«Разделяй и властвуй»: быстрая сортировка

Александр Куликов

Онлайн-курс «Алгоритмы: теория и практика. Методы»
http://stepic.org/217

Функция QuickSort(A, ℓ, r)

если $\ell \geq r$:

  вернуть

$m \leftarrow$ Partition(A, ℓ, r)
QuickSort$(A, \ell, m-1)$
QuickSort$(A, m+1, r)$

Функция QuickSort(A, ℓ, r)

если $\ell \geq r$:

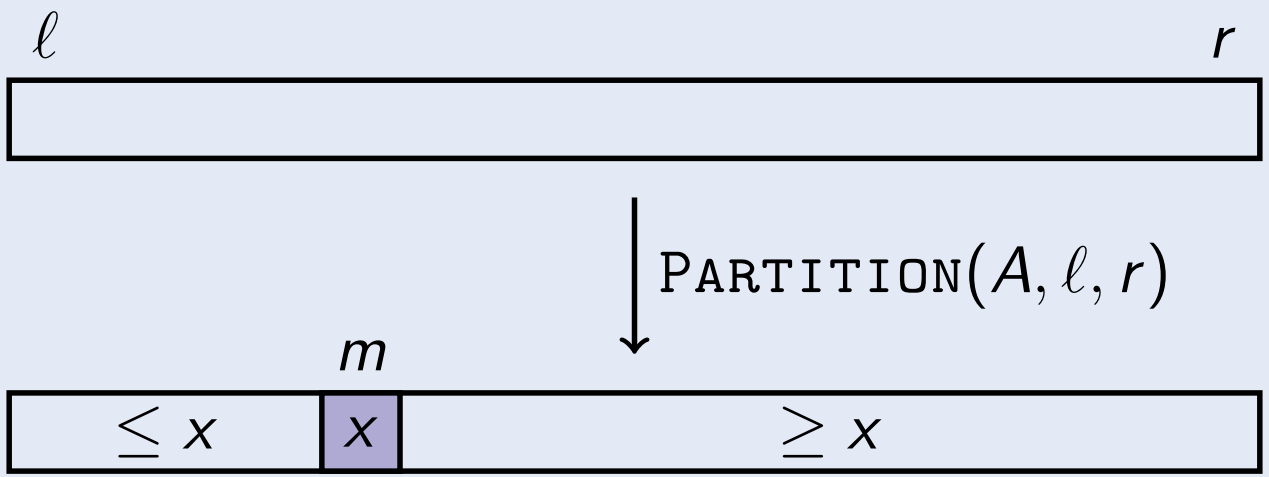
  вернуть

$m \leftarrow$ Partition(A, ℓ, r)

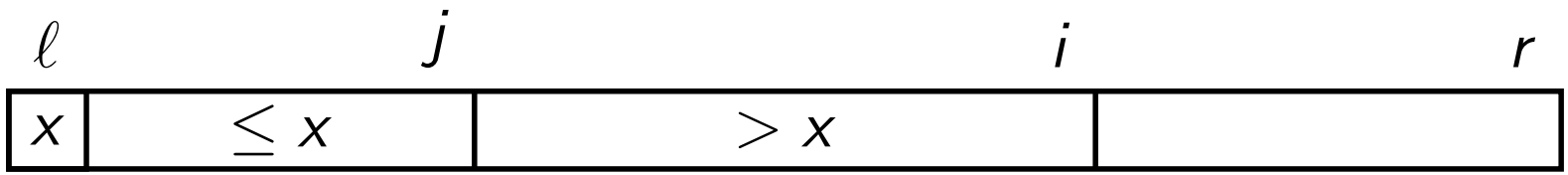
QuickSort$(A, \ell, m-1)$

QuickSort$(A, m+1, r)$

Partition визуально



Разбиение: основные идеи



- $x = A[\ell]$ — опорный элемент
- двигаем i от $\ell + 1$ до r, поддерживая следующий инвариант:
  - $A[k] \leq x$ для всех $\ell + 1 \leq k \leq j$
  - $A[k] > x$ для всех $j + 1 \leq k \leq i$
- пусть i только что увеличился; если $A[i] > x$, не делаем ничего; в противном случае меняем $A[i]$ с $A[j + 1]$ и увеличиваем j

Функция PARTITION(A, ℓ, r)

$x \leftarrow A[\ell]$

$j \leftarrow \ell$
для i от $\ell + 1$ до r:
  если $A[i] \leq x$:
    $j \leftarrow j + 1$
    обменять $A[j]$ и $A[i]$

обменять $A[\ell]$ и $A[j]$
вернуть j

Функция PARTITION(A, ℓ, r)

$x \leftarrow A[\ell]$

$j \leftarrow \ell$
для i от $\ell + 1$ до r:
  если $A[i] \leq x$:
    $j \leftarrow j + 1$
    обменять $A[j]$ и $A[i]$

обменять $A[\ell]$ и $A[j]$
вернуть j

Время работы: $O(n)$.

