Казанский (Приволжский) федеральный университет

Институт Информационных технологий и интеллектуальных систем

Отчет

по семестровой работе

на тему:

“Быстрая сортировка (Quicksort)”

Выполнил: Тихомиров Матвей,

гр. 11-002

2021 г.

**Быстрая сортировка**, **сортировка Хоара** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) quicksort), часто называемая *qsort* (по имени в [стандартной библиотеке языка Си](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%BD%D0%B0%D1%8F_%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_%D1%8F%D0%B7%D1%8B%D0%BA%D0%B0_%D0%A1%D0%B8)) — [алгоритм сортировки](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B8), разработанный английским информатиком [Тони Хоаром](https://ru.wikipedia.org/wiki/%D0%A5%D0%BE%D0%B0%D1%80,_%D0%A7%D0%B0%D1%80%D0%BB%D1%8C%D0%B7_%D0%AD%D0%BD%D1%82%D0%BE%D0%BD%D0%B8_%D0%A0%D0%B8%D1%87%D0%B0%D1%80%D0%B4) во время его работы в [МГУ](https://ru.wikipedia.org/wiki/%D0%9C%D0%93%D0%A3) в [1960 году](https://ru.wikipedia.org/wiki/1960_%D0%B3%D0%BE%D0%B4).

Т. Хоар разработал этот метод для машинного перевода текстов; словарь хранился на [магнитной ленте](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D0%B3%D0%BD%D0%B8%D1%82%D0%BD%D0%B0%D1%8F_%D0%BB%D0%B5%D0%BD%D1%82%D0%B0), и сортировка слов обрабатываемого текста позволяла получить их переводы за один прогон ленты, без перемотки её назад.

* Описание алгоритма

Общая идея алгоритма состоит в следующем:

* Из массива выбирается элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность.
* Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные» и «бóльшие».
* Для отрезков «меньших» и «бóльших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

Как правило, массив обычно делят только на две части: например, «меньшие опорного» и «равные и бóльшие»; такой подход в общем случае эффективнее, так как упрощает алгоритм разделения.

[Один из самых быстрых](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C) известных универсальных алгоритмов сортировки массивов: в среднем *O(n\*log n)* обменов при упорядочении n элементов; из-за наличия ряда недостатков на практике обычно используется с некоторыми доработками.

Алгоритм функционирует по принципу "разделяй и властвуй". Он разбивает сортируемый массив на две части, затем сортирует эти части независимо друг от друга.

* Оценка сложности алгоритма

Операция разделения массива на две части относительно опорного элемента занимает время **O(log2n)**. Поскольку все операции разделения, проделываемые на одной глубине рекурсии, обрабатывают разные части исходного массива, размер которого постоянен, суммарно на каждом уровне рекурсии потребуется также O(n) операций. Следовательно, общая сложность алгоритма определяется лишь количеством разделений, то есть глубиной рекурсии. Глубина рекурсии, в свою очередь, зависит от сочетания входных данных и способа определения опорного элемента.

В наиболее сбалансированном варианте при каждой операции разделения массив делится на две одинаковые (плюс-минус один элемент) части, следовательно, максимальная глубина рекурсии, при которой размеры обрабатываемых подмассивов достигнут 1, составит **log2n**. В результате количество сравнений, совершаемых быстрой сортировкой, было бы равно значению рекурсивного выражения **Cn = 2\*Cn/2 + n**, что даёт общую сложность алгоритма **O(n\*log2n)**.

Среднюю сложность при случайном распределении входных данных можно оценить лишь вероятностно. Необходимо заметить, что в действительности необязательно, чтобы опорный элемент всякий раз делил массив на две одинаковых части. Например, если на каждом этапе будет происходить разделение на массивы длиной 75 % и 25 % от исходного, глубина рекурсии будет равна **log4/3n**, а это по-прежнему даёт сложность **O(n\*log n)**. Вообще, при любом фиксированном соотношении между левой и правой частями разделения сложность алгоритма будет той же, только с разными константами.

В худшем варианте каждое разделение даёт два подмассива размерами 1 и n-1, то есть при каждом рекурсивном вызове больший массив будет на 1 короче, чем в предыдущий раз. Такое может произойти, если в качестве опорного на каждом этапе будет выбран элемент либо наименьший, либо наибольший из всех обрабатываемых. В этом случае потребуется n-1 операций разделения, а общее время работы **O(n2)** операций, то есть сортировка будет выполняться за квадратичное время. Также в таком случае глубина рекурсии при выполнении алгоритма достигнет n, что будет означать n-кратное сохранение адреса возврата и локальных переменных процедуры разделения массивов.

* Достоинства и недостатки алгоритма

Плюсы:

* Один из самых быстродействующих (на практике) из алгоритмов внутренней сортировки общего назначения.
* Алгоритм очень короткий: запомнив основные моменты, его легко написать «из головы», невелика константа при *n\*log n*.
* Требует лишь O(1) дополнительной памяти для своей работы. (Не улучшенный рекурсивный алгоритм в худшем *O(n)* памяти)
* Хорошо сочетается с механизмами кэширования и виртуальной памяти.
* 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 и других структурах с последовательным доступом, допускающих эффективный проход как от начала к концу, так и от конца к началу.

Минусы:

* Сильно деградирует по скорости (до *O(n2)*) 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, так как в худшем случае ей может потребоваться сделать *O(n)* вложенных рекурсивных вызовов.
* Неустойчив (в сортировке возможна смена относительного порядка сортируемых элементов, имеющих одинаковые ключи, по которым происходит сортировка).
* Список использованной литературы (источников):
* [Быстрая сортировка — Википедия (wikipedia.org)](https://ru.wikipedia.org/wiki/%D0%91%D1%8B%D1%81%D1%82%D1%80%D0%B0%D1%8F_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0)
* Кормен. Алгоритмы - построение и анализ.
* Седжвик. Фундаментальные алгоритмы на C++
* Код  
  [aisd/Main.cpp at master · Vedmeyga/aisd (github.com)](https://github.com/Vedmeyga/aisd/blob/master/QuickSort/Main.cpp)