Алгоритм сортировки — это алгоритм для упорядочения элементов в массиве, списке или любом другом контейнере однородных данных. В случае, когда элемент списка имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки. На практике в качестве ключа часто выступает число, а в остальных полях хранятся какие-либо данные, никак не влияющие на работу алгоритма. Мы сегодня сортировать будем массивы целых чисел, и ключом, и полем будет значение массива.

Алгоритмы сортировки оцениваются по скорости выполнения и эффективности использования памяти:

- **Время** — основной параметр, характеризующий быстродействие алгоритма. Называется также вычислительной сложностью.
- **Память** — ряд алгоритмов требует выделения дополнительной памяти под временное хранение данных. Алгоритмы сортировки, не потребляющие дополнительной памяти, относят к *сортировкам на месте*.

Классификация алгоритмов сортировки:

- **Устойчивость** (stability) — устойчивая сортировка не меняет взаимного расположения равных элементов.
- **Естественность поведения** — эффективность метода при обработке уже упорядоченных, или частично упорядоченных данных. Алгоритм ведёт себя естественно, если учитывает эту характеристику входной последовательности и работает лучше.

Ещё одним важным свойством алгоритма является его сфера применения. Здесь основных типов упорядочения два:

- **Внутренняя сортировка** оперирует с массивами, целиком помещающимися в оперативной памяти с произвольным доступом к любой ячейке. Данные обычно упорядочиваются на том же месте, без дополнительных затрат.
- **Внешняя сортировка** оперирует с запоминающими устройствами большого объёма, но с доступом не произвольным, а последовательным (упорядочение файлов), т. е. в данный момент мы 'видим' только один элемент, а затраты на перемотку по сравнению с памятью неоправданно велики. Это накладывает некоторые дополнительные ограничения на алгоритм и приводит к специальным методам упорядочения, обычно использующим дополнительное дисковое пространство. Кроме того, доступ к данным на носителе производится намного медленнее, чем операции с оперативной памятью.

Также алгоритмы классифицируются по:

- потребности в дополнительной памяти или её отсутствии
- потребности в знаниях о структуре данных, выходящих за рамки операции сравнения, или отсутствии таковой

Сортировка пузырьком

Для понимания и реализации этот алгоритм — простейший, но эффективен он лишь для небольших массивов. Сложность алгоритма: $O(n^2)$. Алгоритм считается учебным и практически не применяется вне учебной литературы, вместо него на практике применяются более эффективные алгоритмы сортировки.

Алгоритм состоит в повторяющихся проходах по сортируемому массиву. За каждый проход

элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При проходе алгоритма, элемент, стоящий не на своём месте, «всплывает» до нужной позиции как пузырёк в воде, отсюда и название алгоритма.

Вход: массив A, состоящий из элементов A[1], A[2], ..., A[n-1], A[n]

```
t := истина
цикл пока t:
  t := ложь
  цикл для j = 1, 2, ..., n − 1:
    если A[j] > A[j+1], то:
      обменять местами элементы A[j] и A[j+1]
      t := истина
```

Сортировка вставкой

На каждом шаге алгоритма мы выбираем один из элементов входных данных и вставляем его на нужную позицию в уже отсортированном списке до тех пор, пока набор входных данных не будет исчерпан. Метод выбора очередного элемента из исходного массива произволен; может использоваться практически любой алгоритм выбора. Обычно (и с целью получения устойчивого алгоритма сортировки), элементы вставляются по порядку их появления во входном массиве. Приведенный ниже алгоритм использует именно эту стратегию выбора.

```
void insertionSort(int arr[], int length)
{
    int i, j, tmp;
    for (i = 1; i < length; i++) {
        j = i;
        while (j > 0 && arr[j - 1] > arr[j]) {
            tmp = arr[j];
            arr[j] = arr[j - 1];
            arr[j - 1] = tmp;
            j--;
        }
    }
}
```

Пример:

3 | 1 4 1 5 9

1 3 | 4 1 5 9

1 3 4 | 1 5 9

1 1 3 4 | 5 9

1 1 3 4 5 | 9

Хотя этот алгоритм сортировки уступает в эффективности более сложным (таким как быстрая сортировка), у него есть ряд преимуществ:

- эффективен на небольших наборах данных, на наборах данных до десятков элементов может оказаться лучшим;
- эффективен на наборах данных, которые уже частично отсортированы;
- это устойчивый алгоритм сортировки (не меняет порядок элементов, которые уже отсортированы);
- может сортировать список по мере его получения.

