# Объяснение алгоритмов сортировки с примерами на Python

В этой статье будут рассмотрены популярные алгоритмы, принципы их работы и реализация на Python. А ещё сравним, как быстро они сортируют элементы в списке.

В качестве общего примера возьмём сортировку чисел в порядке возрастания. Но эти методы можно легко адаптировать под ваши потребности.

## 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».

Алгоритм

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 n² раз, даже если список уже отсортирован.

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

Чтобы остановить алгоритм по окончании сортировки, нужно ввести переменную-флаг. Когда значения меняются местами, устанавливаем флаг в значение True, чтобы повторить процесс сортировки. Если перестановок не произошло, флаг остаётся False и алгоритм останавливается.

Алгоритм работает в цикле while и прерывается, когда элементы ни разу не меняются местами. Вначале присваиваем swapped значение True, чтобы алгоритм запустился хотя бы один раз.

### Время сортировки

Если взять самый худший случай (изначально список отсортирован по убыванию), затраты времени будут равны O(n²), где n — количество элементов списка.

## Сортировка выборкой

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

### Алгоритм

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.

По мере увеличения значения i нужно проверять меньше элементов.

### Время сортировки

Затраты времени на сортировку выборкой в среднем составляют O(n²), где n — количество элементов списка.

## Сортировка вставками

Как и сортировка выборкой, этот алгоритм сегментирует список на две части: отсортированную и неотсортированную. Алгоритм перебирает второй сегмент и вставляет текущий элемент в правильную позицию первого сегмента.

### Алгоритм

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

### Время сортировки

Время сортировки вставками в среднем равно O(n²), где n — количество элементов списка.

## 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.

### Алгоритм

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.

### Реализация

Создадим вспомогательную функцию heapify() для реализации этого алгоритма:

### Время сортировки

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

## Сортировка слиянием

Этот алгоритм относится к алгоритмам «разделяй и властвуй». Он разбивает список на две части, каждую из них он разбивает ещё на две и т. д. Список разбивается пополам, пока не останутся единичные элементы.

Соседние элементы становятся отсортированными парами. Затем эти пары объединяются и сортируются с другими парами. Этот процесс продолжается до тех пор, пока не отсортируются все элементы.

### Алгоритм

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.

### Реализация

Обратите внимание, что функция merge\_sort(), в отличие от предыдущих алгоритмов, возвращает новый список, а не сортирует существующий. Поэтому такая сортировка требует больше памяти для создания нового списка того же размера, что и входной список.

### Время сортировки

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

## Быстрая сортировка

Этот алгоритм также относится к алгоритмам «разделяй и властвуй». Его используют чаще других алгоритмов, описанных в этой статье. 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

Быстрая сортировка начинается с разбиения списка и выбора одного из элементов в качестве опорного. А всё остальное передвигаем так, чтобы этот элемент встал на своё место. Все элементы меньше него перемещаются влево, а равные и большие элементы перемещаются вправо.

### Реализация

Существует много вариаций данного метода. Способ разбиения массива, рассмотренный здесь, соответствует схеме Хоара (создателя данного алгоритма).

### Время выполнения

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(n²).

## Встроенные функции сортировки на Python

Иногда полезно знать перечисленные выше алгоритмы, но в большинстве случаев разработчик, скорее всего, будет использовать функции сортировки, уже предоставленные в языке программирования.

Отсортировать содержимое списка можно с помощью стандартного метода sort():

Или можно использовать функцию sorted() для создания нового отсортированного списка, оставив входной список нетронутым:

Оба эти метода сортируют в порядке возрастания, но можно изменить порядок, установив для флага reverse значение True:

В отличие от других алгоритмов, обе функции в Python могут сортировать также списки кортежей и классов. Функция sorted() может сортировать любую последовательность, которая включает списки, строки, кортежи, словари, наборы и пользовательские итераторы, которые вы можете создать.

Функции в Python реализуют алгоритм [Tim Sort](https://ru.wikipedia.org/wiki/Timsort), основанный на сортировке слиянием и сортировке вставкой.

## Сравнение скоростей сортировок

Для сравнения сгенерируем массив из 5000 чисел от 0 до 1000. Затем определим время, необходимое для завершения каждого алгоритма. Повторим каждый метод 10 раз, чтобы можно было более точно установить, насколько каждый из них производителен.

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.

Вы познакомились с шестью различными алгоритмами сортировокна Python. Масштаб сравнения и количество перестановок, которые выполняет алгоритм вместе со средой выполнения кода, будут определяющими факторами в производительности. В реальных приложениях Python рекомендуется использовать встроенные функции сортировки, поскольку они реализованы именно для удобства разработчика.