Лекция 11
Сортировки

Постановка задачи

Имеем массив объектов **[a1,a2,a3, …, aN]**, между которыми установлено **отношение порядка** (то есть про любые два элемента можно сказать: первый больше второго, второй больше первого или они равны).

Необходимо расположить эти элементы в порядке **неубывания**.

Расширенная задача: найти **перестановку σ {1,...,N} -> {1,...,N},** причем функция **σ** это **биекция** (то есть для каждого элемента найти его позицию в упорядоченном массиве)

Теорема

Любой алгоритм сортировки, основанный на сравнениях, требует **Ω(NlogN)** сравнений в худшем случае на массиве длины N.

Лемма. **log(n!) = Θ(nlogn)**

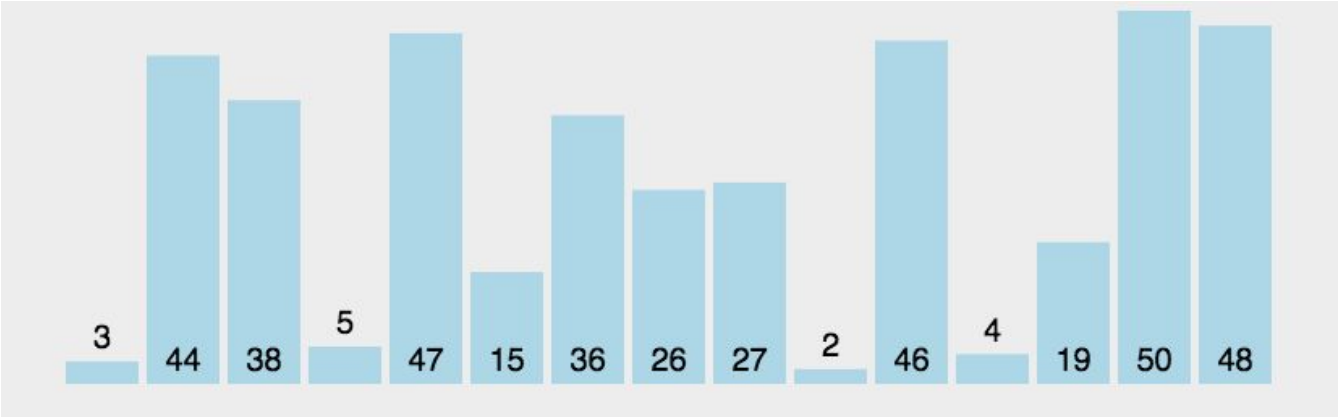
Д-во теоремы: предположим противное ∃ массив длины n, который сортируется быстрее nlogn. Строим решающее дерево алгоритма. Всего возможных перестановок элементов массива n!. Тогда в решающем дереве должно быть n! листьев. Тогда глубина дерева как минимум log(n!). Это значит, что мы совершаем как минимум log(n!) сравнений. А **log(n!) = Ω(nlogn)** (т.к. по лемме **log(n!) = Θ(nlogn)**)

Сортировка пузырьком (bubble sort)

Последовательно сравниваем значения соседних элементов и меняем числа местами, если предыдущее оказывается больше последующего. Таким образом элементы с большими значениями оказываются в конце списка, а с меньшими остаются в начале.

Сложность по времени

Худшее время: $O(n^2)$
Среднее время: $O(n^2)$
Лучшее время: $O(n)$



Сортировка вставками (insertion sort)

При сортировке вставками массив постепенно перебирается слева направо. При этом каждый последующий элемент размещается так, чтобы он оказался между ближайшими элементами с минимальным и максимальным значением.

6 5 3 1 8 7 2 4

Сложность по времени

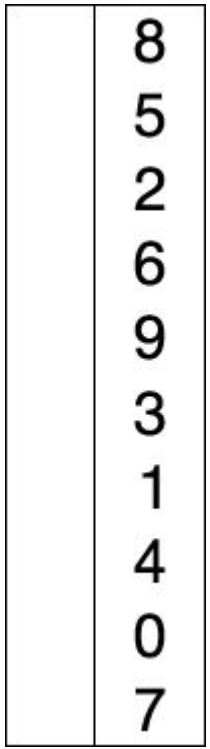
Худшее время: $O(n^2)$ для сравнений и перестановок
Среднее время: $O(n^2)$ для сравнений и перестановок
Лучшее время: $O(n)$ для сравнений, $O(1)$ для перестановок

Сортировка выбором (selection sort)

Сначала нужно рассмотреть подмножество массива и найти в нём максимум (или минимум). Затем выбранное значение меняют местами со значением первого неотсортированного элемента. Этот шаг нужно повторять до тех пор, пока в массиве не закончатся неотсортированные подмассивы.

