دانشگاه صنعتی اصفهان

دانشکده برق و کامپیوتر

جزوه ضمیمه درس ساختمان داده‌ها و الگوریتم‌ها

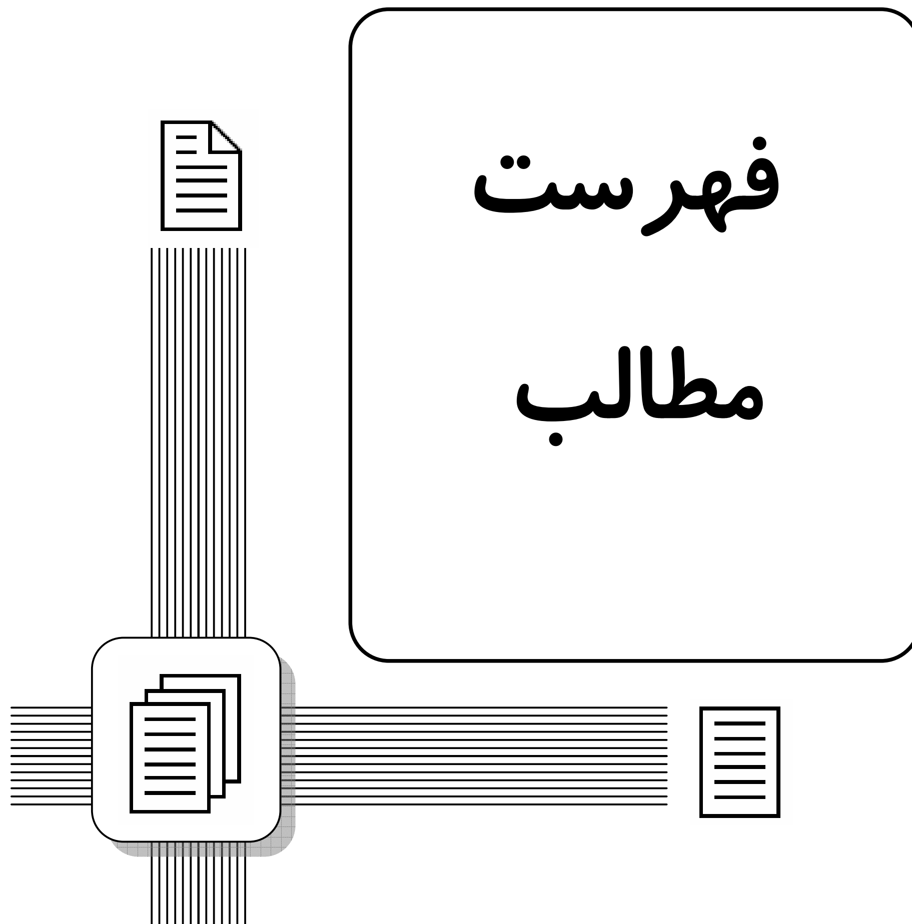
نگاهی گذرا به تحلیل الگوریتم‌ها

با تاکید بر رویکرد ***تحلیل زمانی***

به انضمام مقدمه‌ای بر درس طراحی الگوریتم

نگارش نخست: پاییز ۱۳۸۵

بسم الله الرحمن الرحیم


فهرست
مطالب

فهرست مطالب

پیشگفتار ۵

الگوریتم ۷

تاریخچه ۷

مفهوم الگوریتم ۸

تعریف دقیق الگوریتم ۹

تفاوت الگوریتم و برنامه ۱۰

مقدمه‌ای بر تحلیل الگوریتم‌ها ۱۰

بررسی الگوریتم‌ها از دیدگاه میزان حافظه مورد نیاز ۱۲

بررسی الگوریتم‌ها از دیدگاه زمان مورد نیاز ۱۴

مروری بر تکنیک‌های تبدیل روابط بازگشتی به روابط غیر بازگشتی: ۱۶

علائم مجانبی ۱۸

مثال‌ها ۲۰

از هرچه بگذریم سخن نمره خوش‌تر است ۲۳

ضمیمه) مقدمه‌ای بر درس طراحی الگوریتم ۲۴

الگوریتم‌ها مرتب‌سازی ۲۵

پیشگفتار

به نام رب خدای پشته و صف	خـدای فـوق پـیـونـد مـجوّف
خدای الگوریتم و استـراکـچر	خـداونـد عـلـیـم و حیّ قـادر
خداوند مـرتـب‌سـازی و درج	که عالم را مرتب کرد بی‌خرج
خدای هر درخت و هر گرافی	دگر بس می‌کنم حرف اضافی

آن‌چه که پیش روی شماست حاصل کوششی است در جهت آموزش نحوه تحلیل الگوریتم‌ها از نظر زمان اجرا. ابتدا اجازه بدهید به این نکته بپردازم که چرا این مبحث و چرا در این درس؟ مبحث تحلیل زمان اجرای الگوریتم‌ها یکی از مباحث موجود در کتاب‌هایی است که توسط شورای عالی انقلاب فرهنگی به عنوان مرجع برای درس ساختمان داده‌ها و الگوریتم‌ها معرفی شده‌اند، بنابراین پرداختن یا نپرداختن به آن مانند هر مبحث دیگری از مباحث درس ساختمان داده است.

بسیاری از مباحث موجود در رشته مهندسی کامپیوتر در درس‌های گوناگون وجود داشته و تکرار می‌شوند، بنابراین اساتید این درس‌ها با توافقاتی که بین خود انجام می‌دهند و یا طبق سنّت‌هایی که از گذشته به صورت نانوشته اجرا می‌شده‌اند، ممکن است یک یا چند مبحث را از یک درس حذف نموده و در عوض به یک یا چند مبحث دیگر به طور وسیع‌تری بپردازند، بدون آن که دانشجو از آشنایی با آن مبحث محروم شود.

مبحث تحلیل زمانی الگوریتم‌ها و به طور کلی تحلیل و طراحی الگوریتم‌ها در درس طراحی الگوریتم به طور گسترده مورد بررسی قرار می‌گیرد، به همین دلیل چند سالی است که یا این مبحث در درس ساختمان داده مورد بررسی قرار نمی‌گیرد و یا آن‌که به اشاره‌ای گذرا به آن بسنده می‌شود.

مشکلی که در این میان به وجود آمده این است که دانش‌جویان گرایش سخت‌افزار ملزم به گذراندن درس طراحی الگوریتم نیستند و به عنوان یک درس اختیاری نیز کم‌تر به این درس رغبت نشان می‌دهند، از سوی دیگر از مبحث تحلیل زمانی الگوریتم‌ها در آزمون کارشناسی ارشد ذیل عنوان درس ساختمان داده هر ساله سوالاتی مطرح می‌شود، بنابراین سر مهندسین سخت‌افزار این وسط بی‌کلاه می‌ماند! از سوی دیگر آشنایی با نحوه تحلیل الگوریتم‌ها فواید زیادی برای مهندسین سخت‌افزار داشته و در دست‌یابی به گستره‌ی دید لازم در طراحی‌هایشان آن‌ها را یاری خواهد نمود. علاوه بر این مهندسین نرم‌افزار نیز در صورت آشنایی با مقدمات طراحی الگوریتم در درس ساختمان داده، در درس طراحی الگوریتم مطالب را بهتر خواهند آموخت.

عوامل فوق باعث شد تا برآن شوم تا مقدمات ارائه این مبحث در کنار درس ساختمان‌داده به صورت پررنگ‌تر را فراهم نموده و سعی نمایم انگیزه‌ی لازم را برای گذراندن درس طراحی الگوریتم در یک

فضای رقابتی برای تمامی دانشجویان مهندسی کامپیوتر (چه سخت‌افزار، چه نرم‌افزار و چه حتی فناوری اطلاعات) ایجاد نمایم.

در این جزوه ابتدا به بررسی مقدمات این مبحث می‌پردازیم سپس گریزی به رابطه‌های بازگشتی و نحوه تبدیل آن‌ها به روابط غیر بازگشتی می‌زنیم و سپس بحث تحلیل زمانی را ادامه می‌دهیم. در قسمت پایانی جزوه نگاهی به روش‌های طراحی الگوریتم داشته و شما را به نمونه‌هایی از هر یک از این روش‌ها آشنا می‌نماییم و سعی می‌کنیم شما را هر چه بیش‌تر با مباحث شیرین طراحی الگوریتم آشنا سازیم.

در تدوین این جزوه از جزوه‌های درسی استاد ارجمند سرکار خانم آقایی مربوط به درس‌های ساختمان گسسته و طراحی الگوریتم استفاده شده است، همچنین کتاب‌های گوناگون درس و تست مرتبط با درس ساختمان داده هر جا که لازم بوده است مورد استفاده قرار گرفته‌اند، و نهایتا از دانشنامه‌های ویکی‌پدیا و رشد نیز استفاده شده است.

از آن‌جا که این نخستین بار است که این جزوه مورد استفاده قرار می‌گیرد و از سوی دیگر در تهیه‌ی آن مشکل محدودیت زمانی وجود داشته است، امکان وجود اشکال در آن طبیعی به نظر می‌رسد. شما می‌توانید با ارائه‌ی نظرات پیشنهادات و انتقادات خود در هر چه پربارتر شدن این جزوه این‌جانب را یاری نمایید و زمینه را برای هر چه کامل‌تر شدن آن فراهم کنید. جهت تماس با این‌جانب می‌توانید از نشانی پست الکترونیکی mohsennowruzi@gmail.com **استفاده نموده و برای دریافت و مشاهده نسخه الکترونیکی این جزوه می‌توانید از نشانی** www.computerarea.persiangig.com **استفاده نمایید.**

تا چه قبول افتد و چه در نظر آید

محسن نوروزی

دانشگاه صنعتی اصفهان – پاییز ۱۳۸۵

الگوریتم

پس از بررسی مساله و شناخت آن لازم است تا روشی برای حل مساله ارایه شود. این روش‌ها می‌توانند با استفاده از تجارب قبلی در حل مسایل و به دلخواه کسی که مساله را حل می‌کند به دست آید یا با استفاده از روش‌های علمی مبتنی بر تحلیل‌های ریاضی و منطقی صورت پذیرد .الگوریتم در واقع روشی است که مسایل را با استفاده از تحلیل‌های ریاضی و منطقی مورد بررسی قرار داده و راه‌حل مناسبی برای آن ارایه می‌کند.

