Часть I

Основы

Введение

Эта часть книги заставит вас задуматься о вопросах, связанных с разработкой и анализом алгоритмов. Она была запланирована как вводный курс, в котором рассматриваются способы определения алгоритмов, некоторые стратегии их разработки, использующиеся в этой книге, а также применяемые при анализе алгоритмов различные основополагающие идеи. Кратко рассмотрим содержание глав первой части.

В главе 1 рассматривается понятие алгоритма и его роль в современных вычислительных системах. В ней приводится определение алгоритма и даются некоторые примеры. Здесь также обосновывается положение о том, что алгоритмы — это такой же технологический продукт, как, например, аппаратное обеспечение, графические интерфейсы пользователя, объектно-ориентированные системы или сети.

В главе 2 читатель получит возможность ознакомиться с алгоритмами, с помощью которых решается задача о сортировке последовательности из n чисел. Эти алгоритмы сформулированы в виде псевдокода. Несмотря на то, что используемый псевдокод напрямую не преобразуется ни в один из общепринятых языков программирования, он вполне адекватно передает структуру алгоритма, поэтому достаточно компетентный программист легко сможет реализовать его на любом языке программирования. Для изучения

выбраны алгоритм сортировки вставкой, в котором используется инкрементный подход, и алгоритм сортировки слиянием, который характеризуется применением рекурсивного метода, известного также как метод “разделяй и властвуй” (метод разбиения). В обоих алгоритмах время выполнения возрастает с ростом количества сортируемых элементов, однако скорость этого роста зависит от выбранного алгоритма. В этой главе будет определено время работы изучаемых алгоритмов; кроме того, вы познакомитесь со специальными обозначениями для выражения времени работы алгоритмов.

В главе 3 дается точное определение обозначений, введенных в главе 2 (которые называются асимптотическими обозначениями). В начале главы 3 определяется несколько асимптотических обозначений для оценки времени работы алгоритма сверху и/или снизу. Остальные разделы главы в основном посвящены описанию математических обозначений. Их предназначение не столько в том, чтобы ознакомить читателя с новыми математическими концепциями, сколько в том, чтобы он смог убедиться, что используемые им обозначения совпадают с принятыми в данной книге.

В главе 4 представлено дальнейшее развитие метода “разделяй и властвуй”, введенного в главе 2. В частности, в главе 4 представлены методы решения рекуррентных соотношений, с помощью которых описывается время работы рекурсивных алгоритмов. Б*о*льшая часть главы посвящена обоснованию “метода контроля” (master method), который используется для решения рекуррентных соотношений, возникающих в алгоритмах разбиения. Заметим, что изучение доказательств можно опустить без ущерба для понимания материала книги.

Глава 5 служит введением в анализ вероятностей и рандомизированные алгоритмы (т.е. такие, которые основаны на использовании случайных чисел). Анализ вероятностей обычно применяется для определения времени работы

алгоритма в тех случаях, когда оно может меняться для различных наборов входных параметров, несмотря на то, что эти наборы содержат одно и то же количество параметров. В некоторых случаях можно предположить, что распределение входных величин описывается некоторым известным законом распределения вероятностей, а значит, время работы алгоритма можно усреднить по всем возможным наборам входных параметров. В других случаях распределение возникает не из-за входных значений, а в результате случайного выбора, который делается во время работы алгоритма. Алгоритм, поведение которого определяется не только входными значениями, но и величинами, полученными с помощью генератора случайных чисел, называется рандомизированным алгоритмом.

В приложениях A–B содержится дополнительный математический материал, который окажется полезным в процессе чтения книги. Скорее всего, вы уже знакомы с основной частью материала, содержащегося в приложениях (хотя некоторые из встречавшихся вам ранее обозначений иногда могут отличаться от тех, что приняты в данной книге). Поэтому к приложениям следует относиться как к справочному материалу. С другой стороны, не исключено, что вы еще незнакомы с большинством вопросов, рассматриваемых в части I.

Глава 1

Роль алгоритмов в вычислениях

Что такое алгоритмы? Стоит ли тратить время на их изучение? Какова роль алгоритмов и как они соотносятся с другими компьютерными технологиями? Эта глава дает ответы на поставленные вопросы.

