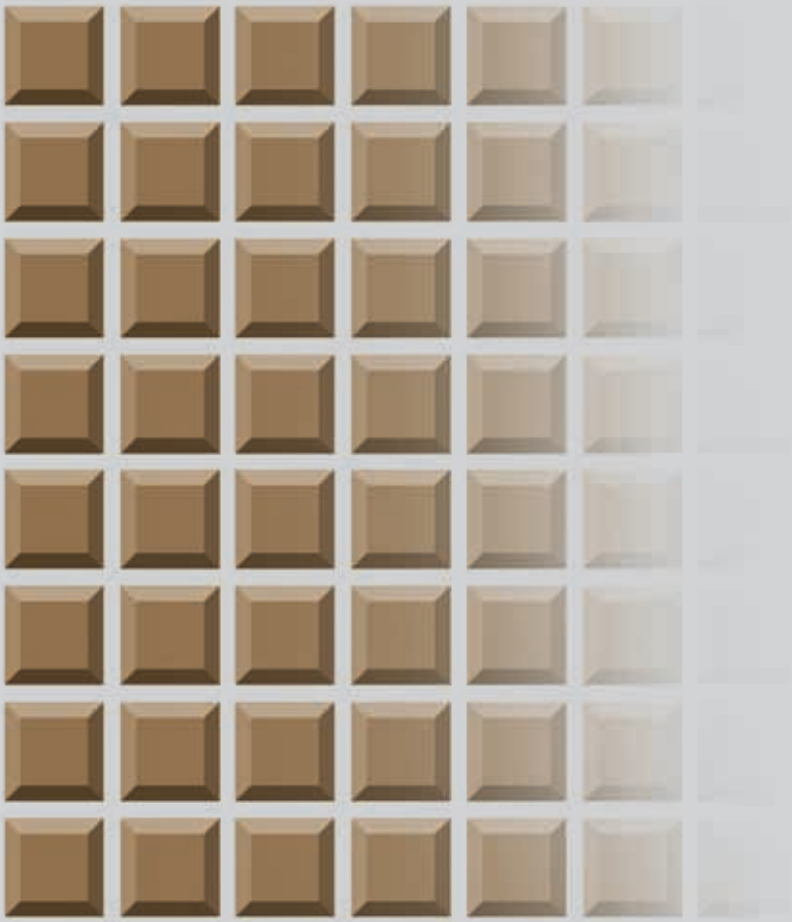
Р. МИЛЛЕР, Л. БОКСЕР

ПОСЛЕДОВАТЕЛЬНЫЕ И ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ



БИНОМ

ПОСЛЕДОВАТЕЛЬНЫЕ И ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ

ОБЩИЙ ПОДХОД

Algorithms Sequential & Parallel

A UNIFIED APPROACH

Russ Miller

Laurence Boxer

Prentice Hall, Upper Saddle River, New Jersey 07458

An Alan R. Apt Book

Р. МИЛЛЕР, Л. БОКСЕР

ПОСЛЕДОВАТЕЛЬНЫЕ И ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ

ОБЩИЙ ПОДХОД

Перевод с английского
А. В. Козвониной
под редакцией
С. М. Окулова

Москва
БИНОМ. Лаборатория знаний
2006

УДК 004.7
ББК 32.973.202
М60

Миллер Р.

М60 Последовательные и параллельные алгоритмы / Р. Миллер, Л. Боксер ; пер. с англ. — М. : БИНОМ. Лаборатория знаний, 2006. — 406 с. : ил.

ISBN 5-94774-325-6

Изложение основывается на объединении в одном курсе вопросов, имеющих отношение к последовательным и параллельным моделям, с большим акцентом на параллельных вычислениях. Подробно излагаются такие темы, как алгоритмы на графах, вычислительная геометрия, фундаментальные модели вычислений, умножение матриц, обработка изображений, вычисление многочленов и нахождение приближенного значения определенных интегралов.

Для студентов и аспирантов, специализирующихся по вычислительным методам и алгоритмам, а также для их преподавателей.

УДК 004.7
ББК 32.973.202

По вопросам приобретения обращаться:
«БИНОМ. Лаборатория знаний»
Телефон: (499) 157-5272
e-mail: binom@Lbz.ru, http://www.Lbz.ru

ISBN 5-94774-325-6

ОГЛАВЛЕНИЕ

Моей жене Селесте и моим детям Мелиссе, Аманде и Брайану.

