تحلیلی از روشهای مرتب کردن*

جاناتان آمستردام *
ترجمهٔ سیدغلامرضا پناهی

چندی پیش دوستی به من گفت که ۹۰٪ کلیهٔ برنامه‌های کامپیوتری دنیا عمل مرتب کردن را انجام می‌دهند.این حرف را می‌توان باور کرد.علاقهٔ وافر جوامع به نظم و سازمان، باعث شده است که عمل سادهٔ قراردادن هر چیزی در جای مناسب آن اهمیت زیادی پیدا کند و چه وسیله‌ای برای این منظور بهتر از حاملان مطیع اطلاعات ـ کامپیوترها ـ ؟ الگوریتمهای مرتب کردن به علت اهمیت زیادشان به تفصیل مورد مطالعه قرار گرفته‌اند.بعضی از این الگوریتمها کند و برخی دیگر سریع هستند.تعدادی از آنها فقط چند داده را مرتب می‌کنند وگروهی میلیونها داده را.در این مقاله، سه نوع الگوریتم مرتب کردن را بررسی می‌کنیم.مرتب کردن انتخابی برای فهرستهای کوچک و مرتب کردن سریع برای فهرستهای بزرگتر و مرتب کردن ادغامی برای فهرستهایی که آنقدر بزرگ‌اند که نمی‌توان آنها را همزمان در حافظه قرار داد.اما ابتدا به مفاهیم ساده‌ای نیاز داریم که ما را در تحلیل الگوریتمها یاری کنند.

تحلیل

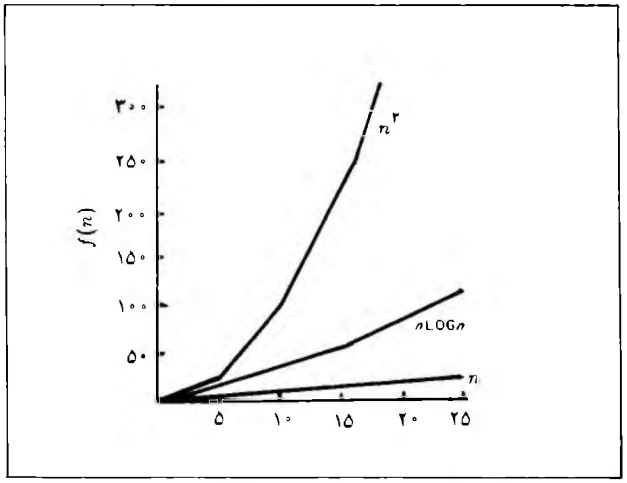
هدف ما بررسی میزان کارایی بعضی از الگوریتمهای مرتب کردن است، اما در آغاز کار با مسأله‌ای مواجه می‌شویم:چگونه می‌توانیم الگوریتمی را به طور مجرد مطالعه کنیم بدون اینکه به زبانی که الگوریتم با آن زبان نوشته می‌شود و به ماشینی که در آن اجرا خواهد شد توجه کنیم؟به عنوان مثال، اگر الگوریتمی به زبان اسمبلی نوشته شود سریعتر اجرا می‌شود تا اینکه به یکی از زبانهای سطح بالا نوشته شود و هر برنامه‌ای که در یک ریزکامپیوتر اجرا می‌شود در کامپیوترهای بزرگ سریعتر اجرا خواهد شد.ما می‌خواهیم بدون پرداختن به این موضوعات در مورد مدت اجرای الگوریتمها مستقل از ماشین و زبان به کار رفته صحبت کنیم.

دانشمندان کامپیوتر قواعد ساده‌ای به دست آورده‌اند که این کار را ممکن می‌سازد.اولین قاعده این است، که مدت اجرا با تعداد عملیات اساسی لازم برای اجرای الگوریتم اندازه‌گیری شود و نه با سرعت یک کامپیوتر یا تعداد دستورالعملهایی که باید اجرا شود.به عنوان مثال، در عمل مرتب کردن، عملیات اساسی عبارت‌اند از مقایسهٔ دو داده و جابه‌جاکردن آنها با هم.این قاعده از اینجا به دست می‌آید که هر برنامه‌ای که از این الگوریتم استفاده کند، بدون توجه به اینکه به چه زبانی نوشته شده یا در چه کامپیوتری انجام شود باید تعداد یکسانی عملیات اساسی را انجام دهد.تعداد مقایسه‌ها معمولاً شاخص بهتری از مدت زمان اجراست زیرا در الگوریتمهای مرتب کردن، تعداد مقایسه‌ها غالباً از تعداد جابه‌جاییها بیشتر است.