1.1. Алгоритмы

Говоря неформально, ***алгоритм*** (algorithm) — это любая корректно определенная вычислительная процедура, на ***вход*** (input) которой подается некоторая величина или набор величин, и результатом выполнения которой является ***выходная*** (output) величина или набор значений. Таким образом, алгоритм представляет собой последовательность вычислительных шагов, преобразующих входные величины в выходные.

Алгоритм также можно рассматривать как инструмент, предназначенный для решения корректно поставленной ***вычислительной задачи*** (computational problem). В постановке задачи в общих чертах задаются отношения между входом и выходом. В алгоритме описывается конкретная вычислительная процедура, с помощью которой удается добиться выполнения указанных отношений.

Например, может понадобиться выполнить сортировку последовательности чисел в неубывающем порядке. Эта задача часто возникает на практике и

служит благодатной почвой для ознакомления на ее примере со многими стандартными методами разработки и анализа алгоритмов. ***Задача сортировки*** (sorting problem) формально определяется следующим образом.

Вход: последовательность из n чисел $\langle a_1, a_2, \ldots, a_n \rangle$.

Выход: перестановка (изменение порядка) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ входной последовательности таким образом, что для ее членов выполняется соотношение $a'_1 \le a'_2 \le \ldots \le a'_n$.

Например, если на вход подается последовательность $\langle 31, 41, 59, 26, 41, 58 \rangle$, то вывод алгоритма сортировки должен быть таким: $\langle 26, 31, 41, 41, 58, 59 \rangle$. Подобная входная последовательность называется ***экземпляром*** (instance) задачи сортировки. Вообще говоря, ***экземпляр задачи*** состоит из ввода (удовлетворяющего всем ограничениям, наложенным при постановке задачи), необходимого для решения задачи.

В информатике сортировка является основополагающей операцией (во многих программах она используется в качестве промежуточного шага), в результате чего появилось большое количество хороших алгоритмов сортировки. Выбор наиболее подходящего алгоритма зависит от многих факторов, в том числе от количества сортируемых элементов, от их порядка во входной последовательности, от возможных ограничений, накладываемых на члены последовательности, а также от того, какое устройство используется для хранения последовательности: основная память, магнитные диски или накопители на магнитных лентах.

Говорят, что алгоритм ***корректен*** (correct), если для каждого ввода результатом его работы является корректный вывод. Мы говорим, что корректный алгоритм ***решает*** данную вычислительную задачу. Если алгоритм некорректный, то для некоторых вводов он может вообще не завершить свою

работу или выдать ответ, отличный от ожидаемого. Правда, некорректные алгоритмы иногда могут оказаться полезными, если в них есть возможность контролировать частоту возникновения ошибок. Такой пример рассматривается в главе 31, в которой изучаются алгоритмы определения простых чисел, намного превышающих единицу. Тем не менее, обычно мы заинтересованы только в корректных алгоритмах.

Алгоритм может быть задан на естественном языке, в виде компьютерной программы или даже воплощен в аппаратном обеспечении. Единственное требование — его спецификация должна предоставлять точное описание вычислительной процедуры, которую требуется выполнить.

Какие задачи решаются с помощью алгоритмов?

Вычислительные задачи, для которых разработаны алгоритмы, отнюдь не ограничиваются сортировкой. (Возможно, об их разнообразии можно судить по объему данной книги.) Практическое применение алгоритмов чрезвычайно широко, о чем свидетельствуют приведенные ниже примеры.

- Целью проекта по расшифровке генома человека является идентификация всех 100000 генов, входящих в состав ДНК человека, определение последовательностей, образуемых 3 миллиардами базовых пар, из которых состоит ДНК, сортировка этой информации в базах данных и разработка инструментов для ее анализа. Для реализации всех перечисленных этапов нужны сложные алгоритмы. Решение разнообразных задач, являющихся составными частями данного проекта, выходит за рамки настоящей книги, однако идеи, описанные во многих ее главах, используются для решения упомянутых биологических проблем. Это позволяет ученым достигать поставленных целей, эффективно используя вычислительные ресурсы. При этом экономится

время (как машинное, так и затрачиваемое сотрудниками) и деньги, а также повышается эффективность использования лабораторного оборудования.