Расс Миллер

Моей жене Линде и моим детям Робину и Мэтью.

Лоренс Боксер

ПРЕДИСЛОВИЕ

Главной задачей компьютерной науки являются разработка, анализ, применение и научная оценка алгоритмов решения важных задач. Кроме того, компьютерным специалистам предлагаются новые задачи в области *вычислительной науки и техники*, включая проблемы из вычислительной биологии, вычислительной динамики жидкостей и вычислительной химии, если перечислять только несколько. По мере того как параллельные процессы занимают все более важное место при вычислениях, для студентов и ученых становится все важнее понимать возможности применения и анализа примеров алгоритмов как в (традиционной) последовательной модели вычислений, так и в различных параллельных моделях.

Многие факультеты информатики предлагают курсы «Анализ алгоритмов», «Алгоритмы», «Введение в алгоритмы» или «Структуры данных и алгоритмы» на младшем и старшем уровнях. Кроме того, курс «Анализ алгоритмов» требуется для большинства аспирантов, получающих степень в компьютерной науке. В 80-х годах XX века в большинстве из этих предлагаемых курсов делался акцент на алгоритмы для последовательных компьютеров (фон Нейман). В действительности курсы, охватывающие введение в параллельные алгоритмы, стали появляться на факультетах с исследовательской направленностью не ранее конца 80-х годов XX в. Кроме того, эти курсы по параллельным алгоритмам обычно предлагались продвинутым аспирантам. Однако к началу 90-х годов курсы по параллельным вычислениям начали появляться и для студентов, особенно в передовых колледжах с четырехлетним сроком обучения.

Интересно отметить, что в течение большей части 90-х годов XX века традиционные курсы по алгоритмам изменились очень

незначительно. Постепенно такие курсы начали включать элементы параллельных алгоритмов, обычно от одной до трех недель где-то в конце семестра. Однако в конце 90-х было уже вполне привычно встретить алгоритмические курсы, где треть материала была посвящена параллельным алгоритмам.

В этой книге мы используем совсем другой подход к традиционным алгоритмическим курсам. Параллельные вычисления занимают более важное место, т. к. прилавки заполнили небольшие многопроцессорные машины (которые могут быть заказаны по почте из вашего любимого списка продаваемых товаров) и потому, что распределенные вычислительные системы успешно эксплуатируются. Следовательно, мы верим, что настало время преподавать фундаментальный курс по алгоритмам, охватывающий примеры как последовательных, так и параллельных моделей. В действительности, наш подход состоит в том, чтобы объединить в курсе рассмотрение параллельных и последовательных алгоритмов.

Философия этой книги состоит в том, чтобы осветить такую парадигму, как «разделяй и властвуй», и затем рассмотреть вопросы ее применения как для последовательных, так и для параллельных моделей. Ввиду того что мы представляем разработку и анализ примеров для последовательных и параллельных моделей, читатель может заметить, что количество примеров, которые мы можем осветить в течение семестра, ограничено.

Некоторые разделы этой книги были успешно опробованы при обучении аспирантов и студентов Университета штата Нью-Йорк в Буффало.

Предварительные условия: мы предполагаем, что читатель имеет основные знания по структурам данных. То есть читателю должны быть ясны понятия стека, очереди, списка и двоичного дерева на том уровне, на котором обычно преподают на курсе CS2 (Computer Science, информатика). Читатель также должен быть знаком с основами дискретной математики и математического анализа. Особенно хорошо читатель должен знать пределы, суммирование и интегралы.

Обзор глав

Исходные данные курса представлены в главах 1, 2 и 3. В главе 1 вводится понятие асимптотического анализа. Хотя читатель, возможно, встречал часть этого материала в курсе по структурам

данных, мы представляем этот материал довольно подробно. Читатель, не знакомый с некоторыми важными, базисными моментами начального курса исчислений последовательностей, возможно, захочет освежить понятия, такие как предел, суммирование и интеграл, и производные от них, т. к. они естественно появятся в представлении и приложении асимптотического анализа. В главе 2 делается акцент на основы индукции и рекурсии. Так как многие студенты встречались с этим материалом в предыдущих курсах информатики и/или математики, мы сочли необходимым рассмотреть этот материал кратко и предоставили студентам ссылки, чтобы у них была возможность составления необходимого обзора. В главе 3 мы представляем Основной метод, очень полезную (типа поваренной книги) систему оценки рекуррентных уравнений, которые часто встречаются, когда речь идет об алгоритмах.

