طراحی الگوریتم - تحلیل الگوریتم

محسن هوشمند

طراحی الگوریتم

فهرست

الگوریتم

اصطلاح الگوریتم به نظر ریشه در نام خوارزمی دارد. مختصری در این باره را عینا از کتاب جبر و مقابله وی از دوازده قرن پیش نقل می‌کنیم.

فأما **الأموال والجذور التی تعدل العدد** فمثل قولک مال و عشرۀ أجذاره یعدل تسعۀ وثلاثین درهماً و معناه أی مال إذا زدت علیه مثل عشرۀ أجذاره بلغ ذلک کله تسعۀ وثلاثین.

فقیاسه أن تنصف الأجذار و هی فی هذه المسألۀ خمسۀ فتضربها فی مثلها فتکون خمسۀ وعشرین فتزیدها علی التسعۀ والثلاثین فتکون أربعۀ وستین فتأخذ جذرها و هو ثمانیۀ فتنقص منه نصف الأجذار و هو خمسۀ فیبقی ثلاثۀ و هو جذر المال الذی ترید و المال تسعۀ.

شش- محمد بن موسی الخوارزمی، جبر و مقابله ۲۱۵ قمری، ۲۰۹ خورشیدی، ۸۳۰ میلادی

طراحی الگوریتم

جذر: x، مال: $x^۲$، مفرد: عدد ثابت

اما مال‌ها و جذرهایی که با عددی برابر می‌شوند:

اگر بگویی: یک مال، به اضافهٔ ده جذر از آن مال، با سی و نه درهم برابر می‌شود، مقصود آن است که اگر به مالی به اندازهٔ ده جذر از آن مال افزوده شود مجموع آن می‌شود سی و نه درهم.

راه حل آن چنین است: باید جذرها را نصف کنی – مقدار نصف آن در این مسئله پنج می‌شود – و آن نصف را در مانند خودش ضرب کنی، در این صورت حاصل‌ضرب بیست و پنج می‌شود. آن‌گاه این عدد را بر سی و نه بیفزایی، مجموعه شصت و چهار می‌شود، سپس جذر این عدد را می‌گیری، هشت می‌شود، آن‌گاه نیمی از شمارهٔ جذرها را که عبارت باشد از پنج، از آن کم می‌کنی که در نتیجه سه باقی می‌ماند، و همین عدد سه، جذر مال مورد نظر است، و آن مال نه است.

تلخیص سخن شیخ: توصیف الگوریتم برای حل نوع خاصی از معادلات درجه دو با استفاده از مثال $x^۲ + ۱۰x = ۳۹$

$$x^۲ + ۱۰x = ۳۹ \Longrightarrow (x + ۵)^۲ = ۳۹ + ۲۵ = ۶۴;$$

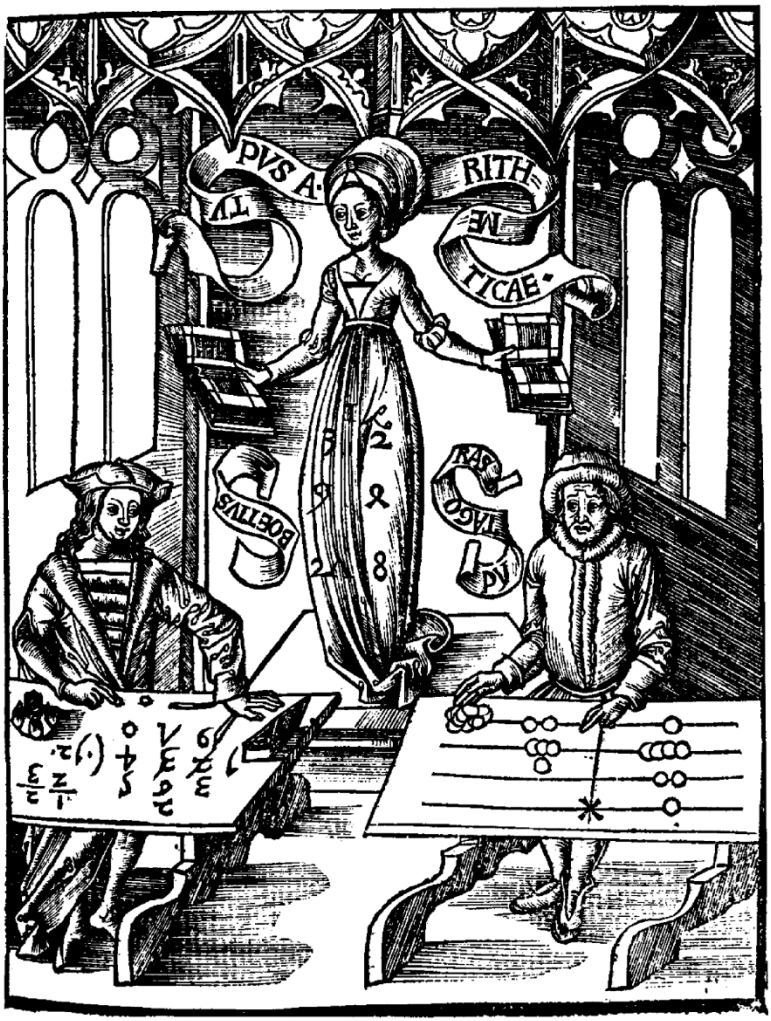
$$x + ۵ = ۸ \Longrightarrow x = ۳; x^۲ = ۹$$

م خوارزمی، جبر و مقابله، حسین خدیوجم، ۱۳۶۳

روش حل بالا را می‌توان از نمونه‌های حل الگوریتمی دانست. خوارزمی این مسئله را به روش هندسی «تکمیل مربع» و نه با جبر نمادین امروزی حل کرد. این روش یکی از نخستین نمونه‌های ثبت‌شده از تفکر الگوریتمی با گام‌های مشخص، بدون ابهام، و قابل تکرار برای دسته‌ای از مسائل است. با آنکه واژه الگوریتم از نام خوارزمی اخذ شده است، خود مفهوم الگوریتم بسیار قدیمی‌تر است. یکی از نخستین نمونه‌های

شناخته‌شده آن، الگوریتم اقلیدس برای محاسبه بزرگ‌ترین مقسوم‌علیه مشترک دو عدد است که بیش از دو هزار سال پیش در کتاب اصول اقلیدس آمده است.

روش حل خوارزمی با ترجمه به لاتین و اروپا منتقل می‌شود. منجر به تلاش افرادی با حساب عددی می‌شود. در نتیجه اندک اندک به معنای هر گونه روش محاسباتی روشمند اطلاق می‌شود.



«روح حساب» به منازعهٔ بین «الگوری‌های» نو که از نوشتن اعداد بهره می‌برند و «چرتکه‌اندازان» سنتی با جداول شمارش‌شان می‌نگرد. برگفته از مروارید فلسفه Margarita Philosophica، اثر گرگور رایش، ۱۵۰۴ مسیحی

سخن کوتاه، معنی لغوی آن روش مخصوص حل دسته‌ای از مسائل است.

در سده‌های میانه، اصطلاح algorithmus در لاتین دقیقاً به معنای «حساب کردن با استفاده از دستگاه شمارش بر پایه ۱۰» بود، و در مقابل abacus که روش سنتی با چرتکه را شامل می‌شد. اما با ترجمه و نشر آثار خوارزمی در اروپا، به تدریج معنای این اصطلاح گسترش یافت. در سال‌های ۱۵۰۳ تا ۱۵۱۷ میلادی کتاب «مروارید فلسفه» (Margarita Philosophica) اثر گرگور رایش (Gregor Reisch) بارها منتشر شد. تصویر معروف این کتاب که در بالا آماده است، نمادین‌ترین سند تاریخی از منازعه «الگوریست‌ها» و «چرتکه‌اندازان» است. در این تصویر، «بوئتیوس» (Boethius) فیلسوف رومی نماینده الگوریست‌هاست که با قلم و کاغذ و ارقام دستگاه شمارش ده‌تایی محاسبه می‌کند و «فیثاغورس» نماینده چرتکه‌اندازان است که با گرد و غبار و سنگ‌ریزه (Calculi) کار می‌کند. این نبرد در نهایت با پیروزی قاطع روش الگوریتمی (ارقام ده‌دهی) به پایان رسید.

در سده هفدهم، به تأثیر از واژه «لگاریتم» (جان نپر، ۱۶۱۴) و نیز تمایل به یونانی‌نمایی، صورت واژه از Algorism به Algorithm تغییر یافت، و دال آن نیز از «روش محاسبه با ارقام» فراتر رفت و به هر فرایند مرحله‌به‌مرحله و روشمند اطلاق شد.

کمی پیشتر از ساخت هر رایانه‌ای در حدود ۱۲۲۲ شمسی، آدا لاولیس نخستین الگوریتم را برای اجرا بر روی «ماشین تحلیلی» چارلز بابیج نوشت. او روشی برای محاسبه اعداد برنولی با استفاده از این ماشین عرضه کرد. به همین دلیل، او را نخستین برنامه‌نویس تاریخ می‌دانند. هرچند، ماشین تحلیلی هرگز ساخته نشد، اما یادداشت‌های او حاوی نخستین الگوریتم منتشر شده برای اجرا با ماشین بود.

در دهه ۱۳۱۰ شمسی انقلابی بنیادین در مفهوم الگوریتم رخ داد. تورینگ نشان داد هر الگوریتم قابل تعریف به عنوان ماشین تورینگ است. همزمان، آلونزو چرچ و استیون

کلین Kleene تعریف‌های معادلی با جبر لامبدا و توابع بازگشتی ارائه دادند. این سه تعریف «تز چرچ-تورینگ» را شکل می‌دهند که بنیان نظری علم رایانش است. ثمره چنین تحقیقاتی به این منجر شد که ماشین تورینگ مدلی انتزاعی اما دقیق از محاسبه شد. همچنین روشن شد هر رایانش مکانیکی قابل تصور را می‌توان با مجموعه‌ای متناهی از دستورالعمل‌ها شبیه‌سازی کرد. از این پس، الگوریتم دیگر فقط «روش» نبود؛ بلکه «شیای ریاضی» شد که می‌توان آن را مطالعه، تحلیل، و حتی اثبات کرد.

با اختراع رایانه‌های الکترونیکی، الگوریتم‌ها از روی کاغذ به درون ماشین‌ها منتقل شدند. در سال‌های مابین دهه ۱۹۴۰ تا ۱۹۶۰ میلادی طوفانی از الگوریتم‌های بنیادین پدیدار می‌شوند. در سال ۱۹۴۵جان فون نویمان الگوریتم «مرتب‌سازی ادغامی» را ابداع کرد. در سال ۱۹۴۷ جرج دانتزیگ روش «سیمپلکس» را برای حل مسائل بهینه‌سازی خطی ارائه داد. در سال ۱۹۵۲ دیوید هافمن الگوریتم «کدگذاری هافمن» را برای فشرده‌سازی داده‌ها ابداع کرد. در سال ۱۹۵۶ دیکسترا الگوریتم معروف خود برای یافتن کوتاه‌ترین مسیر در گراف را منتشر کرد که حتی امروزه در سامانه‌های مسیریابی چون جی‌پی‌اس استفاده می‌شود. در سال ۱۹۶۲ تونی هور الگوریتم مرتب‌سازی سریع را معرفی کرد که به یکی از الگوریتم‌های پرکاربرد در علم کامپیوتر بدل شد. این الگوریتم‌ها امروزه هستهٔ بسیاری از سیستم‌های نرم‌افزاری ایجاد شده‌اند. همچنین در دهه‌های ۱۹۶۰ و ۱۹۷۰ پرسش اساسی «الگوریتم چقدر باید سریع باشد؟» مطرح شد. در این دوره مفاهیم مهمی مانند نماد O-بزرگ (کنوث)، مفهوم زمان اجرا یا پیچیدگی زمانی، و مسئلهٔ تاریخی P در برابر NP (کوک-لوین) شکل گرفتند. این دوره الگوریتم را نه فقط از نظر ماهیت، بلکه از نظر کارآمدی نیز موضوع مطالعه قرار داد.

