Глава 7. Быстрая сортировка

Алгоритм быстрой сортировки имеет в наихудшем случае для входного массива из n элементов время работы, равное $\Theta(n^2)$. Несмотря на такую медленную работу в наихудшем случае, этот алгоритм на практике зачастую оказывается оптимальным благодаря тому, что в среднем время его работы намного лучше: $\Theta(n \lg n)$. Кроме того, постоянный множитель, скрытый в выражении $\Theta(n \lg n)$, достаточно мал по величине. Алгоритм обладает также тем преимуществом, что сортировка в нем выполняется на месте, без использования дополнительной памяти (см. с. 39), поэтому он хорошо работает даже в средах с виртуальной памятью.

В разделе 7.1 описаны сам алгоритм и важная подпрограмма, использующаяся в нем для разбиения массива. Поскольку поведение алгоритма быстрой сортировки достаточно сложное, мы начнем с нестрогого, интуитивного обсуждения производительности этого алгоритма в разделе 7.2, а строгий анализ отложим до конца данной главы. В разделе 7.3 представлена версия быстрой сортировки, в которой используется случайная выборка. У этого алгоритма хорошее ожидаемое время работы, при этом никакие конкретные входные данные не могут ухудшить его производительность до уровня наихудшего случая. Этот рандомизированный алгоритм анализируется в разделе 7.4, где показано, что время его работы в наихудшем случае равно $\Theta(n^2)$, а среднее время работы в предположении, что все элементы различны, составляет $O(n \lg n)$.

7.1. Описание быстрой сортировки

Быстрая сортировка, подобно сортировке слиянием, применяет парадигму "разделяй и властвуй", представленную в разделе 2.3.1. Ниже описан процесс сортировки подмассива $A[p\,..\,r]$, состоящий, как и все алгоритмы с использованием декомпозиции, из трех этапов.

Разделение. Массив $A[p\,..\,r]$ разбивается на два (возможно, пустых) подмассива $A[p\,..\,q-1]$ и $A[q+1\,..\,r]$, таких, что каждый элемент $A[p\ ..\ q-1]$ меньше или равен $A[q]$, который, в свою очередь, не превышает любой элемент подмассива $A[q+1\,..\,r]$. Индекс q вычисляется в ходе процедуры разбиения.

Властвование. Подмассивы $A[p\,..\,q-1]$ и $A[q+1\,..\,r]$ сортируются с помощью рекурсивного вызова процедуры быстрой сортировки.

Комбинирование. Поскольку подмассивы сортируются на месте, для их объединения не требуются никакие действия: весь массив $A[p\,..\,r]$ оказывается отсортированным.

Быстрая сортировка реализуется следующей процедурой.

```
QUICKSORT(A, p, r)
1  if p < r
2      q = PARTITION(A, p, r)
3      QUICKSORT(A, p, q − 1)
4      QUICKSORT(A, q + 1, r)
```

Для сортировки всего массива A начальный вызов процедуры должен иметь вид QUICKSORT$(A, 1, A.\mathit{length})$.

Разбиение массива

Ключевой частью рассматриваемого алгоритма сортировки является процедура PARTITION, изменяющая порядок элементов подмассива $A[p\,..\,r]$ без привлечения дополнительной памяти.