Плохие и хорошие разделители

- $T(n) = T(n-1) + n$:

$$T(n) = n + (n-1) + (n-2) + \ldots = \Theta(n^2)$$

Плохие и хорошие разделители

- $T(n) = T(n-1) + n$:

$$T(n) = n + (n-1) + (n-2) + \ldots = \Theta(n^2)$$

- $T(n) = T(n-5) + T(4) + n$:

$$T(n) \geq n + (n-5) + (n-10) + \ldots = \Theta(n^2)$$

Плохие и хорошие разделители

- $T(n) = T(n-1) + n$:

$$T(n) = n + (n-1) + (n-2) + \ldots = \Theta(n^2)$$

- $T(n) = T(n-5) + T(4) + n$:

$$T(n) \geq n + (n-5) + (n-10) + \ldots = \Theta(n^2)$$

- $T(n) = 2T(n/2) + n$: $T(n) = O(n \log n)$

Плохие и хорошие разделители

- $T(n) = T(n-1) + n$:

$$T(n) = n + (n-1) + (n-2) + \ldots = \Theta(n^2)$$

- $T(n) = T(n-5) + T(4) + n$:

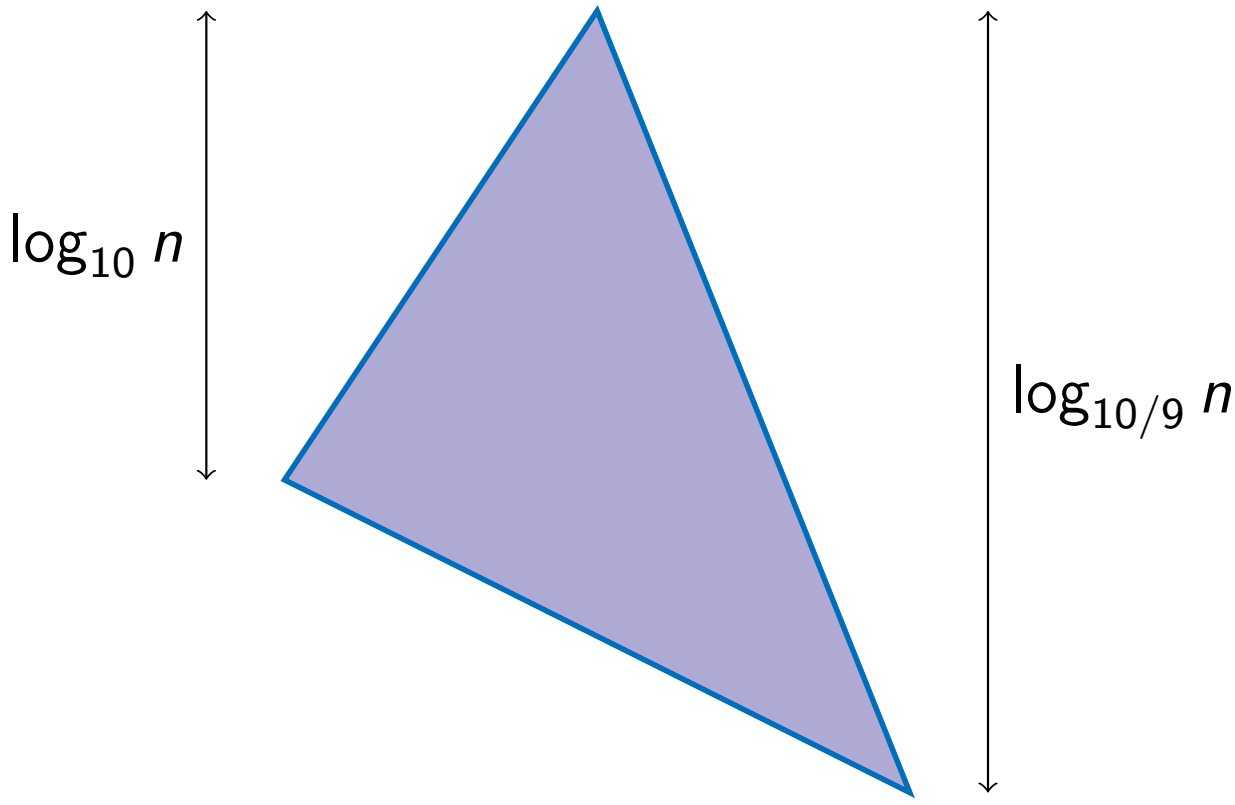
$$T(n) \geq n + (n-5) + (n-10) + \ldots = \Theta(n^2)$$