از ۱۹۷۰ تا ۲۰۰۰ میلادی پیچیدگی و اهمیت الگوریتم‌ها فزونی گرفت. الگوریتم‌های رمزنگاری مانند RSA (۱۹۷۷) امنیت ارتباطات دیجیتال را بر پایه تلاش‌های دیگران و ابداع مخترعان ممکن ساخت. موتورهای جستجو مانند گوگل (تاسیس ۱۹۹۸) با

الگوریتم رتبه‌بندی PageRank انقلابی در دسترسی به اطلاعات ایجاد کردند. الگوریتم‌های توزیع‌شده مانند MapReduce (گوگل، اوایل ۲۰۰۰) برای پردازش داده‌های عظیم به کار رفت. چنین الگوریتم‌هایی به زندگی روزمرهٔ میلیون‌ها حتی میلیاردها نفر وارد شد.

از دهه هشتاد شمسی به بعد روزگار «الگوریتم‌های هوشمند» فرا رسیده است. در سال ۱۹۹۷ کامپیوتر دیپ بلو (Deep Blue) از شرکت آی‌بی‌ام، گری کاسپاروف، قهرمان شطرنج جهان، را شکست می‌دهد. این نخستین باری است که یک ماشین در مسابقه‌ای رسمی، قهرمان جهان را مغلوب کرد. دیپ‌بلو از یادگیری ماشین استفاده نکرد، بلکه بر الگوهای دیگر هوش مصنوعی مبتنی بر جستجوی گسترده و ارزشیابی دست‌ساخت بود. از سال ۱۳۸۹ به بعد یادگیری ماشین و یادگیری عمیق بر پایه الگوریتم‌هایی مانند شبکه‌های عصبی که ایده اولیه آن به دهه ۱۳۲۰ شمسی بازمی‌گردد، کارآمدی فراوانی در حوزه‌های بینائی ماشین، پردازش سیگنال و زبان طبیعی نشان دادند. الگوریتم‌ها دیگر صرفاً پیرو دستورات صریح نیستند، بلکه از داده‌ها «یاد می‌گیرند،»، الگوها را کشف می‌کنند، و تصمیم‌گیری تقریبی اما کارآمد انجام دهند. در سال ۱۴۰۱ چت‌بات ChatGPT منتشر شد. در کنار این پیشرفت‌ها، نظریه پیچیدگی رایانشی (مفاهیمی مانند O بزرگ، P و NP) نشان می‌دهند برخی مسائل ذاتاً سخت‌تر از دیگران هستند و برخی الگوریتم‌ها در مقابل خطا یا داده‌های عظیم رفتارهای جالبی از خود نشان می‌دهند. این سیستم مبتنی بر مدل زبانی بزرگ که خود مجموعه عظیمی از الگوریتم‌هاست، توانست متونی شبیه به انسان تولید کند و توجه جهانی را به قدرت بی‌سابقه الگوریتم‌های مدرن جلب نماید.

در کنار پیشرفت‌های هوش مصنوعی، وجود داده‌های عظیم و بزرگ و جریان داده‌ها موجب توجه به الگوریتم‌هایی شد که در ازای میزان قابل کنترلی از خطا فضا و زمان کمتری را در پردازش داده‌ها برای استخراج و مدیریت داده دارا باشند.

هم‌زمان با رشد عظیم داده‌ها، نوعی جدید از الگوریتم‌ها مانند درهم‌سازی‌های تصادفی، فیلتر بلوم، ابرلگ‌لگ، طرح شمارش-کمینه، MapReduce پدید آمدند که در ازای مقداری خطای کنترل‌شده، حافظه و زمان را به‌شدت کاهش می‌دهند و در مقیاس‌های بزرگ اینترنتی استفاده می‌شوند.

امروز در علم رایانش، الگوریتم به صورت مجموعه‌ای متناهی، دقیق، گام‌به‌گام و قابل اجرا از دستورالعمل‌هاست که ورودی مشخص می‌گیرد، خروجی مشخص تولید می‌کند، در زمان متناهی متوقف می‌شود. تعریف مذکور بیش از ۱۲۰۰ سال تحول تاریخی است.

آنچه با نام «الگوریتم» از خوارزمی بزرگ در قرن دوم شمسی و قرن نهم میلادی با حل معادلاتی از قبیل $x^۲ + ۱۰x = ۳۹$ آغاز شد، گذاری هزار و دویست ساله را پشت سر گذاشته است. در آغاز، (قرن ۹ تا ۱۵)، نام دانشمندی (خوارزمی) بود که به روش محاسبه با ارقام ده‌دهی اطلاق شد. در میانه، (قرن ۱۶ تا ۱۹)، به معنای هر روش گام‌به‌گام و نظام‌مند برای حل مسائل ریاضی تبدیل گشت. در دوران جدید (قرن ۲۰)، تعریف دقیقی در قالب ماشین تورینگ و نظریه پیچیدگی یافت و به هسته مرکزی علم کامپیوتر بدل شد. در زمان حاضر (قرن ۲۱) الگوریتم موجودیتی محاسباتی است که یاد می‌گیرد، تصمیم می‌گیرد، و در زندگی دیجیتال نقش دارد. از مسیریابی در گوشی همراه تا تشخیص بیماری و تولید هنر، هیچ بخشی از زندگی بشر امروز از دسترس الگوریتم‌ها دور نمانده است. بدین ترتیب، واژه‌ای که روزگاری نام ریاضیدان ایرانی بود، اکنون به یکی از بنیادی‌ترین مفاهیم تمدن دیجیتال تبدیل شده است.

واژهٔ «مسئله» به معنای پرسشی که به دنبال پاسخی برای آن هستیم. موارد زیر دو مثال از مسئله هستند.

- مرتب کردن افزایشی فهرستی از چند عدد

- بررسی حضور عددی در فهرست داده شده

مسئله دارای متغیرهایی است که مقدار معینی نگرفته‌اند و معروف به پارامتر هستند.

- پارامترهای مرتب‌سازی: فهرست و طول آن
- پارامترهای جستجو: فهرست و طول آن و عدد منظور

به دلیل داشتن پارامتر، هر مسئله دارای رده‌ای است که هر یک با تخصیص مقداری به پارامتر تعیین می‌گردد. هر تخصیص خاص را یک نمونه می‌خوانیم. راه‌حل نمونه‌ای از یک مسئلهٔ پاسخ به پرسش مسئله در رابطه با آن نمونه است.

نمونه‌ای از مثال نخست

- فهرست $S = [۱۰, ۱۲, ۱, ۵, ۱۴, ۱۵, ۷]$ و $n = ۷$. پاسخ آن $[۱,۵,۷,۱۰, ۱۲,۱۴, ۱۵]$

نمونه‌ای از مثال دوم

- فهرست $S = [۱۰, ۱۲, ۱, ۵, ۱۴, ۱۵, ۷]$ و $n = ۷$ و x=۵. پاسخ آن «آری، x در S موجود است»

حال فرض کنید فهرست دارای هزاران عضو باشد. لزوما به شیوهٔ معمول نمی‌توان پاسخ را یافت. همچنین شیوه اقتضایی انسانی را نمی‌توان در رایانه اعمال کرد. جهت ایجاد برنامه رایانه‌ای که نمونه‌هایی از مسئله‌ای را حل کند، نیاز به تدوین رویهٔ عمومی و گام به گام جهت پاسخ‌دهی به هر نمونه داریم. رویه گام به گام را الگوریتم خوانیم و اظهار می‌کنیم که الگوریتم مسئله را حل می‌کند.

مثال- الگوریتم مثال دوم. با شروع از عضو اول فهرست x را با هر عضو فهرست مقایسه کن تا x یافته شود یا فهرست به اتمام رسد. اگر یافت شد پاسخ آری

وگرنه پاسخ نه. الگوریتم بالا دارای معایبی مانند پیچیدگی نوشتن و غیرصوری بودن است.

. پس تاکنون تعریفی شهودی از الگوریتم به دست داده شد. تعریف غیرصوری الگوریتم در رایانش «دستورالعملی است که مراحل مختلف انجام کاری را به زبان دقیق و با جزئیات کافی به نحوی بیان کند که ترتیب مراحل روشن و شرط خاتمهٔ عملیات نیز آشکار باشد.» پس، الگوریتم مجموعه‌ای متناهی از مراحل، غیرمبهم، اجراپذیر، و دارای شرط خاتمه است.

تمرین- آیا «حدس» الگوریتم است؟

زمینه‌های مطالعه الگوریتم: الگوریتم و مطالعهٔ آن به دلایل زیر انجام می‌پذیرد.

- طراحی الگوریتم: اعم از استقراء ریاضی، تقسیم و غلبه، حریصانه، برنامه‌نویسی پویا، پسگرد، انشعاب و تحدید. البته می‌توان نگاه به طراحی الگوریتم را به دو صورت دید، یا طبقه‌بندی بر اساس فنون همچون برنامه‌نویسی پویا و یا توجه به مسئله‌ای خاص مانند مرتب‌سازی و پیمایش گراف.
- اعتبارسنجی (اثبات درستی الگوریتم‌ها): الگوریتم در صورتی درست است که به ازای هر ورودیْ (هر نمونهْ) خروجی متناظر درست را پاسخ دهد.
- تحلیل الگوریتم (تحلیل مقدم، ارزیابی کارایی): الگوریتم در زمان اجرا روی پردازنده از حافظه جهت ذخیره‌سازی برنامه و داده‌ها استفاده می‌کند. تحلیل الگوریتم به فرایندی اطلاق می‌شود که میزان پیچیدگی زمان و پیچیدگی فضا اجرای الگوریتم را بررسی می‌کند.
- پیاده‌سازی
- آزمون برنامه شامل اشکال‌زدایی و پروفایلینگ برنامه (اندازه‌گیری کارایی و تحلیل موخر)

طراحی الگوریتم

- پروفایلینگ: اجرای صحیح برنامه بر مجموعه داده ورودی و تعیین زمان و فضای لازم برای محاسبه نتیجه

الگوریتم‌ها

- بازگشتی: هرگاه الگوریتم خود را فراخواند. شامل دو نوع مستقیم و غیرمستقیم
- غیربازگشتی

مثال فیبوناتچی- بازگشتی مستقیم

```
Alg. Fib(n):
        if n <= ۱:
                return n
        else:
                return Fib(n – ۱) + Fib(n – ۲)
```

مثال- ب‌م‌م دو عدد صحیح و مثبت و a>b>۰

```
Alg. BMM(a, b):
        If b == ۰:
                return a
        else:
                return BMM(b, a % b)
```

مثال- جستجوی کلید x در آرایه X[۰,...,n-۱]

```
Alg. Jostejo(X, n, i):
        if i >= n:
                return Hich
        else if X[i] == x:
                return i
        else:
                return Jostejo(X, n, i + ۱)
```

در حالی که حالت غیربازگشتی جستجو به صورت زیر است

مثال- یافتن بازگشتی بیشِ آرایه X[۰,...,n-۱]

```
Alg. Bish(X, n, i):
        if i < n - ١:
                j = Bish( X, n, i + ١)
                if X[i] > X[j]:
                        k = i
                else:
                        k = j
        else:
                k = n
        return k
```

تمرین- الگوریتم بازگشتی یافتن بیش آرایه را تحلیل کنید.

تمرین- شبه کد غیربازگشتی یافتن کلید در آرایه و همچنین یافتن بیش آرایه را بنویسید و با نمونه‌های بازگشتی مقایسه کنید.

الگوریتم کارآ و تحلیل الگوریتم‌ها

مثال- جستجوی ترتیبی در مقابل جستجوی دودوئی

در ادامه، یکی از مسائل مشهور الگوریتمی را بررسی می‌کنیم. گاهی اوقات دنبال یافتن مقداری هستیم. به فرایند مذکور «جستجو» گویند. از روش‌های زیر جهت جستجو در فهرست استفاده می‌شود.

```
adadha = [١٢٣, ٤٥, ٣٢, ٧٦, ٨٩, ٤٥, ٣٣, ٩٠, ١٧٨]

adadha.index(٤٥)
١
```

اندیس‌های آغاز و پایان بازهٔ جستجو را می‌توان تعیین کرد.

```
adadha = [۱۲۳, ٤٥, ۳۲, ۷٦, ۸۹, ٤٥, ۳۳, ۹۰, ۱۷۸]

adadha.index(٤٥, ٤, len(adadha))
۵
```

ایراد دستور بالا این است که در صورتی که مقدار در مجموعه نباشد خطا برمی‌گرداند. همچنین، از عملگرهای in و not in نیز جهت تعیین وجود مقداری در فهرست می‌توان بهره برد.