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          Обменять A[i] и A[j]
7  Обменять A[i + 1] и A[r]
8  return i + 1
```

На рис. 7.1 показано, как процедура PARTITION работает с 8-элементным массивом. Эта процедура всегда выбирает элемент $x = A[r]$ в качестве ***опорного*** (pivot). Разбиение подмассива $A[p\,..\,r]$ будет выполняться относительно этого элемента. В начале выполнения процедуры массив разделяется на четыре области (они могут быть пустыми). В начале каждой итерации цикла **for** в строках 3–6 каждая область удовлетворяет определенным свойствам, показанным на рис. 7.2. Эти свойства можно сформулировать в виде инварианта цикла.

В начале каждой итерации цикла в строках 3–6 для любого индекса k массива справедливо следующее:

1. если $p \le k \le i$, то $A[k] \le x$;
2. если $i+1 \le k \le j-1$, то $A[k] > x$;
3. если $k = r$, то $A[k] = x$.

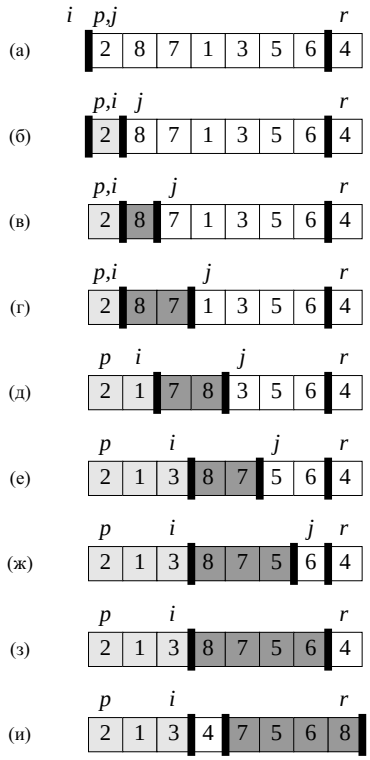

Рис. 7.1. Пример действия процедуры PARTITION на массив. Элемент массива $A[r]$ становится опорным элементом x. Светло-серым цветом обозначены элементы массива, которые попали в первую часть разбиения; их значения не превышают x. Элементы темно-серого цвета образуют вторую часть массива; их величина больше x. Незакрашенные элементы — это элементы, которые пока что не попали ни в одну из первых двух частей; последний незакрашенный элемент является опорным элементом x. **(а)** Начальное состояние массива и значения переменных. Ни один элемент не помещен ни в одну из первых двух частей. **(б)** Элемент со значением 2 "переставлен сам с собой" и помещен в часть с меньшими значениями. **(в)** и **(г)** Элементы со значениями 8 и 7 добавлены в часть с бóльшими значениями. **(д)** Элементы 1 и 8 поменялись местами, в результате чего количество элементов в первой части возросло. **(е)** Обмен местами элементов 3 и 7, в результате чего количество элементов в первой части возрастает. **(ж)** и **(з)** Вторая часть увеличивается за счет включения в нее элементов 5 и 6, после чего цикл завершается. **(и)** В строках 7 и 8 опорный элемент меняется местами с тем, который находится между двумя областями.

Индексы между j и $r-1$ не подпадают ни под один из трех перечисленных случаев, и значения соответствующих им элементов не имеют определенной связи с опорным элементом x.

Нам необходимо показать, что сформулированный выше инвариант цикла справедлив перед первой итерацией, что каждая итерация цикла сохраняет этот инвариант и что он позволяет продемонстрировать корректность алгоритма по завершении цикла.

Инициализация. Перед первой итерацией цикла $i = p-1$ и $j = p$. Между элементами с индексами p и i нет никаких элементов, как нет их и между

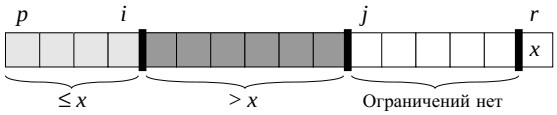

Рис. 7.2. Четыре области, поддерживаемые процедурой PARTITION в подмассиве $A[p..r]$. Все элементы подмассива $A[p\,..\,i]$ меньше либо равны x, все элементы подмассива $A[i+1\,..\,j-1]$ больше x, а $A[r] = x$. Подмассив $A[j\,..\,r-1]$ может иметь любые значения.

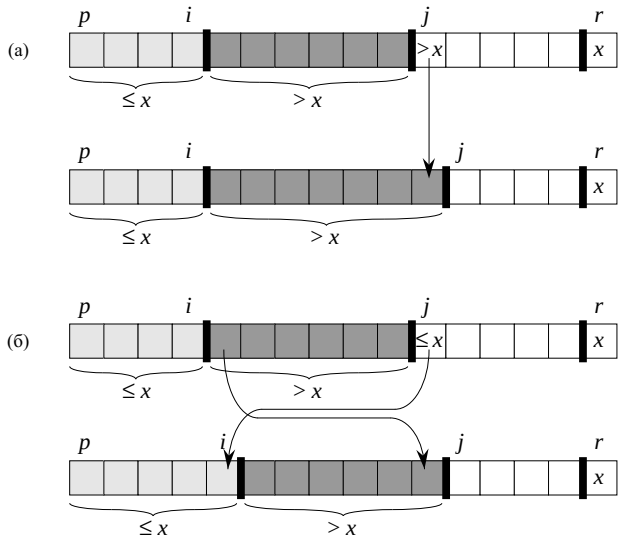

Рис. 7.3. Два варианта итерации в процедуре PARTITION. **(а)** Если $A[j] > x$, единственным действием является увеличение j, что сохраняет инвариант цикла. **(б)** Если $A[j] \le x$, индекс i увеличивается, $A[i]$ и $A[j]$ меняются местами, после чего увеличивается j. Инвариант цикла сохраняется и в этом случае.

элементами с индексами $i + 1$ и $j - 1$, поэтому первые два условия инварианта цикла тривиально выполняются. Присваивание в строке 1 удовлетворяет третьему условию.

Сохранение. Как видно из рис. 7.3, нужно рассмотреть два случая, выбор одного из которых определяется проверкой в строке 4. На рис. 7.3, (а) показано, что происходит, если $A[j] > x$; единственное действие, которое выполняется в этом случае в цикле, — это увеличение на единицу значения j. При увеличении значения j условие 2 выполняется для элемента $A[j-1]$, а все остальные элементы остаются неизменными. На рис. 7.3, (б) показано, что происходит, если $A[j] \le x$; в этом случае увеличивается значение i, элементы $A[i]$ и $A[j]$ меняются местами, после чего на единицу увеличивается значение j. В результате перестановки получаем $A[i] \le x$, и условие 1 выполняется. Аналогично получаем $A[j-1] > x$, поскольку элемент, который был переставлен в позицию элемента $A[j-1]$, согласно инварианту цикла больше x.

Завершение. По завершении работы алгоритма $j = r$. Поэтому каждый элемент массива является членом одного из трех множеств, описанных в инварианте цикла. Таким образом, все элементы массива разбиты на три множества: величина которых не превышает x, превышающие x и одноэлементное множество, состоящее из элемента x.

В последних двух строках процедуры PARTITION опорный элемент перемещается на свое место в средину массива с помощью его перестановки с крайним левым элементом, превышающим величину x. После этого процедура возвращает новый индекс опорного элемента. Выход процедуры PARTITION удовлетворяет требованиям, наложенным шагом разделения. Фактически он удовлетворяет немного более строгому условию: после строки 2 процедуры QUICKSORT элемент $A[q]$ строго меньше любого элемента $A[q+1\,..\,r]$.

Время работы процедуры PARTITION над подмассивом $A[p\,..\,r]$ равно $\Theta(n)$, где $n = r - p + 1$ (см. упр. 7.1.3).

Упражнения

7.1.1

Воспользовавшись рис. 7.1 в качестве образца, проиллюстрируйте работу процедуры PARTITION с массивом $A = \langle 13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11\rangle$.

7.1.2

Какое значение q возвращает процедура PARTITION, если все элементы массива $A[p\,..\,r]$ одинаковы? Модифицируйте эту процедуру так, чтобы в случае, когда все элементы массива $A[p\,..\,r]$ имеют одно и то же значение, q определялось следующим образом: $q = \lfloor (p + r)/2 \rfloor$.

7.1.3

Приведите краткое обоснование утверждения, что время обработки процедурой PARTITION подмассива, состоящего из n элементов, равно $\Theta(n)$.

7.1.4

Как бы вы изменили процедуру QUICKSORT для сортировки в невозрастающем порядке?

7.2. Производительность быстрой сортировки

