# درک الگوریتم

## راهنمای تصویری برای برنامه‌نویس‌ها و افراد کنجکاو

آدیتیا بهارگاوا

مترجم: مهران افشارنادری

عنوان و نام پدیدآور:درک الگوریتم (راهنمای تصویری برای برنامه‌نویس‌ها و افراد کنجکاو)/ آدیتیا بهارگاوا ؛
مترجم: مهران افشارنادری
مشخصات نشر: تهران، ۱۴۰۲-
مشخصات ظاهری: دیجیتال، ۲۸۸ ص، مصور
موضوع: الگوریتم
صفحه‌آرایی: نفیسه عطاران

# درک الگوریتم

## راهنمای تصویری
## برای برنامه‌نویس‌ها
## و افراد کنجکاو

آدیتیا بهارگاوا

مترجم: مهران افشارنادری

تقدیم به پدر و مادرم، یُگِش و سَنگینا

قدردانی مترجم :
سپاس از جادی عزیز که کتاب را پیش از انتشار خواند و پیشنهادهایی در جهت بهتر شدن کیفیت ترجمه داد. با این حال هرگونه اشکال در متن نهایی متوجه مترجم است.

# فهرست

## پیشگفتار

در ابتدا، برنامه‌نویسی برای من تنها یک سرگرمی بود. مبانی برنامه‌نویسی را از کتاب Visual Basic 6 for Dummies آموختم و برای فهم بیشتر به خواندن کتاب‌های دیگر ادامه دادم. اما موضوع الگوریتم سخت و غیر قابل درک باقی ماند. به‌خاطر دارم که فهرست مطالب اولین کتاب الگوریتمی را که داشتم، مزه‌مزه می‌کردم و به این فکر می‌کردم که «بالاخره این موضوعات را درک خواهم کرد!» اما مطالب سنگین بود و پس از چند هفته منصرف شدم. تا بعد اولین استاد الگوریتم خوبم را پیدا کردم و فهمیدم این ایده‌ها چقدر ساده و زیبا هستند.

چند سال پیش، اولین پست تصویری وبلاگم را نوشتم. اصولاً موضوعات را در قالب دیداری بهتر یاد می‌گیرم و سبک مصور را خیلی دوست دارم. از آن زمان، چند پست مصور در مورد برنامه نویسی فانکشنال[1]، گیت[2]، یادگیری ماشین[3] و هم‌زمانی[4] نوشتم. راستی زمانی که شروع به کار کردم نویسنده‌ی متوسطی بودم. توضیح مفاهیم فنی سخت است و ارائه‌ی مثال‌های خوب و توضیح یک مفهومِ دشوار، زمان‌بر. بنابراین راحت‌ترین کار نادیده‌گرفتن مفاهیم سخت است. فکر می‌کردم چقدر کارم را خوب انجام می‌دهم، ولی بعد از محبوبیت یکی از پست‌هایم، یکی از همکارانم گفت: «پست تو را خواندم ولی چیزی سر در نیاوردم.» فهمیدم هنوز راه درازی در مسیر یادگیری نویسندگی در پیش دارم.

در میانه‌ی نوشتن این پست‌های وبلاگ، انتشارات منینگ به سراغ من آمد و از من پرسید که تمایلی دارم که یک کتاب مصور بنویسم؟ خوب، معلوم شد که ویراستاران منینگ در مورد توضیح مفاهیم فنی چیزهای زیادی می‌دانند و به من آموختند که چگونه تدریس کنم. این کتاب را نوشتم تا این وسوسه را عملی کنم: می‌خواستم کتابی

1. functional programming
2. git
3. machine learning
4. concurrency

بنویسم که موضوعات فنی سخت را به خوبی توضیح بدهد، و کتاب الگوریتمی می‌خواستم بنویسم که آسان‌خوان باشد. نوشته‌های من با اولین پست وبلاگ فاصله‌ی زیادی دارد، و امیدوارم این کتاب برای شما آسان و آموزنده باشد.

## قدردانی

از انتشارات منینگ تشکر می کنم که فرصت نوشتن این کتاب را در اختیار من قرار داد و دستم را برای خلاقیت باز گذاشت. از ناشر کتاب مرجان بیس و مایک استفنز برای همراهی، برت بیتس برای آموزش نویسندگی، و جنیفر استات چون ویراستاری فوق‌العاده پاسخگو و یاری‌رسان بود، تشکر می‌کنم. همچنین از افراد تیم تولید منینگ: کوین سالیوان، مری پیرگیز، تیفانی تیلور، لزلی هایمز، و بقیه‌ی افراد دست‌اندرکار متشکرم. علاوه بر این، می‌خواهم از کسانی که دست‌نوشته‌ی کتاب را خواندند و پیشنهاداتی ارائه کردند تشکر کنم: کارن بنسدون، راب گرین، مایکل همرا، اوزن هارلوویچ، کالین هستی، کریستوفر هاپت، چاک هندرسون، پاول کوزلوفسکی، آمیت لامبا، ژان فرانسوا مورین، رابرت موریسون، سانکار. راماناتان، ساندر راسل، داگ اسپارلینگ و دیمین وایت.

با تشکر از افرادی که به من کمک کردند تا به این نقطه برسم: رفقای Flaskhit game board که به من کدنویسی را یاد دادند؛ دوستان زیادی از جمله بن سرکه، کارل پوزون، الکس منینگ، استر چان، آنیش بات، مایکل گلس، نیکراد مهدی، چارلز لی، جرد فریدمن، هما مانیکاواساگام، هاری راجا، مورالی گودیپاتی، سرینیواس وارادان و دیگران که با مرور فصل‌ها، مشاوره دادند و فرصتی برای محک‌زدن اَشکال گوناگون ارائه‌ی توضیحات را برایم فراهم کردند و سپاس از گری بردی، برای آموزش الگوریتم‌ها. یک تشکر اساسی دیگر از اساتید CLRS[1]، کنوت و استرنگ. به‌راستی که روی شانه‌های غول‌ها ایستاده‌ام.

بابا، مامان، پریانکا و بقیه‌ی اعضای خانواده: از حمایت همیشگی شما متشکرم. و یک تشکر بزرگ از همسرم مگی. ماجراهای زیادی در پیش روی ما وجود دارد، که البته محدود به آخر هفته‌ها در خانه ماندن و بازنویسی پاراگراف‌ها نمی‌شود.

در نهایت، سپاس فراوان از خوانندگانی که کتاب را مطالعه کردند و آنانی که به من در انجمن این کتاب بازخورد دادند تشکر می کنم. شما واقعاً به بهتر شدن این کتاب کمک کردید.

۱. توماس اچ کورمن چارلز ای لایسرسان رونالد ریوست کلیفورد استین نویسندگان کتاب مقدمه‌ای بر الگوریتم.

## درباره‌ی این کتاب

این کتاب به گونه‌ای طراحی شده که آسان‌خوان باشد. از جهش‌های فکری بزرگ اجتناب کرده‌ام. هر زمان که مفهوم جدیدی مطرح می‌شود، آن را توضیح می‌دهم یا به شما می‌گویم که چه زمانی آن را توضیح خواهم داد. مفاهیم اصلی با تمرین‌ها و توضیحات متعدد تقویت می‌شوند تا بتوانید مفروضات خود را بررسی کنید و مطمئن شوید که موضوعات را متوجه شده‌اید.

هر موضوع را با مثال شرح می‌دهم. به جای نامفهوم‌نویسی هدف من این است که برای شما تجسم این مفاهیم را آسان کنم. از طرفی فکر می‌کنم با یادآوری آن‌چه از قبل می‌دانیم، یادگیری بهتری خواهیم داشت، و مثال‌ها یادآوری را آسان‌تر می‌کنند. بنابراین وقتی سعی می‌کنید تفاوت بین آرایه‌ها و لیست‌های پیوندی را به خاطر بیاورید(فصل ۲)، می‌توانید صرفا به نشستن افراد در یک سالن برای تماشای فیلم فکر کنید. هرچند شاید بدیهی به نظر برسد، خودم مطالب را به شکل تصویری بهتر یاد می‌گیرم و این کتاب هم سرشار از تصاویر است.