Глава 4 представляет собой обзор комбинационных схем и сортирующих сетей. Этот обзор можно использовать для мотивации естественного использования параллельных моделей и продемонстрировать сочетаемость архитектурного и алгоритмического подходов. В главе 5 вводятся фундаментальные модели вычислений, в том числе RAM (Random Access Machine) (формальная последовательная архитектура) и разнообразные параллельные модели вычислений. Среди прочих изучаются такие параллельные модели как PRAM (Parallel RAM), матричные сети процессоров и гиперкуб. Кроме того, в главе 5 вводятся термины, такие как разделяемая память и распределенная память.

Основным предметом изучения в главе 6 является важная задача умножения матриц, рассматриваемая для различных моделей вычислений. Это очень мощная операция с разнообразными способами приложения. Мы обсудим применение и анализ для некоторых моделей, представленных в главе 5, и приведем примеры приложения. В главе 8 мы введем метод *перехода по указателю* и покажем, как некоторые алгоритмы, основанные на списках, могут эффективно применяться параллельно.

В главе 9 мы расскажем о мощной парадигме (методе) «разделяй и властвуй». Мы обсудим применение «разделяй и властвуй» к проблемам, таким как перемещение данных, включая сортировку, параллельное чтение/запись и т. д. Алгоритмы и их анализ представлены для разнообразных моделей.

Главы 10 и 11 повествуют о двух важных областях применения, а именно вычислительной геометрии и обработке изображений. В этих главах мы остановим внимание на интересных

проблемах, выбранных из этих важных областей, как способ закрепления подхода этой книги в терминах развивающихся стратегий решений, не зависящих от машины, которые могут быть приспособлены для конкретных моделей так, как это требуется.

В главе 12 основное внимание уделяется фундаментальным теоретическим проблемам графов. Вначале мы представляем стандартно обсуждаемые методы, включая поиск в ширину, поиск в глубину и переход по указателю. Затем мы обсуждаем фундаментальные проблемы, включая построение остовного дерева и транзитивного замыкания. В завершение мы связываем эти методы с жадными алгоритмами решения задач, такими как маркировка компонентов графа, определение минимального перекрывающего леса графа и проблема определения кратчайших путей или путей с минимальным весом в графе.

Глава 13 — это дополнительная глава, связанная с некоторыми фундаментальными числовыми задачами. Основной частью главы является рассмотрение последовательных алгоритмов вычисления многочленов и определения приближенного значения определенных интегралов.

Рекомендуем использовать

В Министерстве информатики и инженерии в Университете штата Нью-Йорк в Буффало (SUNY-Буффало) эта книга используется как для факультативного курса младшего/старшего уровня, так и для обязательного курса первого года аспирантуры. Курс рассчитан на 14 недель, включая двадцать четыре (24) 75-минутные лекции, два обзорных занятия и два экзамена. Обзорные лекции были проведены успевающим аспирантом и использовались для помощи студентам с домашними заданиями и пониманием материала. Обзоры были особенно важны в начале семестра, когда вводился основной математический материал. Курс предназначен для студентов имеющих основные, базовые знания в математике.

Переписка

Если у вас есть какие-либо комментарии или вы хотите высказать критические замечания (конструктивные или нет), пожалуйста, свободно связывайтесь прямо с авторами данной книги.

Рассу Миллеру можно написать по адресу miller@cse.buffalo.edu, а Лоренс Боксер доступен по адресу boxer@niagara.edu. Кроме того, доступен Web-сайт http://www.prehall.com/millerboxer. Этот сайт содержит информацию, относящуюся к тематике книги, включая указатели на образовательные страницы и ссылки на относящиеся к параллельным вычислениям сайты. По адресу www.cse.buffulo.edu/pub/www/facylty/miller/SegPar можно найти опечатки, учтенные в данной книге и, возможно, новые.