- Internet позволяет пользователям в любой точке мира быстро получать доступ к информации и извлекать ее в больших объемах. Управление и манипуляция этими данными осуществляется с помощью хитроумных алгоритмов. В число задач, которые необходимо решить, входит определение оптимальных маршрутов, по которым перемещаются данные (методы для решения этой задачи описываются в главе 24), и быстрый поиск страниц, на которых находится та или иная информация, с помощью специализированных поисковых машин (соответствующие методы приводятся в главах 11 и 32).

- Электронная коммерция позволяет заключать сделки и предоставлять товары и услуги с помощью электронных технических средств. Для того чтобы она получила широкое распространение, важно иметь возможность защищать такую информацию, как номера кредитных карт, пароли и банковские счета. В число базовых технологий в этой области входят криптография с открытым ключом и цифровые подписи (они описываются в главе 31), основанные на численных алгоритмах и теории чисел.

- В производстве и коммерции очень важно распорядиться ограниченными ресурсами так, чтобы получить максимальную выгоду. Нефтяной компании может понадобиться информация о том, где пробурить скважины, чтобы получить от них как можно более высокую прибыль. Кандидат в президенты может задаться вопросом, как потратить деньги, чтобы максимально повысить свои шансы победить на выборах. Авиакомпаниям важно знать, какую минимальную цену можно

назначить за билеты на тот или иной рейс, чтобы уменьшить количество свободных мест и не нарушить при этом законы, регулирующие авиаперевозку пассажиров. Поставщик услуг Internet должен уметь так размещать дополнительные ресурсы, чтобы повышался уровень обслуживания клиентов. Все эти задачи можно решить с помощью линейного программирования, к изучению которого мы приступим в главе 29.

Несмотря на то, что некоторые детали представленных примеров выходят за рамки настоящей книги, в ней приводятся основные методы, применяющиеся для их решения. В книге также показано, как решить многие конкретные задачи, в том числе те, что перечислены ниже.

- Пусть имеется карта дорог, на которой обозначены расстояния между каждой парой соседних перекрестков. Наша цель — определить кратчайший путь от одного перекрестка к другому. Количество возможных маршрутов может быть огромным, даже если исключить те из них, которые содержат самопересечения. Как найти наиболее короткий из всех возможных маршрутов? При решении этой задачи карта дорог (которая сама по себе является моделью настоящих дорог) моделируется в виде графа (мы подробнее познакомимся с графами в главе 10 и приложении Б). Задача будет заключаться в определении кратчайшего пути от одной вершины графа к другой. Эффективное решение этой задачи представлено в главе 24.

- Пусть дана последовательность $\langle A_1, A_2, \ldots, A_n \rangle$, образованная n матрицами, и нам нужно найти произведение этих матриц. Поскольку матричное произведение обладает свойством ассоциативности, существует несколько корректных порядков умножения. Например, если $n = 4$, перемножение матриц можно выполнять любым из следующих

способов (определяемых скобками): $\left(A_1\left(A_2\left(A_3A_4\right)\right)\right)$, $\left(A_1\left(\left(A_2A_3\right)A_4\right)\right)$, $\left(\left(A_1A_2\right)\left(A_3A_4\right)\right)$, $\left(\left(A_1\left(A_2A_3\right)\right)A_4\right)$ или $\left(\left(\left(A_1A_2\right)A_3\right)A_4\right)$. Если все эти матрицы квадратные (т.е. одинакового размера), порядок перемножения не влияет на продолжительность процедуры. Если же матрицы различаются по размеру (но при этом их размеры соответствуют правилам матричного умножения), то порядок их перемножения может очень сильно влиять на время выполнения процедуры. Количество всевозможных вариантов, различающихся порядком перемножения матриц, растет с увеличением количества матриц по экспоненциальному закону, поэтому для перебора всех этих вариантов может потребоваться довольно длительное время. Из главы 15 мы узнаем, как эта задача намного эффективнее решается с помощью общего метода, известного как динамическое программирование.