مطالب کتاب به دقت تنظیم شده است. نیازی به نوشتن کتابی نیست که همه‌ی الگوریتم‌های مرتب‌سازی را پوشش‌بدهد - به همین دلیل است که ما ویکی‌پدیا و خان‌آکادمی را داریم. تمام الگوریتم‌هایی که در این کتاب گنجانده‌ام کاربردی هستند و من از آن‌ها در شغلم به عنوان یک مهندس نرم‌افزار استفاده می‌کنم. این‌ها پایه‌ی خوبی برای موضوعات پیچیده‌تر فراهم می‌کنند.

به سلامتی خواندن!

## نقشه‌ی راه

در سه فصل اول به بنیان الگویتم می‌پردازیم:

- **فصل ۱**- اولین الگوریتم عملی خود را خواهید آموخت: جست‌وجوی دودویی[1]. همچنین یاد می‌گیرید که سرعت یک الگوریتم را با استفاده از نماد O بزرگ[2] تجزیه و تحلیل کنید. نماد O بزرگ در سراسر کتاب برای تجزیه و تحلیل سرعت یک الگوریتم استفاده شده است.
- **فصل ۲**- با دو ساختمان‌داده‌ی اساسی آشنا خواهید شد: آرایه و لیست پیوندی. این ساختمان‌داده‌ها در سراسر کتاب مورد استفاده قرار می‌گیرند، و از آنها برای ساختن ساختمان‌داده‌های پیشرفته‌تر مانند جدول‌های هَش[3] استفاده می‌شود(فصل ۵).
- **فصل ۳**- در مورد بازگشت[4]، یک تکنیک مفید که توسط بسیاری از الگوریتم‌ها استفاده می‌شود (مانند مرتب‌سازی سریع[5]، که در فصل ۴ پوشش داده شده است) می‌آموزید.

به تجربه‌ی من، نماد O بزرگ و تابع‌های بازگشتی موضوعاتی چالش‌برانگیز برای مبتدیان است. بنابراین در این فصل با سرعتی آهسته زمان بیشتری را به این بخش‌ها اختصاص داده‌ام.

در بخش‌های دیگر کتاب به بررسی الگوریتم‌هایی با کاربردهای گسترده می‌پردازیم:

- در فصل‌های ۴، ۸، و ۹ تکنیک‌های حل مسئله - پوشش داده شده است. اگر به مشکلی برخورد کردید و مطمئن نبودید که چگونه آن را به طور موثر حل کنید، تقسیم و حل[6](فصل ۴) یا برنامه نویسی پویا[7](فصل ۹) را امتحان کنید. یا ممکن است متوجه بشوید که هیچ راه حل کارآمدی وجود ندارد و به جای آن با استفاده از الگوریتم حریصانه[8] به یک پاسخ تقریبی دست پیدا کنید (فصل ۸).
- جدول هَش در فصل ۵ پوشش داده شده است. جدول هَش ساختمان‌داده‌ای

1. Binary search
2. Big O notation
3. Hash tables
4. recursion
5. quicksort
6. Divide and conquere
7. Dynamique programming
8. Greedy algorithm

بسیار مفید است. این جدول‌ها شامل مجموعه‌ای از جفت‌های کلید[1] و مقدار[2] است. مانند نام شخص و آدرس ایمیل او، یا نام کاربری و رمز عبور آن. جدول‌های هش واقعاً مفید هستند. وقتی می‌خواهم مسئله‌ای را حل کنم، دو استراتژی که در شروع درنظر می‌گیرم عبارتند از: «آیا می‌توانم از جدول هش استفاده کنم؟» و «آیا می‌توانم این را به عنوان یک گراف[3] مدل‌سازی کنم؟»

- الگوریتم گراف در فصل‌های ۶ و ۷ پوشش داده شده است. گراف راهی برای مدل‌سازی یک شبکه است: یک شبکه‌ی اجتماعی، یا شبکه‌ای از مسیرها، یا نورون‌ها، یا هر مجموعه‌ی دیگر از ارتباط‌ها. جست‌وجوی سطح اول[4](فصل ۶) و الگوریتم دایجسترا[5](فصل ۷) راه‌هایی برای یافتن کوتاه‌ترین فاصله بین دو نقطه در یک شبکه هستند: می‌توانید از این روش برای محاسبه‌ی درجات جدایی بین دو نفر یا کوتاه‌ترین مسیر به مقصد استفاده کنید.
- الگویتم K- نزدیکترین همسایه[6] (KNN) - این یک الگوریتم ساده برای استفاده در یادگیری ماشینی است و در فصل ۱۰ به آن پرداخته می‌شود. می‌توانید از KNN برای ایجاد یک سیستم پیشنهادات، یک موتور OCR، سیستم پیش‌بینی ارزش سهام و به طور کلّی هر چیزی که شامل پیش‌بینی یک مقدار باشد، استفاده کنید («ما فکر می‌کنیم اَدیت به این فیلم ۴ ستاره می‌دهد») یا می‌توانید به کمک آن یک شیء را طبقه‌بندی کنید. («این حرف یک Q است»).
- گام‌های بعدی در فصل ۱۱ به ده الگوریتم دیگر پرداخته می‌شود که برای مطالعه‌ی بیشتر مناسب هستند.

## شیوه‌ی خواندن این کتاب

ترتیب و محتوای این کتاب با دقت طراحی شده است. اگر به موضوع خاصی علاقه‌مند هستید، با خیال راحت به آن بخش بروید. در غیر این صورت، فصل‌ها را به‌ترتیب بخوانید. هر فصل زیربنای فصل بعدی است.

1. key
2. value
3. graph
4. Breadth-first search
5. dijkstra algorithm
6. k-nearest neighbors algorithm

اکیداً توصیه می‌کنم کد مثال‌ها را خودتان اجرا کنید. در این مورد هر چه بگویم کم گفته‌ام. نمونه کدها را کلمه به کلمه تایپ کنید (یا آنها را از www.manning.com/books/grok-king-algorithms یا https://github.com/egonschiele/grokking_algorithms دانلود کنید)، و آن‌ها را اجرا کنید. با پیروی از این شیوه چیزهای بیشتری را به خاطر خواهید سپرد.

انجام تمرین‌های این کتاب را هم توصیه می‌کنم. تمرینات کوتاه هستند- حل آن‌ها معمولاً فقط یک یا دو دقیقه، گاهی اوقات ۵ تا ۱۰ دقیقه زمان می برند . این تمرین‌ها به شما کمک می‌کنند تا از شرایط یادگیری خود مطلع بشوید، بنابراین قبل از اینکه دیر شود متوجه می‌شوید که از مسیر خارج شده‌اید یا خیر.

این کتاب برای چه کسانی مناسب است

این کتاب برای کسانی است که اصول کدنویسی را می‌دانند و می‌خواهند الگوریتم‌ها را درک کنند. شاید قبلاً در کدنویسی به مشکل برخورد کرده باشید و در تلاش برای یافتن یک راه حل الگوریتمی باشید. یا شاید می‌خواهید بدانید که الگوریتم‌ها به چه کاری می‌آیند. در اینجا یک لیست کوتاه و ناقص از افرادی است که احتمالاً از این کتاب بهره می‌برند:

- کدنویسان تفریحی
- دانش‌آموزان دوره‌های کدنویسی
- فارغ التحصیلان علوم کامپیوتر که به دنبال بازآموزی هستند
- فارغ‌التحصیلان فیزیک،ریاضی و دیگر رشته‌ها که به برنامه‌نویسی علاقه‌مند هستند

## قراردادهای کدنویسی و آدرس فایل‌ها برای دانلود

تمام نمونه کدهای این کتاب از Python 2.7 استفاده می‌کنند. تمام کدهای موجود در کتاب با فونت با عرض ثابت مانند این (`fixed-width font like this`) ارائه شده است تا از متن معمولی قابل تفکیک باشد. حاشیه‌نویسی کد برخی از لیست‌ها را همراهی می‌کند و مفاهیم مهم را برجسته می‌کند.

می‌توانید کد مثال‌های کتاب را از وب‌سایت ناشر به نشانی www.manning.com/books/grok-king-algorithms یا از https://github.com/egonschiele/grokking_algorithms دانلود کنید.