Благодарности

Авторы хотят поблагодарить нескольких анонимных рецензентов за предоставленные комментарии, которые использовались для улучшения этой книги. Нам бы хотелось поблагодарить студентов SUNY-Буффало, которые использовали первые наброски данной книги на занятиях и предоставили ценную обратную связь. Нам хочется сказать спасибо Кену Смиту, члену группы технической поддержки в SUNY-Буффало, за предоставленное содействие в поддержке Wintel. Нам также хотелось бы поблагодарить наши семьи за предоставленную поддержку, необходимую для того, чтобы завершить этот трудоемкий проект.

Расс Миллер и Лоренс Боксер, 1999
www.cs.buffalo.edu/pub/www/faculty/miller/research.htm

1

АСИМПТОТИЧЕСКИЙ АНАЛИЗ

Асимптотический анализ

Всестороннее исследование алгоритмов, решающих важные задачи, предполагает их разработку, анализ, экспериментальную проверку. В этой главе вводится математический аппарат и рассматриваются методы, позволяющие выполнять как теоретический, так и экспериментальный анализ алгоритмов. Следует ясно осознавать, что без соответствующего анализа достаточно сложно объяснить причины выбора конкретного алгоритма. В связи с этим фактом важной составляющей большинства углубленных курсов по структурам данных и алгоритмам является обучение методам оценки ресурсов (времени работы, памяти и т. д.), требуемых конкретному алгоритму. Стоит отметить и ключевую роль курса по доказательству корректности алгоритмов, ибо иметь очень быстрые алгоритмы, но не всегда выдающие правильные результаты, бессмысленно. Однако по практическим причинам методы доказательства правильности работы алгоритма в целом в данном тексте не рассматриваются.

В этой книге обсуждаются вопросы, связанные с количеством процессоров, требуемых для эффективного выполнения алгоритма, объемом памяти и временем работы, необходимые алгоритму. Это позволит провести разумное сравнение алгоритмов так, чтобы любой специалист мог принимать обоснованные решения. Например, какой алгоритм сортировки следует выбрать для конкретной входной информации, представленной в памяти через определенные структуры данных, при работе на последовательной ЭВМ. Стоит указать на то, что при рассмотрении числовых задач требуется обращать внимание на точность вычислений. К сожалению, эта тема не рассматривается в данной книге. На самом деле, большинство алгоритмов, рассмотренных в книге, можно назвать «нечисловыми» по своей природе.

На практике часто оказывается, что мы больше уделяем внимания времени работы алгоритма, а не используемой памяти. Это может удивить студентов, привыкших думать об относительно небольших проектах классического типа, которые, будучи однажды отлажены (или, по крайней мере, освобождены от бесконечных циклов), начинают выводить результаты через долю секунды. Однако многие важные приложения, обрабатывающие огромные объемы данных, требуют часов или даже дней времени центрального процессора. Примеры таких приложений можно найти в таких областях, как молекулярное моделирование, прогнозирование погоды, анализ изображений, обучение нейронных сетей и т. д. Помимо цены компьютерного времени, человеческое нетерпение или серьезные ограничения по срокам ограничивают использование приложений. Например, прогноз погоды имеет смысл при его доступности заранее, до прогнозируемого периода. С другой стороны, в большинстве известных алгоритмов и связанных с ними структур данных требования к памяти обычно очень разумны и не превосходят величины, кратной (с небольшой константой) объему обрабатываемой информации. Видимо, эти причины и обусловливают первичность оценки времени работы алгоритма.

В этой главе рассматривается математический аппарат анализа ресурсов, используемых компьютерным алгоритмом. Чаще всего речь идет о времени, поэтому используется соответствующая терминология. Однако весь инструментарий с таким же успехом может быть использован и для оценки памяти.

Условные обозначения и терминология

В этом параграфе представлены некоторые условные обозначения и терминология, используемые в дальнейшем в тексте. Мы постараемся придерживаться традиционных условных обозначений и стандартной терминологии. В основном, используется положительное целое n для обозначения размерности задачи (количества входных данных), для решения которой предназначен анализируемый алгоритм. Обрабатываться может, например, массив из n элементов или связный список, дерево, или граф с n вершинами. Запись $T(n)$ используется для обозначения времени работы алгоритма с задачей размерности n. (Иногда алгоритм анализируется с помощью более чем одного параметра, и соответственно используются обозначения, такие как $T(m, n)$, $T(k, n, p)$ и т. д.)

