**Лекция 6.**

Быстрая сортировка. Специализированные алгоритмы внутренней сортировки

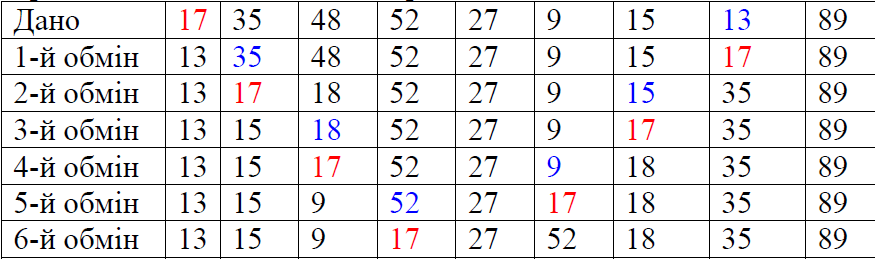
**Быстрая сортировка Хоара**

Усовершенствованный метод сортировки, основанный на обмене, К. Хоар предложил алгоритм QuickSort сортировки массивов, дает на практике отличные результаты и очень просто программируется. Это сортировки называют быстрым, так как на практике оно оказывается самым быстрым методом сортировки из тех, оперирующих сравнениями.

Основная стратегия ускорения алгоритмов сортировки - обмен между как можно более отдаленными элементами исходного файла.

Идея К. Хоара состоит в следующем: на каждом шагу метода мы сначала выбираем "средний" элемент, затем переставляем элементы массива так, что он делится на три части: сначала идут элементы, меньшие "среднего", затем равные ему, а в третьей части - больше. После такого распределения массива остается только отсортировать первую и третью его части, с которыми мы сделаем аналогично (разделим на три части). И так до тех пор, пока эти части не будут состоять из одного элемента, а массив из одного элемента всегда отсортирован.

Выбор "среднего" - задача непростая, так как нужно, не выполняя сортировки, найти элемент со значением максимально близким к среднему. Здесь, конечно, можно просто выбрать произвольный элемент (обычно выбирают элемент, стоящий в середине подмассив, что сортируется), но можем выбирать из трех элементов самого левого, самого правого и стоящего посередине.



**Cложность:**

Анализ сложности алгоритма в среднем, использующий гипотезу о равной вероятности всех входов, показывает, что

C (n) = O (n log2 n), M(n) = O (n log2 n).

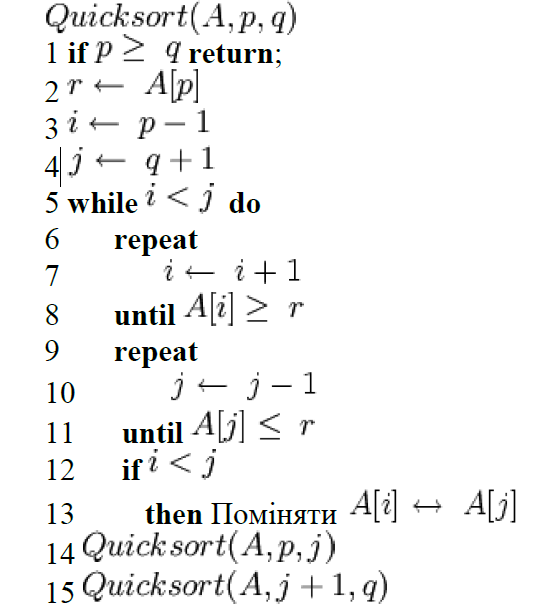
В худшем случае, когда в качестве барьерного выбирается, например, максимальный элемент подмассив, сложность алгоритма квадратичная.

Быстрая сортировка является алгоритмом на основе сравнений, и не является стабильным.

**Классическая реализация**

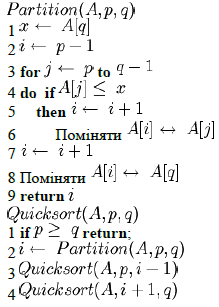
В классическом варианте, предложенном Хоаром, из массива избирался один элемент, и весь массив разбивался на две части по принципу: в первой части - те что не больше данного элемента, во второй части - те что не менее данного элемента. Процедура Quicksort (A, p, q) осуществляет частичное

*упорядочение массива с p-го по q-й индекс*:



**Cовременная реализация**

На сегодня в стандартных библиотеках используют такую ​​реализацию алгоритма:



**Анализ**

Время работы алгоритма сортировки зависит от сбалансированности, характеризующий разбиения. Сбалансированность, в свою очередь зависит от того, какой элемент выбран в качестве опорного (относительно какого элемента выполняется разбивка). Если разбиение сбалансированное, то асимптотически алгоритм работает так же быстро как и алгоритм сортировки слиянием. В худшем случае, асимптотическое поведение алгоритма столь же плоха, как и в алгоритмах сортировка вставками.