قاعدهٔ مهم دیگر، اندازه‌گیری سرعت الگوریتم برحسب اندازهٔ ورودی آن است، و این چیزی است که عقل سلیم حکم می‌کند زیرا مثلاً نمی‌خواهیم مدت زمان لازم برای اجرای الگوریتمی که ۱۰ داده را مرتب می‌کند با مدت اجرای الگوریتمی که ۱۰۰ داده را مرتب می‌کند مقایسه شود.بلکه می‌خواهیم جدولی برای مقایسهٔ زمان مرتب کردن ۱۰، ۱۰۰، ۱۰۰۰ داده یا بیشتر توسط الگوریتمهای مختلف داشته باشیم.

ولی در عمل به چیزی کاراتر از جدول نیاز داریم.ایده‌آل آن است که تابعی داشته باشیم که به ازای هر اندازهٔ ورودی، مدت اجرا را به ما بدهد.در این صورت خواهیم توانست آهنگ رشد تابع را بررسی کنیم و قضاوت نماییم که وقتی تعداد داده‌ها زیاد است کدام الگوریتم بهتر است.مثلاً، تابع $f(n) = n^2$ سریعتر رشد می‌کند تا تابع $g(n) = 100n$.به عبارت دیگر، به ازای مقادیری از n که به اندازهٔ کافی بزرگ باشند، در مورد این مثال $n > 100$، مقدار f از g بیشتر خواهد بود.مقایسهٔ آهنگ رشد به کمک نمودار توابع ساده‌تر می‌شود.در شکل ۱ نمودار چند تابع متداول و از جمله تابعهایی که مورد بحث ما خواهند بود، نشان داده شده است.

مفهوم مقدار یک تابع به ازای داده‌های زیاد (n به قدرکافی بزرگ)، که آن را آهنگ رشد مجانبی می‌گویند، مفهوم اساسی در تحلیل الگوریتمهاست.در مقایسهٔ دو الگوریتم معمولاً حالتی را که n کوچک است بررسی نمی‌کنیم زیرا مسأله در این حالت ساده است.حالت مهم حالتی است،که تعداد داده‌ها زیاد می‌شود و می‌خواهیم بدانیم در این حالت الگوریتمها چگونه عمل می‌کنند مدت زمان لازم برای مرتب کردن یک فهرست ده‌تایی به وسیلهٔ هر یک



شکل ۱.آهنگ رشد بعضی توابع متداول

از الگوریتمهای مرتب کردن عملاً ناچیز است ولی با بزرگ شدن فهرست، الگوریتمهایی که ذاتاً کندترند مشخص می‌شوند.

بنابراین، برای تحلیل یک الگوریتم می‌توانیم تابعی به دست آوریم که عملیات اساسی الگوریتم را به اندازهٔ فهرست ورودی مربوط کند، و آنگاه آهنگ رشد مجانبی آن را تعیین کنیم.ولی اغلب، حتی این اندازه تجرید هم کافی نیست و تحلیل الگوریتم به این ترتیب بسیار طولانی خواهد بود.در یک مسألهٔ بزرگ مهم نیست،که طبق یک الگوریتم مثلاً تعداد $۳۷n + ۴$ و یا $۲۴n - ۶$ مقایسه انجام می‌شود.مهم این است که تعداد مقایسه‌ها متناسب با اندازهٔ ورودی است. به عبارت دیگر، آهنگ رشد مجانبی هر دو تابع از مرتبهٔ n است.اگر به هنگام تحلیل الگوریتمها از مقادیر ثابت در توابع مورد بررسی صرف نظر کرده و تقریبهایی را با توجه به مرتبهٔ بزرگی این توابع در نظر بگیریم به بیراهه نرفته‌ایم.برای این مفهوم نمادی متداول است که می‌توان به سهولت از آن استفاده کرد: اگر آهنگ رشد تابع از مرتبهٔ n باشد، می‌نویسیم $f = O(n)$ (و می‌خوانیم تابع f از مرتبهٔ n است) تعریف دقیقتر نماد O ی بزرگ این است:

اگر f و g دو تابع باشند، آنگاه $f = O(g)$ اگر و تنها اگر دو مقدار ثابت c و k وجود داشته باشند به طوری که $f \leq cg + k$.

به عنوان مثال، اگر f تابعی باشد برابر $۳۷n + ۴$ آنگاه $f = O(n)$، زیرا می‌توانیم مقادیر ثابت c و k را به ترتیب ۳۷ و ۴ اختیار کنیم و در این صورت داریم $۳۷n + ۴ = ۳۷n + ۴$.مثال دیگری که به اندازهٔ مثال قبلی واضح نیست، تابع $۱۵۰n^۲ + ۹۷n - ۳۴$ است که عبارت است از $O(n^۲)$، زیرا در این حالت می‌توانیم مقادیر ثابت c و k را به ترتیب برابر با ۲۰۰ و ۱۳ اختیارکنیم ($۱۵۰n^۲ + ۹۷n - ۳۴ \leq ۲۰۰n^۲ + ۱۳$) مقادیر فراوان دیگری نیز می‌توان برای c و k اختیارکرد.