Алгоритм может быть реализован на различных ЭВМ, с различным быстродействием и программным обеспечением. Мы предполагаем, что один и тот же алгоритм, оперирующий с теми же значениями данных, будет выполняться быстрее, если будет реализован на транслируемом языке суперкомпьютера, а не интерпретируемом языке персонального компьютера, скажем, 80-х годов XX века. Таким образом, редко имеет смысл анализировать алгоритмы в терминах реального времени центрального процессора. Скорее требуется, чтобы анализ отражал истинную эффективность алгоритма безотносительно к таким факторам, как скорость аппаратного/программного обеспечения, с которыми применяется алгоритм. Суть заключается в оценке эффективности алгоритмических (программных) решений, а не их конкретной реализации.

Таким образом, анализ алгоритмов в основном соответствует следующим принципам:

1. **Игнорировать зависимость от конкретной ЭВМ.** Не говорится о том, насколько быстро отдельный процессор выполняет машинную инструкцию.

2. **Анализировать изменение $T(n)$ при $n \to \infty$.** Даже неэффективный алгоритм при работе с небольшим количеством входных данных часто завершает работу за допустимое время. Таким образом, анализ $T(n)$ определяет время работы алгоритма при больших значениях n (напоминаем, что n обычно обозначает размерность данных, обрабатываемых алгоритмом).

3. **Темп роста.** В асимптотическом анализе исследуется общее поведение функции при увеличении входного параметра (т. е. нас интересует поведение $T(n)$ по мере того, как $n \to \infty$), поэтому в выражении $T(n)$ элементы, не оказывающие влияние на поведение функции, опускаются. В действительности, если нас интересует темп роста функции по мере увеличения n, следует игнорировать постоянные члены в выражении. Это не значит, что эти члены выражения неуместны на практике, просто они неуместны для определения темпа роста функции. Так, например, мы говорим, что функция $3n^3 + 10n^2 + n + 17$ увеличивается как n^3. Рассмотрим другой пример. При увеличении n какой из алгоритмов предпочтительнее: с временем работы $95n^2 + 405n + 1997$ или с временем работы, равным $2n^3 + 12$? Мы надеемся, вы выбрали первое, с темпом роста, равным n^2, а не последнее с гораздо большим темпом роста в n^3. Естественно, однако, что если n мало, любой предпочел бы последнее $(2n^3 + 12)$ первому $(95n^2 + 405n + 1997)$. В действительности, требуется определить значение n, при котором происходит пересечение функций, и на

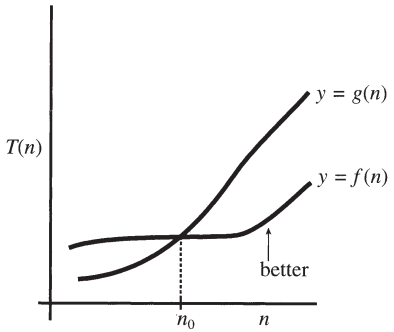


Рис. 1.1. Иллюстрация темпа роста двух функций $f(n)$ и $g(n)$. Заметьте, что для больших значений n алгоритм с асимптотическим временем работы, равным $f(n)$, обычно лучше, чем алгоритм с асимптотическим временем работы, равным $g(n)$. В этом примере «большое» определяется как $n \geq n_0$.

основе этого значения отдавать предпочтение тому или иному алгоритму. На рисунке 1.1 проиллюстрирована эта ситуация.

Асимптотическая форма записи

В этом разделе приводятся некоторые стандартные формы записи, используемые при анализе асимптотического поведения функции. Так как часто поведение функции легче выразить через поведение другой, более простой функции, то для упрощения терминологии лучше использовать две функции. Предположим, что f и g – положительные функции от n. Тогда

1. $f(n) = \Theta(g(n))$ (читается «f от n есть тэта от g от n») тогда и только тогда, когда существуют положительные константы c_1, c_2 и n_0 такие, что $c_1 g(n) \leq f(n) \leq c_2 g(n)$ при $n \geq n_0$; см. рис. 1.2.