تنها زمانی بیشترین میزان یادگیری را تجربه می‌کنید که واقعاً از یادگیری لذت ببرید

پس لذت ببرید و نمونه کدها را اجرا کنید!

## درباره‌ی نویسنده

آدیتیا بهارگاوا[1] مهندس نرم‌افزار در Etsy، (بازار آنلاین برای کالاهای دست‌ساز) است. او دارای مدرک کارشناسی ارشد در رشته‌ی علوم کامپیوتر از دانشگاه شیکاگو است. او همچنین یک وبلاگ فناوری مصور محبوب و پربازدید به آدرس adit.io دارد.

1. Adit Bhargava

# یک | درآمدی بر الگوریتم

در این فصل

- مبانی و اساس مباحث مطرح‌شده در کتاب را یاد می‌گیرید.
- اولین الگوریتم جست‌وجوی خود را می‌نویسید (جست‌وجوی دودویی).
- یاد می‌گیرید چگونه درباره‌ی زمان اجرای یک الگوریتم صحبت کنید (علامت O بزرگ).
- یک تکنیک رایج برای طراحی الگوریتم به شما معرفی می‌شود (تابع بازگشتی).

## مقدمه

الگوریتم مجموعه‌ای از دستورالعمل‌ها برای انجام یک کار است. هرچند هر قطعه کد را می‌توان یک الگوریتم نامید، با این حال در این کتاب به موارد جالب‌تر پرداخته می‌شود. الگوریتم‌هایی را در این کتاب آورده‌ام که عملکرد سریعی دارند یا به واسطه‌ی آن‌ها مسائل قابل توجهی را می‌توان حل کرد یا هر دو این ویژگی‌ها را شامل می‌شوند. در ادامه مباحث مطرح‌شده در این کتاب را مرور می‌کنیم:

• در فصل ۱ در مورد جست‌وجوی دودویی صحبت می‌شود و می‌بینیم که چگونه یک الگوریتم می‌تواند سرعت کد شما را افزایش بدهد. در یک مثال، تعداد مراحل مورد نیاز از ۴ میلیارد مرحله به ۳۲ مرحله کاهش می‌یابد!

یک دستگاه GPS از الگوریتم گراف (همان‌طور که در فصل‌های ۶ و ۷ یاد می‌گیرید) برای محاسبه‌ی کوتاه‌ترین مسیر به مقصد استفاده می‌کند.

• می‌توانید از برنامه‌نویسی پویا (در فصل ۹ به آن پرداخته شده است) برای نوشتن یک الگوریتم هوش مصنوعی برای بازی چکرز[1] استفاده کنید.

در هر مورد، الگوریتم را شرح می‌دهم و مثالی برای شما می‌زنم. سپس در مورد زمان اجرای الگوریتم در نماد O بزرگ صحبت می‌کنم. در انتها، به بررسی مسئله‌های دیگری که می‌توان با همان الگوریتم حل کرد، می‌پردازم.

## آن‌چه درباره‌ی کارایی[2] خواهید آموخت

خبر خوب این که احتمالاً پیش از این نسخه‌ی پیاده‌سازی‌شده‌ی هر الگوریتمی که در این کتاب آمده، در زبان برنامه‌نویسی مورد علاقه‌ی شما در دسترس باشد، در نتیجه نیاز نخواهد شد خود شما کد هر الگوریتم را بنویسید! اما اگر این کدها درک نشوند، پیاده‌سازی آن‌ها بیهوده خواهد بود. در این کتاب، می‌آموزید الگوریتم‌های مختلف را مقایسه کنید: باید از مرتب‌سازی ادغامی[3] استفاده کرد یا مرتب‌سازی سریع[4]؟ از آرایه[5] یا لیست[6]؟ استفاده از یک ساختمان داده‌ی[7] متفاوت می‌تواند تفاوت اساسی ایجاد کند.

## آن‌چه در مورد حل مسئله یاد می‌گیرید

شما تکنیک‌هایی را برای حل مسئله‌هایی که ممکن است تاکنون از درک شما خارج بوده‌اند، یاد می‌گیرید. مثلا:

1. checkers
2. Performance
3. merge sort
4. quicksort
5. array
6. list
7. Data structure

• اگر به ساخت بازی‌های ویدیویی علاقه دارید، می‌توانید یک سیستم هوش مصنوعی بنویسید که حرکات بازیکن را به کمک الگوریتم گراف دنبال بکند.
• می‌آموزید که با استفاده از K نزدیک‌ترین همسایه یک سیستم پیشنهاد بسازید.
• برخی از مسائل قابل حل نیستند! بخشی از این کتاب که در مورد مسائل NP کامل[1] صحبت می‌کند و به شما نشان می‌دهد که چگونه آن مسئله‌ها را شناسایی کنید و الگوریتمی ارائه بدهید که به پاسخ تقریبی برسید.

در پایان این کتاب، با برخی از کاربردی‌ترین الگوریتم‌ها آشنا می‌شوید. سپس می‌توانید از دانش جدید خود برای یادگیری الگوریتم های خاص‌تر برای هوش مصنوعی، پایگاه‌های داده و غیره استفاده کنید. یا می‌توانید با چالش‌های بزرگ‌تری در محل کار روبرو بشوید.

## پیش‌نیاز

پیش از شروع این کتاب دانش مقدماتی از جبر مورد نیاز است. در این حد که در تابع $f(x) = x \times ۲$، پاسخ $f(۵)$ را بدانید. اگر جواب شما عدد ۱۰ است، از دانش کافی ریاضی برخوردارید.

به‌علاوه، دنبال‌کردن این فصل(و این کتاب) اگر با زبان برنامه‌نویسی آشنا باشید آسان‌تر خواهد بود. تمامی مثال‌های این کتاب به زبان پایتون نوشته شده‌اند. اگر با هیچ زبان برنامه‌نویسی آشنایی ندارید و می‌خواهید یاد بگیرید، پایتون را انتخاب کنید- برای تازه‌کارها حرف ندارد. اگر با زبان دیگری مثل روبی هم آشنایی دارید، باز مشکلی نخواهید داشت.

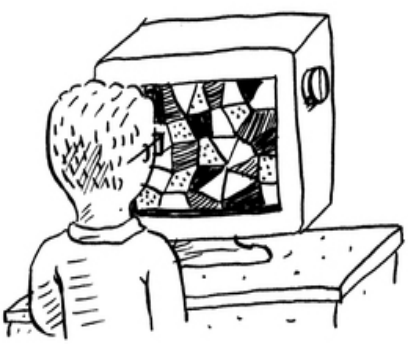
1. NP-complete

## جست‌وجوی دودویی

فرض کنید در یک دفترچه تلفن دنبال نام فردی می‌گردید(چه کار از مُدافتاده‌ای) اسم آن شخص با حرف K شروع می‌شود. می‌توانید از اول شروع کرده و صفحات را ورق بزنید تا به اسم‌هایی برسید که با حرف k شروع می‌شوند. با این حال احتمال این‌که از صفحه‌ای از وسط دفترچه شروع به جست‌وجو کنید بیشتر است، چون می‌دانید که Kها جایی در همان میانه‌ی دفترچه‌ی تلفن هستند.

یا فرض کنید به دنبال کلمه‌ای در دیکشنری می‌گردید که با حرف O شروع شده است. این بار هم، از صفحات وسط شروع می‌کنید.

حالا فرض کنید که وارد فیس‌بوک می‌شوید. فیس‌بوک باید تأیید کند که در این سایت حساب کاربری دارید. پس، باید در پایگاه داده‌ی[1] خود دنبال نام شما بگردد. فرض کنید که نام کاربری شما Karmageddon است. فیس‌بوک می‌تواند از حرف A شروع کرده و به دنبال نام شما بگردد-با این حال برایش منطقی‌تر است که از جایی در میانه‌ی فهرست شروع کند.