نماد O محاسن زیادی دارد.از جمله اینکه، به کمک آن، ثابتها و جملاتی که به کندی رشد می‌کنند از توابع کنار گذاشته می‌شوند و در نتیجه مقایسهٔ توابع آسانتر انجام می‌شود.با استفاده از آن، می‌توان از جزئیات کم‌اهمیت صرف نظر کرد و الگوریتم را در ساده‌ترین شکلش مطالعه کرد و به علاوه چون کار خسته‌کنندهٔ شمارش دقیق لازم نمی‌آید، تحلیل الگوریتم ساده‌تر می‌شود.پس ازکار و تجربهٔ کافی، قضاوت دربارهٔ زمان اجرای یک الگوریتم با یک نگاه به O امکان‌پذیر می‌شود.

مرتب کردن انتخابی

این الگوریتم در تابلو ۱ نشان داده شده است.در این الگوریتم، برای مرتب کردن فهرست به ترتیب صعودی، کوچکترین عضو فهرست مشخص شده و با عضو اول تعویض می‌شود.در مرحلهٔ بعدی لازم نیست که عضو اول بررسی شود، بلکه کوچکترین مقدار در میان عضوهای دوم تا n ام فهرست معین و با عضو دوم جابه‌جا می‌گردد.پیدا کردن کوچکترین عضو به طریق زیر انجام می‌شود:عضو اول را کوچکترین عضو می‌گیریم؛ این عضو با تمام عضوهای فهرست مقایسه می‌شود و هر جا عضوی کوچکتر از آن پیدا شد، این مقدار به عنوان کوچکترین مقدار جدید در نظر گرفته می‌شود.

حال برای تحلیل این الگوریتم ببینیم چند مقایسه انجام می‌شود.هر بار که می‌خواهیم کوچکترین عضو را بیابیم دست به مقایسه می‌زنیم.به ازای هر مقدار ممکن i و j یک مقایسه انجام می‌شود.به ازای $i = ۱$، مقدار j بین ۲ و n تغییر می‌کند که عبارت از $(n - ۱)$ مقدار برای j است؛ لذا در این مرحله $(n - ۱)$ مقایسه داریم.به ازای $i = ۲$، j بین ۳ و n تغییر می‌کند یعنی جمعاً $(n - ۲)$ مقدار اختیار می‌کند و لذا در این مرحله، $(n - ۲)$ مقایسه انجام می‌گیرد.وقتی $i = n - ۱$ ، j فقط یک مقدار اختیار خواهد کرد و بالاخره به ازای $i = n$، حلقهٔ j انجام نخواهد شد.بنابراین، تعداد کل مقایسه‌ها عبارت است از

$$(n - ۱) + (n - ۲) + (n - ۳) + \cdots + ۱$$

که این مجموع مساوی است با $\frac{n(n-۱)}{۲} = \frac{۱}{۲}(n^۲ - n)$.

با توجه به مطالب بالا در مورد نماد O،این تابع $O(n^۲)$ است.پس مرتب کردن انتخابی از مرتبهٔ $n^۲$ است یعنی برای فهرستی به اندازهٔ n، تعداد

تابلو ۱: *الگوریتم مرتب کردن انتخابی*

```
مرتب کردن انتخابی.
ورودی:آرایه‌ای چون A با n عضو
خروجی:  همان فهرست که مرتب شده است

۱.دستور i = ۱ را اجراکن.
۲.دستور m = i را اجراکن.
۳.دستور j = i + ۱ را اجراکن.
۴.اگر A(j) کوچکتر از A(m) است، دستور m=j را اجراکن و
  به شمارهٔ ۶ برو.
۵.جای A(j) و A(m) را عوض کن.
۶.اگر j < n، دستور j = j + ۱ را اجراکن.
۷.اگر i < n،دستور i = i + ۱ را اجراکن و به دستور ۲ برو.
۸.پایان.
```

مقایسه‌های لازم از مرتبهٔ n^2 است.ضمناً الگوریتم مشهور مرتب کردن حبابی هم $O(n^2)$ است.