یک الگوریتم مجوعه‌ی متناهی از دستورالعمل‌های خوش تعریف برای انجام یک عمل است که با داشتن یک حالت اولیه به حالت پایانی مشخص و متناظری خواهد رسید. الگوریتم، مجموعه‌ای متناهی از دستورالعمل‌هاست که به صورت دقیق و بدون ابهام بیان شده‌اند و اگر به ترتیب خاصی اجرا شوند، مساله حل می‌شود. به عبارت دیگر، الگوریتم روشی گام به گام است که برای حل مساله به کار می‌رود.

تاریخچه

واژه الگوریتم از نام *محمد ابن موسی خوارزمی* ریاضی دان قرن نهم گرفته شده است. کتاب معروف *الجبر و المقابله* خوارزمی که حاوی دستورالعمل‌های مختلف برای حل مسایل محاسباتی است از راه ترجمه اسپانیایی آن در اروپا شناخته شد و نام عربی او، الخوارزمی، (از طریق آوانگاری آن در زبان اسپانیایی و سپس ورود آن به دیگر زبان‌های اروپایی) مترادف شد با "دستورهای حل مسایل".

کلمه *الگوریتم حساب* در اصل تنها به قوانین انجام محاسبات با اعداد عربی اطلاق می شد، اما در قرن ۱۸ به "الگوریتم" بسط یافت. در حال حاضر این کلمه شامل تمام روش های معین حل مساله یا انجام یک کار می شود. اولین الگوریتم نوشته شده برای کامپیوتر، یادداشت هایی بر موتورهای تحلیلی از ادا بایرون «Ada Byron» بود که در سال 1842 میلادی نوشته شد و به خاطر آن، بسیاری او را اولین برنامه‌نویس می‌دانند. به هر حال، چون چارلز بابیج هرگز موتور تحلیلی خود را کامل نکرد، این الگوریتم بر آن اجرا نشد. نبود دقت ریاضی درتعریف "رویه‌های خوش تعریف"[1] مشکلاتی را برای ریاضی‌دان‌ها، و منطق‌دانان قرن ۱۹ و اوایل قرن ۲۰ پدیدآورد. این مشکل تا حد زیادی با معرفی ماشین تورینگ، مدلی انتزاعی از کامپیوتر که توسط الن تورینگ تنظیم شد، و این بیان که هر روش توصیف "رویه‌های خوش تعریف" با یک ماشین تورینگ قابل شبیه‌سازی است، رفع شد (این جمله به قضیه Church-Turing معروف است)(با ماشین تورینگ و سایر مطالب مربوط به آن در درس نظریه زبان‌ها و ماشین‌ها به صورت کامل آشنا خواهید شد). تعریف رسمی امروزی یک الگوریتم این است: یک الگوریتم، رویه ای است که بر یک ماشین تورینگ کاملا خاص و یا یکی از شکل‌های مشابه‌اش قابل اجرا باشد.

1-well-defined routines

مفهوم الگوریتم

به مجموعه‌ای از دستورالعمل‌ها که با زبان دقیق و قابل فهم به همراه جزئیات لازم وبه صورت مرحله به مرحله به گونه‌ای اجرا شده که هدف خاصی را دنبال می‌کند و شروع وخاتمه آن‌ها نیز مشخص باشد الگوریتم می‌گویند. به عبارت دیگر می‌توان الگوریتم را به یک ماشین تشبیه کرد که مقادیر معلوم را دریافت کرده روی آن محاسباتی انجام می‌دهد ودر پایان خواسته‌های مساله را ارایه می‌کند.

همان‌طور که می‌بینید رابطه نزدیکی بین مفهوم الگوریتم ونحوه‌ی کار کامپیوتر در حل مسایل وجود دارد، بنابر این با استفاده از روش الگوریتم می‌توانید حل مسایل را به گونه‌ای طراحی کنید که برای تبدیل به زبان کامپیوتر نیز قابل فهم باشد.

الگوریتم در اصطلاح به معنی و مفهوم روش تکنیک و شیوه حل یک مساله است. الگوریتم یکی از مهم‌ترین مفاهیم در کامپیوتر می‌باشد که به فهرستی از دستور العمل‌ها اطلاق می‌شود که برای حل یک مساله می‌بایست گام به گام طی نمود.

برای آشنایی بیش‌تر با الگوریتم بد نیست به این نکته توجه کنید که همه در زندگی روزمره با مسایل مختلفی سروکار دارند که به طریقی نسبت به حل آن‌ها اقدام می‌کنند. در برخورد با یک مساله شیوه‌های گوناگونی برای حل وجود دارد. حل مساله را می‌توان به طور کلی دارای پنج گام دانست.

۱- تعیین نیازهای مساله ۲-تجزیه وتحلیل مساله۳-طراحی الگوریتمی برای حل مساله

۴-پیاده سازی الگوریتم ۵- بررسی وآزمودن نتیجه.

شناسایی نیازهای مساله ما را در رسیدن به روش مناسبی برای حل آن یاری می‌کند. به علاوه در این مرحله ابزارهای مورد نیاز نیز شناخته می‌شوند. گاهی ممکن است به این نتیجه مهم برسید که در جهت حل مساله به همکاری افرادی نیاز دارید که اطلاعات بیشتری در مورد آن دارند. پس از شناسایی کامل مساله نوبت به تجزیه وتحلیل آن می‌رسد. در این گام ورودی‌های مساله یا همان مفروضات داده شده را می‌شناسیم سپس خروجی یا پاسخ مساله را تعیین می‌کنیم و بالا خره روش حل و ابزارهای این کار را انتخاب می‌نماییم. اما بحث اساسی از گام سوم به بعد آغاز می‌شود.

مثال:

شخصی را در نظر بگیرید که تصمیم به تعویض یک چرخ ماشین پنچر شده خود دارد پس الگوریتم همانند زیر طراحی می‌شود:

۱. شروع
۲. جک را زیر اتومبیل بگذارید
۳. پیچ‌های چرخ پنچر شده را باز کنید
۴. چرخ را خارج کنید

۵. چرخ یدکی را جای آن بگذارید
۶. پیچ‌ها را محکم کنید
۷. اگر پیچ‌ها سفت نشده‌اند به مرحله ۶بروید
۸. جک را پایین بیاورید
۹. پایان

تعریف دقیق الگوریتم

اکنون که با الگوریتم تا حدی آشنا شدیم اجازه بدهید تعریف ارایه شده برای الگوریتم را از کتاب ساختمان داده‌ها به زبان c (تالیف الیس هارویتز و دیگران) با هم مرور کنیم:

الگوریتم: مجموعه‌ی محدودی از دستورالعمل‌ها است که با دنبال کردن آن‌ها هدف خاصی حاصل می‌شود.(به عبارت دیگر نوعی توالی از گام‌هایی است که هدف آن حل مساله می‌باشد)

هر الگوریتم باید دارای خصوصیات زیر باشد:

۱. ورودی: هر الگوریتم باید هیچ و یا چند ورودی داشته باشد.
۲. خروجی: هر الگوریتم باید حداقل یک خروجی داشته باشد.
۳. قطعیت: وضوح و خالی بودن از هرگونه ابهام.
۴. محدودیت: هر الگوریتم باید پس از طی مراحل محدودی (با پایان) خاتمه یابد.(از طریق هر کدام از مسیرهای الگوریتم).
۵. کارآیی: انجام‌پذیر بودن هر دستورالعمل(قابلیت اجرا به صورت دستی با قلم و کاغذ).

اجازه بدهید تعریف فوق و معیارهای ذکر شده را در یک جمله متمرکز کنیم(این جمله از جزوه درس طراحی الگوریتم نقل می‌شود): مجموعه محدودی از دستورالعمل‌ها که مراحل انجام کاری را با دقت و جزئیات کافی بیان کند، به طوری که ترتیب اجرای مراحل و شرط خاتمه کار مشخص باشد، الگوریتم نام دارد.

🔔: امکان طرح سوال از تعریف و معیارهای ذکر شده فوق در آزمون کارشناسی ارشد وجود دارد.

✎: عبارت "خالی بودن از هرگونه ابهام در مورد دستورالعمل " بیان‌کننده کدام خاصیت الگوریتم است؟

۱. ابهام ۲. قطعیت ۳. محدودیت ۴. کارآیی

🕮: صحیح بودن گزینه ۲ واضح است!

تفاوت الگوریتم و برنامه

هر الگوریتم باید حتما پایان‌پذیر باشد(محدودیت(مورد ۴ از معیارهای فوق)) اما یک برنامه لزوما پایان پذیر نیست. به عنوان نمونه سیستم عامل برنامه‌ای است که هیچ‌گاه پایان نمی‌پذیرد(به جز در مواقع خرابی یا خاموش شدن سیستم) و دائما در یک سیکل انتظار است تا برنامه بعدی وارد شود.

🔔: امکان طرح سوال از نکته فوق در آزمون کارشناسی ارشد وجود دارد.