- Пусть имеется уравнение $ax \equiv b\left(\bmod n\right)$, где a, b и n — целые числа, и нужно найти все целые x по модулю n, удовлетворяющие этому уравнению. Оно может не иметь решения, может иметь одно или несколько решений. Можно попытаться применить метод, заключающийся в последовательной подстановке в уравнение чисел $x = 0,1,\ldots,n-1$, но в главе 31 представлен более эффективный метод.

- Пусть имеется n принадлежащих плоскости точек, и нужно найти выпуклую оболочку этих точек. Выпуклой оболочкой точек называется минимальный выпуклый многоугольник, содержащий эти точки. Для решения этой задачи удобно воспользоваться такой наглядной картиной: если представить точки в виде вбитых в доску и торчащих из нее гвоздей, то выпуклую оболочку можно получить, намотав на них резинку таким образом, чтобы все гвозди вошли внутрь замкнутой линии, образуемой резинкой. Каждый гвоздь, вокруг которого

обвивается резинка, становится вершиной выпуклой оболочки (рис. 33.6). В качестве набора вершин может выступать любое из 2^n подмножеств множества точек. Однако недостаточно знать, какие точки являются вершинами выпуклой оболочки, нужно еще указать порядок их обхода. Таким образом, чтобы найти выпуклую оболочку, приходится перебрать множество вариантов. В главе 33 описаны два хороших метода определения выпуклой оболочки.

Приведенные выше случаи применения алгоритмов отнюдь не являются исчерпывающими, однако на их примере выявляются две общие характеристики, присущие многим интересным алгоритмам.

1. Есть множество вариантов-кандидатов, большинство из которых решениями не являются. Поиск среди них нужной комбинации является довольно трудоемким.
2. Задачи имеют практическое применение. Простейший пример среди перечисленных задач — определение кратчайшего пути, соединяющего два перекрестка. Любая компания, занимающаяся авто- или железнодорожными перевозками, финансово заинтересована в том, чтобы определить кратчайший маршрут. Это способствовало бы снижению затрат труда и расходов горючего. Маршрутизатору Internet также нужно иметь возможность находить кратчайшие пути в сети, чтобы как можно быстрее доставлять сообщения.

Структуры данных

В книге представлен ряд структур данных. ***Структура данных*** (data structure) — это способ хранения и организации данных, облегчающий доступ к этим данным и их модификацию. Ни одна структура данных не является

универсальной и не может подходить для всех целей, поэтому важно знать преимущества и ограничения, присущие некоторым из них.

Методические указания

Данную книгу можно рассматривать как “сборник рецептов” для алгоритмов. Правда, однажды вам встретится задача, для которой вы не сможете найти опубликованный алгоритм (например, таковы многие из приведенных в книге упражнений и задач). Данная книга научит вас методам разработки алгоритмов и их анализа. Это позволит вам разрабатывать корректные алгоритмы и оценивать их эффективность самостоятельно.

Сложные задачи

Большая часть этой книги посвящена эффективным алгоритмам. Обычной мерой эффективности для нас является скорость и время, в течение которого алгоритм выдает результат. Однако существуют задачи, для которых неизвестны эффективные методы их решения. В главе 34 рассматривается интересный вопрос, имеющий отношение к подобным задачам, известным как NP-полные.

Почему же NP-полные задачи представляют интерес? Во-первых, несмотря на то, что до сих пор не найден эффективный алгоритм их решения, также не доказано, что такого алгоритма не существует. Другими словами, неизвестно, существует ли эффективный алгоритм для NP-полных задач. Во-вторых, набор NP-полных задач обладает замечательным свойством. Оно заключается в том, что если эффективный алгоритм существует хотя бы для одной из этих задач, то его можно сформулировать и для всех остальных. Эта взаимосвязь между NP-полными задачами и отсутствие методов их эффективного решения вызывает еще б*о*льший интерес к ним. В третьих, некоторые NP-полные задачи

похожи (но не идентичны) на задачи, для которых известны эффективные алгоритмы. Небольшое изменение формулировки задачи может значительно ухудшить эффективность самого лучшего из всех известных алгоритмов.