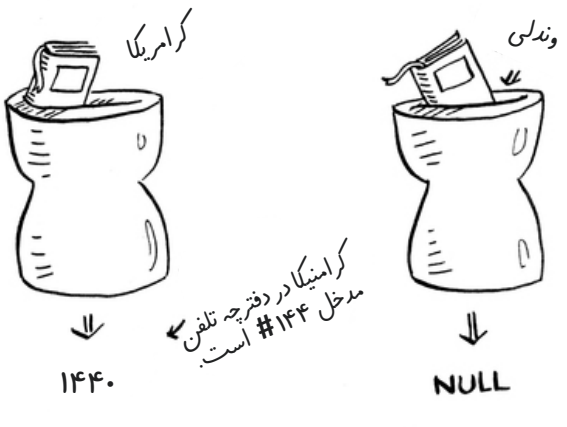
این یک مسئله‌ی جست‌وجو است. و در تمامی این موارد از الگوریتم یکسانی برای حل مسئله استفاده می‌شود.

جست‌وجوی دودویی الگوریتمی است

1. database

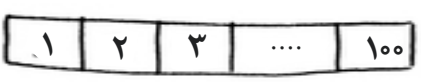
که ورودی آن یک لیست مرتب‌شده از عناصر است(بعدتر توضیح خواهم داد که چرا نیاز است تا این لیست مرتب بشود). اگر عنصری که به دنبال آن می‌گردید در آن لیست باشد، جست‌وجوی دودویی موقعیت مورد نظر را بر می‌گرداند. در غیر این صورت جواب جست‌وجوی دودویی `null` خواهد بود.

به عنوان مثال:




**به دنبال شماره تلفن شرکت‌ها در یک دفترچه‌تلفن از طریق جست‌وجوی دودویی**

در اینجا مثالی داریم از این‌که جست‌وجوی دودویی چگونه کار می‌کند. عددی بین ۱ تا ۱۰۰ را انتخاب می‌کنم.



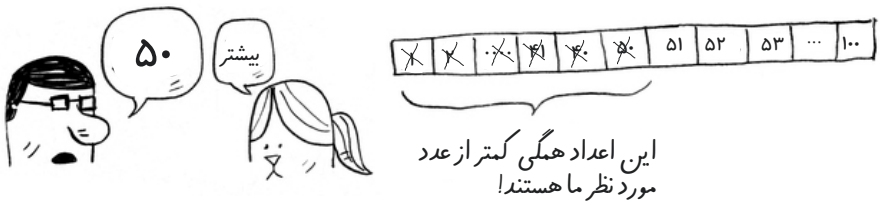

باید بتوانید با کمترین دفعات ممکن عدد مورد نظر را حدس بزنید. در هر نوبت، به شما خواهم گفت که آیا حدس شما کمتر یا بیشتر از عدد مورد نظر است، یا اینکه عدد را درست حدس زده‌اید.

فرض کنید اعدادی که حدس زده‌اید به این شکل باشد: ۱،۲،۳،۴.... با این روال به‌پیش خواهید رفت:


این یک جست‌وجوی ساده است.(شاید جست‌وجوی احمقانه تعبیر درست‌تری باشد). با هر بار حدس‌زدن، تنها یک شماره را حذف می‌کنید اگر شماره‌ی مورد نظر من ۹۹ باشد، ۹۹ بار طول می‌کشد تا درست حدس بزنید!

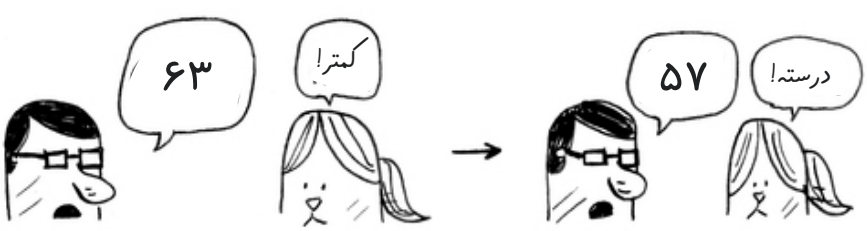
**یک راه بهتر برای جست‌وجو**

این یکی تکنیک بهتر است. با عدد ۵۰ شروع کنید.

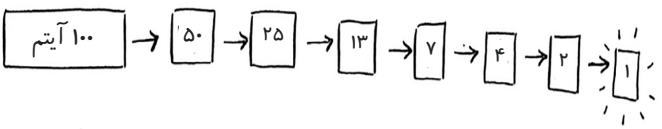



کوچک‌تر از عدد مورد نظر است، با این حال نصف اعداد حذف شدند! حالا می‌دانید که اعداد ۱-۵۰ همگی کمتر هستند. حدس بعدی: ۷۵.


بزرگ‌تر است، اما باز هم نصف اعداد باقی‌مانده را حذف کرده‌اید! با جست‌وجوی دودویی، عدد وسطی را حدس می‌زنید و نصف اعداد باقی‌مانده را در هر نوبت حذف می‌کنید. عدد بعدی ۶۳ است(عدد میانی ۵۰ و ۷۵)



همین الان اولین الگوریتم را یاد گرفتید. به این راه حل جست‌وجوی دودویی گفته می‌شود. در اینجا تعداد اعدادی که در هر نوبت می‌توان حذف کرد به نمایش در آمده است:




**حذف نصف اعداد در هر نوبت با جست‌وجوی دودویی.**

۷ گام

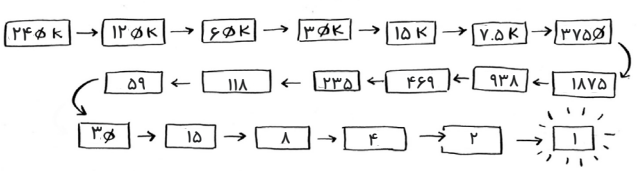
به هر عددی می‌توان با حداکثر ۷ حدس دست یافت - چون با هر حدس اعداد بسیاری را حذف می‌کنید!

فرض کنید که به دنبال یک کلمه در دیکشنری می‌گردید. این دیکشنری از ۲۴۰۰۰۰ کلمه تشکیل شده است. در بدترین وضعیت، فکر می‌کنید چند گام برای هر جست‌وجو مورد نیاز است؟

گام ــــــــ : جست‌وجوی ساده

گام ــــــــ : جست‌وجوی دودویی

جست‌وجوی ساده در شرایطی‌که کلمه‌ی مورد نظر آخرین کلمه‌ی دیکشنری باشد ۲۴۰۰۰۰ مرحله خواهد داشت. در حالی‌که در هر مرحله از جست‌وجوی دودویی، تعداد کلمات را به نصف کاهش می‌دهید تا تنها یک کلمه باقی بماند.




۱۸ گام

پس جست‌وجوی دودویی ۱۸ مرحله طول می‌کشد- تفاوت قابل توجه!

به طور معمول، برای هر فهرستی از n، جست‌وجوی دودویی $\log_2 n$ مرحله برای بدترین وضعیت نیاز دارد، در حالی‌که برای جست‌وجوی ساده n مرحله مورد نیاز است.

## لگاریتم

شاید مفهوم لگاریتم را فراموش کرده باشید، اما احتمالاً می‌دانید که نمایی (توان) چیست. $\log_{10} 100$ مثل این است که بپرسیم، «چندتا ده تا باید در هم ضرب کنیم تا به عدد ۱۰۰ برسیم؟» جواب ۲ است: ۱۰ × ۱۰. بنابراین $\log_{10} 100 = 2$. درواقع لگاریتم برعکس توان است.

$$10^2 = 100 \leftrightarrow \log_{10} 100 = 2$$
$$10^3 = 1000 \leftrightarrow \log_{10} 1000 = 3$$
$$2^3 = 8 \leftrightarrow \log_2 8 = 3$$
$$2^4 = 16 \leftrightarrow \log_2 16 = 4$$
$$2^5 = 32 \leftrightarrow \log_2 32 = 5$$

**لگاریتم برعکس توان است.**

در این کتاب هرگاه از زمان اجرا با علامت O بزرگ صحبت می‌کنم(کمی بعد توضیح می‌دهم)، log همیشه به معنای $\log_2$ است. وقتی که از راه جست‌وجوی ساده به دنبال عضوی می‌گردید، در بدترین حالت باید به هر عضو نگاه کنید. پس برای یک فهرست ۸ عددی، حداکثر باید ۸ عدد را چک کنید. برای جست‌وجوی دودویی، باید در بدترین حالت، log n عضو را چک کنید. برای یک فهرست ۸ عضوری، $\log 8 == 3$، چون $2^3 == 8$. پس برای یک فهرست ۸ عضوی، حداکثر سه عدد را باید چک کنید. برای لیستی با ۱۰۲۴ عضو، $\log 1,024$ مورد، چون $2^{10} == 1,024$ . پس برای یک فهرست ۱۰۲۴ عددی، باید در بدترین حالت ۱۰ عدد را چک کنید.

