**История**. Быстрая сортировка, сортировка Хоара (англ. quicksort — широко известный алгоритм сортировки, разработанный английским информатиком Чарльзом Хоаром во время его работы в МГУ в 1960 году. Один из самых быстрых известных универсальных алгоритмов сортировки массивов: в среднем O(n logn) обменов при упорядочении n элементов; из-за наличия ряда недостатков на практике обычно используется с некоторыми доработками.

**Достоинства**

* Один из самых быстродействующих (на практике) из алгоритмов внутренней сортировки общего назначения.
* Прост в реализации.
* Допускает естественное распараллеливание (сортировка выделенных подмассивов в параллельно выполняющихся подпроцессах).
* Допускает эффективную модификацию для сортировки по нескольким ключам (в частности — алгоритм Седжвика для сортировки строк): благодаря тому, что в процессе разделения автоматически выделяется отрезок элементов, равных опорному, этот отрезок можно сразу же сортировать по следующему ключу.
* Работает на [связных списках](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D1%8F%D0%B7%D0%BD%D1%8B%D0%B9_%D1%81%D0%BF%D0%B8%D1%81%D0%BE%D0%BA) и других структурах с последовательным доступом, допускающих эффективный проход как от начала к концу, так и от конца к началу.

**Недостатки:**

* Сильно деградирует по скорости (до O(n^2)) в худшем или близком к нему случае, что может случиться при неудачных входных данных.
* Прямая реализация в виде функции с двумя рекурсивными вызовами может привести к ошибке [переполнения стека](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%81%D1%82%D0%B5%D0%BA%D0%B0), так как в худшем случае ей может потребоваться сделать O(n) вложенных рекурсивных вызовов.
* [Неустойчив](https://ru.wikipedia.org/wiki/%D0%A3%D1%81%D1%82%D0%BE%D0%B9%D1%87%D0%B8%D0%B2%D0%B0%D1%8F_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0).

**Алгоритм**. Быстрый метод сортировки функционирует по принципу "разделяй и властвуй".

* Массив a[l,..., r] типа T разбивается на два (возможно пустых) подмассива a[l ,..., q] и a[q+1 ,..., r], таких, что каждый элемент a[l ,..., q] меньше или равен a[q], который в свою очередь, не превышает любой элемент подмассива a[q+1 ,…, r]. Индекс вычисляется в ходе процедуры разбиения.
* Подмассивы a[l,...,q] и a[q+1,…, r] сортируются с помощью рекурсивного вызова процедуры быстрой сортировки.
* Поскольку подмассивы сортируются на месте, для их объединения не требуются никакие действия: весь массив a[l ,..., r] оказывается отсортированным.

**Псевдокод**.

**void** quicksort(a: **T**[n], **int** l, **int** r)

**if** l < r

**int** q = partition(a, l, r)

quicksort(a, l, q)

quicksort(a, q + 1, r)

Для сортировки всего массива необходимо выполнить процедуру quicksort(a,0,length|a|-1).

**Разбиение массива**.

Основной шаг алгоритма сортировки — процедура \mathrm{partition}, которая переставляет элементы массива a[l \ldots r] типа T нужным образом. Разбиение осуществляется с использованием следующей стратегии. Прежде всего, в качестве разделяющего элемента произвольно выбирается элемент a[(l + r) / 2] — он сразу займет свою окончательную позицию. Далее начинается просмотр с левого конца массива, который продолжается до тех пор, пока не будет найден элемент, превосходящий по значению разделяющий элемент, затем выполняется просмотр, начиная с правого конца массива, который продолжается до тех пор, пока не отыскивается элемент, который по значению меньше разделяющего. Оба элемента, на которых просмотр был прерван, очевидно, находятся не на своих местах в разделенном массиве, и потому они меняются местами. Так продолжаем дальше, пока не убедимся в том, что слева от левого указателя не осталось ни одного элемента, который был бы больше по значению разделяющего, и ни одного элемента справа от правого указателя, которые были бы меньше по значению разделяющего элемента.

Переменная v сохраняет значение разделяющего элемента a[(l + r) / 2], a i и j представляет собой, соответственно, указатели левого и правого просмотра. Цикл разделения увеличивает значение i и уменьшает значение j на 1, причем условие, что ни один элемент слева от i не больше v и ни один элемент справа от j не меньше v, не нарушается. Как только значения указателей пересекаются, процедура разбиения завершается.

**int** partition(a: **T**[n], **int** l, **int** r)

**T** v = a[(l + r) / 2]

**int** i = l

**int** j = r

**while** (i \leqslant j)

**while** (a[i] < v)

i++

**while** (a[j] > v)

j--

**if** (i \leqslant j)

swap(a[i++], a[j--])

**return** j

**Асимптотика**.

Предположим, что мы разбиваем массив так, что одна часть содержит n - 1 элементов, а вторая — 1. Поскольку процедура разбиения занимает время \Theta(n), для времени работы T(n) получаем соотношение:

T(n) = T(n - 1) + \Theta(n) = \sum\limits_{k=1}^{n} \Theta(k) = \Theta(\sum\limits_{k=1}^{n} k) = \Theta(n^2).