Полезно располагать знаниями о NP-полных задачах, поскольку в реальных приложениях некоторые из них возникают неожиданно часто. Если перед вами встанет задача найти эффективный алгоритм для NP-полной задачи, скорее всего, вы потратите много времени на безрезультатные поиски. Если же вы покажете, что данная задача принадлежит к разряду NP-полных, то можно будет вместо самого лучшего из всех возможных решений попробовать найти достаточно эффективное.

Рассмотрим конкретный пример. Компания грузового автотранспорта имеет один центральный склад. Каждый день грузовики загружаются на этом складе и отправляются по определенному маршруту, доставляя груз в несколько мест. В конце рабочего дня грузовик должен вернуться на склад, чтобы на следующее утро его снова можно было загрузить. Чтобы сократить расходы, компании нужно выбрать оптимальный порядок доставки груза в различные точки. При этом расстояние, пройденное грузовиком, должно быть минимальным. Эта задача хорошо известна как “задача о коммивояжере”, и она является NP-полной. Эффективный алгоритм решения для нее неизвестен, однако при некоторых предположениях можно указать такие алгоритмы, в результате выполнения которых полученное расстояние будет ненамного превышать минимально возможное. Подобные “приближенные алгоритмы” рассматриваются в главе 35.

Упражнения

1.1-1. Приведите реальные примеры задач, в которых возникает одна из таких вычислительных задач: сортировка, определение оптимального порядка перемножения матриц, поиск выпуклой оболочки.

1.1-2. Какими еще параметрами, кроме скорости, можно характеризовать алгоритм на практике?

1.1-3. Выберите одну из встречавшихся вам ранее структур данных и опишите ее преимущества и ограничения.

1.1-4. Что общего между задачей об определении кратчайшего пути и задачей о коммивояжере? Чем они различаются?

1.1-5. Сформулируйте задачу, в которой необходимо только наилучшее решение. Сформулируйте также задачу, в которой может быть приемлемым решение, достаточно близкое к наилучшему.

1.2. Алгоритмы как технология

Предположим, быстродействие компьютера и объем его памяти можно увеличивать до бесконечности. Была бы в таком случае необходимость в изучении алгоритмов? Была бы, но только для того, чтобы продемонстрировать, что метод решения имеет конечное время работы и что он дает правильный ответ.

Если бы компьютеры были неограниченно быстрыми, подошел бы любой корректный метод решения задачи. Возможно, вы бы предпочли, чтобы реализация решения была выдержана в хороших традициях программирования (т.е. качественно разработана и аккуратно занесена в документацию), но чаще всего выбирался бы метод, который легче всего реализовать.

Конечно же, сегодня есть весьма производительные компьютеры, но их быстродействие не может быть бесконечно большим. Память тоже дешевеет, но она не может быть бесплатной. Таким образом, время вычисления — это такой же ограниченный ресурс, как и объем необходимой памяти. Этими ресурсами следует распоряжаться разумно, чему и способствует применение алгоритмов, эффективных в плане расходов времени и памяти.

Эффективность

Алгоритмы, разработанные для решения одной и то же задачи, часто очень сильно различаются по эффективности. Эти различия могут быть намного значительнее, чем те, что вызваны применением неодинакового аппаратного и программного обеспечения.

В качестве примера можно привести два алгоритма сортировки, которые рассматриваются в главе 2. Для выполнения первого из них, известного как ***сортировка вставкой***, требуется время, которое оценивается как $c_1 n^2$, где n — количество сортируемых элементов, а c_1 — константа, не зависящая от n. Таким образом, время работы этого алгоритма приблизительно пропорционально n^2. Для выполнения второго алгоритма, ***сортировки слиянием***, требуется время, приблизительно равное $c_2 n \lg n$, где $\lg n$ — краткая запись $\log_2 n$, а c_2 — некоторая другая константа, не зависящая от n. Обычно константа метода вставок меньше константы метода слияния, т.е. $c_1 < c_2$. Мы убедимся, что постоянные множители намного меньше влияют на время работы алгоритма, чем множители, зависящие от n. Для двух приведенных методов последние относятся как $\lg n$ к n. Для небольшого количества сортируемых элементов сортировка включением обычно работает быстрее, однако когда n становится достаточно большим, все заметнее проявляется преимущество сортировки слиянием, возникающее благодаря тому, что для больших n

незначительная величина $\lg n$ по сравнению с n полностью компенсирует разницу величин постоянных множителей. Не имеет значения, во сколько раз константа c_1 меньше, чем c_2. С ростом количества сортируемых элементов обязательно будет достигнут переломный момент, когда сортировка слиянием окажется более производительной.