ناگفته پیداست جستجو در در بین رکوردها امری شایع در کارهای رایانه‌ای است. به به عنوان مثال، شماره دانشجویی جستجو می‌شود تا ریزنمرات شخص موردنظر بازیابی شود. در مثال‌های قبلی استفاده از توابع پیش ساختهٔ پایتون را جهت جستجو نام برده شد. در ادامه سعی بر پیاده‌سازی تابع آنها روی فهرست‌ها می‌کنیم. الگوریتم پایین فهرستی و عددی را به نام کلید در ورودی دریافت می‌کند و سعی می‌کند کلید مدنظر را از بین اعداد فهرست جستجو کند و در صورتی که موجود باشد اندیس متناظرش را برگرداند.

```
def jostejo(mqadir, kelid):
   """yaftan andis mqdar kelid dar fehrest vorodi."""

   for i in range(len(mqadir)):
      if mqadir[i] == kelid:
         return i
```

اجرای آن:

```
adadha = [۱, ۳, ۵, ۷, ۱۱, ۹]
jostejo(adadha, ۱۱)
۴
```

الگوریتم جستجو را به نحوی دیگر می‌توان نوشت.

```
def jostejo۲(mqadir, kelid):
   """yaftan andis mqdar kelid dar fehrest vorodi."""
   andis = -۱
```

```
  for i in range(len(mqadir)):
    if mqadir[i] == kelid:
      andis = i
      break
  if andis == -١:
    print('dar majmoe nist')
  return andis
```

اجرای آن:

```
adadha = [١, ٣, ٥, ٧, ١١, ٩]
jostejo٢(adadha, ١٢)
dar majmoe nist
-١
```

تمرین- تفاوت دو الگوریتم در چیست؟

در الگوریتم جستجوی اخیر در بدترین حالت به تعداد ورودی‌ها جستجو انجام خواهد شد. اما اگر فهرست مرتب باشد و مقادیر از کوچک به بزرگ مرتب باشند می‌توان بدترین حالت را به میزان زیادی کاهش داد. یکی از الگوریتم‌های مهم جستجو که دارای بدترین زمان جستجوی کمتری باشد «الگوریتم جستجو دودوئی» است. در الگوریتم مذکور در ابتدا تعداد فهرست مشخص و سپس بر اساس آن اندیس عدد میانه را تعیین می‌کند. مقدار کلید با عدد متناظر اندیس میانه مقایسه می‌شود. اگر برابر باشند، اندیس برگردانده می‌شود و اگر کوچکتر باشد در نیمه چپ فهرست و اگر کلید بزرگتر باشد در نیمهٔ راست فهرست جستجو ادامه خواهد یافت. عمل مذکور در هر نیمه ادامه خواهد یافت. در واقع هر دفعه جستجو در نیمی از فهرست ادامه می‌یابد. می‌توان نشان داد که در بدترین حالت لگاریتمی در مبنای دو مقایسه انجام خواهد پذیرفت. به عنوان مثال اگر طول فهرست ۱۰۰۰ باشد، در جستجوی معمولی در بدترین حالت ۱۰۰۰ مقایسه، ولی در جستجوی درختی حداکثر ۱۰ مقایسه انجام خواهد یافت که بهبودی بسیار مناسب است. در ادامه جستجوی دودوئی را معرفی می‌کنیم.

```
def jostejo_dodoee(adadha, kelid):
  """Jostejoye derkhti majomoe morattab jahat yaftan andis kelid"""
  chap = ۰
  rast = len(adadha) - ۱
  while chap <= rast:
    i_miani = int((chap + rast) / ۲ )
    if kelid > adadha[i_miani]:
      chap = i_miani + ۱
    elif kelid < adadha[i_miani]:
      rast = i_miani - ۱
    else:
      return i_miani
  payam = 'dar fehrest nabood'
  return payam
```

پس در ابتدا دو اندیس چپ و راست که معادل اندیس اولین مقدار و آخرین مقدار فهرستند را تعریف می‌کنیم. تا زمانی که چپ‌ترین اندیس از اندیس راست کوچکتر یا مساوی باشد حلقه را ادامه می‌دهیم. در هر تکرار حلقهٔ اندیس میانی برابر میانگین جز صحیح دو اندیس قبلی تعریف می‌شود. اگر کلید بزرگتر از مقدار متناظر اندیس میانی باشد، پرچم اندیس چپ برابر با مقدار بعدی اندیس میانی قرار می‌گیرد و اگر کلید کوچکتر باشد مقدار اندیس راست برابر با اندیس قبل از اندیس میانی قرار می‌گیرد. در غیر این صورت مقدار کلید و مقدار متناظر اندیس میانی برابرند، پس اندیس میانی برگشت داده می‌شود. در صورتی که حلقه به پایان برسد و اندیسی در حلقه برگردانده نشود به معنای نبود مقدار کلید در فهرست است. در نتیجه پیام نبود مقدار ایجاد می‌گردد. کد را برای مثال زیر اجرا می‌کنیم.

```
adadha_morattab = [۳, ۵, ۷, ۹, ۱۱, ۱۳, ۱۷]
andis = jostejo_dodoee(adadha_morattab, ۱۷)
print(andis)
۶
```

در مثال قبل دیدیم که فهرست مرتب چه تاثیر بزرگی بر بهینگی الگوریتم جستجو می‌گذارد.

جدول مقایسه تعداد مقایسه‌ها در دو روش جستجو ترتیبی و جستجو دودوئی را مشاهده کنید. توجه کنید به دلیل استفاده از ساختار فهرست (لیست) پایتون در عنوان اندازه فهرست نوشته شده است است ولی در حالت کلی اندازه آرایه نوشته می‌شود.

اندازه فهرست	تعداد مقایسه‌های جستجو ترتیبی	تعداد مقایسه‌های جستجو دودوئی
۱۲۸	۱۲۸	۸
۱۰۲۴	۱۰۲۴	۱۱
۱۰۴۸۵۷۵	۱۰۴۸۵۷۵	۲۱
۴۴۲۹۴۹۶۷۲۹۶	۴۴۲۹۴۹۶۷۲۹۶	۳۳

بنابراین سعی می‌شود که ساختار داده‌های چند مقداری مرتب باشند تا بتوان جستجو دودوئی را روی آنها اعمال کرد. در نتیجه سعی بر مرتب‌سازی چند مقداری‌ها می‌شود. همچنین، مرتب‌سازی تاثیر فراوان در دانش و فن رایانه داشته است. در مطالب آینده چند نوع الگوریتم مرتب‌سازی معرفی می‌شوند. تفاوت آنها در میزان استفاده بهینه منابع سیستم شامل حافظه و زمان است.

در ادامه، الگوریتم مثال تابع فیبوناتچی را بررسی می‌کنیم. عدد فیبوناتچی را به صورت زیر و بازگشتی حساب می‌کنیم:

```
def Fibonacci(n):
  if n == ۰:
    return ۰
  elif n == ۱:
    return ۱
  else:
```

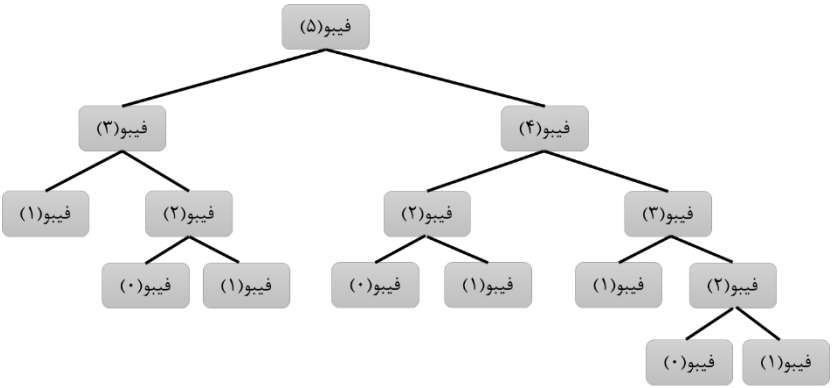
```
    return Fibonacci(n-۱) + Fibonacci(n-۲)

for i in range(۱۱):
  print('Fib(', i, ')=', Fibonacci(i))
```

برنامه بالا زمان زیادی را می‌گیرد و بسیاری از مقادیر را چندین بار محاسبه کرد. به سخن دیگر، تابع بازگشتی لزوما در همه‌جا دارای بهینگی زمان اجرا نیست.

مقدار n	۰	۱	۲	۳	۴	۵	۶
تعداد فراخوانی تابع	۱	۱	۳	۵	۹	۱۵	۲۵

درخت بازگشتی الگوریتم بالا در شکل زیر رسم شده است.



قضیه- تعداد موارد بازگشتی در درخت متناظر با الگوریتم بازگشتی فیبوناتچی

$$T(n) > ۲^{\frac{n}{۲}}, n \geq ۲$$

اثبات- تمرین با استقرا

می‌توان تابع فیبوناتچی را به صورت زیر نیز نوشت تا از مشکل حاصل از برنامهٔ بازگشتی جلوگیری کرد. روش زیر تکراری است.

```
def Fibo(n):
  f = [۰]*(n + ۱)
```

```
  f[۰] = ۱
  f[۱] = ۱
  for i in range(۲, n+۱):
    f[i] = f[i - ۱] + f[i - ۲]
  return f[n]
```

اجرای آن

```
for i in range(۱۱):
  print('Fib(', i, ')=', Fibo(i))
```

می‌توان از استفاده از فهرست نیز اجتناب کرد. پس برنامه فیبوناتچی را به صورت زیر بازنویسی می‌کنیم.

```
def fibo۳(n):
  f_qabli = ۱
  f_۲qabli = ۱

  if n == ۰ or n == ۱:
    return ۱
  for i in range(۲, n+۱):
    f = f_qabli + f_۲qabli
    f_۲qabli = f_qabli
    f_qabli = f

  return f
```

اجرای آن

```
for i in range(۱۱):
  print('fib(', i, ')=', fibo۳(i)
```

تمرین- زمان اجرای دو روش را بررسی کنید. فرض هر مقدار فیبوناتچی زمان $۱۰^{-۹}$ ثانیه داشته باشد.

الگوریتم بازگشتی فیبوناتچی و الگوریتم جستجو دودوئی از نوع تقسیم و غلبه هستند. الگوریتم تکراری فیبوناتچی از نوع برنامه‌ریزی پویاست. از چنین مثال‌هایی روشن می‌شود که فنون متفاوت و آشنایی با آن نقش مهم در طراحی الگوریتم پیدا می‌کند. هر یک در بعضی مسائل شیوه‌ای مناسبتر نسبت به دیگر شیوه‌ها است.

تحلیل الگوریتم-پیچیدگی زمانی

جهت تعیین میزان کارایی الگوریتم در حل مسئله، نیاز به تحلیل الگوریتم داریم. در کار قبلی به صورت غیرصوری کار تحلیل را پیش بردیم. روش صوری آن استفاده از تحلیل پیچیدگی است.

