Основные алгоритмы 8

Ковалев Алексей

1. Слова состоят из k маленьких латинскиз букв, значит можно считать, что мы работаем с числами в 26-ричной системе счисления длина k. Значит сортировать слова в лексикографическом порядке можно с помощью поразрядной сортировки за $O(nk)$.

2. Будем использовать тернарный поиск для поиска максимального элемента за $O(\log n)$. Пусть на вход подается массив длины n. Рассмотрим элементы x и y с индексами $n/3$ и $2n/3$ соответственно. Если $x > y$ – рекурсивно запустимся от $[0;\ 2n/3]$, если $x < y$ – рекурсивно запустимся от $[n/3;\ n]$, если $x = y$ – рекурсивно запускаемся от $[n/3;\ 2n/3]$. Тогда $T(n) \leqslant T(2n/3) + O(1)$. Пользуясь мастер-теоремой получаем требуемую ассимптотику $T(n) = O(\log n)$.

3. Будем взвешивать монеты следующим способом: разделим их на три равные группы и положим на весы две из них. Если какая-то из чаш весов перевесила, то фальшиваю монета лежит на более легкой чаше; если чаши уравновесились, то фальшивая монета находится в третьей группе, которая не взвешивалась в данный момент. После определения, в какой из групп находится фальшивая монета, повторяем те же дейстия с этой группой. Таким образом, на каждой итерации количество монет, среди которых точно есть фальшивая, будет становиться в 3 раза меньше, а значит хватит $\log_3 n + c$ взвешиваний.

4. Пусть мы можем найти фальшивую монету за k взвешиваний. Каждое из этих взвешиваний допускает не более трех различных исходов. То есть всевозможные последоватльности действий можно зашифровать троичными числами длины k, которых существует 3^k. По k взвешиваниям можно определить, какая из монет фальшивая, значит $3^k \geqslant n$. Отсюда $k \geqslant \log_3 n \iff k = \log_3 n + c$. □

5. Будем действовать следующи способо: найдем медиану m_1 и m_2 в каждом из массивов.

- $m_1 > m_2$: будем таким же алгоритмом искать медиану в массивах, состоящих из первой половины первого исходного массива и второй половины второго исодного массива.
- $m_1 = m_2$: $m_1 = m_2$ – медиана слияния исходных массивов.
- $m_1 < m_2$: будем таким же алгоритмом искать медиану в массивах, состоящих из второй половины первого исходного массива и первой половины второго исодного массива.

Корректность алгоритма: докажем корректность второго и третьего пунктов (корректность первого аналогично третьему). Если $m_1 < m_2$, то первая половина первого массива меньше, чем второая половина второго массива, значит слияние будет выглядеть так:

$$a_1, a_2, \ldots, a_{n/2}, \ldots, m_1, \ldots, m_2, \ldots b_{n/2}, \ldots, b_{n-1}, b_n$$

где a_i – элементы первого массива, b_i – элементы второго массива. Так как количество элементов в левой половине первого массива равно количеству элементов в правой половине второго массива, эти элементы можно исключить из массива, не изменив медиану. Если $m_1 = m_2$, то количество элементов, больших $m_1 = m_2$, такое же, как и количество элементов, меньших $m_1 = m_2$, то есть $m_1 = m_2$ – медиана.
Сложность: на каждой итерации размер рассматриваемых массивов сокращается хотя бы вдвое, значит сложность приведенного влгоритма – $O(\log n)$.

6. Из формальных определений каждого алгоритма видно следующее

сортировка	устойчивая	in-place
`QuickSort`	нет	да
`MergeSort`	да	нет
`InsertionSort`	да	да
`HeapSort`	нет	да

7. Пусть требуется отсортировать массив `a`. Построим массив `b`, элементами которого будут являться пары из элемента массива `a` и его индекса, то есть `b[i] = {a[i], i}`. Отсортируем массив `b` некоторой неустойчивой сортировкой, которую мы хотим сделать устойчивой. Затем разделим отсортированный массив `b` на подмассивы, состоящие из равных элементов, и отсортируем каждый из них какой-либо быстрой (работающей за $O(n \log n)$) сортировкой, например `MergeSort`. Далее обратно объединим все отсортированные подмассивы в исходном порядке. Отсортированный массивом `a` будет массив, состоящий из первых элементов пар в полученном массиве.
Алгоритм корректен, так как сортировка подмассивов из равных элементов восстанавливает относительный порядок элементов в этих подмассивах, что и является критерием устойчивости сортировки.
Асимптотически алгоритм работает за такое же время, что и сортировка, которой изначально сортировался массив `b`, так как не существует сортировки сравнениями более быстрой, чем $O(n \log n)$, и

$$\sum_{i=1}^{k} n_i \log n_i \leqslant n \log n$$

где $\sum_{i=1}^{k} n_i = n$. Формально, если сортировка, которой изначально сортировался массив `b`, работает за $O(T(n))$, то полученная из нее устойчивая сортировка работает за

$$O(T(n)) + \sum_{i=1}^{k} O(n_i \log n_i) = O(T(n)) + O(n \log n) = O(T(n))$$

□

8. Многочлен $P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$, где все a_i натуральные, строго возрастает при $x \geqslant 0$, так как $P'(x) = n a_n x^{n-1} + (n-1) a_{n-1} x^{n-2} + \ldots + 2a_2 x + a_1 \geqslant 0$ при $x \geqslant 0$. Уравнение $P(x) = y$ равносильно уравнению $Q(x) = 0$, где $Q(x) = P(x) - y$, то есть $Q(x)$ строго возрастает при $x \geqslant 0$ и является "сдвинутым вниз" $P(x)$. Тогда можно вычислить значение $P(0)$ и, если оно положительно, число y не является значением многочлена. Если же $P(0) \leqslant 0$, то можно запустить экспоненциальный (галопирующий) поиск (exponential or galloping search), пока не найдем точку n, такую что $P(n) > 0$, а затем запустить его же или обычный бинарный поиск на отрезке $[0; n]$. Это либо найдет натуральную точку, где значение многочлена равно y, либо покажет, что такой точки нет.
Сложность вычисления значения многочлена в точке равна $O(n \log n)$, количество итераций экспоненциального (и возможно бинарного) поиска некоторым образом зависит от входных данных. Данный алгоритм гарантированно даст правильный ответ, но непонятно, за какое время.

9. Если мы знаем, что первый шарик не разбился на этаже k, но разбился на этаже m, то нам потребуется проверить все этажи между ними, начиная с $(k+1)$-го линейным поиском, затратив на это $m-k-1$ бросков. Пусть мы бросаем первый шарик с этажей $k_1, k_2, \ldots, k_t$. Тогда при броске с этажа k_i было совершено уже $i-1$ бросков, и при i-том броске шарик может как разбиться, так и не разбиться. Значит сумма бросков первого шарика и второго шарика должна быть максимально близка к некоторому значению (в лучшем случае – равна ему) при любом исходе. Отсюда делаем вывод, что первый шарик надо бросать с этажей $n, 2n-1, 3n-2, \ldots$, так как в таком случае при любом исходе потребуется не более n бросков, чтобы узнать прочность шарика. Сумма $n + (n-1) + (n-2) + \ldots + 2 + 1$ должна быть больше или равна 100 и максимально близка к 100, значит $n = 14$ и всегда потребуется не более 14 бросков шариков.
Ответ: 14.