✎: تفاوت برنامه و الگوریتم در این است که:

۱. الگوریتم باید دارای قطعیت باشد ولی برنامه می‌تواند دارای ابهام باشد.
۲. الگوریتم باید پایان پذیر باشد اما برنامه لزوما پایان‌پذیر نیست!
۳. الگوریتم می‌تواند ورودی نداشته باشد اما یک برنامه حتما ورودی دارد.
۴. برنامه و الگوریتم دو مفهوم معادل هستند.

📖: صحیح بودن گزینه ۲ واضح است!

مقدمه‌ای بر تحلیل الگوریتم‌ها

برای حل هر مساله ممکن است الگوریتم‌های متعددی وجود داشته باشد که هر کدام مزایا و معایب خود را دارد، در عین حال علاوه بر این‌که برای یک مساله خاص راه‌حل‌های گوناگونی وجود دارد، به طور کلی نیز روش‌های گوناگونی برای طراحی الگوریتم برای مسایل مختلف وجود دارد.

🕯 سوال: فکر می‌کنید چند الگوریتم مختلف برای عمل ضرب دو عدد وجود دارد؟

🗐تمرین : تمام الگوریتم‌هایی که برای ضرب می‌شناسید را نوشته و برای هر کدام از آن‌ها معیارهای ۵ گانه‌ای که برای الگوریتم ذکر شد را مورد بررسی قرار دهید.

احتمالا در حین انجام تمرین فوق این سوال به ذهن شما خطور کرده است که واقعاً کدام الگوریتم برای ضرب دو عدد بهترین الگوریتم است و چرا؟ خوب کافی است کمی صبر داشته باشید تا با معیارها و روش‌های تحلیل الگوریتم‌ها آشنا شوید سپس خواهید توانست این الگوریتم‌ها را از دیدگاه‌های گوناگون با هم مقایسه کرده و الگوریتم برتر را با توجه به نوع کاربرد مورد نظرتان انتخاب نمایید.

عوامل گوناگونی در ارزیابی الگوریتم‌ها موثرند، به عبارت دیگر برای تحلیل یک الگوریتم می‌توان از منظرهای گوناگونی به الگوریتم نگریست، اما معمولا الگوریتم‌ها را از دو دیدگاه زمان و حافظه مورد نیاز مورد تجزیه و تحلیل قرار می‌دهند، به عبارت دیگر منظور از تحلیل الگوریتم‌ها تعیین کارآیی الگوریتم برحسب زمان مورد نیاز و فضای حافظه مورد نیاز است، بدون آن‌که نیاز به پیاده‌سازی الگوریتم باشد.

نکته‌ی ظریفی نیز در این بین وجود دارد، ممکن است بگویید که در مورد هر الگوریتم علاوه بر موارد فوق نحوه پیاده‌سازی آن هم اهمیت دارد، مثلا یک الگوریتم را ممکن است دو برنامه‌نویس به دو روش گوناگون پیاده‌سازی نمایند. از سوی دیگر این‌که یک الگوریتم با چه زبانی پیاده‌سازی شود و روی چه سخت‌افزاری مورد اجرا قرار گیرد نیز در تحلیل آن بی‌تاثیر نیست. اجازه بدهید تمام این عوامل را تحت عنوان پیاده‌سازی دسته‌بندی کنیم.

بنابراین در تحلیل یک الگوریتم سه ویژگی حافظه مورد نیاز، زمان اجرا و نیز پیاده‌سازی موثر هستند. معمولا با توجه به پیچیدگی‌های موجود در بررسی پیاده‌سازی در سطح کارشناسی از آن صرف نظر کرده و بیشتر به دو مورد دیگر می‌پردازند.

اکنون که وارد این بحث شدیم اجازه بدهید بگویم که قسمتی از عوامل که مربوط به تعیین حافظه مورد نیاز و زمان اجرای یک الگوریتم به صورت مستقل از سخت‌افزار است قسمت مهمی از علم کامپیوتر است که تحت عنوان تئوری پیچیدگی از آن یاد می‌شود. قسمت دیگر عوامل که سنجش و اندازه‌گیری اجرای برنامه نامیده می‌شود، کارآیی و زمان اجرای الگوریتم را با توجه به سخت‌افزاری که روی آن اجرا می‌شود ارائه می‌کند. در ادامه به بررسی بخش‌هایی از تئوری پیچیدگی می‌پردازیم و در مورد بخش دوم تنها به اشاراتی کوتاه بسنده می‌کنیم.

قبل از آنکه وارد بررسی میزان حافظه و زمان لازم برای الگوریتم‌ها بپردازیم اجازه بدهید با ذکر یک مثال اهمیت تحلیل الگوریتم‌ها را بررسی کنیم. مشخص شدن اهمیت الگوریتم‌ها باعث می‌شود تا با شور و شوق بیش‌تری به ادامه بحث توجه کنید پس ابتدا به این مثال توجه کنید:

✐: فرض کنید الگوریتمی داریم که روی نمونه‌ای به اندازه n به میزان $۲^n \times ۱۰^{-۴}$ ثانیه زمان نیاز دارد. جدول زیر زمان اجرا برای n های گوناگون نشان می‌دهد.

زمان مورد نیاز(تقریبی)	مقدار n
۰/۱ ثانیه	۱۰
۲ دقیقه	۲۰
۱ شبانه روز	۳۰
۱ سال!	۳۸

حال فرض کنید که مهندسین کامپیوتر با تلاش فراوان کامپیوتری ساخته‌اند که سرعتی معادل ۱۰۰ برابر سرعت کامپیوتر فوق دارد، در این صورت زمان اجرا به صورت $۲^n \times ۱۰^{-۶}$ ثانیه تغییر می‌کند و حل یک مساله با نمونه $n=۴۵$ به یک سال زمان نیاز دارد، به عبارت دیگر افزایش سرعت سخت‌افزار تنها ۷ عدد به تعداد نمونه در طول یک سال افزوده است.

حال فرض کنید که مهندسین کامپیوتر به جای کار روی سخت‌افزار روی الگوریتم کار کرده و برای این مساله الگوریتمی طرح کرده‌اند که برای نمونه n به زمانی معادل $n^3 \times 10^{-2}$ ثانیه نیاز دارد. حال به جدول زیر توجه کنید:

زمان مورد نیاز(تقریبی)	مقدار n
۱۰ ثانیه	۱۰
۱ دقیقه	۲۰
۵ دقیقه	۳۰
۱ سال!	۱۵۰۰

می‌بینید که کارآیی به نحو قابل توجهی تغییر کرده است. این مثال میزان تاثیر تحلیل و بهینه نمودن الگوریتم را مشخص می‌کند.

بررسی الگوریتم‌ها از دیدگاه میزان حافظه مورد نیاز:

به طور کلی میزان حافظه مورد نیاز یک برنامه را به دو دسته نیازمندی‌های فضای ثابت و نیازمندی‌های فضای متغیر تقسیم می‌کنند:

۱) نیازمندی‌های فضای ثابت: فضای مورد نیاز برای الگوریتم که به تعداد و اندازه ورودی و خروجی بستگی ندارد. نیازمندی‌های ثابت شامل فضای دستورالعمل‌ها، فضای لازم برای متغیرهای ساده، ساختارها و ثابت‌ها است.

۲) نیازمندی‌های فضای متغیر: در این‌جا میزان حافظه بستگی به نمونه‌ای که الگوریتم بر روی آن اجرا می‌شود دارد، به عبارت دیگر این مقدار حافظه تابعی از اندازه نمونه است، همچنین نیاز به فضای اضافی برای توابع بازگشتی را نیز دربرمی‌گیرد. برای الگوریتم p که روی نمونه I اجرا می‌شود معمولا فضای متغیر مورد نیاز را با $S_p(I)$ نشان می‌دهند. اگر بخواهیم ساده‌تر بگوییم نیازمندهای فضای متغیر به تعداد، اندازه و مقادیر ورودی و خروجی مربوط به نمونه I بستگی دارد.

در صورتی که نیازمندی‌های فضای کل مورد نیاز برای الگوریتم p را با $S(p)$ نشان دهیم، خواهیم داشت: $S(p)=c + S_p(I)$ که در آن c نیازمندی‌های فضای ثابت و $S_p(I)$ نیازمندی‌های فضای متغیر است. همان‌طور که پیشتر گفتیم تاکید ما در این‌جا روی بخش‌های خاصی از تحلیل الگوریتم‌ها است، در این مورد هم بیشتر نیازمندی‌های فضای ثابت مورد نظر ما است چرا که نیازمندی‌های فضای متغیر

ممکن است به زبان برنامه‌نویسی که الگوریتم با آن پیاده می‌شود بستگی داشته باشد. اجازه بدهید با چند مثال بحث را روشن‌تر کنم:

✐: فرض کنید که یک الگوریتم سه عدد صحیح را به عنوان ورودی دریافت کرده و حاصل‌جمع آن‌ها را به عنوان خروجی برمی‌گرداند. این الگوریتم تنها نیازمندی‌های فضای ثابت دارد و نیازمندی‌های فضای متغیر آن برابر صفر است.