تحلیل الگوریتم نیازمند موارد زیر است:

- تعیین عملیات مورداستفاده شامل چهار عمل اصلی، دستورات مقایسه‌ای،خواندن و نوشتن، فراخوانی پردازه‌ها، دستورات انتساب
- تعیین مجموعه داده جهت بررسی همه حالات اعم از بهترین، متوسط، و بدترین زمان اجراء

در تحلیل الگوریتم تحلیل مقدم متفاوت از تحلیل موخر است. مورد متقدم به دنبال تعیین کران تابعی زمان اجراست و در مورد متأخر آمارهای مصرف زمان و حافظه را گزارش می‌کند. در درس جاری منظور از تحلیل مورد متقدم است. بنابراین ساده‌سازی‌هایی را مدنظر قرار می‌دهیم. ذخیره و بازیابی هر مقدار با دیگر مقادیر برابر است. زبان برنامه‌نویسی را وارد ماجرا نمی‌کنیم. افزودگی‌های مقداردهی‌های اولیه یا خواندن و نوشتن را در نظر نمی‌گیریم. در واقع هنگام بررسی کارایی الگوریتم به تعداد حقیقی چرخه‌های پردازنده توجه نمی‌کنیم. زیرا تعداد چرخه‌ها به رایانه خاص مرتبط است. علاوه بر آن به دنبال شمارش تمامی عملیات‌های انجام یافته نیز نیستیم. صرفا به دنبال آنهایی هستیم که از رایانه، زبان، و برنامه‌نویسی مستقل است. در مثال قبل

دیدیم که اندازه مسئله روی پاسخ تاثیر دارد. پس روش معمول استفاده از اندازه ورودی است. به دیگر سخن، روش معیار در تحلیل الگوریتم این است که زمان اجرای الگوریتم با افزایش اندازهٔ ورودی افزایش می‌یابد و کل زمان اجرا با تعداد اجرای عملیات پایه (مثلا مقایسه) تقریبا متناسب است. در نتیجه، کارایی الگوریتم با تعیین تعداد دفعاتی که عملیات پایه‌ای بر اساس تابعی از اندازه ورودی تحلیل می‌شود. صرفا سعی بر تحلیل مسئله بر اساس اندازه ورودی می‌شود. مثلا اگر تعداد ورودی‌های مسئله n باشد سعی بر گزارش تابعی از میزان زمان اجرا بر اساس مقدار مذکور می‌شود.

مثال-

الف	ب	ج

x = x + y	for i ∈ n:
	  x = x + y
		for i ∈ n:
		  for j ∈ n:
		  x = x + y

در حالت الف تعداد اجرا ۱، در حالت ب تعداد اجرا برابر n، و در حالت ج تعداد اجرا برابر با n^2 است. بنابراین مرتبه اجرای الگوریتم را برابر تعداد دفعات اجرای تک تک دستورالعمل‌ها تعریف می‌شود.

همچنین انتخاب ورودی نیز خود به نوع مسئله مرتبط است. اندازه ورودی را می‌توان به صورت‌های زیر تعریف کرد.

- تعداد اعضای فهرست مثل جستجو، مرتب‌سازی. یا تعداد سطر و ستون در ضرب ماتریسی.
- اندازه ورودی با لحاظ دو مقدار حاصل می‌شود. مثال، برای تعیین پیچیدگی الگوریتم‌های گراف دو مقدار تعداد راس و تعداد یال بررسی می‌شود.
- اندازه ورودی با تعداد علائم موردنیازتعیین شود. مثال، تعداد بیت‌ها در فیبوناتچی بازگشتی.

پس از تعیین اندازه ورودی، عملیات یا مجموعه عملیات‌هایی را که کار الگوریتم متناسب با تعداد اجرای آنهاست را انتخاب می‌کنیم. مجموعه اخیر را عملیات پایه می‌خوانیم و تحلیل پیچیدگی زمان الگوریتم برابر با تعداد تکرار عملیات پایه بر اساس مقدار اندازه ورودی است. فرض بر پیاده‌سازی بهینه عملیات پایه است. روش همیشه معینی برای انتخاب بهترین عملیات پایه وجود ندارد. بلکه حاصل تصمیم و تجربه است و به هنر تنه می‌زند. معمولا ساختار کنترلی بررسی نمی‌شود. گاهی اوقات یک تکرار از حلقه یک عملیات به حساب می‌آید. در نگاهی دقیق‌تر می‌توان به اجرای هر دستور ماشین را به عنوان عمل پایه در نظر گرفت. در مرتب‌سازی می‌توان دو عملیات مقایسه و عملیات تخصیص را در نظر گرفت. چون ممکن است که میزان سربار هر یک متفاوت باشد سعی بر بررسی هر دو مورد می‌شود.

چند مثال تحلیل پیچیدگی ساده را بررسی می‌کنیم. در مثال‌های مذکور تمامی داده دیده می‌شود تا عملیات و الگوریتم مدنظر انجام پذیرد. در این هنگام به دنبال یافتن مقدار زمان اجرا هستیم.

مثال- الگوریتم جمع اعداد: n تعداد اعداد. عمل اصلی جمع. فارغ از مقدار ورودی‌ها n تکرار حلقه رخ می‌دهد. پس $Z(n) = n$

مثال- مرتب‌سازی تبادلی

```
Alg. MorattabSazi_Tabadoli(X[۰,...,n-۱], n):
        for i ∈ [۰,..., n-۱]:
          for j ∈ [i+۱,...,n-۱]:
                if X[i] < X[j]:
                        X[i] ↔ X[j]
```

```
def MorattabSazi_Tabadli(X):
  for i in range(len(X)-۱):
    for j in range(i + ۱, len(X) - ۱):
      if X[i] < X[j]:
```

```
            X[i], X[j] = X[j] , X[i]
    return X



X = [۱۲, ۳, ۱, ۲۵, -۱, ٤٤, ۹, ۱۵]
MorattabSazi_Hobabi(X)
[-۱, ۱, ۳, ۹, ۱۲, ۱۵, ۲۵, ٤٤]
```

مقایسه عمل اصلی الگوریتم مرتب‌سازی تبادلی است.

$$Z(n) = (n-۱) + (n-۲) + (n-۳) + \cdots + ۱ = \frac{(n-۱)n}{۲}$$

ضرب ماتریس، عمل اصلی ضرب در حلقه داخلی $Z(n) = n^۳$

مثال- الگوریتم یافتن بیش

```
Alg. YaftanBish(X[۰,...,n-۱], n):
            i = ۱
            for j in [۲,..., n-۱]:
                if  X[i] < X[j]:
                        i = j
```

پیاده‌سازی الگوریتم یافتن بیش را در پایتون انجام می‌دهیم. ابتدا تابع یافتن بیش و اندیس متناظر آن را تعریف می‌کنیم.

```
def bish(feh):
    b = -۲e۷
    b_i = -۱
    for i in range(len(feh)):
        if feh[i] > b:
            b = feh[i]
            b_i = i
    return b_i, b
bish(adadha)
```

تمرین- اگر فهرست دارای مقدار ۳e۷- باشد الگوریتم بالا چگونه عمل خواهد کرد؟ الگوریتم بالا را به نوعی تغییر دهید که نیاز به مقداردهی پیش‌فرض نداشته باشد.

تمرین- چه روشی برای یافتن بیشینه فهرست با پیچیدگی کمتر پیشنهاد می‌دهید؟

تمرین- الگوریتم زیر روشی جهت مرتب‌سازی است که از خواص برنامه‌نویسی پایتون بهره می‌برد. مرتبه زمانی آن را بررسی کنید. تابع زیر اندیس مقدار بیشینه را برمی‌گرداند.

```
def andis_bish(feh):
    b = feh[۰]
    b_i = ۰
    for i in range(۱, len(feh)):
        if b < feh[i]:
            b = feh[i]
            b_i = i

    return b_i
```

تابع زیر با استفاده از روش‌های فهرست (لیست) ورودی را مرتب نزولی می‌کند.

```
def MoratabSazi_badvi(feh):
    feh_morattab = []
    while len(feh) > ۰:
        feh_morattab.append(feh.pop(andis_bish(feh)))
    return feh_morattab
```

اجرای آن

```
adadha = [۱۲۳, ٤٥, ۳۲, ۷٦, ٤٥, ۱۸۲, ۳۳, ٤٥, ۹۰, ۱۳٦]
MoratabSazi_badvi(adadha)
[۱۸۲, ۱۳٦, ۱۲۳, ۹۰, ۷٦, ٤٥, ٤٥, ٤٥, ۳۳, ۳۲]
```

مثال‌های قبلی صرفا وابسته به تعداد ورودی هستند. اما گاهی اوقات ممکن است که پیچیدگی زمانی صرفا وابسته به اندازه ورودی مسئله نباشد. بلکه ممکن است بسته به مقادیر ورودی نیز باشد. به دیگر سخن، و بسته به احتمال در موقعیتی در ابتدا در انتها یا در وسط مشاهده داده‌ها ختم شود. در بخش قبل مثال جستجو ذکر شد. الگوریتم جستجو ترتیبی از این نوع موارد است.

الگوریتم جستجوی ترتیبی:

بهترین زمان اجرا $\mathrm{z(n)} = ۱$ است. چرا؟ بدترین زمان اجرا آن برابر $z(n) = n$ است. متوسط زمان اجرا را به صورت امید ریاضی از طریق زیر بدست می‌آوریم.

$$z(n) = \sum_{k=0}^{n-۱} \left(k \times \frac{۱}{n}\right) = \frac{۱}{n}\sum_{k=۱}^{n} k = \frac{۱}{n}\frac{(n-۱)n}{۲} = \frac{n-۱}{۲}$$

در صورتی که p برابر با احتمال یافتن مقدار کلید در آرایه باشد زمان اجرای متوسط را می‌توان دقیق‌تر بدست آورد.

$$z(n) = \sum_{k=0}^{n-۱} \left(k \times \frac{p}{n}\right) + n(۱-p) = \frac{p}{n}\frac{(n-۱)n}{۲} + n(۱-p)$$
$$= n\left(۱ - \frac{p}{۲}\right) - \frac{p}{۲}$$

تمرین- اگر $\mathrm{p} = \frac{۱}{۲}$ مقدار چقدر خواهد شد؟ تفسیر آن چیست؟

ذکر این نکته ضروری است که میانگین به تنهایی برای تصمیم‌گیری مناسب نیست. ممکن است تمامی مقادیر حول و حوش میانگین باشند و یا دارای وردائی زیاد نسبت به میانگین باشند. در طراحی الگوریتم نیز باید به این مورد توجه کرد وگرنه ممکن است موجب خسارات فراوان شود.

مرتبه

گاه اوقات محاسبه دقیق لازم نیست و محاسبهٔ کران جهت تعیین پیچیدگی زمانی کفایت می‌کند. مثال $۰.۰۱n^۲ > ۱۰۰n$ از مقدار ۱۰۰۰۰ به بعد تابع درجه دو میزان زمان بیشتری را مصرف خواهد کرد. یا در جدول زیر تفاوت‌ها بین درجات مختلف محسوس و ولی با بزرگترین درجه یکسان در مقادیر بزرگتر نسبتا ناچیز است و می‌توان نادیده گرفته شوند. مثال زیر میزان رشد دو چندجمله‌ای درجه دو را نمایش می‌دهد.

n	۱۰	۲۰	۵۰	۱۰۰	۱۰۰۰
$۰.۱n^۲$	۱۰	۴۰	۲۵۰	۱۰۰۰	۱۰۰۰۰۰
$۰.۱n^۲ + n + ۱۰۰$	۱۲۰	۱۶۰	۴۰۰	۱۲۰۰	۱۰۱۱۰۰

نمادگذاری مجانبی: قبلا ذکر شد که مرتبه اجرای الگوریتم برابر تعداد دفعات اجرای تک تک دستورالعمل‌ها (دستورالعمل‌های پایه) تعریف می‌شود. تحلیل الگوریتم معمولا با استفاده از توابع مجانبی انجام می‌پذیرد. در ادامه نمادگذاری مجانبی را بررسی می‌کنیم.

نماد اُمیکرون بزرگ O (بدترین زمان اجرا):

$$f(n) = O\big(g(n)\big) \Leftrightarrow \exists c, n_۰ : \forall n \geq n_۰ \cdot \; ۰ \leq f(n) \leq cg(n)$$

مثال- $f(n) = ۲n^۳ + ۲n^۲ = O(n^۳)$؟ $c = ۴$ و $n_۰ = ۲$

مثال- $f(n) = ۲n^۳ + ۲n^۲ = O(n^۴)$؟ $c = ۴$ و $n_۰ = ۲$

معمولا به دنبال کوچکترین کران بالایی هستیم.

قضیه- اگر $f(n) = a_m n^m + a_{m-۱} n^{m-۱} + \cdots + a_۰, a_m > ۰$، آن‌گاه:
$f(n) = O(n^m)$

تمرین- اثبات کنید.

