
Гуев Тимур @timyrik20
Пользователь
112,7 карма
0,0 рейтинг

Профиль | 22 Публикации | 67 Комментарии | 1,1k Избранное | 48 Подписчики

29 июля 2013 в 00:00

Разработка → Знай сложности алгоритмов

перевод

Алгоритмы*

Эта статья рассказывает о времени выполнения и о расходе памяти большинства алгоритмов используемых в информатике. В прошлом, когда я готовился к прохождению собеседования я потратил много времени исследуя интернет для поиска информации о лучшем, среднем и худшем случае работы алгоритмов поиска и сортировки, чтобы заданный вопрос на собеседовании не поставил меня в тупик. За последние несколько лет я проходил интервью в нескольких стартапах из Силиконовой долины, а также в некоторых крупных компаниях таких как Yahoo, eBay, LinkedIn и Google и каждый раз, когда я готовился к интервью, я подумал: «Почему никто не создал хорошую шпаргалку по асимптотической сложности алгоритмов? ». Чтобы сохранить ваше время я создал такую шпаргалку. Наслаждайтесь!

Хорошо	Приемлемо	Плохо

Поиск

Алгоритм	Структура данных	Временная сложность		Сложность по памяти
		В среднем	В худшем	В худшем
Поиск в глубину (DFS)	Граф с |V| вершинами и |E| ребрами	-	O(|E| + |V|)	O(|V|)
Поиск в ширину (BFS)	Граф с |V| вершинами и |E| ребрами	-	O(|E| + |V|)	O(|V|)
Бинарный поиск	Отсортированный массив из n элементов	O(log(n))	O(log(n))	O(1)
Линейный поиск	Массив	O(n)	O(n)	O(1)
Кратчайшее расстояние по алгоритму Дейкстры используя двоичную кучу как очередь с приоритетом	Граф с |V| вершинами и |E| ребрами	O((|V| + |E|) log |V|)	O((|V| + |E|) log |V|)	O(|V|)
Кратчайшее расстояние по алгоритму Дейкстры используя массив как очередь с приоритетом	Граф с |V| вершинами и |E| ребрами	O(|V|^2)	O(|V|^2)	O(|V|)
Кратчайшее расстояние используя алгоритм Беллмана–Форда	Граф с |V| вершинами и |E| ребрами	O(|V||E|)	O(|V||E|)	O(|V|)

Сортировка

Алгоритм	Структура данных	Временная сложность			Вспомогательные данные
		Лучшее	В среднем	В худшем	В худшем
Быстрая сортировка	Массив	O(n log(n))	O(n log(n))	O(n^2)	O(n)
Сортировка слиянием	Массив	O(n log(n))	O(n log(n))	O(n log(n))	O(n)
Пирамидальная сортировка	Массив	O(n log(n))	O(n log(n))	O(n log(n))	O(1)
Пузырьковая сортировка	Массив	O(n)	O(n^2)	O(n^2)	O(1)
Сортировка вставками	Массив	O(n)	O(n^2)	O(n^2)	O(1)
Сортировка выбором	Массив	O(n^2)	O(n^2)	O(n^2)	O(1)
Блочная сортировка	Массив	O(n+k)	O(n+k)	O(n^2)	O(nk)
Поразрядная сортировка	Массив	O(nk)	O(nk)	O(nk)	O(n+k)

Структуры данных

Структура данных	Временная сложность								Сложность по памяти
	В среднем				В худшем				В худшем
	Индексация	Поиск	Вставка	Удаление	Индексация	Поиск	Вставка	Удаление	
Обычный массив	O(1)	O(n)	-	-	O(1)	O(n)	-	-	O(n)
Динамический массив	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Односвязный список	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Двусвязный список	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Список с пропусками	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Хеш таблица	-	O(1)	O(1)	O(1)	-	O(n)	O(n)	O(n)	O(n)
Бинарное дерево поиска	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Декартово дерево	-	O(log(n))	O(log(n))	O(log(n))	-	O(n)	O(n)	O(n)	O(n)
Б-дерево	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Красно-черное дерево	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Расширяющееся дерево	-	O(log(n))	O(log(n))	O(log(n))	-	O(log(n))	O(log(n))	O(log(n))	O(n)
АВЛ-дерево	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)

Кучи

Куча	Временная сложность						
	Преобразование к куче	Поиск максимума	Извлечение максимума	Увеличить ключ	Вставить	Удалить	Слияние
Связный список (отсортированный)	-	O(1)	O(1)	O(n)	O(n)	O(1)	O(m+n)
Связный список (не отсортированный)	-	O(n)	O(n)	O(1)	O(1)	O(1)	O(1)
Бинарная куча	O(n)	O(1)	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(m+n)
Биномиальная куча	-	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))
Фибоначчева куча	-	O(1)	O(log(n))	O(1)*	O(1)	O(log(n))*	O(1)

Представление графов

Пусть дан граф с |V| вершинами и |E| ребрами, тогда

Способ представления	Память	Добавление вершины	Добавление ребра	Удаление вершины	Удаление ребра	Проверка смежности вершин
Список смежности	O(|E| + |V|)	O(1)	O(1)	O(|E| + |V|)	O(|E|)	O(|V|)
Список инцидентности	O(|E| + |V|)	O(1)	O(1)	O(|E|)	O(|E|)	O(|E|)
Матрица смежности	O(|V|^2)	O(|V|^2)	O(1)	O(|V|^2)	O(1)	O(1)
Матрица инцидентности	O(|V||E|)	O(|V||E|)	O(|V||E|)	O(|V||E|)	O(|V||E|)	O(|E|)