✐: فرض کنید الگوریتم مورد نظر ما یک آرایه از اعداد را دریافت کرده و حاصل‌جمع آن‌ها را برمی‌گرداند. در مورد فضای مورد نیاز آن چه فکر می‌کنید؟ در این‌جا پاسخ به نحوه ارسال داده‌های ورودی به برنامه بستگی دارد. اجازه بدهید بیشتر توضیح بدهم: فرض کنید از زبان پاسکال برای پیاده‌سازی این الگوریتم استفاده کنیم، در این زبان آرایه به صورت مقداری به برنامه ارسال می‌شود به عبارت دیگر تمام آرایه قبل از اجرای برنامه در حافظه موقت کپی می‌شود، بنابراین در این حالت نیازمندی فضای متغیر در این‌جا با تعداد اعضای آرایه برابر است. اگر نام الگوریتم خود را sum بگزاریم داریم: $S_{sum}(I)=n$ که n اندازه آرایه ورودی است. حال فرض کنید از زبان c برای پیاده‌سازی الگوریتم استفاده کنیم، در این حالت آرایه به صورت ارجاعی به برنامه ارسال می‌شود به عبارت دیگر مستقل از این‌که آرایه چند عضو دارد، تنها آدرس اولین عنصر آرایه به برنامه ارسال می‌شود، بنابراین در این حالت $S_{sum}(I)=0$.

تمرین : در مثال فوق به نحوه خروجی توجه نشده است، روی حالات گوناگون خروجی بحث کرده و پاسخ فوق را تکمیل کنید.

✐: فرض کنید می‌خواهیم مثال فوق را به صورت بازگشتی پیاده‌سازی نماییم در این صورت تحلیل فوق چه تغییری می‌کند؟ در حالت بازگشتی کامپایلر باید متغیرهای محلی را ذخیره کند و نیز آدرس برگشت هر فراخوانی بازگشتی را داشته باشد. در این مثال فضای لازم برای یک فراخوانی بازگشتی، تعداد بایت‌های مورد نیاز برای دو پارامتر و آدرس بازگشت است. در صورتی که برای هر عضو آرایه ۲ بایت فضا اختصاص یابد(که وابسته به سخت‌افزار است) و نیز برای هر آدرس بازگشت ۲ بایت فضا اختصاص یابد و برای نوع حاصل‌جمع نیز ۴ بایت فضا در نظر گرفته شود در مجموع هر فراخوانی به ۸ بایت فضا نیاز دارد. بنابراین در صورتی که آرایه n عضو داشته باشد در مجموع به ۸n فضای متغیر نیاز است.

با توجه به پیچیدگی‌های زیادی که در بحث تحلیل فضا مورد نیاز وجود دارد بیش از این به این مبحث نمی‌پردازیم. منظور ما پس از این، از فضای مورد نیاز الگوریتم همان نیازمندی فضای ثابت است مگر آن‌که خلاف آن صراحتا ذکر شود. به این ترتیب بار دیگر باید بگویم که فضای حافظه مورد نیاز برای الگوریتم، میزان حافظه مورد نیاز الگوریتم بدون در نظر گرفتن فضای لازم برای ورودی و خروجی است(این فضا معمولا بر حسب بیت بیان می‌شود).

بررسی الگوریتم‌ها از دیدگاه زمان مورد نیاز:

همان‌طور که مشاهده نمودید در بررسی الگوریتم‌ها سه عامل زمان اجرا، حافظه‌ی مورد نیاز و نیز پیاده‌سازی نقش عمده‌ای بر عهده دارند اما به دلیل پیچیدگی محاسبات موجود از تحلیل پیاده‌سازی جز در موارد خاص صرف‌نظر شده و به بررسی حافظه نیز اهمیت چندان زیادی داده نمی‌شود، بنابراین تنها عامل باقی‌مانده یعنی زمان اجرا اصلی‌ترین عامل در تحلیل الگوریتم‌ها است. در ادامه ابتدا با مقدماتی از تحلیل زمان اجرا آشنا می‌شویم سپس گریزی به روابط بازگشتی زده و پس از آن بحث تحلیل زمان اجرا را دنبال می‌کنیم.

یکی از نکاتی که در تحلیل زمان اجرای الگوریتم‌ها باید مورد توجه قرار گیرد، این است که زمان اجرا بر حسب ثانیه یا هر واحد زمانی متداول دیگر(مانند دقیقه ، ساعت و ...) سنجیده نمی‌شود، چرا که عملیات پایه در سخت‌افزارهای مختلف با سرعت های گوناگون اجرا می‌شوند و ما همان‌طور که در ابتدای این نوشته متذکر شدیم بیش‌تر به بررسی زمان اجرا به صورت مستقل از سخت‌افزار و زبان برنامه‌نویسی علاقه‌مند هستیم.

راه‌کاری که برای حل این مشکل ارائه شده است، این است که زمان اجرای الگوریتم‌ها را برحسب تعداد عملیات پایه یا (elementary operation) ها بسنجیم.

Elementary operation: عملیات ساده‌ای هستند که زمان مورد نیاز آن‌ها از بالا به ثابتی محدود می‌شود، مانند جمع ، ضرب، تقسیم، تفریق، انتساب، مقایسه و سایر عملیات منطقی.

تذکر: عملیاتی هم‌چون یافتن یک مقدار حداقل(minimum) از بین اعضای یک آرایه یا تست بخش‌پذیری یا محاسبه فاکتوریل و ... عملیات پایه محسوب نمی‌شوند.

نکته‌ی دیگری که در تحلیل زمان اجرا حائز اهمیت است این است که تابع در چه حالتی از نمونه مورد تحلیل قرار می‌گیرد. به عنوان مثال هنگامی که قصد داریم الگوریتمی برای مرتب کردن اعضای یک آرایه بنویسیم بهترین حالت برای نمونه هنگامی رخ می‌دهد که در ابتدا اعضای آرایه به همان ترتیبی که ما می‌خواهیم(صعودی یا نزولی)، مرتب شده باشند و بدترین حالت، حالتی است که اعضای آرایه در جهت عکس مرتب شده باشند(البته انتخاب بهترین و بدترین حالت تا حدودی بستگی به الگوریتمی که برای مرتب‌سازی به کار می‌رود هم دارد، چرا؟). علاوه بر تحلیل در بهترین حالت و تحلیل در بدترین حالت، گاهی علاقه‌مند هستیم که الگوریتم را در حالت متوسط تحلیل کنیم.

از آن‌جا که اجرای الگوریتم در بهترین حالت(مثلا مرتب‌سازی یک آرایه که خود مرتب است!) به ندرت رخ می‌دهد و از سوی دیگر تحلیل الگوریتم در حالت متوسط، مستلزم محاسبات پیچیده‌ای بوده و برای قرار گرفتن هر عنصر در هر مکان باید قواعد احتمال را هم دخالت داد، معمولا الگوریتم را در

بدترین حالت تحلیل می‌کنند. یک حسن دیگر در تحلیل زمان اجرای الگوریتم در بدترین حالت این است که در این حالت مطمئن هستیم که تابع به زمانی کم‌تر از زمان محاسبه شده نیاز دارد.

خوب بهتر است کم کم کار تحلیل زمان اجرا را آغاز کنیم، برای این کار ابتدا باید یک تابع تعریف کنیم، همان‌طور که احتمالا حدس زده‌اید این تابع یک تابع از اندازه ورودی است. به عبارت دیگر اگر تابع زمان مورد نیاز را با f(n) نشان دهیم n همان اندازه نمونه‌ای است که الگوریتم بر روی آن اجرا می‌شود. اگر بخواهیم خیلی ساده‌تر بگوییم، هنگامی که یک الگوریتم را از نظر زمان اجرا تحلیل می‌کنیم به تابعی مانند f(n) می‌رسیم که می‌تواند به ما بگوید که به کار بردن این الگوریتم با نمونه‌ای به هر اندازه، به چه زمانی نیاز دارد. برای روشن‌تر شدن بحث به مثال زیر توجه کنید:

✐: **فرض کنید شبه‌کد یک الگوریتم به صورت مقابل باشد، تابع f(n) را برای آن بیابید.**

```
For i=1 to n do
        For j=i to n do
                S=s+i+j;
```

اکنون به تحلیلی که برای به دست آوردن تابع f(n) انجام می‌شود توجه کنید:

i=1	i=2	i=3	...	i=k	...	i=n
j=1...n	j=2...n	j=3...n		j=k...n		j=n
n time	n-1 time	n-2 time		n-k time		1 time

بنابراین در مورد تابع f(n) داریم:

$$F(n)= n + n-1 + n-2 + ... + n-k + ... + 1=\frac{n(n+1)}{2}$$

حتما به این نکته توجه کرده‌اید که تعداد محاسبات پایه موجود درون حلقه هیچ تاثیری در محاسبه f(n) ندارد. مثلا در مثال بالا تعداد elementary operation ها ۳ تا است اما اگر تعداد آن‌ها مثلا ۲ تا یا ۴ تا یا ۵ تا یا ... هم بود جواب f(n) هیچ تغییری نمی‌کرد.

🔔 **یادآوری: روابط زیر می‌تواند در یافتن تابع f(n) شما را یاری نماید.**

$$\sum_{i=1}^{n} 1 = n \qquad \sum_{i=1}^{n} i = \frac{n(n+1)}{2} \qquad \sum cf(i) = c\sum f(i)$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

از آنجا که ما در بسیاری از الگوریتم‌ها از رهیافت بازگشتی استفاده می کنیم و هم‌چنین الگوریتم‌هایی که با روش تقسیم و حل نوشته می‌شوند، از رهیافت بازگشتی استفاده می‌کنند بنابراین قبل از آن‌که با قواعد و تکنیک‌های محاسبه زمان اجرا آشنا شویم، خوب است روش تبدیل روابط بازگشتی به غیر بازگشتی را با هم مرور کنیم (شما در درس ساختمان‌های گسسته با این کار آشنا شده‌اید بنابراین می‌توانید برای یادآوری بیش‌تر به جزوه درس ساختمان‌های گسسته خود مراجعه کنید).