نتایج O

- اگر تعداد اجرا دستوری در الگوریتمی f(n) باشد آن‌گاه زمان اجرای آن از مرتبهٔ $O(n^m)$ است.
- اگر برنامه دارای k بخش با مرتبه بزرگی $O(nm_1)$، $O(nm_2)$، ...، $O(nm_k)$ باشد آن‌گاه مرتبه بزرگی برنامه $O(nm)$ است به طوری که $m =$ بیش$(m_1, m_2, \dots, m_k)$
- قانون جمع (حالت دوتائی)
- قانون حاصل‌ضرب: اگر $f_1(n) = O(g_1(n))$ و $f_2(n) = O(g_2(n))$، آن‌گاه $f_1(n)f_2(n) = O(g_1(n)g_2(n))$
- $O(f(n))O(g(n)) = O(f(n)g(n))$
- $f(n) = O(f(n))$
- $O(cf(n)) = O(f(n)), c > 0$
- $O(f(n)g(n)) = f(n)O(g(n))$
- $O(f(n)) + O(g(n)) = O(f(n) + g(n))$

مثال- $f(n) = (4n^3 + 5n^2 + 7n)(8logn)$ از $O(n^3logn)$

نماد امگا بزرگ Ω (بهترین زمان اجرا)

$$f(n) = \Omega(g(n)) \Leftrightarrow \exists c, n_0 : \forall n \geq n_0 > 0 \leq cg(n) \leq f(n)$$

مثال- $f(n) = 3n^3 + 2n^2 = \Omega(n^3)$؟ $c = 3$ و $n_0 = 1$

همچنین $3n^2$ و $3n$ نیز در مثال اخیر صدق می‌کنند ولی معمولا به دنبال بزرگترین کران پائین هستیم.

نماد تتا بزرگ Θ (میانگین زمان اجراء)

$$f(n) = \Theta\big(g(n)\big) \Leftrightarrow \exists c_1, c_2, n_0 : \forall n \geq n_0 \cdot 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$$

قضیه $f(n) = \Theta\big(g(n)\big)$ اگر و تنها اگر $f(n) = O\big(g(n)\big)$ و $f(n) = \Omega\big(g(n)\big)$

مثال- $f(n) = 3n^3 + 2n^2 = \theta(n^3)$؟

نماد o

$$f(n) = o\big(g(n)\big) \Leftrightarrow \exists c, n_0 : \forall n \geq n_0 \cdot 0 \leq f(n) < cg(n)$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

مثال- $7n^2 + 8n + 3 = o(n^3)$

مثال- $7n^2 + 8n + 3 = O(n^2)$ است ولی $7n^2 + 8n + 3 \neq o(n^2)$

مثال- $\log(n) = o(n)$

تمرین نشان دهید که الف- $n^b = o(a^n)$ ب- به ازای $a>b>0$ داریم $a^n = o(b^n)$. ج- $a^n = o(n!)$.

نماد ω

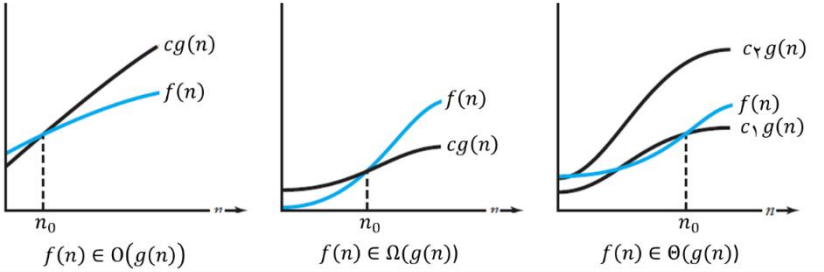
$$f(n) = \omega\big(g(n)\big) \Leftrightarrow \exists c, n_0 : \forall n \geq n_0 \cdot 0 \leq cg(n) < f(n)$$

قضیه $f(n) = \omega\big(g(n)\big)$ اگر و فقط اگر $g(n) = o\big(f(n)\big)$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

مثال- $7n^2 + 8n + 3 = \omega(n)$ است ولی $7n^2 + 8n + 3 \neq \omega\ (n^2)$

مثال- $n= \omega(\log n)$



مثال-

الف- $O(n^2)$

- $3n\log n + 8, 4n^2, 5n + 7, 6n^2 + 9, 2n\log n, n^2 + 2n$

ب- $\Omega(n^2)$

- $4n^2, 4n^3 + 3n^2, n^2 + 9, 6n^6 + n^4, 5n^2 + 2n, 2^n + 4n$

ج- $\Theta(n^2) = O(n^2) \cap \Omega(n^2)$

- $4n^2, 6n^2 + 9, 5n^2 + 2n$

قضیه

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\begin{cases}c, c>0, f(n)=\Theta(g(n)\\ 0, f(n)=o(g(n)\\ \infty, f(n)=\omega(g(n)\end{cases}$$

مثال- تحلیل الگوریتم مرتب‌سازی حبابی

```
Alg. MorattabSazi_Hobabi(X[۰,...,n-۱], n):
        for i in [۰,..., n-۲]:
          for j in [n-۱,..., i +۱]:
                If X[j-۱] > X[j]:
                        (X[j − ۱] ↔ X[j])
```

پیاده‌سازی

```
def MorattabSazi_Hobabi(X):
   for i in range(len(X)-۲):
      for j in range(len(X)-۱, i, -۱):
         if X[j - ۱] > X[j]:
            X[j - ۱], X[j] = X[j] , X[j - ۱]

   return X

X = [۱۲, ۳, ۱, ۲۵, -۱, ٤٤, ۹, ۱٥]
MorattabSazi_Hobabi(X)
[−۱, ۱, ۳, ۹, ۱۲, ۱٥, ۲٥, ٤٤]
```

بدترین حالت، بهترین حالت، حالت میانگین الگوریتم بالا را حساب کنید.

تحلیل الگوریتم مرتب‌سازی درجی

```
Alg. MorattabSazi_Darji(X[۰,...,n-۱], n):
        for j in [۱,..., n-۱]:
                mqdar = X[j]
                i = j - ۱
                while mqdar < X[i]:
```

```
X[i +١] = X[i]
i = i - ١
X[j + ١] = mqdar
```

```
def MorattabSazi_Darji(X):
  for j in range(١, len(X)):
    mqdar = X[j]
    i = j - ١
    while mqdar < X[i] and i >= ٠:
      X[i + ١] = X[i]
      i = i - ١
    X[i + ١] = mqdar
  return X
```

اجرای الگوریتم به صورت زیر است.

```
X = [١٢, ٣, ١, ٢٥, -١, ٤٤, ٩, ١٥]
MorattabSazi_Darji(X)
[-١, ١, ٣, ٩, ١٢, ١٥, ٢٥, ٤٤]
```

صرفا کافی است تعداد دفعات اجرای دستور مقایسه‌ای شمرده شود. بدترین حالت: هنگامی که حلقه داخلی همواره اجرا شود. در صورتی که آرایه نزولی مرتب شده باشد از مرتبه $O(n^{٢})$ است. بهترین حالت هنگامی است که حلقه داخلی هیچ وقت اجرا نشود یا داده‌ها به صورت صعودی مرتب شده باشند و در نتیجه از $\mathrm{O(n)}$ است. حالت میانگین آن نیز از مرتبه $O(n^{٢})$ است.

مرتبه‌های بزرگی معمول به صورت زیر هستند.

$$O(١) < O(\log n) < O(n) < O(nlogn) < O(n^{٢}) < O(n^{٣}) < O(٢^{n})$$

همچنین با داشتن دو متغیر n و m رابطه $٢^{n} \neq O(n^{m})$ همیشه برقرار است. اگر کران پایین از مرتبه نمایی دو باشد الگوریتم بسیار بد عمل می‌کند و همیشه دنباله حل مسئله نمایی و تبدیل آن به مسئلهٔ چندجمله‌ای هستیم.

خاصیت تابع‌های مقایسه مرتبه

- تعدی

$$f(n) = \Theta\big(g(n)\big), g(n) = \Theta\big(h(n)\big) \Longrightarrow f(n) = \Theta\big(h(n)\big)$$

$$f(n) = O\big(g(n)\big), g(n) = \mathrm{O}\big(h(n)\big) \Longrightarrow f(n) = \mathrm{O}\big(h(n)\big)$$

$$f(n) = \Omega\big(g(n)\big), g(n) = \Omega\big(h(n)\big) \Longrightarrow f(n) = \Omega\big(h(n)\big)$$

$$f(n) = \mathrm{o}\big(g(n)\big), g(n) = \mathrm{o}\big(h(n)\big) \Longrightarrow f(n) = \mathrm{o}\big(h(n)\big)$$

$$f(n) = \omega\big(g(n)\big), g(n) = \omega\big(h(n)\big) \Longrightarrow f(n) = \omega\big(h(n)\big)$$

- بازتابی

$$f(n) = \Theta\big(f(n)\big)$$

$$f(n) = O\big(f(n)\big)$$

$$f(n) = \Omega\big(f(n)\big)$$

- تقارن

$$f(n) = \Theta\big(g(n)\big) \Leftrightarrow g(n) = \Theta\big(f(n)\big)$$

- تقارن ترانهاده

$$f(n) = \mathrm{O}\big(g(n)\big) \Leftrightarrow g(n) = \Omega\big(f(n)\big)$$

$$f(n) = o\big(g(n)\big) \Leftrightarrow g(n) = \omega\big(f(n)\big)$$

سه علامت امیکرون و امگا و تتا را می‌توان با کوچکتر و بزرگتر و مساوی در دو مقایسه دو عدد متناظر دانست. مقایسهٔ دو عدد حقیقی a و b از سه حالت یا a=b یا a<b

یا a>b خارج نیستند که به سه شاخگی معروف است. هرچند سه شاخگی برای دو عدد ممکن است ولی برای توابع لزوما برقرار نیست، مثلا چنین رابطهٔ مجانبی بین n و $n^{۱+\sin n}$ وجود ندارد.

قضیه- اگر $f(n) = O(g(n)) - \Omega(g(n))$ آن‌گاه $f(n) = o(g(n))$

تمرین- تابع $f(n) = \begin{cases} n, n\text{زوج} \\ ۱, n\text{فرد} \end{cases}$ است. نشان دهید $f(n) = O(n) - \Omega(n)$ ولی $f(n) \neq o(n)$.

روش‌های تحلیل پیچیدگی مبتنی بر روابط بازگشتی

روش حدسی

مثال فاکتوریل- بازگشتی مستقیم

```
Alg. Factorial(n):
        if n <= ۱:
                return n
        else:
                return n * Factorial(n – ۱)
```

$$\begin{cases} z_n = z_{n-۱} + ۱ \\ z_۰ = ۰ \end{cases}$$

$$z_۱ = z_{۱-۱} + ۱ = z_۰ + ۱ = ۰ + ۱ = ۱$$

$$z_۲ = z_{۲-۱} + ۱ = z_۱ + ۱ = ۱ + ۱ = ۲$$

$$z_۳ = z_{۳-۱} + ۱ = z_۲ + ۱ = ۲ + ۱ = ۳$$

پایه استقراء

طراحی الگوریتم

$$z_{\cdot} = \cdot$$

فرض استقراء

$$z_n = n$$

گام استقراء

$$z_{n+1} = n + 1$$

$$z_{n+1} = z_{(n+1)-1} + 1 = z_n + 1 = n + 1$$

مثال-جستجو دودوئی

$$\begin{cases} z_n = z_{\frac{n}{2}} + 1 \\ z_1 = 1 \end{cases}$$

$$z_2 = z_{2/2} + 1 = z_1 + 1 = 1 + 1 = 2$$

$$z_4 = z_{4/2} + 1 = z_2 + 1 = 2 + 1 = 3$$

$$z_8 = z_{8/2} + 1 = z_4 + 1 = 3 + 1 = 4$$

$$z_{16} = z_{16/2} + 1 = z_8 + 1 = 4 + 1 = 5$$

$$z_n = \log n + 1$$

پایه

$$z_1 = \log 1 + 1 = 1$$

فرض

$$z_n = \log n + 1$$

حکم

$$z_{۲n} = \log ۲n + ۱$$

$$z_{۲n} = z_{۲n/۲} + ۱ = z_n + ۱ = \log n + ۱ + ۱$$
$$= \log n + \log ۲ + ۱ = \log ۲n + ۱$$