**نکته**

در این کتاب به دفعات درباره‌ی log time یا زمان لگاریتمی صحبت می‌کنم، پس باید مفهوم لگاریتم را بفهمید. اگر اطلاعات کافی درباره‌ی آن ندارید، خان آکادمی(khanacademy.org) ویدئوی خوبی در مورد لگلاریتم دارد که درک این مبحث را ممکن می‌کند.

**نکته**

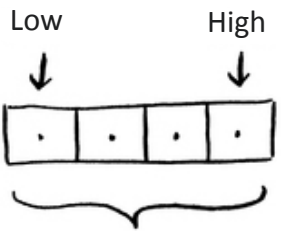
تنها زمانی می‌توان از جست‌وجوی دودویی استفاده کرد که لیست شما مرتب‌شده باشد. برای مثال، اسامی دفتر تلفن به ترتیب حروف الفبا ردیف شده است، پس می‌توانید از جست‌وجوی دودویی برای پیداکردن یک نام استفاده کنید. اگر اسامی مرتب نشده بودند چه اتفاقی میافتاد؟

بیایید با هم نگاهی بیندازیم به اینکه چگونه باید جست‌وجوی دودویی را به زبان پایتون نوشت. در این کد نمونه از آرایه استفاده شده است. اگر نمی‌دانید که آرایه‌ها به چه شکل عمل می‌کنند، نگران نباشید؛ در فصل بعدی به آن می‌پردازیم. فقط باید بدانید که می‌توانید سکانسی از اعضا را در یک ردیف از باکت یا سطل‌های متوالی که آرایه نامیده می‌شود ذخیره کنید. باکت‌ها از شماره‌ی صفر شماره‌گذاری شده‌اند: باکت اول در موقعیت ۰# و باکت دوم ۱# و باکت سوم ۲# است و به همین ترتیب.

تابع `binary_search` به‌عنوان ورودی یک آرایه‌ی مرتب‌شده و یک آیتم می‌گیرد. اگر آیتم در آرایه بود، تابع موقعیت آن را برمی‌گرداند. بخشی از آرایه را که در آن جست‌وجو

انجام می‌شود را زیر نظر می‌گیریم. در گام نخست، آرایه به این شکل است:

```
low = 0
high = len(list) - 1
```




اعدادی که در میان آن‌ها
جست‌وجو می کنیم.

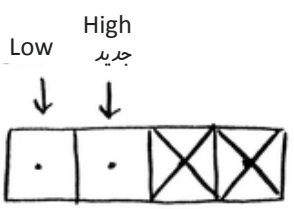
در هر نوبت، عضو میانی را چک می‌کنید:

```
mid = (low + high) // 2
guess = list[mid]
```

اگر حاصل (low + high) زوج نباشد، پایتون mid را کاهشی گرد می کند.

اگر حدس مورد نظر کم‌تر از عدد مورد نظر ما باشد، مطابق زیر مقدار low را تغییر می‌دهید:

```
if guess < item:
    low = mid + 1
```




و اگر حدس مورد نظر بیشتر باشد، high را تغییر می‌دهیم. کد کامل به این شکل است:

```
def binary_search(list, item):
  low = 0
  high = len(list) - 1

  while low <= high:
    mid = (low + high)/2
    guess = list[mid]
    if guess == item:
      return mid
    if guess > item:
      high = mid - 1
    else:
      low = mid + 1
    return None

my_list = [1, 3, 5, 7, 9]

print binary_search(my_list, 3) # => 1
print binary_search(my_list, -1) # => None
```

low و high محدوده‌ی جست‌وجوی شما را مشخص می‌کند.

وقتی دامنه را به یک عنصر محدود نکرده باشید...

عنصر میانی را بررسی می‌کند

عدد مورد نظر پیدا شد!

از عدد مورد نظر بزرگ‌تر است.

از عدد مورد نظر کوچک‌تر است.

آیتم وجود ندارد.

بیایید کد بالا را تست کنیم!

به یاد داشته باشید، لیست‌ها از اندیس صفر شروع می‌شوند. دومین اسلات اندیس یک دارد.

None در پایتون به معنای هیچ است.

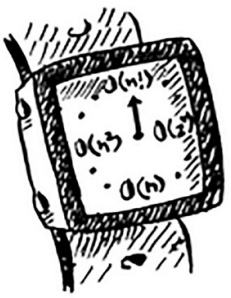
در این حالت آیتم موردنظر پیدا نشده است.

## تمرین

**۱.۱** فرض کنید یک لیست مرتب شده از ۱۲۸ نام دارید، و با جست‌وجوی دودویی در میان آن لیست به جست‌وجو می‌پردازید. حداکثر تعداد مراحل چه‌مقدار خواهد بود؟

**۱.۲** تعداد اعضای لیست را دوبرابر مقدار قبلی در نظر بگیرید. این بار حداکثر تعداد مراحل چقدر خواهد بود؟

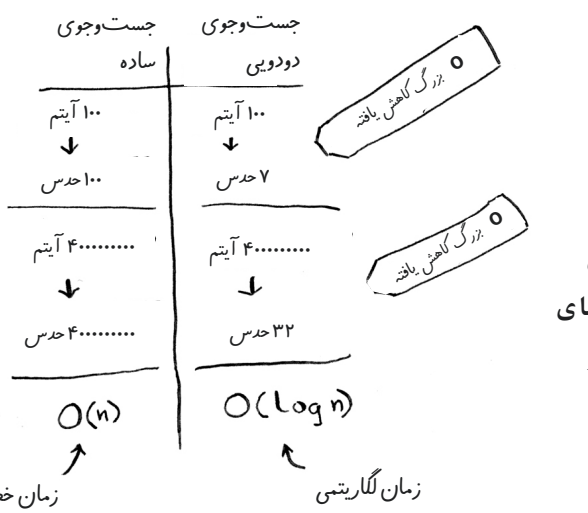
## زمان اجرا



هرگاه از الگوریتم صحبت می‌کنم، به زمان اجرای آن هم اشاره می‌کنم. به طور معمول می‌خواهید کارآمدترین الگوریتم را برای بهینه‌سازی زمان یا فضا انتخاب کنید.

به جست‌وجوی دودویی برگردیم. زمان صرفه‌جویی‌شده با استفاده از این الگوریتم به چه میزان است؟ خب، اولین رویکرد این بود که هر عدد را یک‌به‌یک بررسی کنیم. اگر فهرستی از ۱۰۰ عدد باشد تا ۱۰۰ حدس نیاز دارد. اگر فهرستی از ۴ میلیارد عدد باشد، تا ۴ میلیارد حدس نیاز دارد. بنابراین حداکثر تعداد حدس ها برابر با اندازه‌ی لیست است. به چنین حالتی زمان خطی می‌گویند.

زمان اجرا برای جست‌وجوی دودویی متفاوت است. اگر لیست ۱۰۰ آیتم داشته باشد، حداکثر ۷ حدس نیاز است. اگر لیست ۴ میلیارد آیتم داشته باشد، حداکثر دفعات ۳۲ مرتبه می‌شود. عجب الگوریتم کارا و قدرتمندی، مگر نه؟ جست‌وجوی دودویی در زمان لگاریتمی (یا به قول بومی‌ها لُگ تایم[1]) اجرا می‌شود. در جدول زیر خلاصه‌ای از یافته‌های امروز ما آمده است.