مروری بر تکنیک‌های تبدیل روابط بازگشتی به روابط غیر بازگشتی:

تعریف تابع بازگشتی: تابعی است که برای تعریف از خود تابع استفاده می‌کند!

روش‌های تبدیل روابط بازگشتی به روابط غیر بازگشتی:

۱– روش حدس هوشمندانه:

الف) چند مقدار اولیه از تابع را به دست آورید.

ب) سعی کنید رابطه‌ای بین مقادیر ایجاد کنید.(حدس هوشمندانه)

ج) رابطه را به صورت ریاضی بنویسید.

د) برای اطمینان از درستی رابطه می‌توانید از استقراء استفاده کنید.

✎: سعی کنید با استفاده از روش حدس هوشمندانه رابطه زیر را به فرم غیر بازگشتی تبدیل نمایید.

$F(n)=3\,F(n-1) + 4\,F(n-2) + 6 \qquad F(0)=1 \qquad F(1)=2$

ابتدا چند مقدار اولیه تابع را محاسبه می‌کنیم:

$F(2)=16 \qquad F(3)=62 \qquad F(4)=256 \qquad F(5)=1022$

اکنون زمان حدس زدن یک رابطه و بیان آن به صورت ریاضی است، کمی به مقادیر محاسبه شده دقت کنید توان‌های ۴ در بین مقادیر خودنمایی می‌کنند! یک حدس برای جواب می‌تواند به صورت زیر باشد

$$F(n)=\begin{cases} 4^n & n=2k \\ 4^n-2 & n=2k+1 \end{cases}$$

اثبات این رابطه می‌تواند به عهده‌ی کسانی باشد که به صحت آن اطمینان کافی ندارند!

۲– استفاده از حدس‌های هوشمندانه دیگران:

از آن‌جا که همگان هوشمند نبوده! و از سوی دیگر برخی از روابط بازگشتی شباهت‌های زیادی به هم دارند، در تبدیل دسته‌هایی خاص از روابط بازگشتی از حدس‌های هوشمندانه موجود استفاده می‌کنیم.

تذکر: در صورتی که درس معادلات دیفرانسیل را گذرانده‌اید و یا در حال گذراندن آن هستید، تشابهات موجود بین روش‌هایی که در ادامه معرفی می‌شود و آن‌چه که در درس معادلات دیفرانسیل مورد بحث قرار می‌گیرد، می‌تواند برای شما جالب باشد.

الف) روابط بازگشتی خطی همگن(متجانس) با ضرایب ثابت:

فرم کلی: $a_0F(n) + a_1F(n-1) + a_2F(n-2) + \dots + a_kF(n-k)=0$

حدس هوشمندانه: $F(n)=x^n$

فرم معادله مشخصه: $a_0x^k + a_1x^{k-1} + a_2x^{k-2} + \dots + a_k=0$

فرم جواب عمومی: $F(n)=c_1(r_1)^n + c_2(r_2)^n + \dots + c_k(r_k)^n$

در صورتی که یک ریشه مانند r_i در بین ریشه‌ها m بار تکرار شده باشد، داریم:

فرم جواب عمومی با فرض تکرار r_i :

$F(n)= ...+ c_i(r_i)^n + c_{i+1}n(r_i)^n + c_{i+2}n^2(r_i)^n +...+ c_{i+k}n^{k-1}(r_i)^n + ...$

✐: **رابطه فیبوناچی را به صورت غیر بازگشتی بنویسید:**

$F(n) = F(n-1) + F(n-2) \qquad F(0)=0 \qquad F(1)=1$

حل:

$F(n) - F(n-1) - F(n-2)=0$

معادله مشخصه : $x^2-x-1=0$ **ریشه‌ها** : $r_1=\frac{1+\sqrt{5}}{2}$ $r_2=\frac{1-\sqrt{5}}{2}$

$F(n)=C_1(\frac{1+\sqrt{5}}{2})^n + C_2(\frac{1-\sqrt{5}}{2})^n$

ب) روابط بازگشتی خطی ناهمگن(نامتجانس) با ضرایب ثابت:

فرم کلی: $a_0f(n) + a_1f(n-1) + ... + a_kf(n-k) = b_0^nP_0(n) + b_1^nP_1(n) + b_2^nP_2(n) + ...$

در معادله فوق P_0 از مرتبه d_0 و P_1 از مرتبه d_1 و به همین ترتیب ...

فرم معادله مشخصه: $(a_0x^k + a_1x^{k-1} + ... + a_k)(x-b_0)^{d_0+1}(x-b_1)^{d_1+1}....=0$

در صورتی که یک ریشه مانند r_i در بین ریشه‌ها m بار تکرار شده باشد، مشابه حالت همگن است.

✐: **رابطه زیر را به صورت غیر بازگشتی بنویسید:**

$F(n) - F(n-1) = 3 n^2 \qquad F(0)=7 \qquad F(1)=10$

حل:

$F(n) - F(n-1) = 3 n^2 \times 1^n$

معادله مشخصه : $(X-1)(x-1)^3=0 \qquad (x-1)^4=0$ **ریشه‌ها** : $r_1, r_2, r_3, r_4=1$

$F(n)=C_1(1)^n + C_2n(1)^n + C_3n^2(1)^n + C_4n^3(1)^n$

تذکر: برای محاسبه C1 تا C4 در مثال فوق باید F(2) و F(3) را خودمان بیابیم.

نکته: در تبدیل روابط بازگشتی که در تحلیل الگوریتم‌های تقسیم و حل به دست می‌آیند بعد از اعمال یک تغییر متغیر از روش فوق استفاده می‌کنیم. $T(n)=aT(n/b)+g(n)$

✐: **رابطه زیر را به صورت غیر بازگشتی بنویسید:**

$T(n) = 4 T(n/2) + n$

حل: با فرض $n=2^k$ ($n=b^k$ در حالت کلی)، رابطه به صورت زیر تغییر می‌کند.

$T(2^k) = 4 T(2^{k-1}) + 2^k$

تغییر متغیر: $t(k) = 4 t(k-1) + 2^k \times 1^n$

معادله مشخصه : $(X-4)(X-2)=0$ **ریشه‌ها** : $r_1 =2 \quad , r_2 =4$

$t(k)= C_14^k + C_22^k \qquad T(n)=C_14^{\log_2 n} + C_22^{\log_2 n} \qquad T(n)=C_1n^2 + C_2n$

حالت‌های دیگری نیز وجود دارد که جهت طولانی نشدن بحث از ذکر آن‌ها خودداری می‌شود. پس از معرفی مرتبه اجرا، روابط میان‌بری برای محاسبه مرتبه اجرا بدون تبدیل روابط بازگشتی به غیربازگشتی معرفی خواهند شد.

علائم مجانبی:

احتمالا شما با مبحث مجانب‌ها قبلا در ریاضیات آشنا شده‌اید، (البته ممکن است چیزی از این مبحث اکنون در خاطر شما نباشد!) در واقع علائم مجانبی نیز مانند مجانب‌ها عمل می‌کنند.

اولین و مهم‌ترین علامت مجانبی که با آن آشنا می‌شویم بیگ اُ یا O است. وظیفه‌ای که این علامت بر عهده دارد محدود کردن تابع از بالاست. تعریف این علامت مجانبی به صورت زیر است:

هرگاه دو ثابت C و N پیدا شوند به طوری که به ازای هر $n \geq N$ داشته باشیم $F(n) \leq Cg(n)$ در این صورت می‌نویسیم $F(n)=O(g(n))$ و می‌گوییم $F(n)$ از مرتبه $g(n)$ است. مثلا تابع $F(n)$ را در نظر بگیرید: $F(n)=7n^3 + 4n^2 + 5n + 6$

در مورد این تابع داریم: $F(n)=O(n^3)$ و $F(n)=O(n^4)$ و $F(n)=O(n^5)$ و ...

همان‌طور که مشاهده می‌کنید توابع زیادی در رابطه فوق صدق می‌کنند. بنابراین معمول این است که $F(n)$ را از مرتبه کوچک‌ترین $g(n)$ موجود گرفته و یا از نماد عضویت استفاده می‌کنند، مثلا در مورد مثال فوق یا می‌گویند $F(n)$ از مرتبه n^3است و یا می‌گویند $F(n) \in O(n^3)$.

در صورتی که مطالب بالا کمی برای شما گنگ به نظر می‌رسد حالت حد $F(n)$ وقتی n به سمت بی‌نهایت میل می‌کند را به یاد آورید، این حالت می‌تواند راهنمای خوبی برای شما باشد.

هر چند که در تست‌های مطرح شده از درس ساختمان داده فقط از علامت مجانبی O سوال مطرح می‌شود، اما اجازه بدهید با تعریف سایر علائم مجانبی به صورت کاملا مختصر آشنا شویم:

دومین علامت مجانبی که برای محدود کردن از پایین به کار می‌رود $\Omega(n)$ است. تعریف این علامت مجانبی به صورت زیر است:

هرگاه دو ثابت C و N پیدا شوند به طوری که به ازای هر $n \geq N$ داشته باشیم $F(n) \geq Cg(n)$ در این صورت می‌نویسیم $F(n)=\Omega(g(n))$. مثلا برای مثالی که در قسمت قبل مطرح شد، خواهیم داشت:

$F(n)=\Omega(n^3)$ و $F(n)=\Omega(n^2)$ و $F(n)=\Omega(n)$ و ... در این حالت نیز معمولا می‌نویسیم $F(n) \in \Omega(n^3)$

علامت مجانبی بعدی Θ است. این علامت به نحوی برابری در بی‌نهایت را معین می‌کند، در مورد تعریف این تابع داریم:

اگر داشته باشیم: $F(n)=O(g(n))$ و $F(n)=\Omega(g(n))$ در این صورت: $F(n)=\Theta(g(n))$

مثلا در مورد مثال قبلی داریم $F(n)=\Theta(n^3)$

چند علامت مجانبی دیگر نیز برای به شدت محدود کردن تابع از بالا و پایین و ... وجود دارند که پرداختن به آن‌ها از حوصله این بحث خارج است. احتمالا در مورد $\Theta(n)$ حدس زده‌اید که هنگامی رابطه $F(n)=\Theta(g(n))$ صحیح است که حد کسر $\frac{F(n)}{g(n)}$ در بی‌نهایت برابر با یک مقدار ثابت و غیر صفر باشد.