مثال

$$\begin{cases} z_n = ۷z_{\frac{n}{۲}} \\ z_۱ = ۱ \end{cases}$$

$$z_۲ = ۷z_{۲/۲} = ۷z_۱ = ۷$$

$$z_۴ = ۷z_{۴/۲} = ۷z_۲ = ۷^۲$$

$$z_۸ = ۷z_{۸/۲} = ۷z_۴ = ۷^۳$$

$$z_{۱۶} = ۷z_{۱۶/۲} = ۷z_۸ = ۷^۴$$

$$z_n = ۷^{\log n}$$

تمرین- اثبات کنید.

$$z_n = ۷^{\log n} = n^{\log ۷} \approx n^{۲.۸۱}$$

مثال-

$$\begin{cases} z_n = ۲z_{\frac{n}{۲}} + n - ۱ \\ z_۱ = ۰ \end{cases}$$

$$z_۲ = ۲z_{۲/۲} + ۲ - ۱ = ۲z_۱ + ۱ = ۱$$

$$z_۴ = ۲z_{۴/۲} + ۴ - ۱ = ۲z_۲ + ۳ = ۵$$

$$z_۸ = ۲z_{۸/۲} + ۸ - ۱ = ۲z_۴ + ۷ = ۱۷$$

$$z_{۱۶} = ۲z_{۱۶/۲} + ۱۶ - ۱ = ۲z_۸ + ۱۵ = ۴۹$$

عدم امکان یافتن حدسی بابت یافتن معادله بسته

حل بازگشتی با معادله مشخصه

بازگشت خطی همگن

تعریف-

$$a_۰ z_n + a_۱ z_{n-۱} + \cdots + a_k z_{n-k} = ۰$$

مثال

$$\begin{cases} z_n - ۵z_{n-۱} + ۶z_{n-۲} = ۰ \\ z_۰ = ۰ \\ z_۱ = ۱ \end{cases}$$

$$z_n = r^n$$

$$z_n - ۵z_{n-۱} + ۶z_{n-۲} = r^n - ۵r^{n-۱} + ۶r^{n-۲} = ۰$$

$$r^n - ۵r^{n-۱} + ۶r^{n-۲} = r^{n-۲}(r^۲ - ۵r + ۶) = ۰$$

$$r^۲ - ۵r + ۶ = (r-۳)(r-۲) = ۰$$

$$z_n = ۳^n, z_n = ۲^n$$

$$z_n = c۳^n + d۲^n$$

$$z_۰ = c + d = ۰$$

$$z_۱ = ۳c + ۲d = ۱$$

$$c = -۱, d = ۱ \Longrightarrow z_n = ۳^n - ۲^n$$

تعریف- معادلهٔ $a_۰ z_n + a_۱ z_{n-۱} + \cdots + a_k z_{n-k} = ۰$ دارای معادله مشخصهٔ $r^k + \alpha_۱ r^{k-۱} + \cdots + \alpha_k r^۰ = ۰$ است.

قضیه تعریف- معادلهٔ $a_۰ z_n + a_۱ z_{n-۱} + \cdots + a_k z_{n-k} = ۰$ با معادله مشخصهٔ $\alpha_۰ r^k + \alpha_۱ r^{k-۱} + \cdots + \alpha_k r^۰ = ۰$ دارای k ریشه متمایز باشد، پاسخ بازگشتی برابر است با

$$z_n = c_۱ r_۱^n + c_۲ r_۲^n + \cdots + c_k r_k^n$$

مثال

$$\begin{cases} z_n - ۳z_{n-۱} - ۴z_{n-۲} = ۰ \\ z_۰ = ۰ \\ z_۱ = ۱ \end{cases}$$

$$z_n = r^n$$

$$(r^۲ - ۳r - ۴) = ۰$$

$$r^۲ - ۳r - ۴ = (r-۴)(r+۱) = ۰$$

$$z_n = ۴^n, z_n = (-۱)^n$$

$$z_n = c۴^n + d(-۱)^n$$

$$z_۰ = c + d = ۰$$

$$z_۱ = ۴c - d = ۱$$

$$c = \frac{۱}{۵}, d = -\frac{۱}{۵} \Longrightarrow z_n = ۲^n$$

مثال فیبوناتچی- بازگشتی مستقیم

```
Alg. Fib(n):
        if n <= ۱:
                return n
        else:
                return Fib(n – ۱) + Fib(n – ۲)
```

حل- فرض می‌کنیم a_n تعداد فراخوانی‌های ***Fib(n)*** باشد. آن‌گاه

$$\begin{cases} a_n = a_{n-۱} + a_{n-۲} + ۱ \\ a_۰ = ۱, a_۱ = ۱ \end{cases}$$

حال $b_n = a_n + ۱$ و افزودن یک به طرفین معادلات داریم:

$$\begin{cases} b_n = b_{n-۱} + b_{n-۲} \\ b_۰ = ۲, b_۱ = ۲ \end{cases}$$

رابطه را به دو صورت می‌توان حل کرد.

الف- از معادله مشخصه استفاده می‌کنیم.

$$r^۲ - r - ۱ = ۰ \Longrightarrow r_۱, r_۲ = \frac{۱ \pm \sqrt{۵}}{۲}$$

پس

$$b_n = c_۱ \left(\frac{۱ + \sqrt{۵}}{۲}\right)^n + c_۲ \left(\frac{۱ - \sqrt{۵}}{۲}\right)^n$$

با اعمال شرایط مرزی داریم:

$$b_۰ = ۲ \Longrightarrow ۲ =$$

$$c_۱ + c_۲$$

$$b_۱ = ۲ \Longrightarrow ۲ = c_۱\left(\frac{۱+\sqrt{۵}}{۲}\right) + c_۲\left(\frac{۱-\sqrt{۵}}{۲}\right)$$

در نتیجه

$$c_۱ = -\left(\frac{۱-\sqrt{۵}}{۲}\right) \text{ و } c_۲\left(\frac{۱-\sqrt{۵}}{۲}\right)$$

و

$$b_n = \frac{۲}{\sqrt{۵}}\left(\frac{۱+\sqrt{۵}}{۲}\right)^{n+۱} - \frac{۲}{\sqrt{۵}}\left(\frac{۱-\sqrt{۵}}{۲}\right)^{n+۱}$$

و

$$a_n = \frac{۲}{\sqrt{۵}}\left(\frac{۱+\sqrt{۵}}{۲}\right)^{n+۱} - \frac{۲}{\sqrt{۵}}\left(\frac{۱-\sqrt{۵}}{۲}\right)^{n+۱} - ۱$$

در نهایت

$$a_n = ۲F_{n+۱} - ۱$$

F_n عدد n-ام فیبوناتچی است.

ب- با استفاده از سری مولد

$$\begin{cases} \mathrm{a_n} = a_{n-۱} + a_{n-۲} + ۱ \\ a_. = ۱, a_۱ = ۱ \end{cases}$$

سری مولد را به صورت زیر تعریف می‌کنیم:

$$A(x) = \sum_{n=0}^{\infty} a_n x^n$$

طراحی الگوریتم

$$A(x) = a_{.} + a_{۱}x + \sum_{n=۲}^{\infty} a_n x^n$$

$$A(x) = ۱ + x + \sum_{n=۲}^{\infty} (a_{n-۱} + a_{n-۲} + ۱)x^n$$

$$A(x) = \frac{۱ - x + x^{۲}}{(۱ - x - x^{۲})(۱ - x)}$$

قضیه- اگر معادله مشخصه بازگشتی خطی همگن دارای ریشه‌ای مضاعف باشد، آن‌گاه

$$z_n = r^n, z_n = nr^n, z_n = n^{۲}r^n, \dots, , z_n = n^{m-۱}r^n$$

مثال- حل بازگشتی

$$z_n - ۷z_{n-۱} + ۱۵z_{n-۲} - ۹z_{n-۳} = ۰$$

$$z_{.} = ۰, z_{۱} = ۱, z_{۲} = ۲$$

۱- یافتن معادله مشخصه

$$r^{۳} - ۷r^{۲} + ۱۵r - ۹ = ۰$$

۲- حل معادله مشخصه

$$r^{۳} - ۷r^{۲} + ۱۵r - ۹ = (r - ۱)(r - ۳)^{۱} = ۰$$

۳- اعمال قضیه

$$z(n) = a۱^n + b۳^n + cn۳^n$$

۴- استفاده از مقدار اولیه جهت تعیین ضرائب

$$z_. = ۰ = a۱^۰ + b۳^۰ + c(۰)۳^۰$$

$$z_۱ = ۱ = a۱^۱ + b۳^۱ + c(۱)۳^۱$$

$$, z_۲ = ۲ = a۱^۲ + b۳^۲ + c(۲)۳^۲$$

$$a + b = ۰$$

$$a + ۳b + ۳c = ۱$$

$$a + ۹b + ۱۸c = ۲$$

۵- معادله نهایی

$$z(n) = (-۱)(۱^n) + (۱)(۳^n)\left(-\frac{۱}{۳}\right)(n۳^n) = -۱ + ۳^n - n۳^{n-۱}$$

تمرین- قضیه را اثبات کنید.

تمرین-

$$z(n) = \ ۵z(n - ۱) - ۷z(n - ۲) + ۳z(n - ۳), n > ۲$$

$$z(۰) = ۱, z(۱) = ۲, z(۲) = ۳$$

بازگشتی خطی ناهمگن

تعریف-

$$a_{\cdot}z_n + a_{۱}z_{\mathrm{n-۱}} + \cdots + a_k z_{\mathrm{n-k}} = f(n)$$

روش عمومی شناخته شده جهت حل معادله ناهمگن یافت نشده است. پس صرفا توابعی خاصی را بررسی می‌کنیم. مثالی زیر از موارد خاص است.

$$a_{\cdot}z_n + a_{۱}z_{\mathrm{n-۱}} + \cdots + a_k z_{\mathrm{n-k}} = b^n p(n)$$

حالت خاص اخیر را می‌توان با تبدیل آن به بازگشتی خطی همگن حل کرد. مثال زیر را ببینید.

مثال- $z_n - ۳z_{n-۱} = ۴^n, z_{\cdot} = ۰, z_{۱} = ۴$

۱- جانشینی n با $\mathrm{n-۱}$

$$z_{n-۱} - ۳z_{n-۲} = ۴^{n-۱}$$

۲- تقسیم دو طرف معادله اصلی بر ۴

$$\frac{z_n}{۴} - \frac{۳z_{n-۱}}{۴} = ۴^{n-۱}$$

۳- دو معادله بخش ۱ و ۲ دارای سمت راست یکسان هستند. پس سمت چپ معادله‌ها هم با هم برابر است و در نتیجه داریم،

$$\frac{z_n}{۴} - \frac{۷z_{n-۱}}{۴} + ۳z_{n-۲} = ۰$$

یا

$$z_n - ۷z_{n-۱} + ۱۲z_{n-۲} = ۰$$

امکان حل با معادله بازگشتی خطی همگن

$$r^{۲} - ۷r + ۱۲ = (r - ۳)(r - ۴۴) = ۰$$

$$z(n) = a۳^n + b۴^n$$

با اعمال شرایط اولیه داریم

$$z(n) = +۴^{n+۱} - ۴(۳^n)$$

پس دارای دو جواب ۳^n و ۴^n است. پاسخ نخست از جواب همگن حاصل شده است. پاسخ دوم از بخش ناهمگن معادله بازگشتی حاصل می‌شود.

قضیه معادله بازگشتی خطی ناهمگن به صورت

$$a_۰ z_n + a_۱ z_{n-۱} + \cdots + a_k z_{n-k} = b^n p(n)$$

را می‌توان به معادله خطی باگشتی همگن با معادله مشخصی زیر

$$(\alpha_۰ r^k + \alpha_۱ r^{k-۱} + \cdots + \alpha_k r^۰)(r - b)^{d+۱} = ۰$$

تبدیل کرد که d درجه $p(n)$ است. پس معادله مشخصه دارای دو بخش معادله مشخصه متناظر بازگشت همگن و ضریب حاصل از بخش ناهمگن معادله بازگشتی است.