Нотация асимптотического роста

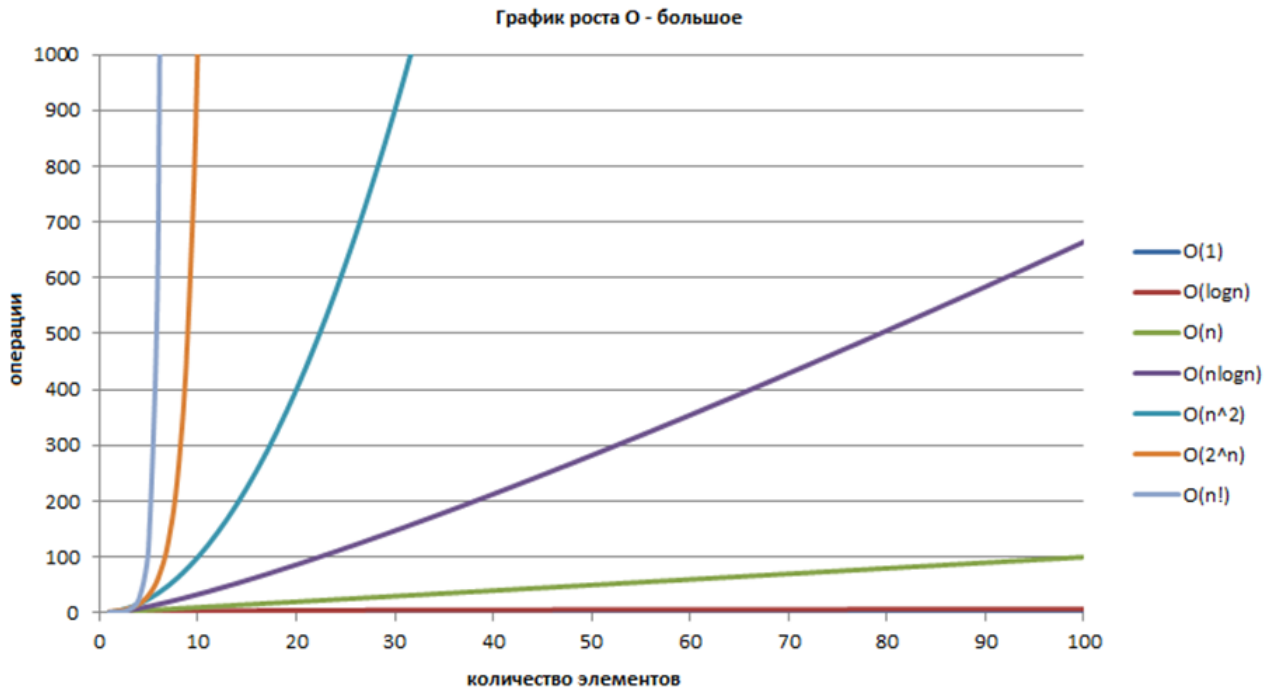
Обозначение	Граница	Рост
(Тета) Θ	Нижняя и верхняя границы, точная оценка	Равно
(О - большое) O	Верхняя граница, точная оценка неизвестна	Меньше или равно
(о - малое) o	Верхняя граница, не точная оценка	Меньше
(Омега - большое) Ω	Нижняя граница, точная оценка неизвестна	Больше или равно
(Омега - малое) ω	Нижняя граница, не точная оценка	Больше

1. (O — большое) — верхняя граница, в то время как (Омега — большое) — нижняя граница. Тета требует как (O — большое), так и (Омега — большое), поэтому она является точной оценкой (она должна быть ограничена как сверху, так и снизу). К примеру, алгоритм требующий Ω (n logn) требует не менее n logn времени, но верхняя граница не известна. Алгоритм требующий Θ (n logn) предпочтительнее потому, что он требует не менее n logn (Ω (n logn)) и не более чем n logn (O(n logn)).
2. f(x)=Θ(g(n)) означает, что f растет так же как и g когда n стремится к бесконечности. Другими словами, скорость роста f(x) асимптотически пропорциональна скорости роста g(n).
3. f(x)=O(g(n)). Здесь темпы роста не быстрее, чем g (n). O большое является наиболее полезной, поскольку представляет наихудший случай.

Короче говоря, если алгоритм имеет сложность __ тогда его эффективность __

Алгоритм	Эффективность
o(n)	< n
O(n)	≤ n
Θ(n)	= n
Ω(n)	≥ n
ω(n)	> n

График роста O — большое



алгоритмы, асимптотический анализ алгоритмов

+280 305k 3080

Перевод: Eric Rowell

Гуев Тимур @timyrik20 карма 112,7 рейтинг 0,0

Похожие публикации

+47 Пессимальные Алгоритмы и Анализ Вычислительной Усложнённости
13,9k 148 15

+20 Несколько советов по эмпирическому анализу алгоритмов
5,4k 72 13

+57 Асимптотический анализ алгоритмов
43,4k 230 81

Самое читаемое

Разработка

Сейчас Сутки Неделя Месяц

+233 Как Skype уязвимости чинил
22,7k 97 131

+19 Улучшение производительности PHP 7
1,9k 22 1

+79 Здравствуй, дорогой Мегафон
8,5k 14 75

+10 Компания Google представила набор тестов Wycheproof
1,1k 6 0

+10 Прототип RFC HTTP-кодов состояния для ошибок разработчиков (диапазон 7XX)
1,1k 4 6

Комментарии (99)

Viktorianec 29 июля 2013 в 00:24 # +78

Теперь я знаю, что можно печатать на обратных страницах тетрадей по информатике или в конце учебников.

petrkozorezov 29 июля 2013 в 00:27 # 0

Спасибо. Простая и полезная статья!

lagranzh 29 июля 2013 в 02:59 +2

чем полезная? проще научиться глядя на алгоритм, говорить какая у него комплексити (сложность?).
имхо, от таблицы один профит — алгоритм в мой голове «на уровне», или можно лучше?