قبل از آن‌که به بیان و بررسی قضایای موجود درباره O بپردازیم اجازه بدهید سرعت رشد چند تابع مختلف را با هم مقایسه کنیم:

با فرض $b>a>1$ **و** $k>i>2$

$\Theta(\log n) < \Theta(n) < \Theta(n \log n) < \Theta(n^2) < \Theta(n^i) < \Theta(n^k) < \Theta(a^n) < \Theta(b^n) < \Theta(n!) < \Theta(n^n)$

سوال: با توجه به رابطه‌ی فوق آیا روابط زیر برقرار هستند؟

$O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^i) < O(n^k) < O(a^n) < O(b^n) < O(n!) < O(n^n)$

$\Omega(\log n) < \Omega(n) < \Omega(n \log n) < \Omega(n^2) < \Omega(n^i) < \Omega(n^k) < \Omega(a^n) < \Omega(b^n) < \Omega(n!) < \Omega(n^n)$

قضیه‌های مربوط به تعیین مرتبه اجرا

قضیه ۱: هرگاه داشته باشیم $F(n)=O(g_1(n))$ **و** $H(n)=O(g_2(n))$ **آن‌گاه در مورد جمع و ضرب داریم:**

$F(n) + H(n) = O(g_1(n) + g_2(n))$

$F(n) \times H(n) = O(g_1(n) \times g_2(n))$

اما در مورد تقسیم و تفریق این روابط لزوما درست نیستند.

$F(n) / H(n) \neq O(g_1(n) / g_2(n))$

$F(n) - H(n) \neq O(g_1(n) - g_2(n))$

قضیه ۲: اگر "مقدار ثابت $= F(n)$ **" آن‌گاه** $F(n)=O(1)$

قضیه ۳: در مورد توابعی که از مرتبه لگاریتمی هستند پایه لگاریتم اهمیتی ندارد.

$F(n)=O(\log_a^n)=O(\log n)$

$g(n)=O(\log_b^n)=O(\log n)$

$\log_a^n= \log_b^n / \log_b^a = \log_b^n / C$

قضیه ۴: هر تابع چند جمله‌ای از مرتبه بزرگترین توانش است.

$F(n)= a_m n^m + ... + a_1 n + a_0 \in O(n^m)$

قضیه ۵: هر تابع به صورت مجموع جملات توانی از مرتبه جمله با پایه بزرگ‌تر است.

$F(n)= a^n + b^n \in O(a^n) \quad \text{if } a > b$

مثال: مرتبه اجرای تابع زیر را به دست آورید:

$F(n)=3n (\log n!) + (n^2 + 3) \log n$

حل : $F(n)= O(3n (\log n!)) + O((n^2 + 3) \log n)$

با صرف نظر کردن از ثابت‌ها و استفاده از رابطه $n!<n^n$ **خواهیم داشت:**

$F(n)= O(n (\log n^n)) + O(n^2 \log n) = O(n (n \log n)) + O(n^2 \log n) = O(n^2 \log n)$

نکته: اگر از انتهای قسمت تبدیل روابط بازگشتی به یاد داشته باشید، گفتیم که روابط میان‌بری را به شما خواهیم آموخت. البته تمامی این روابط به سادگی قابل به دست آوردن هستند اما دانستن آن‌ها سرعت شما را در پاسخ‌گویی به سوالات اضافه می‌نماید.

این روابط در جدول زیر خلاصه شده‌اند:

مرتبه‌ی اجرا	رابطه‌ی بازگشتی
$O(2^n)$	$T(n) = T(n-1)\times T(n-1)$
$O(2^{n/2})$	$T(n) = T(n-2)\times T(n-2)$
$O(2^{\log_2 n})=O(n)$	$T(n) = \sqrt{T(n/2)+T(n/2)}$
$O(n)$	$T(n) = 2T(n-1)$
$O(2^n)$	$T(n) = T(n-1)/(T(n-1)+1)$
$O(3^n)$	$T(n) = T(n-1) + T(n-1)/(T(n-1)+1)$

جدول روابط میان‌بر

از آن‌جا که بهترین روش برای آموختن تحلیل الگوریتم‌ها استفاده از مثال است، به جای آن‌که بیشتر به گفتگو در این باره بپردازیم مثال‌هایی چند را با هم مرور می‌کنیم و هر جا لازم بود نکات جدیدی را مطرح می‌نماییم

مثال‌ها:

✐: فرض کنید زمان اجرای الگوریتمی روی n ورودی، $T(n)$ بوده که به صورت زیر تعریف می‌شود:

$$T(n)=\begin{cases}1 & n=2\\ T(n-1)+n & n>2\end{cases}$$

زمان اجرای الگوریتم مزبور برابر کدام گزینه است؟

۱) $O(n)$ **۲)** $O(n \log n)$ **۳)** $O(n^{2/3})$ **۴)** $O(n^2)$ «آزمون کارشناسی ارشد ۱۳۷۴»

پاسخ: رابطه بازگشتی از نوع ناهمگن با ضرایب ثابت است که می‌توانید آن را به سادگی از حالت بازگشتی خارج نموده و سپس با استفاده از قضایای ذکر شده مرتبه اجرا را به دست آورید، این روش را به شما واگذار کرده و سعی می‌کنیم مساله را از روش دیگری حل نماییم. در این روش به جای تبدیل رابطه بازگشتی آن‌را بسط می‌دهیم:

$$T(n) = T(n-1) + n = T(n-2) + n-1 + n = ... = n + n-1 + ... + 1 = n(n-1)/2=O(n^2)$$

بنابراین گزینه ۴ صحیح است.

✐: تابع بازگشتی زیر را در نظر بگیرید:

```
int test ( int n)
    { if (n ≤2)
            return 1;
      else
            return test(n-2) * test(n-2);
    }
```

زمان اجرای تابع فوق برابر است با:

۱) $O(n^2)$ **۲)** $O(n \log n)$ **۳)** $O(2^{n/2})$ **۴)** $O(2^n)$ «آزمون کارشناسی ارشد ۱۳۷۵»

پاسخ : ابتدا باید رابطه بازگشتی کد فوق را به دست آوریم:

$$T(n)=\begin{cases} O(1) & n \le 2 \\ T(n-2)*T(n-2) & n>2 \end{cases}$$

اگر از جدول روابط میان‌بر به خاطر داشته باشید در مورد رابطه بازگشتی فوق باید گزینه ۳ را انتخاب نمایید.

✐: یک آرایه از اعداد صحیح به صورت A[1..m] مفروض است، به طوری که $\sum_{i=1}^{m} A[i] = s$ می‌باشد. در این صورت مرتبه‌ی اجرای الگوریتم زیر کدام یک از گزینه‌های زیر است؟

```
T:=0;
for i:=1  to m do
        for j:=1 to A[i] do
                T:=T+1;
```

۱) $O(s^2)$ **۲)** $O(m+s)$ **۳)** $O(m^2)$ **۴)** $O(m*s)$ «آزمون کارشناسی ارشد ۱۳۷۶»

پاسخ: در قطعه کد داده شده، حلقه خارجی هیچ تاثیری بر تعداد تکرار حلقه داخلی ندارد و در هر شرایطی تعداد تکرار این دو حلقه مستقل از یک‌دیگر است. حلقه خارجی m بار و حلقه داخلی s بار اجرا می‌شود. البته این جواب در صورتی صحیح است که آرایه حاوی اعداد طبیعی باشد(چرا؟).

تذکر: علامت =: در زبان پاسکال به معنای انتساب است(معادل علامت = در زبان C) و علامت = در این زبان به معنای تساوی است(معادل علامت == در زبان C). دانستن نحو زبان پاسکال ممکن است شما را در حل برخی سوالات یاری نماید!

✐: بر روی پشته S اعمال زیر تعریف شده‌اند:

Push(S,x) : درج x در بالای پشته با هزینه O(1).

Pop(S,x) : حذف عنصر بالای پشته با هزینه O(1).

Multiple(S,k) : حذف k عنصر بالای پشته (با فرض آن‌که k حداکثر برابر تعداد عناصر موجود S است) با هزینه O(k).

اگر N عمل از اعمال فوق به ترتیب دلخواه، بر روی پشته S که در ابتدا تهی است انجام شود، مجموع هزینه این N عمل در بدترین حالت چقدر است؟

۱) $O(N \log N)$ **۲)** $O(N^2)$ **۳)** $O(NK)$ **۴)** $O(N)$ «آزمون کارشناسی ارشد ۱۳۷۸»

پاسخ: از بین سه عمل تعریف شده ، دستور Multiple دارای هزینه O(K) است که در بدترین حالت برابر با O(N) خواهد شد. بنابراین بالاترین هزینه اجرایی دستورات برابر با O(N) خواهد بود.(توجه داشته باشید که با توجه به تهی بودن استک در ابتدا، اجرای دستورات با هر ترتیبی میسر نیست).