مثال -

$$z(n) - ۳z(n-۱) = ۴^n(۲n + ۱), n > ۱$$

$$z(۰) = ۰, z(۱) = ۱۲$$

۱- یافتن معادله مشخصه از بخش همگن معادله

$$z(n) - ۳z(n-۱) = ۰ \Longrightarrow r - ۳ = ۰ \Longrightarrow r = ۳$$

۲- یافتن بخش متناظر ناهمگن معادله

$$(r-b)^{d+۱} = (r-۴)^{۱+۱} = (r-۴)^{۲}$$

۳- اعمال قضیهٔ اخیر و حل معادله

$$(r-۳)(r-۴)^{۲} = ۰$$

۴- اعمال قضیه قبلی

$$z(n) = a۳^n + b۴^n + cn۴^n$$

۵- با بررسی شرایط اولیه

$$z(n) = ۲۰(۳^n) - ۲۰(۴^n) + ۸(n۴^n)$$

تمرین- $z(n) - z(n-۱) = n - ۱, z(۰) = ۰$

تمرین- قضیه را اثبات کنید.

تغییر متغیر (تبدیل دامنه)

مثال-

$$Z(n) = Z\left(\frac{n}{۲}\right) + ۱, Z(۱) = ۱$$

$$n = ۲^k, k = \log n$$

$$Z(۲^k) = Z\left(\frac{۲^k}{۲}\right) + ۱ = Z(۲^{k-۱}) + ۱$$

$$z_k = Z(۲^k)$$

$$z_k = z_{k-۱} + ۱$$

$$z_k = k$$

$$Z(۲^k) = z_k = k \Longrightarrow Z(n) = \log n$$

$$Z(n) = ۱ + logn$$

مثال-

$$Z(n) = ۲Z\left(\frac{n}{۲}\right) + n - ۱, Z(۱) = ۰$$

$$n = ۲^k, k = \log n$$

$$Z(۲^k) = ۲Z\left(\frac{۲^k}{۲}\right) + ۲^k - ۱ = ۲Z(۲^{k-۱}) + ۲^k - ۱$$

$$z_k = Z(۲^k)$$

$$z_k = ۲z_{k-۱} + ۲^k - ۱$$

$$z_k = ۲^k + k۲^k$$

$$Z(۲^k) = ۲^k + k۲^k = k \Longrightarrow Z(n) = n + n\log n$$

$$Z(n) = nlogn - (n - ۱)$$

تمرین-

$$Z(n) = ۷Z\left(\frac{n}{۲}\right) + ۱۸\left(\frac{n}{۲}\right)^۲, Z(۱) = ۰$$

حل معادلات بازگشتی با جانشینی

در صورتی که نتوان با روش‌های قبل به پاسخ رسید می‌توان از صنعت جانشینی بهره برد.

مثال-

$$z_n = z_{n-۱} + \frac{۲}{n}, z_۱ = ۰$$

$$z_n = z_{n-۱} + \frac{۲}{n}$$

$$z_{n-۱} = z_{n-۲} + \frac{۲}{n-۱}$$

$$z_{n-۲} = z_{n-۳} + \frac{۲}{n-۲}$$

$$\vdots$$

$$z_۲ = z_۱ + \frac{۲}{۲}$$

$$z_۱ = ۰$$

$$z_n = z_{n-۱} + \frac{۲}{n}$$

$$= z_{n-۲} + \frac{۲}{n-۱} + \frac{۲}{n}$$

$$= z_{n-۳} + \frac{۲}{n-۲} + \frac{۲}{n-۱} + \frac{۲}{n}$$

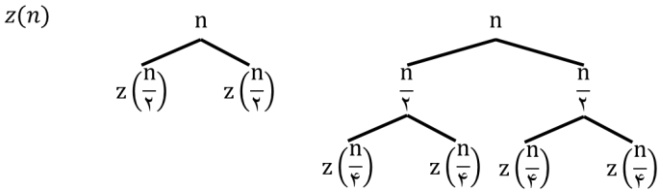
$$\vdots$$

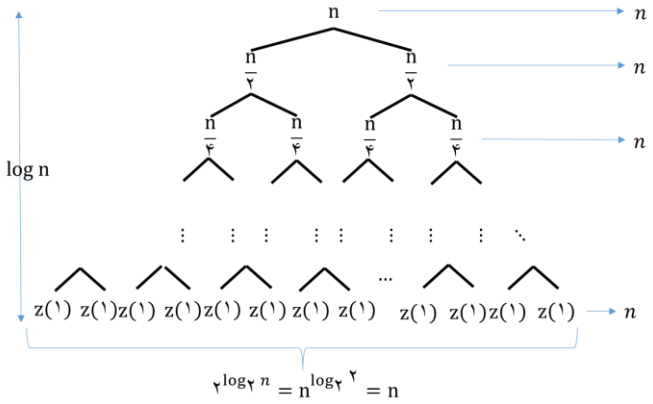
$$= ۰ + \frac{۲}{۲} + \frac{۲}{n-۲} + \frac{۲}{n-۱} + \frac{۲}{n}$$

$$= ۲\sum_{i=۲}^{n}\frac{۱}{i} = ۲\ln n$$

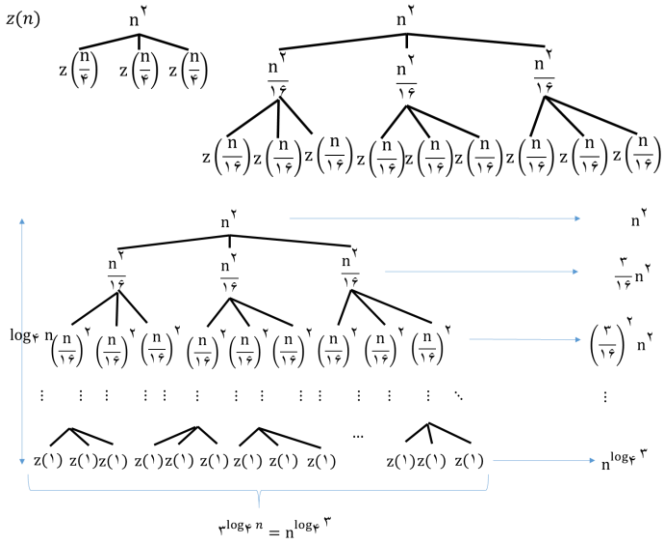
روش درختی

$$z(n) = ۲z\left(\frac{n}{۲}\right) + \theta(n)$$





$$z(n) = ۳z\left(\frac{n}{۴}\right) + \theta(n^۲)$$



$$z(n) = n^2 + \frac{3}{16}n^2 + \left(\frac{3}{16}\right)^2 n^2 + \cdots + \left(\frac{3}{16}\right)^{\log_4 n} n^2 + \theta\left(n^{\log_4 3}\right)$$

$$= \sum_{i=0}^{\log_4 n} \left(\frac{3}{16}\right)^i n^2 + \theta\left(n^{\log_4 3}\right)$$

$$< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i n^2 + \theta\left(n^{\log_4 3}\right)$$

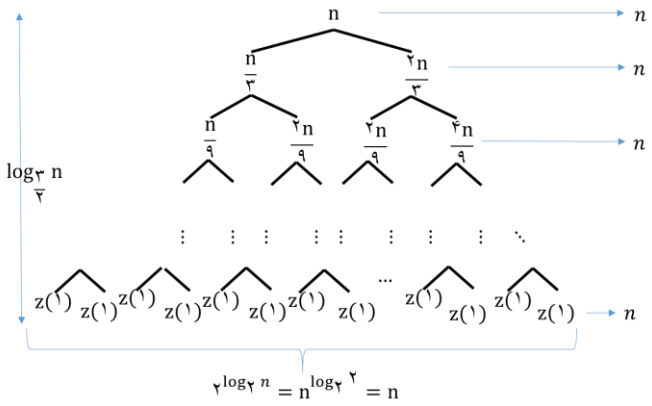
$$= \frac{1}{1 - \left(\frac{3}{16}\right)} n^2 + \theta\left(n^{\log_4 3}\right)$$

$$= \frac{16}{13} n^2 + \theta\left(n^{\log_4 3}\right)$$

$$= O(n^{۲})$$

مثال نامنظم

$$T(n) = T\left(\frac{n}{۳}\right) + T\left(\frac{۲n}{۳}\right) + \theta(n)$$



$$nlogn$$

قضیه اصل حل روابط تقسیم و غلبه. فرض کنیم که

$$\begin{cases} z(n) = az\left(\frac{n}{b}\right) + cn^{k} \\ t(۱) = \theta(۱) \end{cases}$$

که a و b و c اعداد مثبت و $a \geq ۱$ و $b > ۱$، آن‌گاه

الف- اگر $b^{k} < a$ آن‌گاه $z(n) = \theta\left(n^{\log_b a}\right)$

ب- اگر $b^k = a$ آن‌گاه $z(n) = \theta\left(n^k \log n\right)$

ج- اگر $b^k > a$ آن‌گاه $z(n) = \theta\left(n^k\right)$

اثبات-

نکته: درصورت جانشینی $z(n) = az\left(\frac{n}{b}\right) + cn^k$ با $z(n) \leq at\left(\frac{n}{b}\right) + cn^k$ ($z(n) \geq at\left(\frac{n}{b}\right) + cn^k$) نتایج با O یا Ω جانشین خواهد شد.

مثال- $z(n) = ۸z\left(\frac{n}{۴}\right) + ۵n^۲$

$$۸ < ۴^۲ \Longrightarrow z(n) \in \Theta(n^۲)$$

مثال- $z(n) = ۹z\left(\frac{n}{۳}\right) + ۵n^۱$

$$۹ > ۳^۱ \Longrightarrow z(n) \in \Theta\left(n^{\log_۳ ۹}\right) = \Theta(n^۲)$$

مثال- $z(n) = ۲z\left(\frac{n}{۲}\right) + ۱$

مثال- $z(n) = ۲z\left(\frac{n}{۲}\right) + n$

مثال- $z(n) = z\left(\frac{۲n}{۳}\right) + ۱$

مثال- $z(n) = ۳z\left(\frac{n}{۳}\right) + n^۲$

قضیه عمومی حل روابط تقسیم و غلبه

فرض کنیم که معادله بازگشتی زمان اجرای الگوریتمی عبارت است از

$$\begin{cases} z(n) = az\left(\frac{n}{b}\right) + f(n), n = b^m \\ z(۱) = \theta(۱) \end{cases}$$

که a و b و c اعداد مثبت و $a \geq ۱$ و $b > ۱$، C>۰ و f(n)>۰ آن‌گاه

الف- اگر $f(n) = o\left(n^{\log_b a}\right)$ آن‌گاه $z(n) = \theta\left(n^{\log_b a}\right)$

ب- اگر $f(n) = \theta\left(n^{\log_b a}\right)$ آن‌گاه $z(n) = \theta\left(n^{\log_b a} \log n\right)$

ج- اگر $f(n) = \omega\left(n^{\log_b a}\right)$ آن‌گاه $z(n) = \theta\left(f(n)\right)$

مثال- $z(n) = ۱۰۰z\left(\frac{n}{۹۹}\right) + \log n!$

مثال- $z(n) = ۸z\left(\frac{n}{۹}\right) + n \log n$

مثال- $z(n) = ۴z\left(\frac{n}{۲}\right) + ۵n^۲ + nlog\ n + ۶n$

لم- اگر در رابطه $z(n) = az\left(\frac{n}{b}\right) + f(n)$ داشته باشیم $f(n) = \theta\left(n^{\log_b a} \log^j n\right)$ آن‌گاه $z(n) = \theta\left(n^{\log_b a} \log^{j+۱} n\right)$

تمرین- اثبات کنید.