Время работы алгоритма быстрой сортировки зависит от степени сбалансированности, которой характеризуется разбиение. Сбалансированность, в свою очередь, зависит от того, какой элемент выбран в качестве опорного. Если разбиение сбалансированное, асимптотически алгоритм работает так же быстро, как и сортировка слиянием. В противном случае асимптотическое поведение этого алгоритма столь же медленное, как и у сортировки вставкой. В данном разделе

будет проведено неформальное исследование поведения быстрой сортировки при условии сбалансированного и несбалансированного разбиений.

Разбиение в наихудшем случае

Наихудшее поведение алгоритма быстрой сортировки имеет место в случае, когда подпрограмма, выполняющая разбиение, порождает одну подзадачу с $n-1$ элементами, а вторую — пустую. (Это будет доказано в разделе 7.4.1.) Предположим, что такое несбалансированное разбиение возникает при каждом рекурсивном вызове. Для выполнения разбиения требуется время $\Theta(n)$. Поскольку рекурсивный вызов процедуры разбиения, на вход которой подается массив размером 0, приводит к немедленному возврату из этой процедуры без выполнения каких-либо операций, $T(0) = \Theta(1)$. Таким образом, рекуррентное соотношение, описывающее время работы процедуры в указанном случае, записывается следующим образом:

$$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(n) \\ &= T(n-1) + \Theta(n) \ . \end{aligned}$$

Интуитивно понятно, что при суммировании промежутков времени, затрачиваемых на каждый уровень рекурсии, получается арифметическая прогрессия (уравнение (A.2)), что приводит к результату $\Theta(n^2)$. В самом деле, с помощью метода подстановок легко доказать, что решением рекуррентного соотношения $T(n) = T(n-1) + \Theta(n)$ является $T(n) = \Theta(n^2)$ (см. упр. 7.2.1).

Таким образом, если на каждом уровне рекурсии алгоритма разбиение максимально несбалансированное, то время работы алгоритма равно $\Theta(n^2)$. Следовательно, производительность быстрой сортировки в наихудшем случае не превышает производительности сортировки вставкой. Более того, это же время требуется алгоритму быстрой сортировки для обработки уже полностью отсортированного массива (распространенная ситуация, в которой время работы алгоритма сортировки вставкой равно $O(n)$).

Разбиение в наилучшем случае

В наиболее благоприятном случае процедура PARTITION приводит к двум подзадачам, размер каждой из которых не превышает $n/2$, поскольку размер одной из них равен $\lfloor n/2 \rfloor$, а второй — $\lceil n/2 \rceil - 1$. В такой ситуации быстрая сортировка работает намного производительнее, и время ее работы описывается следующим рекуррентным соотношением:

$$T(n) = 2T(n/2) + \Theta(n) \ ,$$

где мы не обращаем внимания на неточность, связанную с игнорированием функций "пол" и "потолок", и вычитанием 1. В соответствии со случаем 2 основной теоремы (теорема 4.1) это рекуррентное соотношение имеет решение $T(n) = \Theta(n \lg n)$. При сбалансированности двух частей разбиения на каждом уровне рекурсии мы получаем асимптотически более быстрый алгоритм.

Сбалансированное разбиение

Как станет ясно из анализа, проведенного в разделе 7.4, в асимптотическом пределе поведение алгоритма быстрой сортировки в среднем случае намного ближе к его поведению в наилучшем случае, чем к поведению в наихудшем случае. Чтобы стало ясно, почему это так, нужно понять, как баланс разбиения отражается на рекуррентном соотношении, описывающем время работы алгоритма.

Предположим, например, что разбиение всегда происходит в соотношении один к девяти, что, на первый взгляд, весьма далеко от сбалансированности. В этом случае для времени работы алгоритма быстрой сортировки мы получим рекуррентное соотношение

$$T(n) = T(9n/10) + T(n/10) + cn \ ,$$

в которое явным образом входит константа c, скрытая в члене $\Theta(n)$. На рис. 7.4 показано дерево рекурсии, отвечающее этому рекуррентному соотношению. Обратите внимание, что каждый уровень этого дерева имеет стоимость cn, и так до тех пор, пока не будет достигнуто граничное условие на глубине $\log_{10} n = \Theta(\lg n)$; после этого стоимость более глубоких уровней не превышает величину cn. Рекурсия прекращается на глубине $\log_{10/9} n = \Theta(\lg n)$; таким образом, общая стоимость быстрой сортировки равна $O(n \lg n)$. Итак, если на каждом уровне рекурсии разбиение производится в соотношении один к девяти (что интуитивно воспринимается как сильный дисбаланс), время работы алгоритма быстрой сортировки равно $O(n \lg n)$ — асимптотически такое же, как и при сбалансированном разбиении. Фактически любое разбиение, характеризующееся конечной *константой* пропорциональности, приводит к образованию дерева рекурсии высотой $\Theta(\lg n)$ со стоимостью каждого уровня, равной $O(n)$. Следовательно, при любой постоянной пропорции разбиения полное время работы быстрой сортировки составляет $O(n \lg n)$.

Интуитивные рассуждения для среднего случая

Чтобы получить ясное представление о рандомизированном поведении алгоритма быстрой сортировки, необходимо сделать предположение о том, как часто ожидается появление тех или иных входных данных. Поведение быстрой сортировки зависит от относительного расположения значений во входном массиве, а не от их конкретных величин. Как и при вероятностном анализе задачи о найме в разделе 5.2, пока что предположим, что все перестановки входных чисел равновероятны.

Если алгоритм быстрой сортировки работает со случайным образом выбранным входным массивом, то маловероятно, чтобы разбиение на каждом уровне происходило в одном и том же соотношении, как это было в ходе проведенного выше неформального анализа. Ожидается, что одни разбиения будут хорошо сбалансированы, в то время как другие окажутся сбалансированными плохо. Например, в упр. 7.2.6 нужно показать, что соотношение размеров подзадач на выходе процедуры PARTITION примерно в 80% случаев сбалансировано лучше, чем девять к одному, и примерно в 20% случаев — хуже.

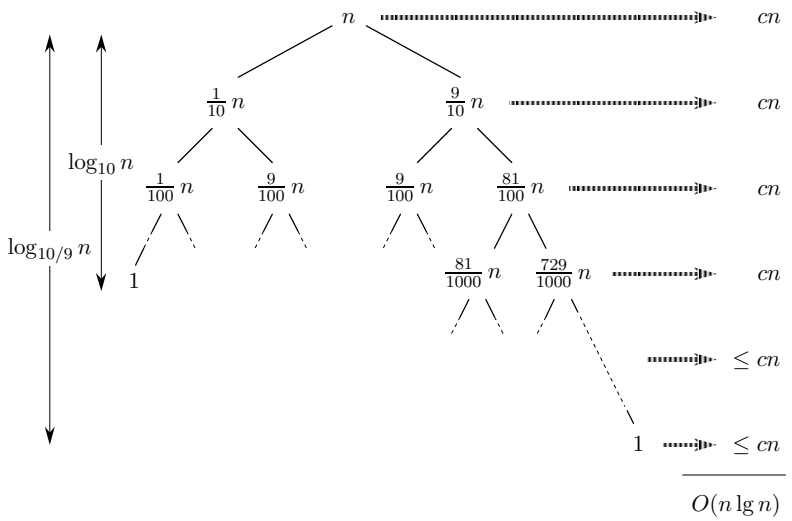