**زمان اجرای الگوریتم های جست‌وجو**

1. Log time

## نماد O بزرگ

O بزرگ یک نماد ویژه است که به شما می‌گوید الگوریتم مورد نظر تا چه اندازه سریع است. چه اهمیّتی دارد؟ خوب، شما اغلب از الگوریتم‌هایی که دیگران نوشته‌اند استفاده می‌کنید - و وقتی این کار را انجام می‌دهید، خوب است که بدانید این الگوریتم‌ها چقدر سریع یا کند هستند. در این بخش توضیح می‌دهم نماد O بزرگ چیست و فهرستی از متداول‌ترین زمان‌های اجرای الگوریتم‌ها را به شما ارائه می‌دهم.

### زمان اجرای الگوریتم‌ها با سرعت‌های متفاوتی رشد می‌کند

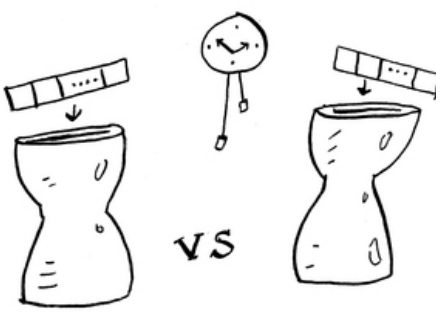
باب در حال نوشتن یک الگوریتم جست‌وجو برای ناسا است. الگوریتم او در آستانه‌ی فرود موشک بر روی ماه آغاز می‌شود و به محاسبه‌ی محل فرود کمک می کند.

این نمونه ای است که نشان می‌دهد چگونه زمان اجرای دو الگوریتم می‌تواند با نسبت متفاوت رشد کند.

باب در تلاش است میان جست‌وجوی ساده و جست‌وجوی دودویی تصمیم بگیرد. الگوریتم باید هم‌زمان سریع و صحیح باشد. از یک طرف، جست‌وجوی دودویی سریع‌تر است و باب فقط ۱۰ ثانیه فرصت دارد تا بفهمد کجا باید فرود بیاید - در غیر این صورت، موشک از مسیر خود خارج خواهد شد. از سوی دیگر، نوشتن جست‌وجوی ساده آسان‌تر است و احتمال بروز مشکلات کمتر است و باب واقعاً نمی‌خواهد باگ‌هایی در کد برای فرود موشک ایجاد شود! برای دقت بیشتر، باب تصمیم می گیرد هر دو الگوریتم را با لیستی از ۱۰۰ عنصر زمان‌بندی کند.

بیایید فرض کنیم برای بررسی یک عنصر ۱ میلی ثانیه طول می‌کشد. با جست‌وجوی ساده، باب باید ۱۰۰ عنصر را بررسی کند، بنابراین جستجو ۱۰۰ میلی‌ثانیه طول می‌کشد تا اجرا شود. از طرف دیگر، او فقط باید ۷ عنصر را با جست‌وجوی دودویی بررسی کند ($\log_2 100$ تقریباً برابر عدد ۷ است)، بنابراین جست‌وجو ۷ میلی‌ثانیه طول می‌کشد تا

اجرا شود. اما به طور واقع‌بینانه، این لیست بیش از یک میلیارد عنصر خواهد داشت. اگر این‌طور باشد، جست‌وجوی ساده چقدر طول می‌کشد؟ جست‌وجوی دودویی چقدر ؟ قبل از خواندن ادامه‌ی مطلب، مطمئن شوید که برای هر یک از پرسش‌ها پاسخی دارید.



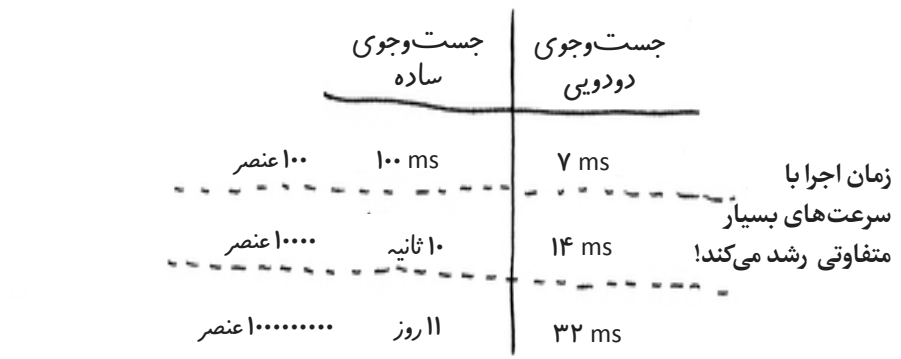

زمان اجرای جست‌وجوی ساده در مقایسه با جست‌وجوی دودویی در لیستی با ۱۰۰ عضو

جست‌وجوی ساده
۱۰۰ ms

جست‌وجوی دودویی
۷ ms

باب جست‌وجوی دودویی را با ۱ میلیارد عنصر اجرا می کند و ۳۰ میلی‌ثانیه طول می‌کشد ($\log_2$ ۱,۰۰۰,۰۰۰,۰۰۰ حدوداً برابر است با ۳۰). باخودش فکر می‌کند « ۳۲ Ms تقریباً ۳۰ است. جست‌وجوی دودویی حدود ۱۵ برابر سریع‌تر از جست‌وجوی ساده است، زیرا جست‌وجوی ساده ۱۰۰ میلی‌ثانیه با ۱۰۰ عنصر و جست‌وجوی دودویی ۷ میلی‌ثانیه طول کشید. بنابراین جست‌وجوی ساده ۴۵۰ = ۱۵× ۳۰ میلی‌ثانیه طول می‌کشد، درست است؟ بسیار کم‌تر از آستانه‌ی ۱۰ ثانیه‌ای.» باب تصمیم می گیرد با یک جست‌وجوی ساده به آن ادامه بدهد. این انتخاب درستی است؟

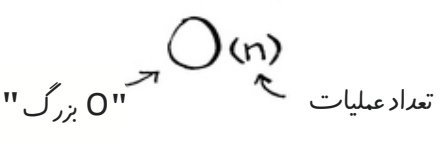
نه. معلوم شد، باب اشتباه می‌کند. آن هم چه جور. زمان اجرای جست‌وجوی ساده با ۱ میلیارد آیتم ۱ میلیارد میلی‌ثانیه خواهد بود که ۱۱ روز می‌شود! مشکل این است که زمان اجرا برای جست‌وجوی دودویی و جست‌وجوی ساده با سرعت یکسانی رشد نمی‌کند.




به عبارتی با افزایش تعداد آیتم‌ها، جست‌وجوی دودویی برای اجرا اندکی زمان بیشتر برای اجرا نیاز دارد. اما جست‌وجوی ساده زمان بسیار بیشتری را برای اجرا نیاز دارد. بنابراین با بزرگ‌تر شدن فهرست اعداد، جست‌وجوی دودویی ناگهان بسیار سریع‌تر از جست‌وجوی ساده می‌شود. باب گمان می‌کرد جست‌وجوی دودویی ۱۵ برابر سریع‌تر از جست‌وجوی ساده است، اما چنین تصوّری درست نیست. اگر لیست دارای ۱ میلیارد آیتم باشد، ۳۳ میلیون بار سریع‌تر است. به همین دلیل است که دانستن مدت زمان اجرای یک الگوریتم کافی نیست - باید بدانید با افزایش اندازه‌ی لیست زمان اجرا چگونه افزایش می‌یابد. اینجاست که نماد O بزرگ وارد می‌شود.

نماد O بزرگ به شما می‌گوید که یک الگوریتم چقدر سریع است. برای مثال، فرض کنید فهرستی با اندازه‌ی n دارید. جست‌وجوی ساده باید هر عنصر را بررسی کند، بنابراین n عملیات طول خواهد کشید. زمان اجرا در نماد O بزرگ O(n) است. واحد زمان و ثانیه‌ی آن چه شد؟ خبری ازش نیست - O بزرگ سرعت را بر حسب ثانیه به شما نمی‌گوید. نماد O بزرگ به شما امکان می‌دهد تعداد عملیات را مقایسه کنید. به شما می‌گوید که الگوریتم با چه سرعتی رشد می‌کند.

در اینجا مثال دیگری داریم. جست‌وجوی دودویی برای بررسی لیستی با اندازه‌ی n به log n عملیات نیاز دارد. زمان اجرا در نماد O بزرگ چقدر است؟ O(log n). به طور کلی نماد O بزرگ به شکل زیر نوشته می‌شود.