Mrrl 29 июля 2013 в 03:03 +2

> от таблицы один профит — алгоритм в мой голове «на уровне», или можно лучше?

Вам этого мало???

Krypt 29 июля 2013 в 10:55 +13

Эти значения не нужно знать, эти значения нужно уметь считать.

Mrrl 29 июля 2013 в 18:08 +2

Вопрос «можно лучше?» предполагает, что в табличке перечислено больше алгоритмов или структур, чем знает тот, кто спрашивает. Поэтому «посчитать» до того, как он увидит в табличке потенциально лучший алгоритм и разберётся в нём, не получится.

imwode 29 июля 2013 в 07:59 +12

Научите меня? (напишите статью, как научиться в смысле). Прошел два курса, прослушал кучу лекций — я так и не понял как определяется сложность алгоритмов.

rayevg 29 июля 2013 в 08:28 +21

«Алгоритмы — Построение и Анализ» (Кормен, Лейзерсон, Ривест, Штайн). Часть I, главы 1-4.

imwode 29 июля 2013 в 16:33 0

Спасибо!

igor_kleiner 1 октября 2014 в 23:28 −2

Вредный совет вам дали

imwode 2 октября 2014 в 00:16 +1

Ваш, безусловно, значительно полезнее!

igor_kleiner 2 октября 2014 в 15:43 0

Да разумеется, человек съэкономит себе время и не будет тратить его на эту книгу.

Rasifiel 29 июля 2013 в 08:34 +4

Есть две отличных книги с разделами про: CLRS ака Алгоритмы: построение и анализ и Конкретная математика. В обеих книгах хорошие вводные разделы про анализ сложности.

igor_kleiner 1 октября 2014 в 23:29 −1

Очень сложные книги, зачем их советовать,

Rasifiel 2 октября 2014 в 07:59 0

Потому что они хорошо и тщательно описывают анализ сложности, а в CLRS еще и разбираются сложности для базовых структур.

igor_kleiner 2 октября 2014 в 15:44 0

Они сложные для понимания большинства студентов, педагогически для самостоятельной работы они мало приспособлены.

Увы можно было учиться по энциклопедиям будь вы правы.

Rasifiel 2 октября 2014 в 16:00 0

Да ладно. Как раз и Алгоритмы и Конкретная математика вполне применимы как учебник. Просто их надо использовать вдумчиво и тщательно. И задачки там не просто так даны. Я же не предлагаю Искусство программирования, вот оно уже харкорно)

igor_kleiner 2 октября 2014 в 17:05 0

Вы читали главу конкретной математики про производящии функции? И готовы дать ответ на материал в этой главе?

Я уверен что если и читали то большую часть не поняли и ответ не дадите.

Не стоит писать то в чем не разбираетесь.

Это очень сложный учебник, для одаренных студентов или специалистов, но не как не для самообучения

chersanya 2 октября 2014 в 20:11 0

Ну Кормена вполне можно самому изучать, вместе с задачами. С преподавателем, разумеется, получается более эффективно (есть опыт обеих путей).

IIIEB4YK 2 октября 2014 в 22:20 0

Попробуем сделать дискуссию конструктивнее: какие книги порекомендовали бы вы? :)

Mrrl 2 октября 2014 в 23:41 0

Интересные задачки. Выглядят вполне решаемыми, даже если не читать главу перед ними. Правда, трудно судить, насколько для решения надо быть специалистом — или достаточно просто уровня олимпиадника по математике.

Mrrl 4 октября 2014 в 11:44 0

Немного порешал. Мой вывод: для решения не обязательно быть специалистом. Это типичные олимпиадные задачи на отработку конкретного приёма.

Danov 29 июля 2013 в 09:01 +2

«Алгоритмы и структуры данных» Н.Вирт, 360с

kuduh 29 июля 2013 в 10:53 +4

«Алгоритмы. Введение в разработку и анализ» (Ананий Левитин). Советую!

Milfgard 29 июля 2013 в 13:34 0

«Жемчужины программирования» (Бентли).

Krypt 29 июля 2013 в 14:35 0

Как бы в 2х словах объяснить… Допустим, у нас есть некоторый счётчик. При каждой выполненной операции он увеличивается на 1. Сложность алгоритма — это зависимость конечного значения этого счётчика от размера входных данных. Константный множитель не учитывается.

По факту — каждый вложенный цикл, цикл увеличивает сложность в n раз.
Поиск значения в упорядоченном массиве разбиением диапазонов пополам имеет сложность log n: Отрезок длинной n можно поделить пополам log2 n раз. Но так как основание логарифма можно изменить, вынеся константный множитель — основание не указывают.

Самый большой треш, что я писал имел сложность n^11 — подбор 11-ти точек под определённые условия полным перебором — лаба в универе 100 точек обрабатывались за 20-30 секунд.