تذکر: نکته‌ی مورد نظر طراح چنین است، فرض کنید یک عمل دارای مرتبه $O(x)$ است، حال m بار این عمل را در خطوط مختلف الگوریتم خود به کار می‌بریم، اکنون مرتبه اجرای الگوریتم ما چند است؟ $O(x)$ یا $O(mx)$ چرا؟

✐: **تابع ذیل برای محاسبه بزرگ‌ترین مقسوم‌علیه دو عدد نوشته شده است:**

```
int gcd(int a, int b)
{
        if (b==0)
                Return a;
        else
                return gcd(b, amod b);
}
```

در این صورت می‌توان گفت: مرتبه زمانی الگوریتم .... است.

۱) $O(\log_2^{a/b})$ **۲)** $O(\log_b^{a})$ **۳)** $O(\log_2^{a})$ **۴)** $O(\log_2^{(a-b)})$ «آزمون کارشناسی ارشد ۱۳۸۰»

پاسخ: اگر به یاد داشته باشید در یکی از قضایا در مورد مرتبه لگاریتمی متذکر شدیم که پایه لگاریتم تاثیری در مرتبه اجرا ندارد چرا که لگاریتم در هر پایه‌ای را با ضرب یک مقدار ثابت می‌توان به لگاریتم با پایه دیگر تبدیل نمود. از سوی دیگر اگر از ابتدای بحث به خاطر داشته باشید، گفتیم که برای یک تابع مانند $F(n)$ مرتبه‌های اجرای گوناگونی وجود دارد، این مقدمه را در ذهن خود داشته باشید تا به جواب بپردازیم.

در مورد تابع gcd می‌توانیم بگوییم(چرا؟): $T(a)T(a/b) = b^2T(a/b^2)=...=b^mT(a/b^m)$

رابطه‌ی بازگشتی فوق دارای مرتبه $\log_b^a$ است اما داریم:

$\mathrm{Log}_b^a=(1/\log_a^2)\,\log_a^2$ **و** $b\geq 2 \;\; \rightarrow \log_b^a \leq \log_2^a$

پس تابع $\log_2^a$ حد بالای تابع $\log_b^a$ است و بنابراین $\log_b^a=O(\log_2^a)$ پس گزینه ۳ صحیح است!

✐: **فرض کنید F و g دو تابع دلخواه به شکل $F,g: N\rightarrow R^+$. به علاوه فرض کنید:**

$\lim F(n)/g(n)=+\infty$ **و** $n\rightarrow \infty$ **. کدام‌یک از گزینه‌های زیر درست است؟**

۱) $F(n) \in O(g(n))\ ,\ g(n) \notin \Omega(F(n))$

۲) $g(n) \in O(F(n))\ ,\ F(n) \in O(g(n))$

۳) $F(n) \in \Omega(g(n))\ ,\ F(n) \notin \theta(g(n))$

۴) $F(n) \in \theta(g(n))\ ,\ g(n) \notin \Omega(F(n))$ «آزمون کارشناسی ارشد ۱۳۸۲»

پاسخ: با توجه به تعاریف علائم مجانبی و نیز سایر موارد گفته شده در مورد آن‌ها گزینه ۳ صحیح است.(چرا؟)

تذکر ۱: همان‌طور که مشاهده می‌کنید تقریبا هر ساله یک سوال از مبحث فوق مطرح می‌شود. این در حالی است که از آوردن سوالات ذیل عنوان درس طراحی الگوریتم که از همین مباحث بوده‌اند خودداری شده است.

تذکر ۲: دسته‌ی دیگری از سوالات که از مبحث تحلیل الگوریتم‌ها و ذیل عنوان درس ساختمان‌داده‌ها و الگوریتم‌ها مطرح می‌شوند مربوط به الگوریتم‌های خاص هستند. به مثال زیر توجه کنید:

✐: **درایه‌ی** n **خانه‌ای** A **را در نظر می‌گیریم که برای ذخیره‌سازی عناصر یک درخت دودوئی کامل مورد استفاده قرار گرفته. فرض کنیم الگوریتمی کارا برای بررسی این که این درخت دودوئی یک** heap **است یا خیر در اختیار داریم. پیچیدگی این الگوریتم در بدترین حالت چقدر است؟**

۱) $O(n)$ ۲) $O(n^2)$ ۳) $O(\log n)$ ۴) $O(n \log n)$ «آزمون کارشناسی ارشد ۱۳۸۱»

برای پاسخ دادن به این سوالات باید دیگر دانسته‌ها خود را با دانسته‌های خود در زمینه تحلیل الگوریتم درهم بیامیزید. برای نمونه پاسخ سوال فوق به شرح زیر است:

هنگامی که یک heap **به وسیله بردار پیاده‌سازی می‌شود، فرزندان گره** I **در خانه** 2i **و** 2i+1 **قرار می‌گیرند، از این‌رو برای بررسی این‌که یک درخت دودوئی، یک** heap **است، کافی است عناصر اول تا نیمه بردار با فرزندانشان مقایسه شوند. چنین کدی دارای مرتبه اجرایی** $O(n)$ **است. برای نمونه کد زیر بردار** A **با** n **عنصر را بررسی می‌کند که آیا یک** MAXheap **است یا نه؟ کد** MINheap **مشابه است.**

```
Bool IsMAXheap(int A[], int n)
{
        For ( int i=0;  i < (n-1)/2 ; i++)
                {
                        if (A[i]< A[2*i]) return false;
                        if (A[i]< A[2*i+1]) return false;
                }
        Return true;
}
```

از هر چه بگذریم سخن نمره خوش‌تر است!

از آن‌جا که از این مبحث در امتحان پایان ترم یک یا دو سوال مطرح خواهد شد،و از سوی دیگر در تهیه این جزوه سعی شده است تا جزوه علاوه بر داشتن حالت خودآموز، خسته‌کننده هم نباشد، حجم جزوه کمی زیاد شده است. در پایان این قسمت خالی از لطف نیست که مشخص شود، دقیقا از کدام قسمت‌ها احتمال طرح سوال وجود دارد.

از این بخش‌ها امکان طرح سوال وجود دارد:

تعریف دقیق الگوریتم(صفحه ۹)　　　تفاوت برنامه با الگوریتم (صفحه ۱۰)

تحلیل الگوریتم‌ها با رویکرد زمانی(صفحات ۱۴ تا ۲۳)

به غیر از مطالب فوق از سایر مطالب جزوه در امتحان پایان ترم سوالی طرح نخواهد شد و حالت مطالعه آزاد خواهد داشت. به علاوه مطالبی که در ادامه و تحت عنوان ضمیمه ذکر می‌شوند نه تنها در امتحان پایان ترم مورد سوال واقع نخواهند شد، بلکه در کلاس حل تمرین هم تنها در صورتی که زمان اضافه وجود داشته باشد مورد بحث قرار خواهند گرفت.

ضمیمه) مقدمه‌ای بر درس طراحی الگوریتم:

همان‌طور که در ابتدای جزوه اشاره شد، یکی از اهداف تدوین این جزوه علاقه‌مند نمودن دانشجویان به درس طراحی الگوریتم است. بد نیست قبل از ادامه مطالعه جزوه سری به نشانی اینترنتی زیر بزنید:

http://ce.sharif.edu/~nasirishargh/dac/

نشانی فوق نشانی سایت مسابقه درس طراحی الگوریتم دانشگاه صنعتی شریف است. مشاهده می‌کنید که درس به صورت کاملا رقابتی ارائه می‌شود و دانشجویان در کنار آشنایی با مفاهیم طراحی الگوریتم به رقابت با یکدیگر پرداخته و علاوه بر بروز خلاقیت‌های خود و گاها حل نمودن مسایل حل نشده، روحیه شادابی و نشاط را نیز توسعه می‌دهند. امید آن است که شما، دانشجویانی که در حال حاضر مشغول مطالعه این جزوه هستید با حضور پرشور در کلاس درس طراحی الگوریتم، این کلاس را متحول نموده و نحوه برگزاری آن را تغییر دهید. در حال حاضر با توجه به روحیه کسل و بی‌حوصله دانشجویان در این کلاس دانشجویان تنها با مفاهیم و اصول طراحی الگوریتم آشنا می‌شوند.

اجازه بدهید با هم نگاهی به سرفصل‌هایی که در درس طراحی الگوریتم مورد بررسی قرار می‌گیرند، بیاندازیم.

در ابتدا شما با مفاهیم تحلیل الگوریتم آشنا می‌شوید و سپس با روش‌های طراحی الگوریتم. در کنار ارائه هر روش طراحی شما با نمونه‌هایی از مسائل که با آن روش حل می‌شوند، نیز آشنا می‌شوید.

روش اول مورد بررسی روش حریصانه نام دارد که در هر مرحله الگوریتم بهترین انتخاب ممکن را انجام می‌دهد و امیدوار است که در پایان جواب بهینه را پیدا نماید.

یک نمونه از مسایلی که با روش حریصانه حل می‌شود مساله زمان‌بندی است. صورت این مساله چنین است: یک سرویس دهنده داریم که تعدادی مشتری از آن سرویس دریافت می‌کنند، هر مشتری میزان زمانی را برای تکمیل سرویس نیاز دارد. ترتیب سرویس دادن را به نحوی معین نمائید که زمان توقف مشتریان در سیستم حداقل ممکن باشد. یک نمونه دیگر مساله کوله پشتی نام دارد. صورت این مساله چنین است: یک کوله پشتی با ظرفیت A داریم که می‌خواهیم آن‌را با n شی پر کنیم. هر یک از اشیا دارای وزن و ارزش مخصوص به خود هستند. می‌خواهیم طوری اشیا را انتخاب کنیم که حداکثر ارزش را کسب کنیم و در عین حال جمع وزن‌ها از ظرفیت کوله‌پشتی بیشتر نباشد. نمونه دیگر مساله

فروشنده دوره‌گرد است. صورت این مساله چنین است: یک فروشنده دوره‌گرد می‌خواهد به n شهر فقط و فقط یک‌بار سر بزند و دوباره به شهر اول بازگردد. او فاصله‌ها شهرها را می‌داند، می‌خواهیم مسیر او را طوری تعیین کنیم که حداقل مسافت را طی کند. همچنین یک نمونه از الگوریتم‌های فشره‌سازی نیز در این روش مورد بررسی قرار می‌گیرد که به روش هافمن معروف است.

