На входе массив a[0]...a[N] и опорный элемент p, по которому будет производиться разделение.

1. Введем два указателя: i и j. В начале алгоритма они указывают, соответственно, на левый и правый конец последовательности.
2. Будем двигать указатель i с шагом в 1 элемент по направлению к концу массива, пока не будет найден элемент a[i] >= p. Затем аналогичным образом начнем двигать указатель j от конца массива к началу, пока не будет найден a[j] <= p.
3. Далее, если i <= j, меняем a[i] и a[j] местами и продолжаем двигать i,j по тем же правилам...
4. Повторяем шаг 3, пока i <= j.

Рассмотрим работу процедуры для массива a[0]...a[6] и опорного элемента p = a[3].

Каждое разделение требует, очевидно, Theta(n) операций. Количество шагов деления(глубина рекурсии) составляет приблизительно log n, если массив делится на более-менее равные части. Таким образом, общее быстродействие: O(n log n), что и имеет место на практике.

Однако, возможен случай таких входных данных, на которых алгоритм будет работать за O(n2) операций. Такое происходит, если каждый раз в качестве центрального элемента выбирается максимум или минимум входной последовательности. Если данные взяты случайно, вероятность этого равна 2/n. И эта вероятность должна реализовываться на каждом шаге... Вообще говоря, малореальная ситуация.

Метод неустойчив. Поведение довольно естественно, если учесть, что при частичной упорядоченности повышаются шансы разделения массива на более равные части.

Сортировка использует дополнительную память, так как приблизительная глубина рекурсии составляет O(log n), а данные о рекурсивных подвызовах каждый раз добавляются в стек.