позиции искомого элемента.

Выбор алгоритма поиска данных зависит от характеристик набора данных и требований к эффективности и точности поиска.

### **Линейный поиск**

Линейный поиск – это простой алгоритм, который последовательно проверяет каждый элемент в наборе данных, пока не будет найден нужный элемент или пока не будут просмотрены все элементы. Если элемент найден, алгоритм возвращает его позицию в наборе данных.

Преимущества линейного поиска:

* Простота реализации.
* Работает для неупорядоченных данных.

Недостатки линейного поиска:

* Неэффективен для больших наборов данных, так как требует последовательного просмотра каждого элемента.

### **Бинарный поиск**

Бинарный поиск – это алгоритм, который применяется к упорядоченному набору данных. Он делит набор данных на две части и сравнивает искомый элемент с элементом в середине набора данных. Если элемент равен искомому, алгоритм возвращает его позицию. Если элемент меньше искомого, алгоритм продолжает поиск в правой половине набора данных. Если элемент больше искомого, алгоритм продолжает поиск в левой половине набора данных. Алгоритм повторяет этот процесс до тех пор, пока не будет найден нужный элемент или пока не будет определено, что элемента в наборе данных нет.

Преимущества бинарного поиска:

* Эффективен для больших упорядоченных наборов данных, так как исключает половину набора данных на каждом шаге.

Недостатки бинарного поиска:

* Требует, чтобы набор данных был упорядоченным.
* Неэффективен для неупорядоченных данных.

### **Интерполяционный поиск**

Интерполяционный поиск – это улучшенная версия бинарного поиска, которая использует интерполяцию для приближенного определения позиции искомого элемента. Вместо деления набора данных пополам, интерполяционный поиск использует формулу для вычисления приблизительной позиции искомого элемента. Затем алгоритм сравнивает искомый элемент с элементом в приближенной позиции и продолжает поиск в соответствующей половине набора данных.

Преимущества интерполяционного поиска:

* Эффективен для больших упорядоченных наборов данных, так как приближается к искомому элементу быстрее, чем бинарный поиск.

Недостатки интерполяционного поиска:

* Требует, чтобы набор данных был упорядоченным.
* Неэффективен для неупорядоченных данных.

Выбор алгоритма поиска данных зависит от характеристик набора данных и требований к эффективности и точности поиска.

**Применение:**

Выбор конкретного алгоритма сортировки или поиска зависит от размера данных, характеристик данных (отсортированность, наличие дубликатов) и требуемой производительности. Вот некоторые примеры применения:

* Пузырьковая сортировка: Простая и легко реализуемая, но не подходит для больших массивов.
* Сортировка выбором: Неэффективна для больших массивов, но может использоваться для небольших списков.
* Сортировка вставками: Эффективна для почти отсортированных массивов и малых массивов.
* Сортировка слиянием: Эффективна для больших массивов и является стабильной (сохраняет относительный порядок равных элементов).
* Быстрая сортировка: Очень эффективна для больших массивов, но нестабильна.
* Линейный поиск: Простота реализации, но неэффективен для больших массивов.
* Бинарный поиск: Очень эффективен для больших отсортированных массивов.

**Вывод:**

Изучение и понимание различных алгоритмов сортировки и поиска позволит вам эффективно выбирать наилучший алгоритм для конкретных задач, помогая организовывать и искать данные с максимальной производительностью.

**Тест:**

1. Какой основной принцип работы алгоритма сортировки пузырьком?

a) Разделяй и властвуй

b) Многократный проход по списку, сравнение соседних элементов и обмен ими, если они стоят в неправильном порядке

c) Пошаговая сортировка списка

2. Для какого типа данных подходит сортировка вставками?

a) Только для чисел

b) Только для строк

c) Для любого типа данных

3. Какой алгоритм поиска эффективен только для отсортированных списков?

a) Линейный поиск

b) Бинарный поиск

c) Интерполяционный поиск

4. Какой алгоритм сортировки использует стратегию "разделяй и властвуй"?

a) Сортировка пузырьком

b) Сортировка выбором

c) Быстрая сортировка

5. Для какого алгоритма поиска характерно использование линейной интерполяции?

a) Линейный поиск

b) Бинарный поиск

c) Интерполяционный поиск

6. Какой алгоритм сортировки является наиболее эффективным?

a) Сортировка пузырьком

b) Сортировка выбором

c) Быстрая сортировка

7. Что делает алгоритм сортировки выбором на каждом шаге?

a) Ищет наименьший элемент и меняет его местами с первым элементом

b) Сравнивает соседние элементы и меняет их местами, если они стоят в неправильном порядке

c) Берет каждый элемент и вставляет его на правильное место в уже отсортированной части списка

8. Какой алгоритм поиска имеет наихудшую производительность для больших списков?

a) Линейный поиск

b) Бинарный поиск

c) Интерполяционный поиск

9. Какой принцип лежит в основе алгоритма сортировки слиянием?

a) Разделяй и властвуй

b) Пошаговая сортировка списка

c) Сравнение соседних элементов и обмен ими

10. Для какого типа данных подходит алгоритм сортировки слиянием?

a) Только для чисел

b) Только для строк

c) Для любого типа данных

**Видео:**

https://www.youtube.com/watch?v=kaIP2dkBlbQ&embeds\_referring\_euri=https%3A%2F%2Flaweba.net%2F&source\_ve\_path=Mjg2NjY&feature=emb\_logo