غلبه بر n^2 : مرتب کردن ادغامی

آیا می‌توانیم نتیجه‌ای بهتر از $O(n^2)$ به دست آوریم؟بله، و الگوریتمی که این امر را تحقق می‌بخشد، الگوریتم مرتب کردن ادغامی است.در این روش از قاعدهٔ معروف «تجزیه و حل » برای حل مسائل بزرگ استفاده می‌شود به این معنی که یک مسألهٔ بزرگ را به چند مسألهٔ کوچکتر از همان نوع تقسیم می‌کنند و آنها را به طور مشابه حل می‌کنند.

در الگوریتم ادغامی، دو فهرست مرتب شده مانند A و B در هم ادغام می‌شوند و یک فهرست مرتب شده مانند C حاصل می‌شود.به این منظور، ابتدا عضوهای اول دو فهرست با هم مقایسه می‌شوند و چون فهرستها مرتب شده هستند، عضو کوچکتر از میان این دو، کوچکترین عضو فهرست کلی خواهد بود.اگر مثلاً، عضو اول A کوچکتر باشد، این عضو به عنوان عضو اول C قرار می‌گیرد.آنگاه عضو بعدی A با عضو اول B مقایسه می‌شود و مجدداً آنکه کوچکتر است به عنوان عضو بعدی C قرار داده می‌شود.این عمل ادامه می‌یابد تا آنکه تمام عضوهای A و B مقایسه شوند.در پایان این کار، C شامل همهٔ عضوهای A و B به ترتیب صعودی خواهد بود.

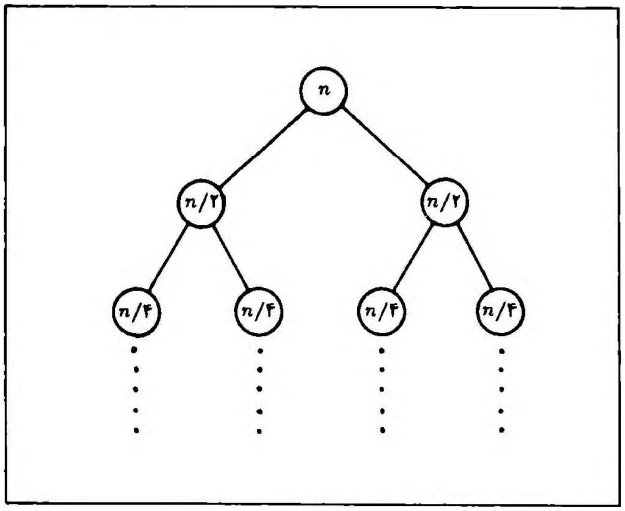
حال که چگونگی ادغام کردن دو فهرست مرتب شده معلوم شد، ببینیم چگونه باید یک فهرست را مرتب کرد.اگر فهرستی که باید مرتب شود بیش از یک عضو داشته باشد آن را به دو نیم (یا تقریباً به دو نیم) تقسیم می‌کنیم و این دو نیم را به وسیلهٔ الگوریتم ادغامی مرتب می‌کنیم.الگوریتم مرتب کردن ادغامی در تابلو ۲ نشان داده شده است.اگر فهرستی که باید مرتب شود فقط یک عضو داشته باشد کاری برای انجام دادن نیست.به عبارت دیگر، **فهرست خود به خود مرتب است.**

تحلیل الگوریتم مرتب کردن ادغامی مشکلتر از الگوریتم مرتب کردن انتخابی است زیرا این الگوریتم، بازگشتی است.ابتدا به مرحلهٔ ادغام در این الگوریتم توجه کنید.حداکثر n مقایسه لازم است تا دو فهرست مرتب شده

تابلو ۲ *الگوریتم مرتب کردن ادغامی*

مرتب کردن ادغامی.
ورودی: فهرستی به نام L.
خروجی: فهرستی به نام S.

۱. اگر L فقط یک عضو دارد، آنگاه S=L در غیر این صورت،
۲. L را به دو فهرست L۱ و L۲، هر کدام تقریباً به اندازهٔ نصف L، تقسیم‌کن.
۳. L۱ را به وسیلهٔ الگوریتم ادغامی مرتب کن و مرتب شدهٔ آن را در S۱ قرار بده.
۴. L۲ را به وسیلهٔ الگوریتم ادغامی مرتب کن و مرتب شدهٔ آن را در S۲ قرار بده.
۵. S۱ و S۲ را ادغام کن و حاصل را در S قرار بده.
۶. پایان



شکل ۲.الگوریتم مرتب کردن ادغامی

به اندازهٔ $n/2$ را در فهرستی به اندازهٔ n قرار دهیم.برای پی بردن به دلیل این امر، توجه کنید که هر بار که یک مقایسه انجام می‌دهیم یک عضو در فهرست نهایی وارد می‌کنیم.لذا تعداد مقایسه‌ها نمی‌تواند بیش از تعداد کل عضوهای فهرست باشد.پس، این قسمت الگوریتم،$O(n)$ است.