**худшее разбиения**

Худшая поведение имеет место в том случае, когда процедура, выполняющая разбиения, порождает одну подзадачу с n-1 элементом, а другу- с 0 элементами. Пусть такое несбалансированное разбиения возникает при каждом рекурсивном вызове. Для самого разбиения нужно время. Тогда, рекуррентное соотношение для времени работы, можно записать так:

Решением такого соотношения является.

**лучше разбиение**

В лучшем случае процедура Partition делит задачу на две подзадачи, размер каждой не превышает n / 2. Время работы, описывается неравенством:

Тогда: -асимптотично лучшее время.

**средний случай**

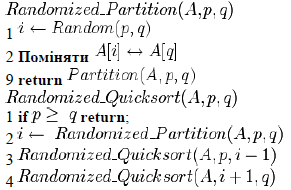
Математическое ожидание времени работы алгоритма на всех возможных входных массивах есть, то есть средний случай ближе к лучшему.

**модификации**

В среднем алгоритм работает очень быстро, но на практике, не все возможные входные массивы имеют одинаковую вероятность. Тогда, путем добавления рандомизации удается получить среднее время работы в любом случае.

**Рандомизованний алгоритм**

В рандомизованному алгоритме, при каждом разбиении случайный элемент избирается в качестве опорного:



**Сортировка подсчетом**

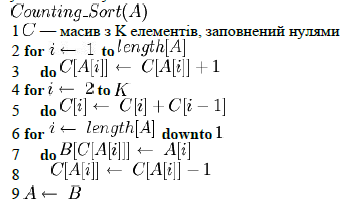
**Сортировка подсчетом** - алгоритм упорядочения, что применяется при малом количестве различных элементов (ключей) в массиве данных. Время его работы линейно зависит как от общего количества элементов в массиве так и от количества различных элементов.

**идея алгоритма**

Идея алгоритма заключается в следующем: сначала подсчитать сколько раз каждый элемент (ключ) встречается в исходном массиве. Опираясь на эти данные можно сразу вычислить на каком месте должен стоять каждый элемент, а потом за один проход поставить все элементы на свои места.

**псевдокод алгоритма**

Для простоты будем считать, что все элементы (ключи) являются натуральными числами лежащие в диапазоне 1..K. Процедура выполняет сортировку массива:



**анализ алгоритма**

В алгоритме присутствуют только простые циклы: в строках 2, 6, 9 - цикл длины N (длина массива), в строке 4 - цикл длины K (величина диапазона). Итак сложность работы алгоритма есть.

В алгоритме используются два дополнительных массивы: и. Поэтому алгоритм требует дополнительной памяти.

В такой реализации алгоритм является стабильным. Именно это его свойство позволяет использовать его как часть других алгоритмов сортировки (например. Сортировка по разрядам).

Использование данного алгоритма целесообразно только в случае малых K (порядка N).

**Сортировка по разрядам**

**Сортировка по разрядам** (англ. Radix sort) - быстрый стабильный алгоритм упорядочения данных. Применяется для упорядочения элементов, являющихся цепочками над любым конечным алфавитом (например. Строки или целые числа). В качестве вспомогательного использует любой другой стабильный алгоритм сортировки.

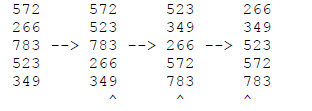
Алгоритм применялся для упорядочения перфокарт.

**идея алгоритма**

Идея заключается в том, чтобы сначала упорядочить все элементы за младшим разрядом, потом стабильно упорядочить по второму разряду, потом с третьим и так далее вплоть до старшего. Поскольку предполагается, что каждый разряд принимает значения из небольшого диапазона, то каждый цикл упорядочения можно выполнять быстро и с малыми затратами памяти.

**пример работы**

В примере показано, как упорядочить таким алгоритмом массив трехзначных чисел:



**анализ**

Время работы каждого цикла сортировки зависит от того алгоритма, который используется в качестве вспомогательного. Чаще всего используют сортировка подсчетом, что работает за время (где N - количество элементов в массиве; K - количество символов в алфавите, если упорядочиваются десятичные числа, то K = 10) и использует дополнительно памяти. Всего осуществляется столько циклов благоустройства, сколько разрядов максимальной Элементы.

Общая сложность работы алгоритма с использованием сортировки подсчетом есть (D - количество разрядов). Если упорядочить этим алгоритмом целые числа, то сложность будет, где M -наибольший элемент массива.