GrigoryPerepechko 29 июля 2013 в 15:31 +6

Зачем вам статья. Это тривиальный материал который изложен в сотнях учебников. Вы же понимаете что такое циклы?
Вам надо просто напрячь голову, хотя бы минимально. Просто читая каждое предложение в учебнике не идите к следующему пока не поймете текущее.

Занудство
Меня вообще удивляет любовь людей читать книги/лекции/чтоугодно сотнями ни разу не понимая что же там написано.
Неужели не тошно самим от собственного нежелания остановиться немного, не спешить, но зато понять каждую мелочь и деталь которую хотел сказать автор?

VenomBlood 29 июля 2013 в 15:38 0

Это ни разу не тривиальный материал. Вот оцените амортизированную сложность в куче фиббоначи? Или в Van Emde Boas дереве. Даже в динамическом массиве не достаточно посчитать циклы/вызовы методов для того, чтобы дать амортизированную оценку сложности.
Циклы подходят только для грубой оценки сложности сверху в худшем случае.

imwode 29 июля 2013 в 16:27 −1

А меня удивляет любовь людей комментировать направо и налево вместо того, чтобы начать заполнять звенящую пустоту в голове.

imwode 29 июля 2013 в 16:33 *(комментарий был изменён)* –5

Давай-ка навскидку, не заглядывая никуда, сообщи нам сложности:
5n
3n^2+2n−100
10log(n)+5n
10log(n)+5n^2
3n^3−2000n^2
2n^2
50n+nlog(n)
1000+2000000
2n+n^2
logn+1000

А также:

```
def recurPowerNew(a, b):
   print a, b
   if b == 0:
      return 1
   elif b%2 == 0:
      return recurPowerNew(a*a, b/2)
   else:
      return a * recurPowerNew(a, b-1)
```

И еще:

```
def unionNew(L1, L2):
   '''
   L1 & L2 are lists of the same length, n
   '''
   temp = []
   for e1 in L1:
      flag = False
      for e2 in L2:
         if e1 == e2:
            flag = True
            break
      if not flag:
         temp.append(e1)
   return temp + L2
```

Ну и вот:

```
def isIn(a, s):
   '''
   a is a character, or, singleton string.
   s is a string, sorted in alphabetical order.
   '''
   if len(s) == 0:
      return False
   elif len(s) == 1:
      return a == s
   else:
      test = s[len(s)/2]
      if test == a:
         return True
      elif a < test:
         return isIn(a, s[:len(s)/2])
      else:
         return isIn(a, s[len(s)/2+1:])
```

Эти задания — элементарные. Для тех, кто впервые коснулся вопроса.

chersanya 29 июля 2013 в 18:17 *(комментарий был изменён)* +1

Что вы имеете в виду под «сообщить сложность 5n» (например)? Если алгоритм выполняет 5n шагов, то и сложность его будет как раз 5n. Так-то конечно можно догадаться, что вы спрашиваете скорее всего про точную оценку (которая обозначается большой буквой тета) с учётом только старших степеней и без константы. Иначе можно много всего напридумывать, подходящего под вопрос, скажем 5n = O(n! — 3n^2).

imwode 30 июля 2013 в 17:48 0

Ступил. Задание на самом деле не указать сложность, а выбрать из списка наиболее близкую:
O(1),O(log(n)),O(n),O(nlog(n)),O(nc) or O(cn).

kriokamera 31 июля 2013 в 10:23 +2

Чтобы посчитать сложность, надо описать количество операций для входных данных размера n и посмотреть на асимптотику на бесконечности. Алгоритм линеен — значит, ему на каждый бит входных данных потребуется (примерно) пропорциональное количество операций. Пример нелинейного алгоритма — сортировка пузырьком. Для каждого из n элементов потребуется в среднем n/2 перестановок (это реальные цифры), в результате массив из n чисел мы отсортируем за n^2 операций.
Чем это плохо: большие массивы становится отсортировать все труднее и труднее. Массив из всего 1000 чисел будет сортироваться миллион операций. Дальше — больше. Это сложность O(n^2). Математическое значение записи «f(x)=O(g(x)) при x->a» представляет собой «существует конечный предел отношения f(x)/g(x) при x->a». То есть f примерно пропорциональна g при x, близких к a.
Гораздо лучше квадратичных алгоритмов линейные — O(n). Радиксная (поразрядная) сортировка отсортирует ваш массив n интов за примерно kn (где k — постоянная) операций процессора. У меня получалось k~4. Это уже зависит от компьютера и от реализации. Главное — что в теории сортировка требует пропорциональное количество операций количеству входных данных. Милионный массив она отсортирует за, скажем, пять миллионов операций перестановок. В это же время сортировка, работающая за квадрат (то есть O(n^2)), например, пузырек, будет сортировать его миллион миллионов операций, что может занять ну очень много времени. Примерно в двести тысяч раз дольше, чем радикс в выбранных нами условиях. Дальше отрыв становится все больше.
Надеюсь, что-нибудь было полезно.

chersanya 31 июля 2013 в 14:22 0

Раз уж стали приводить математические определения, то давайте делать это правильно :) А по определению математическое обозначение f(x)=O(g(x)) обозначает, что f(x) < C*g(x) в некоторой области, где C — некоторая константа.