اما در مورد " بقیهٔ الگوریتم " وضع چگونه است؟از یک نظر در واقع "بقیهٔ الگوریتم" درکار نیست.تنها کار واقعی (یعنی مقایسه) در مرحلهٔ ادغام صورت می‌گیرد.پس سؤال این است که ادغام چند دفعه صورت می‌گیرد و هر دفعه، چند عضو ادغام می‌شوند.

بهترین راه برای جواب دادن به این سؤال در مورد الگوریتم ادغامی، این است که آن را به صورت درختی دودویی در نظر بگیریم (شکل ۲).در این درخت، هرگره نمایندهٔ یک بار فراخوانی الگوریتم ادغامی است.قلهٔ (ریشهٔ) درخت، نمایندهٔ فراخوانی اولیهٔ الگوریتم ادغامی برای فهرستی با اندازهٔ n است.سپس این الگوریتم دوبار به طور بازگشتی، هر بار برای فهرستی به اندازهٔ $n/2$ فرا خوانده می‌شود.هر یک از این فراخوانیها نیز باعث دوبار فراخوانی الگوریتم می‌شود تا آنکه بالاخره به فهرستهایی با اندازهٔ یک عضو برسیم.

در هر یک از سطوح این درخت، تعداد مقایسه‌ها$O(n)$ است.در سطح اول (قله) مقایسه‌ها تنها هنگامی انجام می‌شوند که دو فهرست هر یک به اندازهٔ $n/2$ در هم ادغام می‌شوند و فهرستی به اندازهٔ n حاصل می‌شود.در این عملیات، حداکثر n مقایسه انجام می‌گیرد.در سطح بعد، دو ادغام انجام می‌شوند که بر اثر هر یک، فهرستی با اندازهٔ $n/2$ تولید می‌شود.چون در هر یک از این ادغامها $n/2$ مقایسه انجام می‌گیرد، جمعاً n مقایسه صورت می‌پذیرد.سطح سوم معرف چهار عمل ادغام است که در هر یک $n/4$ مقایسه انجام می‌شود، و به همین ترتیب برای سطوح بعدی.

بنابراین، می‌توانیم بگوییم مدت اجرای الگوریتم ادغامی عبارت است از $O(nk)$که در آن k نمایندهٔ تعداد سطوح درخت مربوطه است.چون تعداد سطوح درخت درست مساوی تعداد دفعاتی است که می‌توان فهرستی به اندازهٔ n را مکرراً نصف کرد تا جایی که فهرستی به اندازهٔ ۱ حاصل شود،

k را می‌توان برحسب n بیان کرد.به عبارت دیگر، این تعداد عبارت است از تعداد دفعاتی که می‌توانیم n را بر ۲ تقسیم کنیم تا به عدد ۱ برسیم.این تعداد تقریباً مساوی $\log_2 n$ است.برای روشن شدن مطلب، فرض کنیم n توانی از ۲ باشد.به عنوان مثال، $4 = 2^2$.در این صورت، ۴ تقسیم بر ۲ مساوی است با ۲ و ۲ تقسیم بر ۲ مساوی است با ۱، و تعداد دفعات تقسیم، ۲ است.

با معلوم شدن این خاصیت، می‌توان نتیجه گرفت که الگوریتم ادغامی، $O(n\log_2 n)$ است که در آن، $\log_2 n$ تعداد سطوح درخت و n تعداد مقایسه‌ها در هر سطح است.این نتیجه بهتر از $O(n^2)$ است و هرچه n بیشتر باشد، باز هم بهتر خواهد بود.نگاهی به شکل ۱ نشان می‌دهد که تابع n^2 به مراتب سریعتر از تابع $n\log_2 n$ افزایش می‌یابد.همچنین با استفاده از چنین شکلی می‌توان این توابع را به ازای مقادیر مخصوصی مقایسه کرد.به ازای $n = 8$ داریم $n^2 = 64$ در حالی که $n\log_2 n = 24$ که اختلاف زیادی با هم ندارند.اما برای $n = 256$ داریم $n^2 \sim 65000$ در حالی که $n\log_2 n \sim 2000$.