2. $f(n) = O(g(n))$ (читается «f от n есть о большое от g от n») тогда и только тогда, когда существуют положительные константы c и n_0 такие, что $f(n) \leq cg(n)$ при $n \geq n_0$; см. рис. 1.3.

3. $f(n) = \Omega(g(n))$ (читается «f от n есть омега большое от g от n») тогда и только тогда, когда существуют положительные константы c и n_0 такие, что $cg(n) \leq f(n)$ при $n \geq n_0$; см. рис. 1.4.

4. $f(n) = o(g(n))$ (читается «f от n есть о малое от g от n») тогда и только тогда, когда для любой положительной

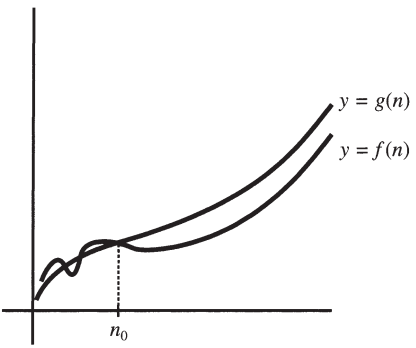


Рис. 1.2. Иллюстрация Θ записи. $f(n) = \Theta(g(n))$, т. к. функции $f(n)$ и $g(n)$ возрастают с одной скоростью при всех $n \geq n_0$.

[. . .]

нения при решении рассматриваемой *задачи*. Например, любой алгоритм вычисления минимального элемента неотсортированного массива из n элементов должен проверить каждый элемент массива (потому что любой пропущенный элемент может быть минимальным элементом). Следовательно, любой последовательный алгоритм решения этой задачи тратит время $\Omega(n)$. Итак, алгоритм, выполняемый за время $\Theta(n)$, оптимален.

Заметьте, что мы используем термин *оптимальный* в смысле *асимптотически оптимальный*. Оптимальный алгоритм – это необязательно самый быстрый возможный алгоритм, дающий верное решение поставленной перед ним задачи, но он должен быть ограничен постоянным множителем или должен быть самым быстрым возможным алгоритмом решения задачи. Доказать оптимальность часто бывает сложно, и существует много задач, для которых неизвестно оптимальное время выполнения. Существуют, однако, задачи, для которых доказать оптимальность достаточно просто, некоторые из этих задач будут рассмотрены в этой книге.

Примечания к главе

Идея применения асимптотического анализа к алгоритмам часто приписывается Дональду Э. Кнуту (Donald E. Knuth) (www-cs-faculty.Stanford.EDU/~knuth/). Хотя асимптотический анализ входит составной частью в книги Д. Э. Кнута серии «Искусство программирования», в действительности, источником O-записи считается учебник по теории чисел Бачмана (Bachmann) 1892 года. o-запись была, очевидно, введена Ландау (Landau) в 1909, а современное использование этой формы записи в алгоритмах приписывается работе Д. Э. Кнута, появившейся в 1976 («Большой омикрон и большая омега и большая тета» («Big omicron and big omega and big theta», ACM SIGACT News, 8(2), 18–23). Историческое развитие асимптотической формы записи в информатике можно найти в обзорах Д. Э. Кнута и в книге Брассара (Brassard) и Брэтли (Bratley) «Алгоритмика: Теория и Практика» (Algorithmics: Theory and Practice) (Prentice Hall, 1988). Одна из ранних книг, получивших статус «классической», – это книга А. В. Ахо (A. V. Aho), Дж. Э. Хопкрофта (J. E. Hopcroft) и Дж. Д. Ульмана (J. D. Ullman) «Построение и анализ вычислительных алгоритмов»[1] (The Design and Analysis of Computer Algorithms), выпущен-