Dim0FF 31 июля 2013 в 21:46 −1

f(x)=O(g(x)): f(x) **<=** C*g(x)
f(x)=o(g(x)): f(x) **<** C*g(x)

VenomBlood 31 июля 2013 в 21:50 0

В вашем описании нет различия между O большое и o малое.

chersanya 31 июля 2013 в 21:50 0

Нет, вы абсолютно не правы. Очевидно же, что ваше и моё определение O(..) совпадают! А следовательно, вводить ещё такое же o(..) смысла нет — на самом деле у него другое математическое определение: f=o(g), если lim f/g = 0 (при x -> куда-то).

Dim0FF 31 июля 2013 в 21:53 0

> Очевидно же, что ваше и моё определение O(..) совпадают!

Мне не очевидно.

chersanya 31 июля 2013 в 22:55 0

Тогда вот доказательство эквивалентности по шагам: в одну сторону — $[f(x)<C\cdot g(x)\to f(x)\le C\cdot g(x)]\to[(\exists C{:}f(x)<C\cdot g(x))\to(\exists C{:}f(x)\le C\cdot g(x))]$, в обратную — $[f(x)\le C\cdot g(x)\to f(x)<2C\cdot g(x)]\to[(\exists C_1{:}f(x)\le C_1\cdot g(x))\to(\exists C_2=2C_1{:}f(x)<C_2\cdot g(x))]$.
Как ещё более подробно написать, я не знаю.

Dim0FF 31 июля 2013 в 23:44 0

Я вижу различия между определениями O:
$$\exists(C>0),n_0:\forall(n>n_0)\ |f(n)|\le|Cg(n)|$$

и o:
$$\forall(C>0),\exists n_0:\forall(n>n_0)\ |f(n)|<|Cg(n)|$$

Если в случае O существует такой коэффициент C, что выполняется условие, то в случае o оно выполняется для любого C.

Почему в моём случае определения совпадают до сих пор не понятно.

chersanya 1 августа 2013 в 00:06 0

В этом комментарии определения верные и не совпадают, но в предыдущий раз, а именно

> f(x)=O(g(x)): f(x) <= C*g(x)
> f(x)=o(g(x)): f(x) < C*g(x)

про разные кванторы перед C не говорилось, и те два определения равносильны.

kriokamera 1 августа 2013 в 03:57 0

f(x)=o(g(x)) по базе B <=> f(x)/g(x) -> 0 по базе B.

kriokamera 1 августа 2013 в 03:55 *(комментарий был изменён)* 0

Ваше определение эквивалентно моему, когда функции непрерывны, а g не бывает нулем.

И да, o-обозначения бессмысленны без указания базы предела, чего у вас явно не хватает.

chersanya 1 августа 2013 в 13:46 0

Во-первых здесь всё-таки речь о теории сложности, и там обычно используются o, O и т.п. обозначения подразумевая поведение на бесконечности, поэтому (как по мне) можно в таких случаях упускать базу и всем будет понятно. А вообще, я таки указал, что

> … f(x) < C*g(x) в **некоторой области** ...

.

По поводу равносильности определений — какие именно вы имеете в виду? Если ваше

> Математическое значение записи «f(x)=O(g(x)) при x->a» представляет собой «существует конечный предел отношения f(x)/g(x) при x->a».

и моё

> математическое обозначение f(x)=O(g(x)) обозначает, что f(x) < C*g(x) в некоторой области, где C — некоторая константа

, то они не равносильны. Например, sin(n)=O(1) (при n->inf), но по вашему определению это не подходит.

Mrrl 29 июля 2013 в 23:30 0

Последнее задание совсем неочевидно. Для него надо знать, как конкретно выполняется операция s[:len(s)/2] — происходит ли копирование фрагмента строки, или создаётся новая ссылка внутрь содержимого строки s.

imwode 30 июля 2013 в 17:47 −1

ну так мож и так написать: O(log(len(s))) ?? :-)

Mrrl 30 июля 2013 в 18:32 +1

Так ведь если строчка копируется, то ответом будет O(len(s)). Так что это вопрос не на сложность алгоритма, а на знание конкретной реализации языка.

imwode 30 июля 2013 в 20:23 0

круть.
может я тоже. когда-нибудь. смогу так.

Velitsky 29 июля 2013 в 15:57 0

Согласен с предыдущими комментариями. Кормен и Конкретная математика — очень хорошие книги, мне нравятся больше Вирта, но он, так сказать, один из столпов.
Вообще для начала хватит одной книги и я из них бы рекомендовал Кормена — хорошо и при этом достаточно доступно объясняет.

petrkozorezov 29 июля 2013 в 09:58 0

Профит в том, что сразу и наглядно видна общая картина алгоритмов, и это не исключает умение определять сложность на глаз.

VenomBlood 29 июля 2013 в 00:38 +16

Я понимаю что перевод, но достаточно много неточностей/недоговорок. Например со временем вставки — не понятно почему рассматривается только вставка в начало, достаточно редкая операция, обычно или вставка в общем случае или добавление в конец, а у них временная сложность другая.

Плюс — дается понятие Тета-нотации, но нигде в таблицах она не используется. Звездочкой, как я понял — обозначена амортизированная сложность, об этом тоже нигде не сказано. Где-то указано лучшее/среднее/худшее время, а где-то только одно время (надо понимать — среднее, и амортизированное — если со звездочкой).
В табличке по памяти для QuickSort видимо ошибка, т.к. там видимо должно быть log(n), раз даже цвет желтый, ну и там вообще раскраска странная в этом столбце.

С цветовой раскраской по графам в целом не согласен, т.к. это сильно зависит от типа графа, особенно разница между O(|E|) и O(|V|) и между O(|E||V|) и O(|V|^2), O(|E|^2)
Ну и так далее.