В качестве примера рассмотрим два компьютера — А и Б. Компьютер А более быстрый, и на нем работает алгоритм сортировки вставкой, а компьютер Б более медленный, и на нем работает алгоритм сортировки методом слияния. Оба компьютера должны выполнить сортировку множества, состоящего из миллиона чисел. Предположим, что компьютер А выполняет миллиард инструкций в секунду, а компьютер Б — лишь десять миллионов. Таким образом, компьютер А работает в 100 раз быстрее, чем компьютер Б. Чтобы различие стало еще б*о*льшим, предположим, что код для метода вставок (т.е. для компьютера А) написан самым лучшим в мире программистом с использованием команд процессора, и для сортировки n чисел надо выполнить $2n^2$ команд (т.е. $c_1 = 2$). Сортировка же методом слияния (на компьютере Б) реализована программистом-середнячком с помощью языка высокого уровня. При этом компилятор оказался не слишком эффективным, и в результате получился код, требующий выполнения $50n\lg n$ команд (т.е. $c_2 = 50$). Для сортировки миллиона чисел компьютеру А понадобится

$$\frac{2\cdot\left(10^6\right)^2 \text{ команд}}{10^9 \text{ команд/с}} = 2000 \text{ с},$$

а компьютеру Б —

$$\frac{50\cdot 10^6\cdot \lg 10^6 \text{ команд}}{10^7 \text{ команд/с}} \approx 100 \text{ с}.$$

Как видите, использование кода, время работы которого возрастает медленнее, даже при плохом компиляторе на более медленном компьютере требует на порядок меньше процессорного времени! Если же нужно выполнить сортировку 10 миллионов чисел, то преимущество метода слияния становится еще более очевидным: если для сортировки вставкой потребуется приблизительно 2.3 дня, то для сортировки слиянием — меньше 20 минут. Общее правило таково: чем больше количество сортируемых элементов, тем заметнее преимущество сортировки слиянием.

Алгоритмы и другие технологии

Приведенный выше пример демонстрирует, что алгоритмы, как и аппаратное обеспечение компьютера, представляют собой ***технологию***. Общая производительность системы настолько же зависит от эффективности алгоритма, как и от мощности применяющегося аппаратного обеспечения. В области разработки алгоритмов происходит такое же быстрое развитие, как и в других компьютерных технологиях.

Возникает вопрос, действительно ли так важны алгоритмы, работающие на современных компьютерах, если и так достигнуты выдающиеся успехи в других областях высоких технологий, таких как:

- аппаратное обеспечение с высокими тактовыми частотами, конвейерной обработкой и суперскалярной архитектурой;
- легкодоступные, интуитивно понятные графические интерфейсы (GUI);
- объектно-ориентированные системы;
- локальные и глобальные сети.

Ответ — да, безусловно. Несмотря на то, что иногда встречаются приложения, которые не требуют алгоритмического наполнения (например, некоторые

простые Web-приложения), для большинства приложений определенное алгоритмическое наполнение необходимо. Например, рассмотрим Web-службу, определяющую, как добраться из одного места в другое (во время написания книги существовало несколько таких служб). В основе ее реализации лежит высокопроизводительное аппаратное обеспечение, графический интерфейс пользователя, глобальная сеть и, возможно, объектно-ориентированный подход. Кроме того, для определенных операций, выполняемых данной Web-службой, необходимо использование алгоритмов — например, таких как вычисление квадратных корней (что может потребоваться для определения кратчайшего пути), визуализации карт и интерполяции адресов.