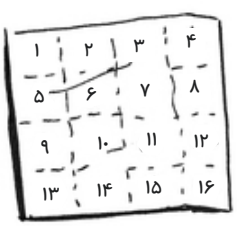

**شکل نماد O بزرگ**

این نماد O بزرگ نامیده می‌شود زیرا شما یک «O بزرگ» را در جلوی تعداد عملیات قرار می‌دهید (به نظر شوخی می رسد، اما حقیقت دارد!). و تعداد عملیاتی را که یک الگوریتم انجام می‌دهد به شما می گوید.

چند مثال را مرور کنیم. ببینید می‌توانید زمان اجرای این الگوریتم‌ها را مشخص کنید.

## به تصویر کشیدن چند زمان اجرا با O بزرگ متفاوت

در اینجا یک مثال عملی داریم که می توانید در خانه با چند تکه کاغذ و یک مداد دنبال کنید. فرض کنید باید یک چهارخانه‌ی ۱۶ تایی بکشید.

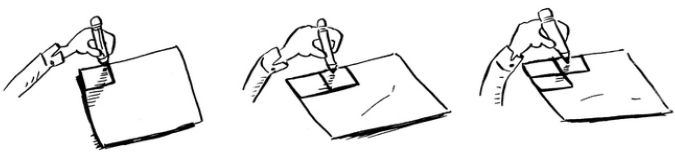



**کدام الگوریتم برای کشیدن این جدول مناسب است؟**

### الگوریتم یک

یکی از راه‌های انجام این کار، کشیدن به نوبت ۱۶ چهارخانه، است. به یاد داشته باشید که نماد O بزرگ تعداد عملیات را می‌شمارد. در این مثال، رسم هر چهارخانه برابر با یک عملیات است. شما باید ۱۶ چهارخانه بکشید. کشیدن یک چهارخانه در هر بار چند عملیات طول می‌کشد؟

**کشیدن یک چهارخانه در هر نوبت**



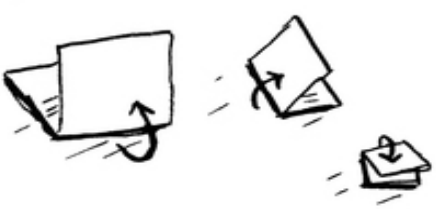
برای کشیدن ۱۶ چهارخانه ۱۶ مرحله طول می‌کشد. زمان اجرای این الگوریتم چقدر است؟

**الگوریتم دو**

حالا این الگوریتم را امتحان کنید. کاغذ را از وسط تا کنید.

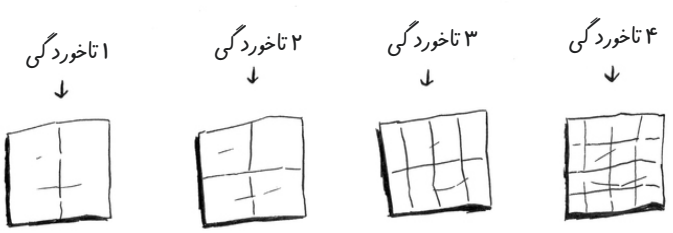
در این مثال، هر بار تا کردن کاغذ یک عملیات است. شما فقط با این عملیات دو چهارخانه درست کردید!

کاغذ را دوباره و دوباره و دوباره تا کنید.



بعد از چهار بار تا زدن آن را باز کنید و یک جدول زیبا خواهید داشت! هر تا زدن تعداد چهارخانه‌ها را دو برابر می‌کند. شما با ۴ عملیات ۱۶ چهارخانه درست کردید!

**کشیدن یک جدول با چهار نوبت تا کردن کاغذ**




شما می‌توانید با هر تا زدن تعداد چهارخانه را دو برابر کنید. در نتیجه می‌توانید ۱۶ چهارخانه را در ۴ مرحله رسم کنید. زمان اجرای این الگوریتم چقدر است؟ قبل از حرکت، زمان‌های اجرای هر دو الگوریتم را مشخص کنید.

پاسخ : الگوریتم اول زمان اجرای $O(n)$ و الگوریتم دوم زمان اجرای $O(\log n)$ دارد.

## O بزرگ برای بدترین زمان اجرا

فرض کنید از جست‌وجوی ساده برای جست‌وجوی شخصی در دفترچه تلفن استفاده می‌کنید. می دانید که جست‌وجوی ساده زمان اجرای $O(n)$ دارد، به این معنی که در بدترین حالت، باید تک تک ورودی های دفترچه تلفن خود را بررسی کنید. در این مورد، شما به دنبال نام Adit هستید. این فرد اولین نام در دفترچه‌ی تلفن شما است. بنابراین لازم نبود تا تمام نام‌ها را مرور کنید - در اولین تلاش آن را پیدا کردید. آیا این الگوریتم زمان $O(n)$ را صرف کرده است؟ یا به دلیل اینکه در اولین تلاش فرد مورد نظر را پیدا کردید، زمان $O(1)$ طول کشید؟

جست‌وجوی ساده همچنان به زمان $O(n)$ نیاز دارد. در این صورت، فوراً چیزی را که دنبالش می‌گشتید پیدا کردید. این بهترین سناریو است. اما نماد O بزرگ در مورد بدترین سناریو است. بنابراین می توانید بگویید که در بدترین حالت، باید هر نام واردشده در دفترچه تلفن را یک بار نگاه کنید. زمان آن $O(n)$ است. این برای آگاهی از بدترین حالت ممکن است - می‌دانید که جست‌وجوی ساده هرگز کندتر از زمان $O(n)$ نخواهد بود.

**نکته**

همراه با بدترین زمان اجرا، توجه به زمان اجرای متوسط[1] هم مهم است. در فصل ۴، بدترین حالت در مقابل حالت متوسط مورد بحث قرار گرفته است.

## برخی از زمان اجراهای معمول O بزرگ

در ادامه پنج زمان اجرای O بزرگ که به دفعات با آن‌ها مواجه می‌شوید، از سریع‌ترین به کندترین حالت فهرست شده است:

- $O(\log n)$، همچنین به عنوان log time شناخته می‌شود. به‌عنوان مثال: جست‌وجوی دودویی.
- $O(n)$، همچنین به عنوان زمان خطی شناخته می‌شود. به‌عنوان مثال: جست‌وجوی ساده
- $O(n * \log n)$، یک الگوریتم مرتب‌سازی سریع، مانند مرتب‌سازی سریع (که در فصل ۴ آمده است).
- $O(n^2)$، یک الگوریتم مرتب‌سازی آهسته، مانند مرتب‌سازی انتخابی (در فصل ۲ آمده است).
- $O(n!)$، یک الگوریتم بسیار کند، مانند فروشنده‌ی دوره‌گرد (مورد بعدی که به آن پرداخته می‌شود!).

فرض کنید یک بار دیگر جدولی با ۱۶ چهارخانه ترسیم می‌کنید و می‌توانید از ۵ الگوریتم مختلف برای این کار استفاده کنید. در صورت انتخاب الگوریتم اول، برای ترسیم شبکه زمان $O(\log n)$ طول می کشد و می توانید ۱۰ عملیات در ثانیه انجام بدهید.

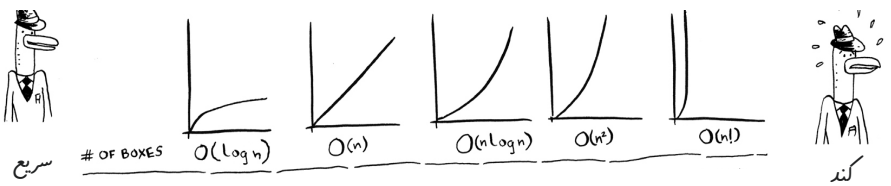
با زمان $O(\log n)$، ۴ عملیات طول می کشد تا یک جدول ۱۶ چهارخانه‌ای بکشید ($\log$ ۱۶ برابر ۴ است). بنابراین ۰.۴ثانیه طول می‌کشد تا جدول را بکشید. اگر بخواهید ۱۰۲۴ چهارخانه رسم کنید چه؟ $\log$ ۱۰۲۴=۱۰ عملیات طول می‌کشد. یا ۱ ثانیه برای رسم جدولی از ۱۰۲۴ چهارخانه. این اعداد از الگوریتم اول استفاده می‌کنند.