[1] Имеется русский перевод: – М.: Мир, 1979. – *Прим. ред.*

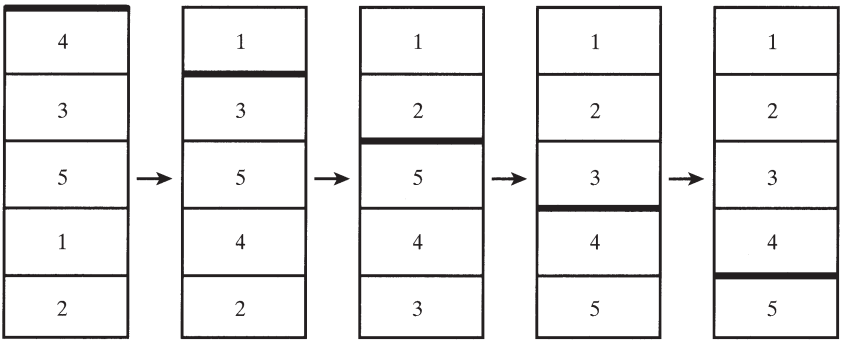


Рис. 1.13. Пример СортировкиВыбором. Полный просмотр элементов массива осуществляется для того, чтобы определить минимальный элемент, который должен быть первым в массиве. Осуществляется перестановка минимального элемента и элемента, стоящего на первом месте. Далее осуществляется просмотр оставшихся четырех элементов и определяется минимальный из них. Найденный элемент переставляется со вторым элементом массива. Работа алгоритма продолжается до тех пор, пока не будет верно упорядочен $n - 1$ элемент, что означает правильность упорядочения всех n элементов.

ная в 1974 году Addison Wesley. Более поздние книги, посвященные алгоритмам и их анализу, – это книги «Алгоритм: построение и анализ»[1] (Introduction to Algorithms) Т. Х. Кормена (T. H. Cormen), С. Е. Лейзерсона (C. E. Leiserson) и Р. Л. Ривеста (R. L. Rivest) (McGraw-Hill, New York, 1989) и «Компьютерные алгоритмы / C++» (Computer Algorithms / C++) Е. Хоровица (E. Horowitz), С. Сани (S. Sahni) и С. Раджасекарана (S. Rajasekaran) (Computer Science press, New York, 1996).

Упражнения

1. Расположите следующее по возрастанию: n, $n^{1/2}$, $\log n$, $\log(\log n)$, $\log^2 n$, $(1/3)^n$, 4, $(3/2)^n$, $n!$

2. Докажите или опровергните каждое из следующих утверждений.

a) $f(n) = O(g(n)) \Rightarrow g(n) = o(f(n))$
b) $f(n) + g(n) = \Theta(\max(f(n), g(n)))$
c) $f(n) = O((f(n))^2)$
d) $f(n) = O(g(n)) \Rightarrow g(n) = \Omega(f(n))$
e) $f(n) + o(f(n)) = \Theta(f(n))$

[1] Имеется русский перевод: – М.: БИНОМ: МЦНМО, 2004. – *Прим. ред.*

3. Используйте O, o, ω и Θ для описания отношений между следующими парами функций:

a) $\log^k n$, n, где k и ϵ — положительные константы
b) n^k, c^n, где k и c — константы, $k > 0$, $c > 1$
c) 2^n, $2^{n/2}$

4. Докажите, что $17n^{1/6} = O(n^{1/5})$.

5. Докажите, что $\sum_{k=1}^{n} k^{1/6} = \Theta(n^{7/6})$.

6. Для данного набора n **целых** чисел из интервала [1,..., 100] приведите эффективный последовательный алгоритм их сортировки. Обсудите время, память и оптимальность вашего решения.

7. (Функция суммы): Определить асимптотическое время выполнения следующего алгоритма, находящего сумму элементов массива. Покажите, что он оптимален.

Function Total(*list*)
Дано: Массив *list* числовых элементов с индексами от 1 до *n*.
Получить: Сумму элементов массива.
Локальные переменные: целый *index*, числовая *subtotal*
Действия:
    subtotal = 0
    Для *index* = 1 to *n*, do
        subtotal = *subtotal* + *list*[*index*]
    Return *subtotal*

8. (СортировкаВыбором): Определить асимптотическое время выполнения алгоритма, идея которого рассматривается на рис. 1.13. Определить требования к памяти для хранения данных и дополнительной памяти.

Subprogram SelectionSort(*List*)
Дано: Массив *List*[1...*n*] для сортировки по возрастанию в соответствии с *key* полем записей
Получить: Упорядоченный *List*.
Алгоритм: СортировкаВыбором
    Для каждой позиции в массиве *List* мы
    Определяем индекс, соответствующий элементу из неотсортированной части *List*, являющемуся минимальным.
    Обмениваем элемент на только что определенной позиции с текущим элементом.

[. . .]