Рис. 7.4. Дерево рекурсии для алгоритма QUICKSORT, в котором процедура PARTITION всегда выполняет разбиение в соотношении девять к одному, дающее время работы $O(n \lg n)$. Узлы показывают размеры подзадач; стоимости уровней показаны справа. Стоимость уровня неявно включает константу c в члене $\Theta(n)$.

В среднем случае процедура PARTITION производит как "хорошие", так и "плохие" деления. В дереве рекурсии, соответствующем среднему случаю, хорошие и плохие разбиения равномерно распределены по всему дереву. Для упрощения интуитивных рассуждений предположим, что уровни дерева с плохими и хорошими разбиениями чередуются, и что хорошие разбиения получаются такими, как в наилучшем случае, а плохие — такими, как в наихудшем. На рис. 7.5, (а) показаны разбиения на двух соседних уровнях такого дерева рекурсии. В корне дерева стоимость разбиения равна n, а размеры полученных подмассивов равны $n-1$ и 0, что соответствует наихудшему случаю. На следующем уровне подмассив размером $n-1$ делится оптимальным образом на подмассивы размерами $(n-1)/2-1$ и $(n-1)/2$. Будем считать, что для подмассива размером 0 стоимость равна 1.

Комбинация плохого и хорошего разбиений приводит к образованию трех подмассивов с размерами 0, $(n-1)/2-1$ и $(n-1)/2$ и суммарной стоимостью $\Theta(n) + \Theta(n-1) = \Theta(n)$. Эта ситуация, определенно, не хуже показанной на рис. 7.5, (б), на котором представлен один уровень разбиения со стоимостью $\Theta(n)$, создающий два подмассива размерами $(n-1)/2$. Однако в этом случае разбиение получается сбалансированным! Интуитивно понятно, что время $\Theta(n-1)$, которое требуется для плохого разбиения, поглощается временем $\Theta(n)$, которое требуется для хорошего разбиения, а полученное в результате разбиение оказывается хорошим. Таким образом, время работы алгоритма быстрой сортировки, при которой на последовательных уровнях чередуются плохие и хорошие разбиения, ведет себя так же, как время работы быстрой сортировки для одних лишь хороших разбиений. Оно также равно $O(n \lg n)$, просто константа, скрытая в O-обозначении,

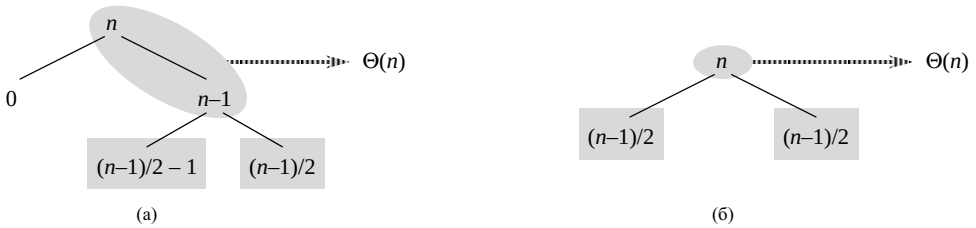

Рис. 7.5. **(а)** Два уровня дерева рекурсии для быстрой сортировки. Стоимость разбиения в корне равна n и генерирует плохое разбиение, т.е. подмассивы размерами 0 и $n-1$. Разбиение подмассива размером $n-1$ имеет стоимость $n-1$ и генерирует хорошее разбиение — подмассивы размерами $(n-1)/2-1$ и $(n-1)/2$. **(б)** Хорошо сбалансированный уровень дерева рекурсии. В обеих частях стоимость разбиения для задач в заштрихованных эллипсах составляет $\Theta(n)$. Задачи, которые остается решить в случае (а), и показанные в заштрихованных прямоугольниках, оказываются не больше подзадач, которые следует решить в случае (б).

в этом случае несколько больше. Строгий анализ ожидаемого времени работы рандомизированной версии быстрой сортировки содержится в разделе 7.4.2.

Упражнения

7.2.1

С помощью метода подстановки докажите, что решение рекуррентного соотношения $T(n) = T(n-1) + \Theta(n)$ имеет вид $T(n) = \Theta(n^2)$, как утверждалось в начале раздела 7.2.

7.2.2

Чему равно время работы процедуры QUICKSORT в случае, когда все элементы массива A одинаковы по величине?

7.2.3

Покажите, что если все элементы массива A различаются по величине и расположены в убывающем порядке, то время работы процедуры QUICKSORT равно $\Theta(n^2)$.

7.2.4

Сведения о банковских операциях часто записываются в хронологическом порядке, но многие предпочитают получать отчеты о состоянии своих банковских счетов в порядке нумерации чеков. Чеки чаще всего выписываются в порядке возрастания их номеров, а торговцы обычно обналичивают их с небольшой задержкой. Таким образом, задача преобразования упорядочения по времени операций в упорядочение по номерам чеков — это задача сортировки почти упорядоченных массивов. Обоснуйте утверждение, что при решении этой задачи процедура INSERTION-SORT обычно превосходит по производительности процедуру QUICKSORT.

7.2.5

Предположим, что в ходе быстрой сортировки на каждом уровне происходит разбиение в пропорции $1-\alpha$ к α, где $0 < \alpha \leq 1/2$ — константа. Покажите, что минимальная глубина, на которой расположен лист дерева рекурсии, приблизительно равна $-\lg n/\lg \alpha$, а максимальная глубина — приблизительно $-\lg n/\lg(1-\alpha)$. (Об округлении до целочисленных значений можно не беспокоиться.)

7.2.6 $\star$

Докажите, что для любой константы $0 < \alpha \leq 1/2$ вероятность того, что процедура PARTITION поделит случайно выбранный входной массив в более сбалансированной пропорции, чем $1-\alpha$ к α, приблизительно равна $1-2\alpha$.

7.3. Рандомизированная быстрая сортировка

Исследуя поведение алгоритма быстрой сортировки в среднем случае, мы делали предположение, что все перестановки входных чисел встречаются с равной вероятностью. Однако на практике это далеко не всегда так (см. упр. 7.2.4). Как мы знаем из раздела 5.3, можно добавить в алгоритм рандомизацию, чтобы получить хорошую ожидаемую производительность для всех входных данных. Многие считают такую рандомизированную версию алгоритма быстрой сортировки оптимальным выбором для обработки достаточно больших массивов.

В разделе 5.3 рандомизация алгоритма проводилась путем явной перестановки его входных элементов. В алгоритме быстрой сортировки можно было бы поступить точно так же, однако анализ упростится, если применить другой метод рандомизации, получивший название ***случайной выборки*** (random sampling). Вместо того чтобы в качестве опорного элемента всегда использовать $A[r]$, такой элемент будет выбираться в массиве $A[p\,..\,r]$ случайным образом. Подобная модификация, при которой опорный элемент выбирается случайным образом среди элементов с индексами от p до r, обеспечивает любому из $r-p+1$ элементов подмассива равную вероятность оказаться опорным. Благодаря случайному выбору опорного элемента можно ожидать, что разбиение входного массива в среднем окажется довольно хорошо сбалансированным.

Изменения, которые нужно внести в процедуры PARTITION и QUICKSORT, незначительны. В новой версии процедуры разбиения непосредственно перед разбиением достаточно реализовать обмен.