دوباره این سؤال مطرح می‌شود که آیا می‌توانیم روشی بهتر از این داشته باشیم؟اگر از قبل بدانیم مقادیری که باید مرتب شوند در حدود معینی واقع هستند، مثلاً اعداد صحیح بین ۱ و ۱۰۰، می‌توان آنها را در مدتی از مرتبهٔ n مرتب کرد و به علاوه، احتیاجی به مقایسهٔ آنها نیست.در غیر این صورت، تنها راه مرتب کردن، مقایسهٔ اعداد خواهد بود.ثابت می‌شود که اگر مرتب کردن تنها به وسیلهٔ مقایسه انجام شود، تعداد بهینهٔ مقایسه‌ها $O(n\log_2 n)$ است.اثبات این مطلب از حوصلهٔ این مقاله خارج است ولی می‌توان این اثبات را درکتاب ساختارهای داده‌ها و الگوریتمها [۱] یافت.مفصلترین بحث دربارهٔ الگوریتمهای مرتب کردن در کتاب دانلد کنوت به نام هنر برنامه‌نویسی، جلد ۳، مرتب کردن و جستجو [۲] به عمل آمده است.

گرچه از لحاظ نظری، الگوریتم ادغامی بهترین الگوریتم ممکن برای مرتب کردن است، به هنگام کاربرد آن درمی‌یابیم مشکلاتی که در بحث نظری از نظر دور می‌مانند باعث کندشدن آن می‌شوند (به خصوص لزوم به کاربردن یک آرایهٔ اضافی در مرحلهٔ ادغام). الگوریتمی وجود دارد که مانند الگوریتم ادغامی، $O(n\log_2 n)$ است.اما در همهٔ کامپیوترهای امروزی، مدت اجرای آن کسری ثابت از مدت اجرای الگوریتم ادغامی است (به خاطر آوریم که $2n\log_2 n$ و $1000n\log_2 n$ هر دو $O(n\log_2 n)$ می‌باشند).این الگوریتم، موسوم به الگوریتم مرتب کردن سریع، سریعترین تکنیک شناخته شده برای مرتب کردن براساس مقایسه است.

مرتب کردن سریع

در الگوریتم سریع به صورت زیر عمل می‌کنیم: اگر فهرستی که قرار است مرتب شود فقط یک عضو داشته باشد، کاری انجام نمی‌دهیم.در غیر این صورت عضوی از فهرست را انتخاب کرده نام آن را مبدأ می‌گذاریم.حال فهرست را به دو قسمت تقسیم می‌کنیم، عضوهایی که کوچکتر از، یا مساوی با، مبدأ هستند و عضوهایی که از این عضو بزرگترند.سپس این دو قسمت را مرتب می‌کنیم.پس از مرتب کردن دو قسمت کار دیگری لازم نیست انجام دهیم زیرا در هنگام قسمت کردن، تمام عضوهای کوچکتر را در ابتدا و تمام عضوهای بزرگتر در انتهای فهرست قرار داده‌ایم.

الگوریتم سریع شبیه الگوریتم ادغامی است و در واقع از آن نشأت گرفته‌است.اختلاف اصلی دو الگوریتم آن است که در الگوریتم ادغامی عمل ادغام پس از عملیات بازگشتی صورت می‌گیرد اما در الگوریتم سریع، عمل **قسمت کردن** قبل از عملیات بازگشتی صورت می‌گیرد.چیزی که باعث کارایی بیشتر الگوریتم سریع نسبت به الگوریتم ادغامی می‌شود آن است که مانند الگوریتم مرتب کردن انتخابی، تمام عملیات درجا صورت می‌گیرد.به عبارت دیگر، همهٔ عملیات را در فهرست اصلی انجام می‌دهیم.تابلو ۳ الگوریتم مرتب کردن سریع را نشان می‌دهد.

فهرست را به طریق زیر تقسیم می‌کنیم.ابتدا و انتهای فهرست را در نظر می‌گیریم.مثلاً فرض کنید i نشان‌دهندهٔ ابتدا و j نشان‌دهندهٔ انتهای فهرست باشد.i را به جلو حرکت می‌دهیم تا به عضوی بزرگتر از مبدأ برسیم؛ j را به طرف عقب حرکت می‌دهیم تا به عضوی کوچکتر از مبدأ برسیم.آنگاه عضوی را که i نشان‌دهندهٔ آن است با عضوی که j آن را نشان می‌دهد، تعویض می‌نماییم.این کار را تکرار می‌کنیم تا i و j به هم برسند؛ در این هنگام فهرست به نحو دلخواه تقسیم شده است.شکل ۳ نشان می‌دهد که چگونه فهرستی شامل پنج عضو به این روش تقسیم می‌گردد.