Обосную недовольство: в принципе эта табличка приведет только к тому, что заучившие ее люди будут вместо первичного собеседования отсеиваться на последующих. Т.к. заучивание всей таблички никому не нужно. Плюс ко всему важно понимание того, что стоит за сложностью каждого алгоритма, в каких данных этот алгоритм себя показывает хорошо, в каких не очень ит.д. В текущей таблица между деревьями поиска вообще различий не видно.

Видимо все это и есть причина, отвечающая на первый вопрос статьи: «Почему никто не создал хорошую шпаргалку по асимптотической сложности алгоритмов?».

SowingSadness 29 июля 2013 в 00:55 +4

> Обосную недовольство: в принципе эта табличка приведет только к тому, что заучившие ее люди будут вместо первичного собеседования отсеиваться на последующих.

Можно узнать, где это так людей отсеивают, которые на зубок не знают какой алгоритм, какую сложность имеет?

VenomBlood 29 июля 2013 в 00:59 *(комментарий был изменён)* +1

Я как раз говорил об обратном, не нужно знать на зубок какой алгоритм какую сложность имеет. Нужно иметь базовое представление об алгоритмах. Мне сложно представить где на собеседовании могут спросить про кучу фиббоначи. Но вот, например, про quicksort могут, т.к. он очень распространен, а эта табличка дает очень обрезанную и неверную картину касательно этого quicksort (и большинства остального).
А когда человек заучит табличку и будет на зубок знать эту мнимую сложность алгоритмов — как раз это знание на 95% бесполезно. И если в первичном тесте может попасться вопрос «напишите оценки сложности 2-3 известных вам алгоритмов» (обычно речь идет или об очень распространенных алгоритмах или собеседуемому предоставляется возможность самому выбрать алгоритмы для детального разговора), то далее будет детальный разбор тех алгоритмов которые собеседуемый написал — и тут он, заучив только эту табличку, провалится.

Mrrl 29 июля 2013 в 02:53 +1

А если подойти не со стороны собеседования, а с реальной работы? Допустим, человек ещё не очень свободно ориентируется в множестве существующих алгоритмов. Тогда для конкретной задачи ему было бы неплохо заглянуть в табличку, посмотреть, какой из указанных там алгоритмов даёт лучшие результаты (для конкретных условий), и либо удовлетворённо заметить, что тот алгоритм, о котором он думал изначально, действительно лучше всех, либо разобраться с тем, что предложит табличка. Возможно, ему повезёт, и найдётся действительно подходящий и эффективный алгоритм.
Думаю, что толк от таблички есть. Пойду посмотрю подробнее, что это за фибоначчева куча (на первый взгляд она на меня впечатления не произвела).

VenomBlood 29 июля 2013 в 03:21 0

Тогда эта табличка должны иметь десяток измерений и миллион ячеек, чтобы быть хоть сколько полезной. Иначе сказать «лучше всех» по табличке не выйдет (да и с миллионом ячеек думаю что не выйдет на сколько нибудь реальной задаче), и она будет только во вред.

Mrrl 29 июля 2013 в 03:26 0

В конечном итоге она такой и станет (или превратится в дерево табличек, или в программу по выбору оптимального алгоритма). Но принцип «Доверяй, но проверяй» никто не отменял.

Mrrl 29 июля 2013 в 02:39 +1

В худшем случае у Quicksort дополнительная память действительно O(n). Это тот случай, когда тот, кто реализовывал алгоритм, не догадался сравнить длину кусков массива и рекурсивно вызвать сортировку только для короткого куска, а честно написал два рекурсивных вызова.

А вот память в поразрядной сортировке можно ограничить (числом значений разряда)*(количество разрядов): если сортировать начиная со старших разрядов, на каждом шагу сначала посчитать статистику значений каждой цифры, а потом положить каждый объект сразу на место (за O(n)). И рекурсивно вызвать сортировку для следующего разряда.

VenomBlood 29 июля 2013 в 03:19 0

А еще можно реализовать не in-place, копировать на каждой итерации и получить до n^2 памяти. Речь то о нормальной реализации, смысл рассматривать наивные реализации, если они хуже?

Mrrl 29 июля 2013 в 03:21 0

Да, согласен.

Roman_Pekhov 29 июля 2013 в 06:43 0

Не согласен. Вариант с двумя рекурсиями может оказаться предпочтительным, потому что код проще. Думаю что, если бы этот вариант был хуже однозначно и всегда, его бы вовсе не упоминали в описаниях алгоритма.

GrigoryPerepechko 29 июля 2013 в 15:34 0

> не понятно почему рассматривается только вставка в начало, достаточно редкая операция, обычно или вставка в общем случае или добавление в конец, а у них временная сложность другая.

Что значит «вставка в общем случае», и почему у неё временная сложность отличается от вставки в начало?

VenomBlood 29 июля 2013 в 15:36 0

Обычно рассматривают вставку в начало, в конец и в произвольное место. Если мы говорим об амортизированном времени, в списке в начало и в конец — Theta(1), в произвольное место — Theta(n), а для динамического массива в начало и в произвольное место — Theta(n), т.к. нужно сдвигать, а в конец — Theta(1).

GrigoryPerepechko 29 июля 2013 в 15:43 0

Понял вашу идею. Согласен, лучше 3 характеристика показывать.