```
RANDOMIZED-PARTITION(A, p, r)
1  i = RANDOM(p, r)
2  Обменять A[r] и A[i]
3  return PARTITION(A, p, r)
```

В новой процедуре быстрой сортировки вместо процедуры PARTITION вызывается процедура RANDOMIZED-PARTITION.

RANDOMIZED-QUICKSORT(A, p, r)

```
1  if p < r
2      q = RANDOMIZED-PARTITION(A, p, r)
3      RANDOMIZED-QUICKSORT(A, p, q − 1)
4      RANDOMIZED-QUICKSORT(A, q + 1, r)
```

Мы проанализируем этот алгоритм в следующем разделе.

Упражнения

7.3.1

Почему мы анализируем ожидаемое время работы рандомизированного алгоритма, а не его время работы в наихудшем случае?

7.3.2

Сколько раз в ходе выполнения процедуры RANDOMIZED-QUICKSORT в наихудшем случае вызывается генератор случайных чисел RANDOM? В наилучшем случае? Выразите свой ответ через Θ-обозначения.

7.4. Анализ быстрой сортировки

В разделе 7.2 были приведены некоторые интуитивные рассуждения по поводу поведения алгоритма быстрой сортировки в наихудшем случае и обосновывалось, почему следует ожидать достаточно высокой производительности его работы. В данном разделе проведен более строгий анализ поведения этого алгоритма. Начнем этот анализ с наихудшего случая. Подобный анализ применим как к процедуре QUICKSORT, так и к процедуре RANDOMIZED-QUICKSORT. Завершается раздел анализом ожидаемого времени работы процедуры RANDOMIZED-QUICKSORT.

7.4.1. Анализ наихудшего случая

В разделе 7.2 было показано, что при самом неудачном разбиении на каждом уровне рекурсии время работы алгоритма быстрой сортировки равно $\Theta(n^2)$. Интуитивно понятно, что это наихудшее время работы рассматриваемого алгоритма. Докажем это утверждение.

С помощью метода подстановки (см. раздел 4.3) можно показать, что время работы алгоритма быстрой сортировки равно $O(n^2)$. Пусть $T(n)$ — наихудшее время обработки процедурой QUICKSORT входных данных размером n. Тогда мы получаем следующее рекуррентное соотношение:

$$T(n) = \max_{0 \le q \le n-1} (T(q) + T(n - q - 1)) + \Theta(n) , \tag{7.1}$$

где параметр q изменяется в интервале от 0 до $n-1$, поскольку на выходе процедуры PARTITION мы получаем две подзадачи, общий размер которых равен $n-1$. Мы предполагаем, что $T(n) \leq cn^2$ для некоторой константы c. Подставив это неравенство в рекуррентное соотношение (7.1), получим

$$\begin{aligned} T(n) &\leq \max_{0 \leq q \leq n-1} (cq^2 + c(n-q-1)^2) + \Theta(n) \\ &= c \cdot \max_{0 \leq q \leq n-1} (q^2 + (n-q-1)^2) + \Theta(n) \ . \end{aligned}$$

Выражение $q^2 + (n-q-1)^2$ достигает максимума на обоих концах интервала $0 \leq q \leq n-1$, что подтверждается тем, что вторая производная от него по q положительна (см. упр. 7.4.3). Это наблюдение позволяет нам сделать оценку $\max_{0 \leq q \leq n-1}(q^2 + (n-q-1)^2) \leq (n-1)^2 = n^2 - 2n + 1$. Продолжая работу с границей для $T(n)$, получаем

$$\begin{aligned} T(n) &\leq cn^2 - c(2n-1) + \Theta(n) \\ &\leq cn^2 \ , \end{aligned}$$

поскольку константу c можно выбрать настолько большой, чтобы слагаемое $c(2n-1)$ доминировало над слагаемым $\Theta(n)$. Таким образом, $T(n) = O(n^2)$. В разделе 7.2 мы встречались с частным случаем быстрой сортировки, при котором требовалось время $\Omega(n^2)$, — это случай несбалансированного разбиения. В упр. 7.4.1 нужно показать, что рекуррентное соотношение (7.1) имеет решение $T(n) = \Omega(n^2)$. Таким образом, время работы алгоритма быстрой сортировки (в наихудшем случае) равно $\Theta(n^2)$.

7.4.2. Ожидаемое время работы

Мы уже приводили интуитивный аргумент в пользу того, что ожидаемое время работы процедуры RANDOMIZED-QUICKSORT равно $O(n \lg n)$: если на каждом уровне рекурсии в разделении, производимом процедурой RANDOMIZED_PARTITION, в одну часть массива помещается произвольная фиксированная доля элементов, то высота дерева рекурсии равна $\Theta(\lg n)$, а время работы каждого его уровня — $O(n)$. Даже после добавления некоторого количества новых уровней с наименее сбалансированным разбиением время работы останется равным $O(n \lg n)$. Можно провести точный анализ математического ожидания времени работы процедуры RANDOMIZED-QUICKSORT. Для этого сначала нужно понять, как работает процедура разбиения, а затем получить для математического ожидания времени работы оценку $O(n \lg n)$. Эта верхняя граница математического ожидания времени работы в сочетании с полученной в разделе 7.2 оценкой для наилучшего случая, равной $\Theta(n \lg n)$, позволяют сделать вывод о том, что математическое ожидание времени работы равно $\Theta(n \lg n)$. Мы предполагаем, что значения всех сортируемых элементов различны.

Время работы и сравнения

Процедуры QUICKSORT и RANDOMIZED-QUICKSORT отличаются только выбором опорного элемента; во всех прочих аспектах они идентичны. Таким образом, анализ процедуры RANDOMIZED-QUICKSORT можно провести, рассматривая процедуры QUICKSORT и PARTITION, но в предположении, что опорные элементы выбираются из передаваемого процедуре RANDOMIZED-PARTITION подмассива случайным образом.

Время работы процедуры QUICKSORT определяется преимущественно временем работы, затраченным на выполнение процедуры PARTITION. При каждом выполнении последней происходит выбор опорного элемента, который впоследствии не принимает участия ни в одном рекурсивном вызове процедур QUICKSORT и PARTITION. Таким образом, на протяжении всего времени выполнения алгоритма быстрой сортировки процедура PARTITION вызывается не более n раз. Один вызов процедуры PARTITION выполняется в течение времени $O(1)$, к которому нужно прибавить время, пропорциональное количеству итераций цикла **for** в строках 3–6. В каждой итерации цикла **for** в строке 4 опорный элемент сравнивается с другими элементами массива A. Поэтому, если известно общее количество выполнений строки 4, можно оценить полное время, которое затрачивается на выполнение цикла **for** в процессе работы процедуры QUICKSORT.

Лемма 7.1

Пусть X является количеством сравнений, выполняемых в строке 4 процедуры PARTITION за время работы процедуры QUICKSORT над n-элементным массивом. Тогда время работы процедуры QUICKSORT составляет $O(n + X)$.

Доказательство. Как следует из приведенных выше рассуждений, процедура PARTITION вызывается не более n раз, выполняя при этом фиксированный объем работы и некоторое количество итераций цикла **for**. Каждая итерация цикла **for** выполняет строку 4. ■

Таким образом, наша цель заключается в том, чтобы вычислить величину X, т.е. полное количество сравнений, выполняемых при всех вызовах процедуры PARTITION. Не будем пытаться проанализировать, сколько сравнений производится при *каждом* вызове этой процедуры. Вместо этого получим общую оценку полного количества сравнений. Для этого необходимо понять, в каких случаях в алгоритме производится сравнение двух элементов массива, а в каких — нет. Для упрощения анализа переименуем элементы массива A как $z_1, z_2, \ldots, z_n$, где z_i — i-й наименьший по порядку элемент. Кроме того, определим множество $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$, которое содержит элементы, расположенные между элементами z_i и z_j включительно.