الگوریتم دوم کندتر است: زمان $O(n)$ طول می‌کشد. برای رسم ۱۶ چهارخانه ۱۶

1. Average-case run time

عملیات و برای رسم ۱۰۲۴ چهارخانه ۱۰۲۴ عملیات طول می‌کشد. در واقع چند ثانیه؟

در ادامه زمان لازم برای ترسیم‌کردن جدول از طریق دیگر الگوریتم‌ها، از سریع‌ترین به کندترین، آمده است:




| | | | | | |
|---|---|---|---|---|---|
| ۱۶ | ۰.۴ ثانیه | ۱.۶ ثانیه | ۶.۴ ثانیه | ۲۵.۶ ثانیه | ۶۶۳۰۱ سال |
| ۲۵۶ | ۰.۸ ثانیه | ۲۵.۶ ثانیه | ۳.۴ دقیقه | ۱.۸ ساعت | $8.6 \times 10^{505}$ سال |
| ۱۰۲۴ | ۱.۰ ثانیه | ۱.۷ ثانیه | ۱۷ دقیقه | ۱.۲ روز | $5.4 \times$ سال |

زمان اجراهای دیگری نیز وجود دارد، اما این پنج مورد رایج‌ترین آن‌ها هستند.

البته در واقعیت نمی‌توانید زمان اجرای O بزرگ را به این سرراستی به تعدادی عملیات تبدیل کنید، اما فعلا تا همین‌جا کافی است. پس از اینکه چند الگوریتم دیگر را یاد گرفتید، در فصل ۳ به نماد O بزرگ باز خواهیم گشت. نکات اصلی تا به این جا به شرح زیر است:

- سرعت الگوریتم نه بر اساس ثانیه بلکه بر مبنای رشد تعداد عملیات اندازه‌گیری می‌شود.
- در واقع، مقصود این است که با افزایش اندازه‌ی داده‌های ورودی زمان اجرای یک الگوریتم با چه سرعتی افزایش می‌یابد.
- زمان اجرای الگوریتم‌ها با نماد O بزرگ بیان می‌شود.
- $O(\log n)$ سریع تر از $O(n)$ است، اما با رشد لیست مواردی که جست‌وجو می‌کنید، سرعت آن بسیار بیشتر می‌شود.

**تمرین**

زمان اجرا برای هر یک از این سناریوها را بر حسب O بزرگ مشخص کنید.

**۱.۳** نام یک نفر را در نظر دارید و می خواهید شماره تلفن آن شخص را در دفترچه‌تلفن پیدا کنید.

**۱.۴** شما یک شماره تلفن دارید و می‌خواهید نام شخص را در دفترچه‌تلفن پیدا کنید. (نکته: باید تمام صفحات دفترچه را بگردید!)

**۱.۵** شما می‌خواهید شماره‌ی هر فرد نوشته‌شده در دفترچه‌تلفن را بخوانید.

**۱.۶** شما می خواهید شماره‌ی افرادی را بخوانید که نام آن‌ها با حرف A شروع شده است. (این مسئله پیچیده است! مفاهیمی را دربرمی‌گیرد که در فصل ۴ بیشتر به آن پرداخته می‌شود. احتمالا با دیدن پاسخ آن متعجب خواهید شد.)

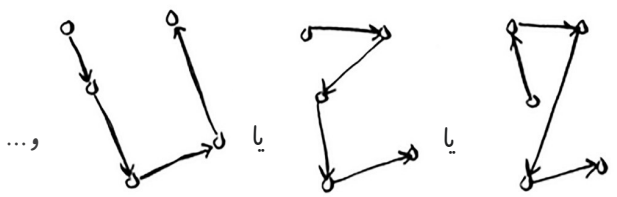
## فروشنده‌ی دوره‌گرد

ممکن است با خواندن بخش آخر فکر کرده باشید، «امکان ندارد با الگوریتمی مواجه شوم که $O(n!)$ زمان نیاز داشته باشد.» خوب، اجازه دهید تا به شما ثابت کنم که اشتباه می‌کنید! در ادامه نمونه‌ای از یک الگوریتم با زمان اجرای بسیار بد آمده است. این یک مسئله‌ی معروف در علوم کامپیوتر است، زیرا رشد آن وحشتناک است و بنا بر نظر افراد خبره و باهوش راهی برای بهبود زمان آن .وجود ندارد به این مسئله فروشنده‌ی دوره‌گرد می‌گویند.

یک فروشنده را فرض کنید.

این فروشنده باید به پنج شهر سفر کند.


این فروشنده، که اُپوس صدایش می‌کنم، می‌خواهد به هر پنج شهر برود با این شرط که حداقل مسافت را طی کند،. یک روش این است: مشاهده‌ی تمامی حالت‌های ممکن که می توان به شهرها سفر کرد.




او کل مسافت را جمع می کند و سپس مسیری را که کمترین فاصله را دارد انتخاب می کند. ۱۲۰ جایگشت [1] با ۵ شهر وجود دارد، بنابراین برای حل مسئله‌ی ۵ شهر، به ۱۲۰ عملیات نیاز است. برای ۶ شهر، ۷۲۰ عملیات انجام می‌شود (۷۲۰ جایگشت وجود دارد). برای ۷ شهر، ۵۰۴۰ عملیات طول می‌کشد!

1. permutation

| تعداد شهر | تعداد عملیات |
|---|---|
| ۶ | ۷۲۰ |
| ۷ | ۵۰۴۰ |
| ۸ | ۴۰۳۲۰ |
| ... | ... |
| ۱۵ | ۱،۳۰۷،۶۷۴،۳۶۸،۰۰۰ |
| ... | ... |
| ۳۰ | ۲،۶۵۲،۵۲،۸۵۹،۸۱۲،۱۹۱،۰۵۸،۶۳۶،۳۰۸،۴۸۰،۰۰۰،۰۰۰ |

**تعداد عملیات به شدت افزایش می یابد.**

به طور کلی، برای $n$ آیتم، $n!$ ($n$ فاکتوریل) عملیات برای محاسبه‌ی نتیجه طول می‌کشد. در نتیجه $O(n!)$ یا مرتبه اجرایی فاکتوریل است. تعداد بالای عملیات در تمامی شرایط (به استثنای ارقام بسیار کوچک) صرف خواهد شد. هنگامی که با بیش از ۱۰۰ شهر سر و کار دارید، محاسبه‌ی پاسخ در زمان مورد نظر ناممکن است و پیش از اینکه به جواب برسید دنیا به پایان رسیده است.

این یک الگوریتم وحشتناک است! اپوس باید از الگوریتم دیگری استفاده کند، درست است؟ اما اطلاعی از آن ندارد. چون این یکی از مسائل حل‌نشده در علوم کامپیوتر است. هیچ الگوریتم سریع شناخته شده‌ای برای آن وجود ندارد و متخصص‌ها بر این باورند که دسترسی به یک الگوریتم هوشمند برای این مسئله ناممکن است. بهترین کاری که می‌توان انجام داد این است که یک راه‌حل تقریبی ارائه بدهیم. برای اطلاعات بیشتر به فصل ۱۰ مراجعه کنید.

یک نکته‌ی پایانی: اگر به مسئله‌های پیچیده‌تر علاقه‌مند هستید، درخت‌های جست‌وجوی دودویی را بررسی کنید! شرح مختصری از آن‌ها در فصل آخر آمده است.

## جمع‌بندی

- جست‌وجوی دودویی بسیار سریع‌تر از جست‌وجوی ساده است.
- $O(\log n)$ سریع‌تر از $O(n)$ است، اما زمانی که لیست آیتم‌هایی که در آن جستجو می‌کنید بزرگ شود، بسیار سریع‌تر هم خواهد شد.
- سرعت الگوریتم به ثانیه اندازه‌گیری نمی‌شود.
- زمان الگوریتم بر حسب رشد یک الگوریتم اندازه‌گیری می‌شود.
- زمان الگوریتم با نماد O بزرگ نوشته می‌شود.