Единственное смутило, если список — это Linked List, то тогда вставка стоит
* Начало — O(1)
* Конец/Произвольное место — O(1) + O(n) для поиска

VenomBlood 29 июля 2013 в 15:48 0

Можно хранить доп. указатель на конец, тогда будет в конец тоже Theta(1).

Вообще 3х характеристик мало. Нужно всегда смотреть на то, какие данные обрабатываются, возможно из ихней специфики (если они не случайны) можно извлечь большое ускорение.

eresik 29 июля 2013 в 01:58 *(комментарий был изменён)* 0

С кучами не работаю, но навскидку непонятны первые две строки в таблице с информацией о кучах.
А именно, почему временная сложность в случае отсортированного списка больше чем в случае неотсортированного (увеличить ключ, вставить ключ).
Как может быть отсортированная структура данных «хуже» неотсортированной (знаю что такое иногда бывает, но вряд ли в данном случае)

VenomBlood 29 июля 2013 в 02:06 0

Имеется ввиду не что входные данные отсортированы, а что структура данных всегда поддерживается в отсортированном состоянии, из за этого требования и увеличиваются показатели времени.

IIaDoHaK 29 июля 2013 в 02:20 0

Ого! Вот это спасибо! Распечатаю и повешу на стену!

gasya 29 июля 2013 в 02:36 +15

«Почему никто не создал хорошую шпаргалку по асимптотической сложности алгоритмов? »

Потому, что толку от такой таблички — ноль.

Sayonji 29 июля 2013 в 05:02 +2

А можно для незнающих пояснения к дополнительным данным в сортировках, пожалуйста? Особенно где O(1) получается.

chersanya 29 июля 2013 в 06:22 0

Просто те алгоритмы, где O(1) в этом столбце, используют некоторое константное значение памяти для сортировки, независимо от размера массива (собственно, именно это и написано).

Sayonji 29 июля 2013 в 08:02 0

Спасибо, я неправильно понял смысл слов «вспомогательные данные». Подумал, что говорится об ускорении сортировок засчет чего-то заранее известного. Что-то в роде поразрядной сортировки за N, хотя быстрее NlogN изначально невозможно. Сбился, наверное, из-за того, что в предыдущей таблице это еще было подписано памятью.

Mrrl 29 июля 2013 в 06:31 +1

Например, для сортировки пузырьком вам кроме исходного массива понадобится ещё 3-4 переменные. Их число не зависит от того, массив какого размера сортируется, поэтому дополнительная память, которую они занимают, считается равной O(1). На самом деле это неправда, потому что число битов в представлении индекса растёт как логарифм от длины массива, но все предпочитают работать в модели, где индекс занимает одну ячейку памяти.
В случае быстрой сортировки у нас идут рекурсивные вызовы. В худшем случае, их глубина будет равна двоичному логарифму длины массива, а каждый вызов захватывает на стеке свой набор переменных. Так что общий размер дополнительной памяти — O(log(N)).
Для классической сортировки слиянием нам нужен второй массив, куда мы будем складывать результат слияния отсортированных половинок

исходного массива. Можно написать алгоритм так, что дополнительной памяти нужно вдвое меньше, чем размер исходного массива, тем не менее, это O(N). Рекурсия в этом алгоритме не обязательна, но если бы она и была, то много памяти бы не съела.

MrEsp 29 июля 2013 в 11:24 +1

> В случае быстрой сортировки у нас идут рекурсивные вызовы. В худшем случае, их глубина будет равна двоичному логарифму длины массива, а каждый вызов захватывает на стеке свой набор переменных. Так что общий размер дополнительной памяти — O(log(N)).

Уверены?

Mrrl 29 июля 2013 в 18:20 0

Что можно добиться O(log(N)) — уверен. Следующий уровень рекурсии идёт только для сортировки меньшего из кусков, на которые разделился массив. Его длина меньше половины исходного массива, так что глубина рекурсии не больше O(log(N)). Сортировка оставшегося большего куска организуется с помощью цикла.
Можно ли обойтись без рекурсии и без явно захваченного стека индексов (т.е. используя память O(1)), не уверен. Наверное, можно что-нибудь придумать, но это, скорее всего, будет дольше.

MrEsp 1 августа 2013 в 10:55 0

Да, правильно. а на счет O(1)- ну это уже heapsort получается какой-то. Без кардинальных изменений алгоритма такая магия не сработает.

Mrrl 1 августа 2013 в 17:57 0

С Mergesort работает (но теряется устойчивость и немного увеличивается время)

MuLLtiQ 1 августа 2013 в 16:08 0

> Сортировка оставшегося большего куска организуется с помощью цикла.

Можно поподробнее

Mrrl 1 августа 2013 в 18:03 *(комментарий был изменён)* +1

Если в двух словах:

```
void qsort(T *arr,int len){
  while(len>1){
      int medIndex=split(arr,len);
      if(2*medIndex>=len){
        qsort(arr+(medIndex+1),len-(medIndex+1));
        len=medIndex;
      }else{
         qsort(arr,medIndex);
         arr+=(medIndex+1);
         len-=(medIndex+1);
      }
  }
}
```

Здесь split() выполняет один шаг быстрой сортировки и возвращает индекс, на который встал разделяющий элемент.

MuLLtiQ 1 августа 2013 в 19:07 0

Выигрыш в том, что нет второго рекурсивного вызова (для «длинной» части)? Ну да, стек вызовов будет гораздо короче.

chersanya 29 июля 2013 в 06:21 +3

Не совсем понятно, для кого и для чего эти таблицы.