Когда в алгоритме выполняется сравнение элементов z_i и z_j? Прежде чем ответить на этот вопрос, заметим, что сравнение каждой пары элементов производится не более одного раза. Почему? Дело в том, что элементы сравниваются с опорным элементом, который никогда не используется в двух разных вызовах процедуры PARTITION. Таким образом, после некоторого вызова этой процедуры

используемый в качестве опорного элемент впоследствии не будет сравниваться с другими элементами.

Воспользуемся в нашем анализе индикаторными случайными величинами (см. раздел 5.2). Определим величину

$$X_{ij} = \mathrm{I}\{z_i \text{ сравнивается с } z_j\} \ ,$$

с помощью которой учитывается, произошло ли сравнение в течение работы алгоритма (но не в течение определенной итерации или определенного вызова процедуры PARTITION). Поскольку каждая пара элементов сравнивается не более одного раза, полное количество сравнений, выполняемых на протяжении работы алгоритма, легко выразить следующим образом:

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij} \ .$$

Применяя к обеим частям этого выражения операцию вычисления математического ожидания и используя свойство ее линейности и лемму 5.1, получим

$$\begin{aligned} \mathrm{E}[X] &= \mathrm{E}\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathrm{E}[X_{ij}] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ сравнивается с } z_j\} \ . \end{aligned} \tag{7.2}$$

Осталось вычислить величину $\Pr\{z_i \text{ сравнивается с } z_j\}$. Предполагается, что опорные элементы выбираются случайным образом и независимо один от другого.

Полезно поразмышлять о том, когда два элемента *не* сравниваются. Рассмотрим в качестве входных данных для алгоритма быстрой сортировки множество, состоящее из целых чисел от 1 до 10 (в произвольном порядке), и предположим, что в качестве первого опорного элемента выбрано число 7. Тогда в результате первого вызова процедуры PARTITION все числа разбиваются на два множества: $\{1, 2, 3, 4, 5, 6\}$ и $\{8, 9, 10\}$. При этом элемент 7 сравнивается со всеми остальными. Очевидно, что ни одно из чисел, попавшее в первое подмножество (например, 2), больше не будет сравниваться ни с одним элементом второго подмножества (например, с 9).

В общем случае ситуация такая. Поскольку предполагается, что значения всех элементов различаются, при выборе в качестве опорного элемента такого x, что $z_i < x < z_j$, элементы z_i и z_j впоследствии сравниваться не будут. С другой стороны, если в качестве опорного элемент z_i выбран до любого другого элемента Z_{ij}, то он будет сравниваться с каждым элементом множества Z_{ij}, кроме

себя самого. Аналогичное утверждение можно сделать по поводу элемента z_j. В рассматриваемом примере значения 7 и 9 сравниваются, поскольку элемент 7 — первый из множества $Z_{7,9}$, выбранный в качестве опорного. По той же причине (поскольку элемент 7 — первый из множества $Z_{2,9}$, выбранный в качестве опорного) элементы 2 и 9 сравниваться не будут. Таким образом, элементы z_i и z_j сравниваются тогда и только тогда, когда первым в роли опорного в множестве Z_{ij} выбран один из них.

Теперь вычислим вероятность этого события. Перед тем как в множестве Z_{ij} будет выбран опорный элемент, все это множество является не разделенным, и любой его элемент с одинаковой вероятностью может стать опорным. Поскольку всего в этом множестве $j - i + 1$ элемент, а опорные элементы выбираются случайным образом и независимо один от другого, вероятность того, что какой-либо фиксированный элемент первым будет выбран в качестве опорного, равна $1/(j - i + 1)$. Таким образом, мы имеем

$$\begin{aligned}
\Pr\{z_i \text{ сравнивается с } z_j\} &= \Pr\{z_i \text{ или } z_j \text{ — первый опорный элемент, выбранный из } Z_{ij}\} \\
&= \Pr\{z_i \text{ — первый опорный элемент, выбранный из } Z_{ij}\} \\
&\quad + \Pr\{z_j \text{ — первый опорный элемент, выбранный из } Z_{ij}\} \\
&= \frac{1}{j-i+1} + \frac{1}{j-i+1} \\
&= \frac{2}{j-i+1} \, . \qquad (7.3)
\end{aligned}$$

Вторая строка в приведенной выше цепочке равенств следует из того, что рассматриваемые события взаимоисключающие. Комбинируя уравнения (7.2) и (7.3), получаем

$$\mathrm{E}[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \, .$$

Эту сумму можно оценить, если воспользоваться заменой переменных ($k = j - i$) и границей для гармонического ряда (уравнение (A.7)):

$$\begin{aligned}
\mathrm{E}[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} \\
&< \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k}
\end{aligned}$$

$$= \sum_{i=1}^{n-1} O(\lg n)$$
$$= O(n \lg n) \ . \tag{7.4}$$

Таким образом, можно сделать вывод, что при использовании процедуры RANDOMIZED-PARTITION математическое ожидание времени работы алгоритма быстрой сортировки различающихся по величине элементов равно $O(n \lg n)$.

Упражнения

7.4.1

Покажите, что в рекуррентном соотношении

$$T(n) = \max_{0 \le q \le n-1} (T(q) + T(n - q - 1)) + \Theta(n) \ ,$$

$T(n) = \Omega(n^2)$.

7.4.2

Покажите, что время работы быстрой сортировки в наилучшем случае равно $\Omega(n \lg n)$.

7.4.3

Покажите, что выражение $q^2 + (n - q - 1)^2$ достигает максимума на множестве значений $q = 0, 1, \ldots, n - 1$, когда $q = 0$ или $q = n - 1$.

7.4.4

Покажите, что ожидаемое время работы процедуры RANDOMIZED-QUICKSORT равно $\Omega(n \lg n)$.

7.4.5

На практике время работы алгоритма быстрой сортировки можно улучшить, воспользовавшись тем, что алгоритм сортировки по методу вставок быстро работает для "почти" отсортированных входных последовательностей. Для этого можно поступить следующим образом. Когда процедура быстрой сортировки начнет рекурсивно вызываться для обработки подмассивов, содержащих менее k элементов, в ней не будет производиться никаких действий, кроме выхода из процедуры. После возврата из процедуры быстрой сортировки, вызванной на самом высоком уровне, запускается алгоритм сортировки по методу вставок, на вход которого подается весь обрабатываемый массив. На этом процесс сортировки завершается. Докажите, что математическое ожидание времени работы такого алгоритма сортировки равно $O(nk + n \lg(n/k))$. Как следует выбирать значение k, исходя из теоретических и практических соображений?

7.4.6 $\star$

Рассмотрим следующую модификацию процедуры PARTITION. В ней случайным образом выбираются три элемента массива A, и разбиение массива выполняет-

ся по медиане выбранных элементов (т.е. по среднему из этих трех элементов). Найдите приближенную величину вероятности того, что в худшем случае разбиение будет произведено в отношении α к $(1 - \alpha)$, как функцию от α в диапазоне $0 < \alpha < 1$.

Задачи

7.1. Корректность разбиения по Хоару

Представленная в этой главе версия процедуры PARTITION не является реализацией первоначально предложенного алгоритма. Ниже приведен исходный алгоритм разбиения, разработанный Ч.Э.Р. Хоаром (C.A.R. Hoare).

HOARE-PARTITION(A, p, r)