Мы видим, что при максимально несбалансированном разбиении время работы составляет \Theta(n^2). В частности, это происходит, если массив изначально отсортирован.

Заполним сначала массив a длины n элементами от 1 до n, затем применим следующий алгоритм (нумерация с нуля):

**void** antiQsort(a: **T**[n])

**for** i = 0 **to** n - 1

swap(a[i], a[i / 2])

n = 4: 2 4 1 3

n=7: 2 4 6 7 1 5 3

**Улучшенная быстрая сортировка**.

Выбор медианы из первого, среднего и последнего элементов в качестве разделяющего элемента и отсечение рекурсии меньших подмассивов может привести к существенному повышению эффективности быстрой сортировки. Функция \mathrm{median} возвращает индекс элемента, являющегося медианой трех элементов. После этого он и средний элемент массива меняются местами, при этом медиана становится разделяющим элементом. Массивы небольшого размера (длиной M = 11 и меньше) в процессе разделения игнорируются, затем для окончания сортировки используется сортировка вставками.

**const int** M = 10

**void** quicksort(a: **T**[n], **int** l, **int** r)

**if** (r - l \leqslant M)

insertion(a, l, r)

**return**

**int** med = median(a[l], a[(l + r) / 2], a[r])

swap(a[med], a[(l + r) / 2])

**int** i = partition(a, l, r)

quicksort(a, l, i)

quicksort(a, i + 1, r)

**История. Сортировка слиянием** (англ. *Merge sort*) — алгоритм сортировки, пред­ло­женный Джо­ном фон Ней­ма­ном в 1945 го­ду. Это устойчивый алгоритм, использующий O(n) дополнительной памяти и работающий за O(n \log n) времени.

**Алгоритм**.

Алгоритм использует принцип «разделяй и властвуй»: задача разбивается на подзадачи меньшего размера, которые решаются по отдельности, после чего их решения комбинируются для получения решения исходной задачи. Конкретно процедуру сортировки слиянием можно описать следующим образом:

1. Если в рассматриваемом массиве один элемент, то он уже отсортирован — алгоритм завершает работу.
2. Иначе массив разбивается на две части, которые сортируются рекурсивно.
3. После сортировки двух частей массива к ним применяется процедура слияния, которая по двум отсортированным частям получает исходный отсортированный массив.

**Достоинства**:

* устойчивая,
* можно написать эффективную многопоточную сортировку слиянием,
* сортировка данных, расположенных на периферийных устройствах и не вмещающихся в оперативную память.

**Недостатки**:

* при любых входных данных время работы — O(n\log{n}),
* требуется дополнительно O(n) памяти, но можно модифицировать до O(1).

**Псевдокод**.

Функция сортирует подотрезок массива с индексами в полуинтервале [left; right).

**function** mergeSortRecursive(a : **int[n]**; left, right : **int**):

**if** left + 1 >= right

**return**

mid = (left + right) / 2

mergeSortRecursive(a, left, mid)

mergeSortRecursive(a, mid + 1, right)

merge(a, left, mid, right)

**Слияние двух массивов**.

У нас есть два массива a и b (фактически это будут две части одного массива, но для удобства будем писать, что у нас просто два массива). Нам надо получить массив c размером |a| + |b|. Для этого можно применить процедуру слияния. Эта процедура заключается в том, что мы сравниваем элементы массивов (начиная с начала) и меньший из них записываем в финальный. И затем, в массиве у которого оказался меньший элемент, переходим к следующему элементу и сравниваем теперь его. В конце, если один из массивов закончился, мы просто дописываем в финальный другой массив. После мы наш финальный массив записываем заместо двух исходных и получаем отсортированный участок.

Множество отсортированных списков с операцией \mathrm{merge} является моноидом, где нейтральным элементом будет пустой список.

Ниже приведён псевдокод процедуры слияния, который сливает две части массива a — [left; mid) и [mid; right)

**function** merge(a : **int[n]**; left, mid, right : **int**):

it1 = 0

it2 = 0

result : **int[right - left]**

**while** left + it1 < mid **and** mid + it2 < right

**if** a[left + it1] < a[mid + it2]

result[it1 + it2] = a[left + it1]

it1 += 1

**else**

result[it1 + it2] = a[mid + it2]

it2 += 1

**while** left + it1 < mid

result[it1 + it2] = a[left + it1]

it1 += 1

**while** mid + it2 < right

result[it1 + it2] = a[mid + it2]

it2 += 1

**for** i = 0 **to** it1 + it2

a[left + i] = result[i]

**Сложность**.

Чтобы оценить время работы этого алгоритма, составим рекуррентное соотношение. Пускай T(n) — время сортировки массива длины n, тогда для сортировки слиянием справедливо T(n)=2T(n/2)+O(n)   
O(n) — время, необходимое на то, чтобы слить два массива. Распишем это соотношение:

T(n)=2T(n/2)+O(n)=4T(n/4)+2O(n)=\dots=2^kT(1)+kO(n).

Осталось оценить k. Мы знаем, что 2^k=n, а значит k=\log n. Уравнение примет вид T(n)=nT(1)+ \log n O(n). Так как T(1) — константа, то T(n)=O(n)+\log n O(n)=O(n\log n).