روش دومی که مورد بررسی قرار می‌گیرد روش تقسیم و حل نام دارد. در این روش نمونه را به زیر نمونه‌های کوچک‌تر تقسیم می‌کنیم و به حل مساله برای آن زیر نمونه‌ها می‌پردازیم.

از مسایلی که با روش تقسیم و حل، حل می‌شوند می‌توان به Quicksort و mergesort اشاره کرد. در همین قسمت از درس شما با شایرالگوریتم‌ها مرتب‌سازی و جستجو نیز آشنا می‌شوید و به مقایسه آن‌ها با یکدیگر می‌پردازید. الگوریتم ضرب اعداد n رقمی و نیز ضرب ماتریس‌های بزرگ از دیگر نمونه‌های روش تقسیم و حل هستند.

روش سومی که در طراحی الگوریتم مورد بحث قرار می‌گیرد روش برنامه‌سازی پویا نام دارد.این روش ابتدا نمونه‌های کوچک مساله را حل می‌کند و جواب‌ها را در یک جدول ذخیره می‌کند، با ادغام جواب‌های نمونه‌های کوچک، جواب برای نمونه‌هاب بزرگ به دست می‌آید.یک نمونه از مسایلی که با این روش حل می‌شودع ضرب زنجیری ماتریس‌ها است. همچنین مساله فروشنده دوره‌گرد در این‌جا نیز مورد بحث قرار می‌گیرد.

علاوه بر روش‌های فوق روش‌های دیگری نیز در این درس مطرح می‌شود و برای هر یک نمونه‌های گوناگونی مورد بحث قرار می‌گیرد(مانند مساله ۸ وزیر یا کوله پشتی صفر و یک یا بازی Nim یا ...) که برای جلوگیری از طولانی شدن مطلب به آن‌ها اشاره نمی‌کنیم.

در پایان مقایسه الگوریتم‌ها مرتب‌سازی را از دانشنامه ویکی‌پدیا با هم مرور می‌کنیم.

الگوریتم‌های مرتب‌سازی(Sorting algorithms)

در علوم کامپیوتر و ریاضی، یک الگوریتم مرتب‌سازی الگوریتمی است که لیستی از داده‌ها را به ترتیبی مشخص می‌چیند. پر استفاده‌ترین ترتیب‌ها، ترتیب‌های عددی و لغت‌نامه‌ای هستند. مرتب‌سازی کارا در بهینه‌سازی الگوریم‌هایی که به لیست‌های مرتب شده نیاز دارند (مثل جستجو و ترکیب) اهمیت زیادی دارد. از ابتدای علم کامپیوتر مسائل مرتب‌سازی تحقیقات فراوانی را متوجه خود ساختند، شاید به این علت که در عین ساده بودن، حل آن به صورت کارا پیچیده است. برای مثال مرتب‌سازی حبابی در سال ۱۹۵۶ به وجود آمد. در حالی که بسیاری این را یک مساله‌ی حل شده می‌پندارند، الگوریتم‌های کارآمد جدیدتر هم‌چنان ابداع می‌شوند (مثلاً مرتب‌سازی کتاب‌خانه‌ای در سال ۲۰۰۴ مطرح شد).

در علم کامپیوتر معمولاً الگوریتم‌های مرتب‌سازی بر اساس این معیارها طبقه‌بندی می‌شوند:

- **پیچیدگی (بدترین و بهترین عملکرد و عملکرد میانگین) : با توجه به اندازه‌ی لیست(n). در مرتب‌سازی‌های معمولی عملکرد خوب $O(n \log n)$ و عملکرد بد $O(n^2)$ است. بهترین عملکرد برای مرتب‌سازی $O(n)$ است. الگوریتم‌هایی که فقط از مقایسه‌ی کلیدها استفاده می‌کنند در حالت میانگین حداقل $O(n \log n)$ مقایسه نیاز دارند.**
- **حافظه (و سایر منابع کامپیوتر) : بعضی از الگوریتم‌های مرتب‌سازی «در جا» هستند. یعنی به جز داده‌هایی که باید مرتب شوند، حافظه‌ی کمی ($O(1)$) مورد نیاز است؛ در حالی که سایر الگوریتم‌ها به ایجاد مکان‌های کمکی در حافظه برای نگه‌داری اطلاعات موقت نیاز دارند.**
- **پایداری : الگوریتم‌های مرتب‌سازی پایدار ترتیب را بین داده‌های دارای کلیدهای برابر حفظ می‌کنند. فرض کنید می‌خواهیم چند نفر را بر اساس سن با یک الگوریتم پایدار مرتب کنیم. اگر دو نفر با نام‌های الف و ب هم‌سن باشند و در لیست اولیه الف جلوتر از ب آمده باشد، در لیست مرتب شده هم الف جلوتر از ب است.**
- **مقایسه‌ای بودن یا نبودن. در یک مرتب‌سازی مقایسه‌ای داده‌ها فقط با مقایسه به وسیله‌ی یک عملگر مقایسه مرتب می‌شوند.**
- **روش کلی : درجی، جابجایی، گزینشی، ترکیبی و غیره. جابجایی مانند مرتب‌سازی حبابی و مرتب‌سازی سریع و گزینشی مانند مرتب‌سازی پشته‌ای.**

لیست الگوریتم‌های مرتب‌سازی

در این جدول، n تعداد داده‌ها و k تعداد داده‌ها با کلیدهای متفاوت است. ستون‌های بهترین، میانگین و بدترین، پیچیدگی در هر حالت را نشان می‌دهد و حافظه بیانگر مقدار حافظه‌ی کمکی (علاوه بر خود داده‌ها) است.

نام	بهترین	میانگین	بدترین	حافظه	پایدار	مقایسه ای	روش	توضیحات
مرتب سازی حبابی (Bubble sort)	$O(n)$	—	$O(n^2)$	$O(1)$	بله	بله	جابجایی	Times are for best variant
Cocktail sort	$O(n)$	—	$O(n^2)$	$O(1)$	بله	بله	جابجایی	
Comb sort	$O(n \log n)$	—	$O(n \log n)$	$O(1)$	خیر	بله	جابجایی	
Gnome sort	$O(n)$	—	$O(n^2)$	$O(1)$	بله	بله	جابجایی	
Selection sort	$O(n^2)$	$(O(n^2)$	$O(n^2)$	$O(1)$	خیر	بله	گزینشی	
Insertion sort	$O(n)$	—	$O(n^2)$	$O(1)$	بله	بله	درجی	
Shell sort	$O(n \log n)$	—	$O(n \log^2 n)$	$(O(1)$	خیر	بله	درجی	Times are for best variant
Binary tree sort	$O(n \log n)$	—	$O(n \log n)$	$O(1)$	بله	بله	درجی	

Library sort	O(n)	O(n log) n	$O(n^2)$	(ε+1)n	بله	بله	درجی	
Merge sort	O(n log n)	—	O(n log n)	O(n)	بله	بله	Merging	
In-place merge sort	O(n log n)	—	O(n log n)	O(1)	بله	بله	Merging	Times are for best variant
Heapsort	(O(n log n	—	O(n log n)	O(1)	خیر	بله	گزینشی	
Smoothsort	O(n)	—	O(n log n)	O(1)	خیر	بله	گزینشی	
Quicksort	O(n log n)	O(n log n)	$O(n^2)$	O(n)	خیر	بله	Partitioning	Naive variants O(n space)use
Introsort	O(n log n)	O(n log n)	O(n log n)	O(n)	خیر	بله	Hybrid	
Pigeonhole sort	O(n+k)	—	O(n+k)	O(k)	بله	خیر	Indexing	
Bucket sort	O(n)	O(n)	$O(n^2)$	O(k)	بله	خیر	Indexing	
Counting sort	O(n+k)	—	O(n+k)	O(n+k)	بله	خیر	Indexing	
Radix sort	O(nk)	—	O(nk)	O(n)	بله	خیر	Indexing	
Patience sorting	O(n)	—	O(n log n)	O(n)	خیر	بله	درجی	تمام زیر دنباله‌های صعودی را با log(n)O(n log) پیدا می‌کند.

این جدول الگوریتم‌هایی را توضیح می‌دهد که به علت اجرای بسیار ضعیف و یا نیاز به سخت‌افزار خاص، کاربرد زیادی ندارند.

نام	بهترین	میانگین	بدترین	حافظه	پایدار	مقایسه‌ای	توضیحات
Bogosort	O(n)	O(n × n!)	بدون حد	O(1)	خیر	بله	
Stooge sort	$O(n^{2.71})$	—	$O(n^{2.71})$	O(1)	خیر	بله	
Bead sort	O(n)	—	O(n)	—	N/A	خیر	به سخت‌افزار مخصوص نیاز دارد.
Pancake sorting	O(n)	—	O(n)	—	خیر	بله	به سخت‌افزار مخصوص نیاز دارد.
Sorting networks	O(log n)	—	O(log n)	—	بله	بله	Requires a custom circuit of size O(log n)