Более того, даже приложение, не требующее алгоритмического наполнения на высоком уровне, сильно зависит от алгоритмов. Ведь известно, что работа приложения зависит от производительности аппаратного обеспечения, а при его разработке применяются разнообразные алгоритмы. Все мы также знаем, что приложение тесно связано с графическим интерфейсом пользователя, а для разработки любого графического интерфейса пользователя требуются алгоритмы. Вспомним приложения, работающие в сети. Чтобы они могли функционировать, необходимо осуществлять маршрутизацию, которая, как уже говорилось, основана на ряде алгоритмов. Чаще всего приложения составляются на языке, отличном от машинного. Их код обрабатывается компилятором или интерпретатором, которые интенсивно используют различные алгоритмы. И таким примерам нет числа.

Кроме того, ввиду постоянного роста вычислительных возможностей компьютеров, они применяются для решения все более сложных задач. Как мы уже убедились на примере сравнительного анализа двух методов сортировки, с ростом сложности решаемой задачи различия в эффективности алгоритмов проявляются все значительнее.

Знание основных алгоритмов и методов их разработки — одна из характеристик, отличающих умелого программиста от новичка. Располагая современными компьютерными технологиями, некоторые задачи можно решить и без основательного знания алгоритмов, однако знания в этой области позволяют достичь намного большего.

Упражнения

1.2-1. Приведите пример приложения, для которого необходимо алгоритмическое наполнение на уровне приложений, и дайте характеристику функций, входящих в состав этих алгоритмов.

1.2-2. Предположим, на одной и той же машине проводится сравнительный анализ реализаций двух алгоритмов сортировки, работающих по методу вставок и по методу слияния. Для сортировки n элементов методом вставок необходимо $8n^2$ шагов, а для сортировки методом слияния — $64n\lg n$ шагов. При каком значении n время сортировки методом вставок превысит время сортировки методом слияния?

1.2-3. При каком минимальном значении n алгоритм, время работы которого определяется формулой $100n^2$, работает быстрее, чем алгоритм, время работы которого выражается как 2^n, если оба алгоритма выполняются на одной и той же машине?

Задачи

1-1. **Сравнение времени работы алгоритмов**

Ниже приведена таблица, строки которой соответствуют различным функциям $f(n)$, а столбцы — значениям времени t. Определите максимальные значения n, для которых задача может быть решена за

время t, если предполагается, что время работы алгоритма, необходимое для решения задачи, равно $f(n)$ микросекунд.

	1 секунда	1 минута	1 час	1 день	1 месяц	1 год	1 век
$\lg n$							
$\sqrt{n}$							
n							
$n \lg n$							
n^2							
n^3							
2^n							
$n!$							

Заключительные замечания

Имеется множество учебников, посвященных общим вопросам, которые имеют отношение к алгоритмам. К ним относятся книги Ахо (Aho), Хопкрофта (Hopcroft) и Ульмана (Ullman) [5,6], Бейза (Baase) и Ван Гельдера (Van Gelder) [26], Брассарда (Brassard) и Бретли (Bratley) [46,47], Гудрича (Goodrich) и Тамазии (Tamassia) [128], Горовица (Horowitz), Сани (Sahni) и Раджисекарана (Rajasekaran) [158], Кингстона (Kingston) [179], Кнута (Knuth) [182,183,185], Козена (Kozen) [193], Манбера (Manber) [210], Мельхорна (Mehlhorn) [217,218,219], Пурдома (Purdom) и Брауна (Brown) [252], Рейнгольда (Reingold), Ньевергельта (Nievergelt) и Део (Deo) [257], Седжевика (Sedgewick) [269], Скьены (Skiena) [280] и Вильфа (Wilf) [315]. Некоторые аспекты разработки алгоритмов, имеющие большую практическую направленность, обсуждаются в книгах Бентли (Bentley) [39,40] и Гоннета (Gonnet) [126]. Обзоры по алгоритмам можно также найти в книгах *Handbook of Theoretical Computer Science, Volume A* [302] и *CRC Handbook on Algorithms and Theory of Computation* [24]. Обзоры алгоритмов, применяющихся в вычислительной биологии, можно найти в учебниках Гасфилда (Gusfield) [136], Певзнера

(Pevzner) [240], Сетубала (Setubal) и Мейдениса (Meidanis) [272], а также Вотермена (Waterman) [309].