Сложность по времени

Худшее время: $O(n^2)$
Среднее время: $O(n^2)$
Лучшее время: $O(n^2)$



Устойчивость сортировок

Представим, что мы сортируем не просто числа, а, например, пары {ключ : значение}. При этом на множестве ключей задано отношение порядка, так что сортировку будем производить по ключу. В общем случае, добавим каждому элементу массива какую-то дополнительную характеристику, которая отличает его от других элементов, равных ему согласно отношению порядка.

Устойчивой сортировкой называется сортировка, не меняющая порядка объектов с одинаковыми ключами. Пример, нам надо отсортировать массив

[2:’ca’, 1:’b’, 2:’ab’, 3:’abc’, 1:’a’]

Устойчивой будет такая сортировка, которая расположит элементы так:

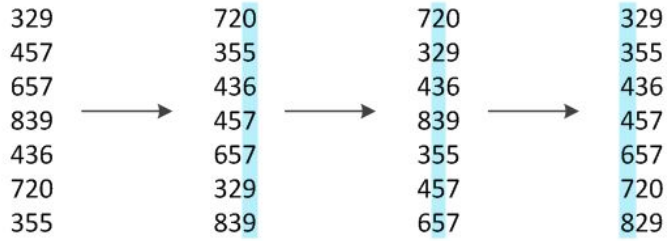
[1:’b’, 1:’a’, 2:’ca’, 2:’ab’, 3:’abc’]

Поразрядная сортировка (radix sort)

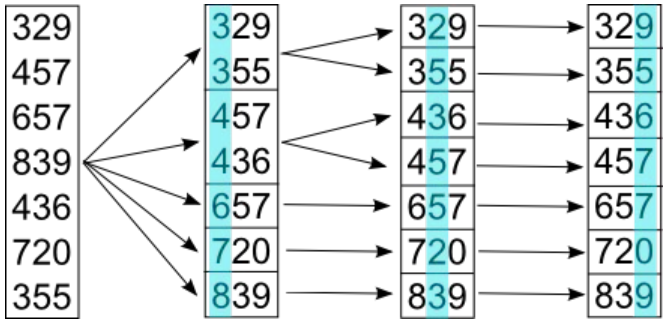
Еще иногда называется цифровой. Имеем множество последовательностей, состоящих из элементов (разрядов), на которых задано отношение порядка. Примерами объектов, которые удобно разбивать на разряды и сортировать по ним, являются числа и строки. Требуется отсортировать эти последовательности.

Сам алгоритм состоит в последовательной сортировке объектов какой-либо устойчивой сортировкой по каждому разряду, в порядке от младшего разряда к старшему (**LSD-сортировка** – Least Significant Digit), или от старшего к младшему **MSD-сортировка** (Most Significant Digit) после чего последовательности будут расположены в требуемом порядке.

Результаты MSD и LSD будут отличаться, если наши последовательности имеют разное количество разрядов.



LSD-сортировка



MSD-сортировка

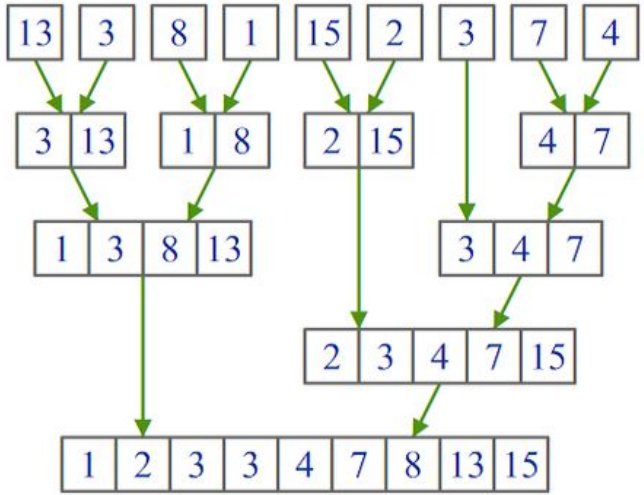
Сортировка подсчетом (counting sort)

Алгоритм сортировки целых чисел в диапазоне от 0 до некоторой константы k.
Создадим вспомогательный массив C[0..k], состоящий из нулей, затем последовательно прочитаем элементы входного массива A. Для каждого A[i] увеличим C[A[i]] на единицу. Теперь достаточно пройти по массиву C, и для каждого j записать число j C[j] раз.

Можно реализовать устойчивую версию используя, например, вместо массива словарь. Каждому ключу будет соответствовать ключ в словаре, по которому будет содержаться список элементов в порядке, в котором они встречались в изначальном массиве.

Сортировка слиянием (merge sort)

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



Может быть написана как рекурсивно, так и итеративно.
Рекурсивное соотношение времени работы:
$T(n)=2T(n/2)+O(n)$ ($O(n)$ – время на слияние)

Алгоритм реализации можно посмотреть здесь

Полезные ссылки

https://academy.yandex.ru/journal/osnovnye-vidy-sortirovok-i-primery-ikh-realizatsii

https://neerc.ifmo.ru/wiki/index.php?title=%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B8

Быстрая сортировка (quicksort)