- $T(n) = 2T(n/2) + n$: $T(n) = O(n \log n)$
- $T(n) = T(n/10) + T(9n/10) + n$:

$$T(n) = O(n \log n)$$

Сбалансированные разбиения

$$T(n) = T(n/10) + T(9n/10) + O(n)$$



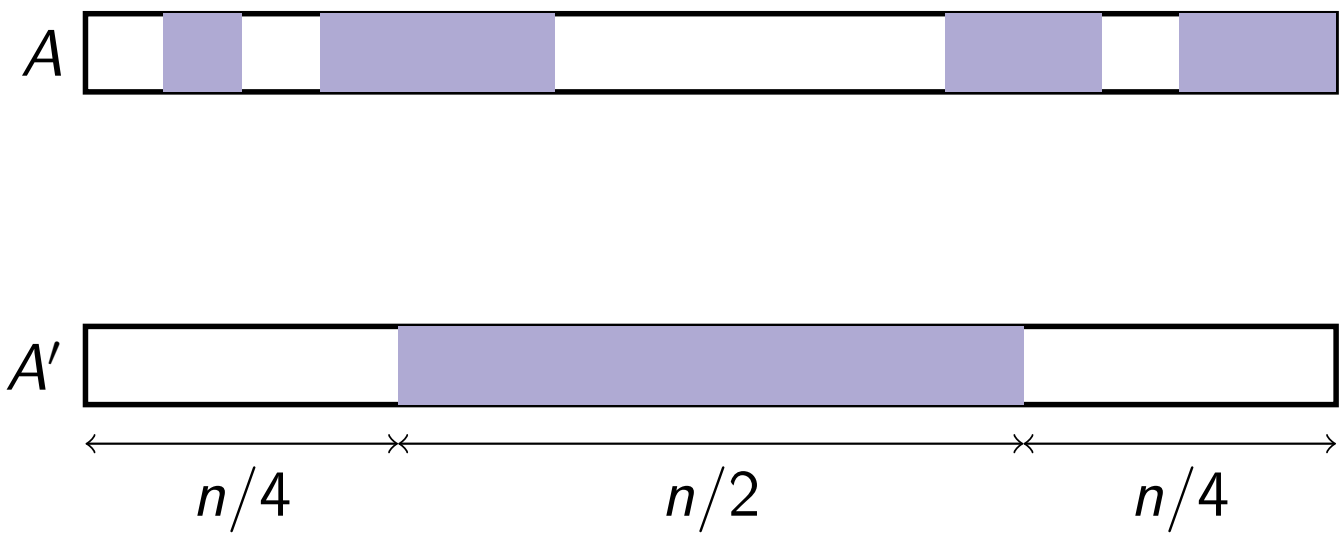
$$T(n) = O(n \log n)$$

Случайный разделитель

- чтобы разбить A относительно случайного разделителя, обменяем $A[\ell]$ со случайным элементом и вызовем $\textsc{Partition}(A, \ell, r)$

Случайный разделитель

- чтобы разбить A относительно случайного разделителя, обменяем $A[\ell]$ со случайным элементом и вызовем $\textsc{Partition}(A, \ell, r)$
- важное наблюдение: половина элементов A дают сбалансированное разбиение:



Время работы

Теорема

Допустим, что все элементы массива $A[1 \ldots n]$ различны и что разделитель всегда выбирается равномерно случайным образом. Тогда среднее время работы алгоритма `QUICKSORT`(A) есть $O(n \log n)$, в то время как время работы в худшем случае есть $O(n^2)$.

Время работы

Теорема

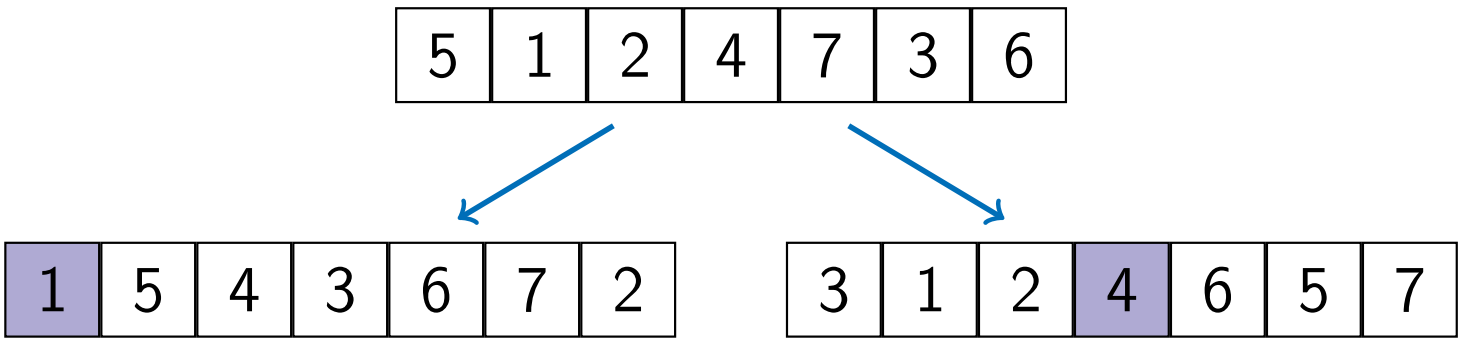
Допустим, что все элементы массива $A[1 \ldots n]$ различны и что разделитель всегда выбирается равномерно случайным образом. Тогда среднее время работы алгоритма `QUICKSORT`(A) есть $O(n \log n)$, в то время как время работы в худшем случае есть $O(n^2)$.