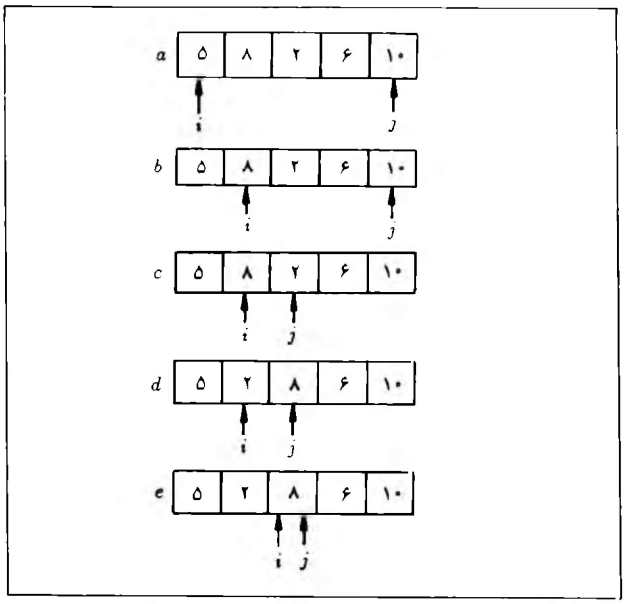
تحلیل الگوریتم سریع مشابه تحلیل الگوریتم ادغامی است.در مرحلهٔ تقسیم، $O(n)$ مقایسه انجام می‌شود و در صورتی که فهرست به دو قسمت تقریباً مساوی تقسیم شود، درخت الگوریتم سریع دارای $\log_2 n$ سطح خواهد بود.بنابراین، الگوریتم سریع در اغلب موارد، $O(n\log_2 n)$ است.این تحلیل متکی به این فرض است که فهرست به دو قسمت نسبتاً مساوی تقسیم شود.این فرض نیز بستگی به چگونگی انتخاب مبدأ خواهد داشت.گیریم همیشه عضو اول را به عنوان مبدأ انتخاب کنیم.حال اگر فهرستها را دارای توزیع تصادفی فرض کنیم، عضو اول به طور متوسط، در اطراف میانهٔ بزرگترین و کوچکترین عضوها خواهد بود و لذا فهرست تقریباً به دو نیم تقسیم خواهد شد.به این ترتیب می‌توان گفت که روش سریع، به طور متوسط، $O(n\log_2 n)$ خواهد بود.

حال نگاهی به بدترین حالت بکنیم.فرض کنیم مبدأ انتخاب شده همواره بزرگترین (کوچکترین) عضو فهرست باشد.به این ترتیب، فهرست به دو

تابلو ۳ *الگوریتم مرتب کردن سریع*

مرتب کردن سریع.
ورودی: آرایه‌ای چون A با عضوهایی از ۱ تا n.
خروجی: همان آرایه که مرتب شده است.

۱.مبدئی انتخاب کن.
۲.فهرست را به دو قسمت کن و تمام عضوهای نابیشتر از مبدا را در خانه‌های ۱ تا $i-1$ قرار بده.
۳.مرتب کردن سریع را در مورد عضوهای A به ازای ۱ تا $i-1$ انجام بده.
۴.مرتب کردن سریع را در مورد عضوهای A به ازای i تا n انجام بده.
۵.پایان



شکل ۳. تقسیم کردن یک آرایه در الگوریتم سریع : در ردیف (الف)، i و j به ترتیب ابتدا و انتهای آرایه را نشان می‌دهند.عضو اول، یعنی ۵، را به عنوان مبدأ در نظر گرفته‌ایم.در ردیف (ب)، i به طرف جلو می‌رود تا جایی که عضو نظیر آن بزرگتر از مبدأ باشد.در ردیف (پ)، j به طرف عقب حرکت می‌کند تا به عضوی برسد که کمتر از، یا مساوی با، مبدأ است.در ردیف (ت)، دو عضوی که پاهٔ i و j نشان داده شده بودند، تعویض شده‌اند.در ردیف (ث)، i مجدداً به طرف جلو می‌رود و به j می‌رسد.به این ترتیب، مرحلهٔ تقسیم تمام می‌شود.

قسمت تقسیم می‌شود که یکی به اندازهٔ ۱ و دیگری به اندازهٔ $n-1$ است.فهرست دومی نیز به دو قسمت تقسیم می‌شود که یکی به اندازهٔ ۱ و دیگری به اندازهٔ $n-2$ خواهد بود و الی آخر.مجموعاً n بار عمل تقسیم و برای آنها به ترتیب، تعداد $n-1$، $n-2$، و ... مقایسه لازم خواهد بود.ملاحظه می‌شود که تحلیل این الگوریتم شبیه الگوریتم انتخابی است و البته، نتیجهٔ این دو تحلیل هم یکی است.پس در بدترین حالت، الگوریتم سریع، $O(n^2)$ است.