```
 1  x = A[p]
 2  i = p − 1
 3  j = r + 1
 4  while TRUE
 5      repeat
 6          j = j − 1
 7      until A[j] ≤ x
 8      repeat
 9          i = i + 1
10      until A[i] ≥ x
11      if i < j
12          Обменять A[i] и A[j]
13      else return j
```

а. Продемонстрируйте, как работает процедура HOARE-PARTITION с массивом $A = \langle 13, 19, 9, 5, 12, 8, 7, 4, 11, 2, 6, 21 \rangle$. Для этого укажите, чему будут равны значения элементов массива и значения вспомогательных переменных после каждой итерации цикла **while** в строках 4–13.

В следующих трех заданиях предлагается привести аргументы в пользу корректности процедуры HOARE-PARTITION. В предположении, что подмассив $A[p\,..\,r]$ содержит как минимум два элемента, докажите следующие утверждения.

б. Индексы i и j принимают такие значения, что никогда не произойдет обращение к элементам массива A, находящимся за пределами подмассива $A[p\,..\,r]$.

в. По завершении процедуры HOARE-PARTITION она возвращает значение j, такое, что $p \le j < r$.

г. По завершении процедуры HOARE-PARTITION каждый элемент подмассива $A[p\,..\,j]$ не превышает значений каждого элемента подмассива $A[j + 1\,..\,r]$.

В описанной в разделе 7.1 процедуре PARTITION опорный элемент (которым изначально является элемент $A[r]$) отделяется от двух образованных с его помощью подмассивов. В процедуре же HOARE-PARTITION, напротив, опорный элемент (которым изначально является элемент $A[p]$) всегда помещается в один из двух полученных подмассивов: $A[p\,..\,j]$ или $A[j+1\,..\,r]$. Поскольку $p \le j < r$, это разбиение всегда нетривиально.

д. Перепишите процедуру QUICKSORT таким образом, чтобы в ней использовалась процедура HOARE-PARTITION.

7.2. Быстрая сортировка с равными элементами

Анализ ожидаемого времени работы рандомизированной быстрой сортировки в разделе 7.4.2 предполагает, что все значения элементов различны. В данной задаче мы рассмотрим, что произойдет в противном случае.

а. Предположим, что значения всех элементов одинаковы. Каким будет время работы рандомизированной быстрой сортировки в этом случае?

б. Процедура PARTITION возвращает индекс q, такой, что каждый элемент $A[p\,..\,q-1]$ не превышает $A[q]$, а каждый элемент $A[q+1\,..\,r]$ больше $A[q]$. Модифицируйте процедуру PARTITION таким образом, чтобы получить процедуру PARTITION$'(A, p, r)$, которая переставляет элементы $A[p\,..\,r]$ и возвращает два индекса q и t, где $p \le q \le t \le r$, такие, что

- все элементы $A[q\,..\,t]$ равны;
- каждый элемент $A[p\,..\,q-1]$ меньше $A[q]$;
- каждый элемент $A[t+1\,..\,r]$ больше $A[q]$.

Как и процедура PARTITION, ваша процедура PARTITION$'$ должна иметь время работы $\Theta(r-p)$.

в. Модифицируйте процедуру RANDOMIZED-PARTITION так, чтобы она вызывала процедуру PARTITION$'$, и назовите новую процедуру именем RANDOMIZED-PARTITION$'$. Затем модифицируйте процедуру QUICKSORT таким образом, чтобы, в свою очередь, получить процедуру QUICKSORT$'(A, p, r)$, которая вызывает RANDOMIZED-PARTITION$'$ и рекурсивно работает только с теми частями, элементы которых не равны один другому.

г. Каким образом процедура QUICKSORT$'$ позволяет изменить анализ из раздела 7.4.2, чтобы избежать предположения о том, что все элементы различны?

7.3. Альтернативный анализ быстрой сортировки

Можно предложить альтернативный анализ рандомизированной быстрой сортировки, в ходе которого внимание сосредоточивается на математическом ожидании времени, которое требуется для выполнения каждого рекурсивного вызова процедуры QUICKSORT, а не на количестве производимых сравнений.

а. Докажите, что вероятность того, что какой-либо заданный элемент n-элементного массива будет выбран в качестве опорного, равна $1/n$. На основании этого факта определите индикаторную случайную величину

$$X_i = \mathrm{I}\{i\text{-й в порядке возрастания элемент выбран опорным}\} .$$

Что собой представляет величина $\mathrm{E}[X_i]$?

б. Пусть $T(n)$ — случайная величина, обозначающая время работы быстрой сортировки n-элементного массива. Докажите, что

$$\mathrm{E}[T(n)] = \mathrm{E}\left[\sum_{q=1}^{n} X_q \left(T(q-1) + T(n-q) + \Theta(n)\right)\right] . \tag{7.5}$$

в. Покажите, что уравнение (7.5) можно представить в виде

$$\mathrm{E}[T(n)] = \frac{2}{n}\sum_{q=2}^{n-1} \mathrm{E}[T(q)] + \Theta(n) . \tag{7.6}$$

г. Покажите, что

$$\sum_{k=2}^{n-1} k \lg k \le \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2 . \tag{7.7}$$

(*Указание:* разбейте сумму на две части, в одной из которых суммирование проводится по индексам $k = 2, 3, \ldots, \lceil n/2 \rceil - 1$, а в другой — по индексам $k = \lceil n/2 \rceil, \ldots, n-1$.)

д. Покажите с помощью границ из (7.7), что решение рекуррентного соотношения (7.6) имеет вид $\mathrm{E}[T(n)] = \Theta(n \lg n)$. (*Указание:* с помощью подстановки покажите, что $\mathrm{E}[T(n)] \le an \lg n$ для достаточно больши́х n и некоторой положительной константы a.)

7.4. Глубина стека быстрой сортировки

Алгоритм QUICKSORT из раздела 7.1 содержит два рекурсивных вызова самого себя. После того как процедура QUICKSORT вызывает процедуру PARTITION, она рекурсивно сортирует левый подмассив, а затем так же рекурсивно сортирует правый подмассив. Второй рекурсивный вызов в QUICKSORT в действительности не является необходимым; его можно избежать с помощью итеративной управляющей структуры. Этот метод, получивший название ***оконечной рекурсии*** (tail recursion), автоматически применяется хорошими компиляторами. Рассмотрите приведенную ниже версию быстрой сортировки, в которой имитируется оконечная рекурсия.

TAIL-RECURSIVE-QUICKSORT(A, p, r)