بازگشت اکرا-باتزی

$$z(n) = f(n) + \sum_{i=۱}^{k} a_i z(\frac{n}{b_i})$$

$$z(n) = \theta(n^p) + \theta\left(n^p \int_{n'}^{n} \frac{f(x)}{x^{n+1}} dx\right)$$

تحلیل احتمالی الگوریتم‌ها

در بسیاری از الگوریتم‌های جدید، رفتار الگوریتم نه تنها به ورودی بلکه به انتخاب‌های تصادفی درون الگوریتم نیز وابسته است. این امر باعث می‌شود تحلیل کلاسیکِ زمانی و فضایی، که صرفاً بر اساس تعداد عملیات در بدترین، میانگین، یا بهترین حالت انجام می‌شد، برای درک کامل رفتار چنین الگوریتم‌هایی کافی نباشد. بنابراین لازم است با مفاهیم تحلیل احتمالی آشنا شد که با آن، عملکرد الگوریتم با استفاده از ابزارهای احتمال و آمار مورد بررسی قرار می‌گیرد. در این بخش، مفاهیم کلیدی متغیرهای تصادفی، امید ریاضی، احتمال صحت، کران‌های خطا، اطمینان، و تکنیک‌های تقویت را مختصرا معرفی می‌کنیم. مفاهیم مزبور زیربنای تحلیل الگوریتم‌هایی نظیر مرتب‌سازی سریع تصادفی، الگوریتم‌های درهم‌ساز، فیلتر بلوم، و امثالهم هستند.

متغیرهای تصادفی

در تحلیل الگوریتم‌های تصادفی، یک مرحلهٔ کلیدی آن است که رفتار الگوریتم را با متغیر تصادفی مدل کنیم. متغیر تصادفی مقداری است که نتیجهٔ یک فرآیند تصادفی را نمایش می‌دهد. مثلاً ممکن است زمان اجرای الگوریتم، تعداد تصادم‌های درهم، تعداد گام‌ها، یا حتی خروجی الگوریتم را با متغیر تصادفی مدل کنیم. برای نمونه، اگر در الگوریتم مرتب‌سازی سریع تصادفی، محور به طور تصادفی انتخاب شود، تعداد مقایسه‌ها متغیری تصادفی است که بر اساس انتخاب‌های الگوریتم تغییر می‌کند. در تحلیل، معمولاً این متغیر را با X نمایش می‌دهیم و هدف درک بهتری رفتار آماری آن به جای حالتی خاص از آن است.

امید ریاضی

امید ریاضی مهم‌ترین ابزار در تحلیل الگوریتم‌های تصادفی است. امید ریاضی یک متغیر تصادفی X میانگین وزنی مقادیر ممکن آن است، طوری که وزن‌ها احتمال وقوع هستند. به بیان ساده‌تر، امید ریاضی روشن می‌کند «اگر الگوریتم بی‌نهایت بار اجرا شود، میانگین زمان اجرا یا تعداد عملیات چقدر خواهد بود؟»

مثال- تحلیل مرتب‌سازی تصادفی: در مرتب‌سازی معمولی، بدترین حالت $O(n^2)$ و بهترین حالت $O(n\log n)$ است. اما درصورتی که محور را به صورت تصادفی انتخاب کنیم، زمان اجرا به ورودی وابسته نیست و امید ریاضی زمان اجرا برابر با $E[T(n)] = O(n\log n)$ است.

اثبات: احتمال اینکه هر دو عنصر با هم مقایسه شوند برابر $\frac{2}{j-i+1}$ است. با جمع زدن روی همه جفت‌ها:

$$E[T(n)] = \sum_{i=0}^{n}\sum_{i=0}^{n}\frac{2}{j-i+1} = O(n\log n)$$

در الگوریتم‌هایی مثل Count-Min Sketch نیز، مقدار خطا در تخمین فراوانی عناصر با استفاده از امید ریاضی تحلیل می‌شود.

احتمال صحت

در الگوریتم‌های تصادفی همیشه تنها زمان اجرا مهم نیست. بلکه باید دانست که «الگوریتم با چه احتمالی جواب درست تولید می‌کند؟»

الگوریتم‌های احتمالی از نظر صحت به دو دسته مهم تقسیم می‌شوند:

الف) الگوریتم‌های مونت‌کارلو (Monte Carlo) که همیشه سریع هستند، اما احتمال دارد خروجی آن‌ها نادرست باشد، و میزان خطا معمولاً کوچک و قابل کنترل است. مثال: فیلتر بلوم که می‌تواند مثبت کاذب بدهد، اما منفی کاذب ندارد.

ب) الگوریتم‌های لاس‌وگاس (Las Vegas): همیشه جواب درست تولید می‌کنند، اما زمان اجرای آن‌ها تصادفی است. مثال: انتخاب سریع تصادفی که میانگین اجرا سریع است، ولی در موارد نادر ممکن است کند شود. در تحلیل، معمولاً احتمال سلامت خروجی را با نمادهایی چون $Pr[\text{صحت}]$ یا $1 - \delta$ نمایش می‌دهیم، که δ احتمال خطا است.

کران‌های خطا

یکی از ویژگی‌های کلیدی تحلیل احتمالی آن است که به کمک آن می‌توانیم محدودهٔ خطا یا دقت خروجی را با روابط ریاضی دقیق بیان کنیم. این کران‌ها معمولاً از مفاهیم نامساوی مارکوف، نامساوی چبیشف، کران چرنوف، و قانون اعداد بزرگ (LLN) به‌دست می‌آیند. این ابزارها کمک می‌کنند فهم دقیقی از رفتار الگوریتم داشته باشیم و بدانیم خطا هرگز از حد معقولی بیشتر نخواهد شد. برای کران‌زدن خطا، از نامساوی‌های زیر استفاده می‌شود.

الف) نامساوی مارکوف: برای متغیر تصادفی غیرمنفی X و $a > 0$:

$$P(X \geq a) \leq \frac{E[X]}{a}$$

کاربرد: اگر میانگین زمان اجرا $O(n\log n)$ باشد، احتمال اینکه زمان اجرا از $100n\log n$ بیشتر شود حداکثر $\frac{1}{100}$ است.

ب) نامساوی چبیشف:

$$P(|X - E[X]| \geq a) \leq \frac{Var(X)}{a^2}$$

کاربرد: در الگوریتم‌های مونت‌کارلو برای کنترل ورودایی کاربرد دارد.

اطمینان و تقویت

در بسیاری از الگوریتم‌ها، می‌توان با تکرار مستقل الگوریتم و ترکیب نتایج، احتمال خطا را به صورت چشمگیری کاهش داد. به این فرایند Amplification تقویت احتمال موفقیت گفته می‌شود. تقویت پیروزی بر این اساس است که اگر اجرای الگوریتم احتمال موفقیت p داشته باشد، تکرار چندین بارهٔ آن احتمال موفقیت را به فاصله‌ای دلخواه از ۱ می‌رساند. برای مثال: درصورتی که الگوریتمی با احتمال ۸۰ درصد پاسخ صحیح را تولید کند، اجرای ده باره و تصمیم بر اساس رای اکثرت احتمال خطای نهایی را به صورت نمایی کاهش می‌دهد. ، با اجرای ۱۰ بار و گرفتن رأی اکثریتْ احتمال خطای نهایی به شکل نمایی کوچک می‌شود.

قضیه (تقویت چرنوف-هوفدینگ): اگر الگوریتمی با احتمال $p = \frac{۱}{۲} + \gamma$ پاسخ درست را برگرداند، با t بار تکرار و رأی اکثریت:

$$P(\text{خطا}) \leq e^{-۲\gamma^۲ t}$$

مثال- برای $p = ۰.۶$ یا $\gamma = ۰.۱$ و $t = ۱۰۰$ احتمال خطا برابر است با

$$P(\text{خطا}) \leq e^{-۲\times۰.۰۱\times۱۰۰} = e^{-۲} \approx ۰.۱۳۵$$

با $t = ۲۰۰$ آن‌گاه خطا برابر است با $pr \leq e^{-۴} \approx ۰.۰$.

این روش در تحلیل بسیاری از ساختارها مثل LSH و الگوریتم‌های احتمال‌محور بسیار مهم است. در LSH، تعداد باندها (bands) و تعداد درهم‌سازها (rows per band) دقیقاً نقش تقویت‌کنندهٔ احتمال پیدا کردن همسایه‌های نزدیک را دارد.

تحلیل احتمالی الگوریتم‌ها به ما امکان می‌دهد تا زمان اجرای الگوریتم‌های تصادفی را با **امید ریاضی** بسنجیم، احتمال **خطا** یا **صحت** را با کران‌هایی مانند مارکف، چبیشف، و چرنوف کنترل کنیم، با **تقویت**، احتمال موفقیت را به مقدار دلخواه نزدیک

به ۱ برسانیم، و ساختارهای داده‌ای تقریبی مانند Bloom Filter، Count-Min Sketch، HyperLogLog، MinHash و LSH را طراحی و تحلیل کنیم.

پیوست مقدمات ریاضی

خاصیت یکنوائی:

- تابع $f(n)$ افزایشی یکنوا است اگر $m \leq n$ آن‌گاه $f(m) \leq f(n)$.
- تابع $f(n)$ کاهشی یکنوا است اگر $m \leq n$ آن‌گاه $f(m) \geq f(n)$.
- تابع $f(n)$ اکیدا افزایشی یکنوا است اگر $m < n$ آن‌گاه $f(m) < f(n)$.
- تابع $f(n)$ اکیدا کاهشی یکنوا است اگر $m < n$ آن‌گاه $f(m) > f(n)$.

سقف و کف

- $\lfloor n \rfloor = n = \lceil n \rceil$
- $x - ۱ < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + ۱$
- $-\lceil x \rceil = \lfloor -x \rfloor$ یا $-\lfloor x \rfloor = \lceil -x \rceil$
- $\left\lceil \frac{n}{۲} \right\rceil + \left\lfloor \frac{n}{۲} \right\rfloor = n$
- $\left\lceil \frac{\lceil \frac{n}{a} \rceil}{b} \right\rceil = \left\lceil \frac{n}{ab} \right\rceil$
- $\left\lfloor \frac{\lfloor \frac{n}{a} \rfloor}{b} \right\rfloor = \left\lfloor \frac{n}{ab} \right\rfloor$
- $\left\lceil \frac{a}{b} \right\rceil \leq \frac{(a+(b-۱))}{b}$
- $\left\lfloor \frac{a}{b} \right\rfloor \geq \frac{(a+(b-۱))}{b}$
- $\lfloor n + x \rfloor = n + \lfloor x \rfloor$

- $\lceil n + x \rceil = n + \lceil x \rceil$

ریاضی پیمانه

- $a\%n = a - n\left\lfloor \frac{a}{n} \right\rfloor$
- $۰ \leq a\%n < n$

نمایی‌ها

- $\lim_{n\to\infty} \frac{n^b}{a^n} = ۰$
- $n^b = o(a^n)$
- $e^x = ۱ + x + \frac{x^۲}{۲} + \cdots + \frac{x^n}{n!} + \cdots$
- $۱ + x \leq e^x$
- اگر $|x| \leq ۱$، آن‌گاه $۱ + x \leq e^x \leq ۱ + x + x^۲$
- $e^x = ۱ + x + \theta(x^۲)$
- $\lim_{n\to\infty} \left(۱ + \frac{x}{n}\right)^n = e^x$

لگاریتم

- $a = b^{\log_b a}$
- $\log_b a = \frac{\log_c a}{\log_c b}$
- $\log_b \left(\frac{۱}{a}\right) = -\log_b a$
- $\log_b a = \frac{۱}{\log_a b}$
- $a^{\log_b n} = n^{\log_b a}$
- اگر $x < y$ آن‌گاه $\log x < \log y$

دوجمله‌ای‌ها

- $(x+y)^n = \sum_{r=0}^{n} \binom{n}{r} x^{n-r} y^r$
- $\binom{n}{r} = \binom{n}{n-r}$
- $\binom{n}{r} = \binom{n-1}{r} + \binom{n-1}{r-1}$
- $\sum_{r=0}^{k} \binom{m}{r}\binom{n}{k-r} = \binom{m+n}{k}$

فاکتوریل‌ها

- $n! = o(n^n)$
- $n! = \omega(2^n)$
- $\log(n!) = \Theta(n \log n)$
- $n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \theta\left(\frac{1}{n}\right)\right)$
- $n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\alpha_n}$ به طوری که $\frac{1}{12n+1} < \alpha_n < \frac{1}{12n}$
- معادلهٔ استرلینگ $n! \cong \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$

سری‌ها

$$\sum_{r=0}^{n} r = \frac{n(n+1)}{2}$$

$$\sum_{r=0}^{n} a^r = \frac{a^{n+1}-1}{a-1}$$

$$\sum_{r=0}^{\infty} a^r = \frac{1}{1-a}, 0 < a < 1$$