Для начинающих программистов, использовать для выбора оптимального алгоритма? Но тогда зачем там например достаточно экзотические Фибоначчиевы пирамиды, и сравнение алгоритмов сортировки (всё равно практически всегда используется функция из библиотеки языка)?

Для изучения теории алгоритмов, асимптотической сложности? Но ведь там куча ошибок и неточностей, даже в определениях всяких о малых и прочих (и в основных таблицах тоже).

aml 29 июля 2013 в 08:17 −2

Это уже третье определение o-малого, которое я вижу. Обычно под O-большим понимается любая оценка сложности сверху. Например:

2*n^2 = O(n^2)
2*n = O(n^2)

это две корректные O-оценки. Тогда как o-малое — это асимптотически верная оценка сверху. Т.е.:

2*n^2 = o(n^2) < — корректная
2*n = o(n^2) < — некорректная

Аналогично с омегами, но снизу.

Википедия даёт другое определение o-малого. Вы — третье. Перед тем, как использовать o-нотацию в своих публикациях или на собеседованиях, лучше всего сначала уточнить определение, которое будет использоваться.

Dim0FF 29 июля 2013 в 09:39 +5

> We can draw an analogy between the asymptotic comparison of two functions f and g and the comparison of two real numbers a and b:
> $f(n) = O(g(n))$ is like $a \leq b$,
> $f(n) = \Omega(g(n))$ is like $a \geq b$,
> $f(n) = \Theta(g(n))$ is like $a = b$,
> $f(n) = o(g(n))$ is like $a < b$,
> $f(n) = \omega(g(n))$ is like $a > b$.

> Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein — Introduction to Algorithms, Third Edition

VenomBlood 29 июля 2013 в 14:45 0

О нотация с небольшими незначимыми нюансами — одна. И то что вы написали — это что-то странное.

> 2*n^2 = o(n^2) < — корректная
> 2*n = o(n^2) < — некорректная

Это с точностью до наоборот, первое — некорректное, второе — корректное.

aml 29 июля 2013 в 15:40 +1

Виноват, вы правы.

kriokamera 31 июля 2013 в 10:30 +1

Определения — подгонка некого осязаемого математического смысла в буквы. Их можно написать миллионом способов, и зависеть что-то будет очень мало. У каждого лектора свои формулировки теорем и определений.

pushist1y 29 июля 2013 в 11:17 *(комментарий был изменён)* +1

bigocheatsheet.com/
не заметил вовремя, что статья — перевод

spiff 29 июля 2013 в 12:13 −2

А я вот не понимаю зачем переводить название алгоритмов и структур. Я не стараюсь сколько-нибудь приубавить ценности стати, но честно не понимаю мотивацию. Меня все эти «сортировка слиянием» (а в Кормене «пирамидальная сортировка») только путают. Давно просто merge не перевожу как «слияние» перевожу как «мержить», поэтому долго сначала не мог понять, что за сортировка такая о которой я ничего не слышал :)

spiff 29 июля 2013 в 12:18 +1

Имел ввиду, что в Кормене есть отличный пример «трудностей перевода» — «пирамидальная сортировка» она же «heap sort». Лучше не переводить такие вещи. ИМХО.

Roman_Pekhov 29 июля 2013 в 12:26 −3

Во так мало-помалу все русские слова выйдут из употребления. Чтобы не иметь трудностей перевода. Вас устроит, если все слова станут не требующими перевода?

malan 29 июля 2013 в 12:40 +3

На последней диаграмме я вижу 6 графиков, а в легенде 7. Это потому что O(1) и O(log n) сливаются?

PavloG 29 июля 2013 в 13:37 0

На графике отношение осей 1/10
И линейна функция поэтому смотрится странно (как и все другое)
Не удачный график как по мне.
И да скорее всего сливается.

evgeny_boger 29 июля 2013 в 16:31 0

У вас рёбра и вершины в первой таблице перепутаны. E — количество рёбер, V — вершин.

Aux 29 июля 2013 в 21:27 −2

Зачем это знать прикладнику, если все алгоритмы написали за него?

hell0w0rd 29 июля 2013 в 22:39 +3

видимо чтобы выбирать какой алгоритм взять?
Хотя на мой взгляд от простого знания эффективности алгоритма — в мозгу не прибавится. Вот если знаешь как действительно алгоритм работает, и что в конкретной ситуации можно сделать чтобы его оптимизировать — вот это действительно нужные знания

MrEsp 1 августа 2013 в 10:57 +1

Какие «все алгоритмы» написали? Проблем, для которых нет эффективных алгоритмов предостаточно.

alexnikleo 10 сентября 2013 в 00:06 −1

Не очень корректные обозначения сложности цветами. Все-таки, O(1) у Фибоначчиевой кучи на всех адекватного размера данных, это далеко не «хорошо».

Quilin 6 декабря 2013 в 09:57 0

Вы хотите еще отметку шкалы «ОФИГЕННО»?

alexnikleo 6 декабря 2013 в 18:10 −1

Я о другом. У Фибоначчиевой кучи хоть и асимптотика O(1), но очень большая константа, поэтому лучше использовать другие структуры данных, пусть с худшей асимптотикой, но с лучшим временем работы на практике.

andy_p 4 октября 2014 в 12:41 0

На самом деле, не все так просто.
В реальной жизни большую роль играет константа перед o-большим.
Поэтому надо различать эффективность алгоритма и его масштабируемость.