Минусом же является высокая сложность алгоритма: O(n^2).

Сортировка Шелла

Сортировка Шелла — алгоритм сортировки, являющийся усовершенствованным вариантом сортировки вставками. Идея метода Шелла состоит в сравнении элементов, стоящих не только рядом, но и на определённом расстоянии друг от друга. Иными словами — это сортировка вставками с предварительными «грубыми» проходами.

При сортировке Шелла сначала сравниваются и сортируются между собой значения, отстоящие один от другого на некотором расстоянии d. После этого процедура повторяется для некоторых меньших значений d, а завершается сортировка Шелла упорядочиванием элементов при $d = 1$ (то есть, обычной сортировкой вставками). Эффективность сортировки Шелла в определённых случаях обеспечивается тем, что элементы «быстрее» встают на свои места (в простых методах сортировки, например, пузырьковой, каждая перестановка двух элементов уменьшает количество инверсий в списке максимум на 1, а при сортировке Шелла это число может быть больше).

Сортировка выбором

Шаги алгоритма:

1. находим минимальное значение в текущем списке
2. производим обмен этого значения со значением на первой неотсортированной позиции
3. теперь сортируем хвост списка, исключив из рассмотрения уже отсортированные элементы

Псевдокод:

for i := 1 to n-1 do

выбрать среди A[i]..A[n] элемент с наименьшим ключом и поменять его местами с A[i].

Пример:

| 3 1 4 1 5 9

1 | 3 4 1 5 9

1 1 | 4 3 5 9

1 1 3 | 4 5 9

1 1 3 4 | 5 9

1 1 3 4 5 | 9

Сортировка вставкой имеет наименьший оверхэд по сравнению со всеми остальными сортировками вообще, к тому же может применяться к данным по мере их поступления. Зато сортировка выбором требует наименьшего числа копирований данных.

(Тут спрашивают, когда какую применять, надо объяснить случаи, когда дорогая операция копирования, и когда - сравнения. Пример от Гоги - шкафы из Икеи. Померить линейкой их легко, а таскать туда-сюда - сложно).

Быстрая сортировка

Один из быстрых известных универсальных алгоритмов сортировки массивов (в среднем O(*n* log *n*) обменов при упорядочении *n* элементов).

Выбираем некоторое значение v в качестве "опорного элемента", относительно которого переупорядочиваются все элементы массива - меньшие - в начало, большие либо равные - в конец. Получаем какой-то индекс j, такой, что A[1]...A[j] < v, и A[j+1],...,A[n] >= v

Повторяем процесс для кусков A[1]...A[j] и A[j+1]...A[n]

Пример:

3 1 4 1 5 9 2 6 5 3 опорный элемент выбираем как наибольшее значение из двух самых левых различных элементов. Если все элементы кусков массива равны, ничего, естественно, не делаем.

Получаем:

Шаг 1:

3 1 4 1 5 9 2 6 5 3 v = 3

2 1 1 4 5 9 3 6 5 3

Шаг 2:

2 1 1 v = 2 4 5 9 3 6 5 3 v = 5

1 1 2 4 3 3 9 6 5 5

Шаг 3:

1 1 готово 2 готово 4 3 3 v = 4 9 6 5 5 v = 9

3 3 4 5 6 5 9

Шаг 4:

3 3 готово 4 готово 5 6 5 v = 6 9 готово

5 5 6

Шаг 5:

5 5 готово 6 готово

Псевдокод:

```
if A[i],...,A[j] имеют не менее двух различных ключей then begin
   пусть v - наибольший из первых двух найденных различных ключей;
   переставляются элементы A[i], ..., A[j] так, чтобы
       для некоторого k, i+1<=k<=j, A[i], ..., A[k-1] имели ключи,
       меньшие, чем v, а A[k], ..., A[j] - большие либо равные v;
   quicksort(i, k-1);
   quicksort(k, j);
end
```

А еще есть сайт http://www.sorting-algorithms.com/, там все эти алгоритмы в анимированном виде.