Замечание

Усреднение берётся по случайным числам алгоритма, а не по входам.

Идея доказательства: сравнения

- время работы пропорционально количеству сравнений
- сбалансированные разбиения лучше, потому что они лучше уменьшают количество необходимых сравнений:



узнали только, что 1 — минимум

3,1,2 уже никогда не сравнятся с 6,5,7

Идеи доказательства: вероятность

A	5	1	8	9	2	4	7	3	6
A'	1	2	3	4	5	6	7	8	9

$$\text{Prob}\,(1 \text{ и } 9 \text{ сравнятся}) = \frac{2}{9}$$

$$\text{Prob}\,(3 \text{ и } 4 \text{ сравнятся}) = 1$$

Доказательство

- для $i < j$ положим

$$\chi_{ij} = \begin{cases} 1 & A'[i] \text{ и } A'[j] \text{ сравнились} \\ 0 & \text{в противном случае} \end{cases}$$

- для всех $i < j$ элементы $A'[i]$ и $A'[j]$ либо сравниваются ровно один раз, либо не сравниваются вообще (мы всегда сравниваем с разделителем)
- это, в частности, означает, что время работы в худшем случае не больше $O(n^2)$

Доказательство

- важное замечание: $\chi_{ij} = 1$, если и только если первый разделитель, выбранный из $A'[i \ldots j]$, — это $A'[i]$ или $A'[j]$
- тогда $\mathrm{Prob}(\chi_{ij}) = \frac{2}{j-i+1}$ и $\mathrm{E}(\chi_{ij}) = \frac{2}{j-i+1}$
- тогда среднее время работы есть

$$\begin{aligned}
\mathrm{E}\sum_{i=1}^{n}\sum_{j=i+1}^{n}\chi_{ij} &= \sum_{i=1}^{n}\sum_{j=i+1}^{n}\mathrm{E}(\chi_{ij}) \\
&= \sum_{i<j}\frac{2}{j-i+1} \\
&\leq 2n\cdot\left(\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n}\right) \\
&= \Theta(n\log n) \quad \square
\end{aligned}$$

Равные элементы

- визуализация:
  www.sorting-algorithms.com/quick-sort

Равные элементы

- визуализация:
  www.sorting-algorithms.com/quick-sort
- если все элементы массива равны между собой, то рассмотренная реализация алгоритма быстрой сортировки будет работать квадратичное время

Равные элементы

- визуализация: www.sorting-algorithms.com/quick-sort
- если все элементы массива равны между собой, то рассмотренная реализация алгоритма быстрой сортировки будет работать квадратичное время
- чтобы обойти это препятствие, массив можно разбивать на три части вместо двух: $< x$, $= x$ и $> x$ (3-разбиение)

Элиминация хвостовой рекурсии

Процедура QuickSort(A, ℓ, r)

пока $\ell < r$:
  $m \leftarrow$ Partition(A, ℓ, r)
  QuickSort$(A, \ell, m-1)$
  $\ell \leftarrow m+1$

Элиминируя рекурсивный вызов для более длинного массива, мы гарантируем, что глубина рекурсии (а значит, и дополнительная память) будет в худшем случае не более $O(\log n)$.

Интроспективная сортировка: $O(n \log n)$ в худшем случае

- запускает быструю сортировку с простой эвристикой выбора разделителя (например, медиана из первого, среднего и последнего элементов)
- если глубина рекурсии превышает порог $c \log n$, быстрая сортировка прерывается и запускается алгоритм с гарантированным временем $O(n \log n)$ в худшем случае (например, сортировка кучей)

Заключение

- Быстрая сортировка работает за время $O(n \log n)$ в среднем случае и за $O(n^2)$ в худшем случае.
- Усреднение берётся по случайным числам алгоритма, но не по входам.
- Очень эффективен на практике.
- Если в массиве могут быть равные числа, стоит использовать 3-разбиение вместо 2-разбиения.
- Элиминация хвостовой рекурсии позволяет сделать так, чтобы алгоритм быстрой сортировки использовал не более $O(\log n)$ дополнительной памяти.