تا اینجا معلوم شد که انتخاب مبدأ بسیار مهم است.اگر این نقطه را خوب انتخاب کنیم می‌توانیم این احتمال را که الگوریتم مرتب کردن کند باشد کاهش دهیم.روشهای متعددی وجود دارند که این نقطه خوب انتخاب شود.مثلاً ممکن است سه عضو را به تصادف انتخاب کنیم و سپس میانهٔ آن سه عضو (عضو وسطی) را به عنوان مبدأ در نظر بگیریم.با این کار، احتمال اینکه مدت اجرای الگوریتم سریع از مرتبهٔ $O(n\log_2 n)$ باشد و در عین حال، چندان هم به درازا نکشد، افزایش خواهد یافت.در صورتی که فهرست مفروض، مرتب شده یا دارای ترتیب عکس باشد، روش انتخاب عضو اول به عنوان مبدأ به همان بدترین حالت منجر می‌شود.بنابراین، اگر قرار باشد فهرستهای زیادی که تقریباً مرتب‌اند با الگوریتم سریع مرتب شوند، بهتر است مبدأ با دقت بیشتری انتخاب شود.

مسألهٔ دیگری در مورد الگوریتم سریع مربوط به مقدار فضایی است که اشغال می‌کند.تا این مرحله از بحث ، مقدار فضای مصرفی را بررسی نکرده‌ایم ولی این موضوع می‌تواند مهم باشد.در الگوریتم انتخابی فضای کمی اشغال می‌شود و از آن مهمتر، فضای اشغال‌شده در این الگوریتم مستقل از اندازهٔ فهرست است.اما به دلیل اینکه الگوریتم سریع بازگشتی است، هر دفعه که روند به کار رفته فراخوانده می‌شود مقداری اطلاعات در پشتهٔ حین اجرا قرار می‌گیرد.در بدترین حالت، الگوریتم سریع ممکن است n دفعه به طور بازگشتی فرا خوانده شود و لذا پشتهٔ مذکور ممکن است $O(n)$ فضا اشغال نماید.برای مقابله با این مسأله، روش زیرکانه‌ای را می‌توان به کار برد که تضمین می‌کند فضای اشغال‌شده به وسیلهٔ پشته بیش از $O(\log_2 n)$ نباشد.به این منظور، اولین فراخوان بازگشتی را به قسمت کوچکتر فهرست اختصاص داده و به جای فراخوان قسمت بزرگتر، این عمل را به وسیلهٔ یک حلقهٔ بازگشتی انجام می‌دهیم.این کار خسته کننده و تا حدی پیچیده است و در مواردی ممکن است ارزش نتیجهٔ به دست آمده را نداشته باشد.لذا بهتر است تنها در مواردی که فضای به کار رفته واقعاً مهم است، آن را به کار برد.

کار با داده‌های واقعی

گرچه دستیابی به تقریبی از مدت اجرای الگوریتمها، با توجه به مرتبهٔ این مدت، مفید است، دیدن مثالی واقعی نیز جالب است.به این منظور، الگوریتمهای سریع در مورد فهرستهایی با اندازه‌های متفاوت که عضوهای آنها به تصادف از میان اعداد صحیح انتخاب شده بودند، اعمال شد.نتایج را در جدول زیر می‌بینید.از این جدول معلوم است که تفاوت مدت اجرا در مورد فهرستهای کوچک قابل صرف نظر کردن است اما در مورد فهرستهای بزرگ، این تفاوت خیلی زیاد است.زبان برنامه‌نویسی به کار رفته، مک پاسکال و کامپیوتر مورد استفاده، مکینتاش بوده است.

جدول ۱. مقایسهٔ مدت اجرای الگوریتمهای انتخابی و سریع (برحسب دقیقه:ثانیه)

اندازهٔ فهرست	مرتب کردن انتخابی	مرتب کردن سریع
۱۰	۰/۰۱	۰/۰۱
۵۰	۰/۰۵	۰/۰۵
۱۰۰	۰/۲۰	۰/۰۹
۵۰۰	۷/۴۹	۱/۰۲
۱۰۰۰	۳۱/۰۱	۲/۱۰

مراجع

1. Aho, Hopcroft, and Ullman, *Data Structures and Algorithms* (Reading, MA: Addison - Wesley.1983, p. 382).

2.Donald E. Knuth, *Art of Computer Programming, Volume 3: Sorting and Searching* (Reading, MA: Addison - Wesley.1973).

****

• Jonathan Amsterdam. " An analysis of sorts, " *Byte*, September 1985, 105 -112.

قسمتهایی از اواخر این مقاله که برای خوانندگان فنر ریاضی چندان جالب‌تشخیص داده نشده در ترجمه حذف شده است.

* جاناتان آمستردام در هنگام نوشتن این مقاله در دانشگاه ام.آی.تی. دانشجوی دکتری بوده است.