```
1  while p < r
2      // Разбиение и сортировка левого подмассива
3      q = PARTITION(A, p, r)
4      TAIL-RECURSIVE-QUICKSORT(A, p, q − 1)
5      p = q + 1
```

а. Докажите, что TAIL-RECURSIVE-QUICKSORT$(A, 1, A.length)$ корректно сортирует массив A.

Обычно компиляторы выполняют рекурсивные процедуры с помощью ***стека***, содержащего необходимую информацию, в том числе и значения параметров, применяющихся для каждого рекурсивного вызова. Информация о последнем вызове находится на вершине стека, а информация о начальном вызове — на его дне. При вызове процедуры информация ***заносится в стек*** (pushed), а при ее завершении — ***выталкивается*** (popped) из него. Поскольку предполагается, что массивы-параметры представлены указателями, при каждом обращении процедуры к стеку передается информация объемом $O(1)$. Глубина стека (stack depth) — это максимальная величина стекового пространства, которая используется в ходе вычисления.

б. Опишите сценарий, в котором глубина стека, необходимая для сортировки процедурой TAIL-RECURSIVE-QUICKSORT n-элементного массива, равна $\Theta(n)$.

в. Модифицируйте код процедуры TAIL-RECURSIVE-QUICKSORT так, чтобы необходимая для ее работы глубина стека в наихудшем случае была равна $\Theta(\lg n)$. Математическое ожидание времени работы алгоритма $O(n \lg n)$ должно при этом остаться неизменным.

7.5. Разбиение по медиане трех элементов

Один из способов улучшения процедуры RANDOMIZED-QUICKSORT заключается в том, чтобы производить разбиение по опорному элементу, определенному аккуратнее, чем путем случайного выбора. Один из распространенных подходов — метод ***медианы трех элементов***. Согласно этому методу в качестве опорного элемента выбирается медиана (средний элемент) подмножества, составленного из трех случайно выбранных в подмассиве элементов (см. упр. 7.4.6). В этой задаче предполагается, что все элементы входного массива $A[1\,..\,n]$ различны и что $n \geq 3$. Обозначим отсортированный выходной массив как $A'[1\,..\,n]$. Используя метод медианы трех элементов для выбора опорного элемента x, определим $p_i = \Pr\{x = A'[i]\}$.

а. Приведите точную формулу для величины p_i как функции от n и i для $i = 2, 3, \ldots, n-1$. (Заметим, что $p_1 = p_n = 0$.)

б. На сколько увеличится вероятность выбора в массиве $A[1\,..\,n]$ в качестве опорного элемента медианы $x = A'[\lfloor (n+1)/2 \rfloor]$, если используется не обычный

способ, а метод медианы? Чему равно предельное отношение этих вероятностей при $n \to \infty$?

в. Будем считать выбор опорного элемента $x = A'[i]$ "удачным", если $n/3 \le i \le 2n/3$. На сколько возрастает вероятность удачного выбора в методе медианы по сравнению с обычной реализацией? (*Указание:* используйте приближение суммы интегралом.)

г. Докажите, что при использовании метода медианы трех элементов время работы алгоритма быстрой сортировки, равное $\Omega(n \lg n)$, изменится только на постоянный множитель.

7.6. Нечеткая сортировка интервалов

Рассмотрим задачу сортировки, в которой отсутствуют точные сведения о значениях чисел. Вместо них для каждого числа на действительной числовой оси задается интервал, которому оно принадлежит. Таким образом, в нашем распоряжении имеется n закрытых интервалов, заданных в виде $[a_i, b_i]$, где $a_i \le b_i$. Задача в том, чтобы произвести ***нечеткую сортировку*** (fuzzy-sort) этих интервалов, т.е. получить такую перестановку $\langle i_1, i_2, \ldots, i_n \rangle$ интервалов, чтобы для каждого $j = 1, 2, \ldots, n$ можно было найти значения $c_j \in [a_{i_j}, b_{i_j}]$, удовлетворяющие неравенствам $c_1 \le c_2 \le \cdots \le c_n$.

а. Разработайте алгоритм нечеткой сортировки n интервалов. Общая структура предложенного алгоритма должна совпадать со структурой алгоритма, выполняющего быструю сортировку по левым границам интервалов (т.е. по значениям a_i). Однако время работы нового алгоритма должно быть улучшено за счет возможностей перекрывающихся интервалов. (По мере увеличения степени перекрытия интервалов задача их нечеткой сортировки все больше упрощается. В представленном алгоритме следует, насколько это возможно, воспользоваться указанным преимуществом.)

б. Докажите, что в общем случае математическое ожидание времени работы рассматриваемого алгоритма равно $\Theta(n \lg n)$, но если все интервалы перекрываются (т.е. если существует такое значение x, что $x \in [a_i, b_i]$) для всех i, то оно равно $\Theta(n)$. В предложенном алгоритме этот факт не следует проверять явно; его производительность должна улучшаться естественным образом по мере увеличения степени перекрытия.

Заключительные замечания

Процедуру быстрой сортировки впервые разработал Хоар (Hoare) [169]; предложенная им версия описана в задаче 7.1. Представленная в разделе 7.1 процедура PARTITION появилась благодаря Н. Ломуто (N. Lomuto). Содержащийся в разделе 7.4 анализ выполнен Авримом Блюмом (Avrim Blum). Хороший обзор литературы, посвященной особенностям реализации алгоритмов и их влиянию

на производительность, можно найти у Седжвика (Sedgewick) [303] и Бентли (Bentley) [42].

Мак-Илрой (McIlroy) [246] показал, как создать конструкцию, позволяющую получить массив, при обработке которого почти каждая реализация быстрой сортировки будет выполняться в течение времени $\Theta(n^2)$. Если реализация рандомизированная, то данная конструкция может создать данный массив на основании информации о том, каким образом в рандомизированном алгоритме быстрой сортировки осуществляется случайный выбор.