باسمه تعالی

طراحی الگوریتم ها

دانشکده مهندسی برق و کامپیوتر

فروردین ۱۴۰۳

استاد:

دکتر فلسفین

سپهر عبادی

۹۹۳۳۲۴۳

سوال ۱:

آرایه مقابل را به دو روش merge sort و quick sort مرتب کرده و تمامی مراحل برای اجرای این الگوریتم را توضیح دهید (همراه با رسم شکل همانند ترسیم های کتاب نیپولیتن)

Array = [9, 4 , 2 , 7 , 11 , 5 , 1 , 6 , 8 , 10 , 3 , 12]

مراحل: Merge Sort

۱.تقسیم:

آرایه را به دو نیمه تقسیم می‌کنیم:

[۹, ٤, ۲, ۷, ۱۱, ٥]و[۱, ٦, ۸, ۱۰, ۳, ۱۲]

۲.تقسیم بازگشتی:

برای هر یک از نیمه‌ها، عملیات تقسیم را به صورت بازگشتی انجام می‌دهیم.

بخش اول:

- [۹, ٤, ۲, ۷, ۱۱, ٥] => [۹, ٤], [۲, ۷], [۱۱, ٥]
- [۹, ٤] => [۹], [٤]
- [۲, ۷] => [۲], [۷]
- [۱۱, ٥] => [۱۱], [٥]

بخش دوم:

- [۱, ٦, ۸, ۱۰, ۳, ۱۲] => [۱, ٦], [۸, ۱۰], [۳, ۱۲]
- [۱, ٦] => [۱], [٦]
- [۸, ۱۰] => [۸], [۱۰]
- [۳, ۱۲] => [۳], [۱۲]

۳.ادغام:

حالا باید بخش‌های مرتب شده را ادغام کنیم.

بخش اول:

- [۹], [٤] => [٤, ۹]

- [۲], [۷] => [۲, ۷]

- [۱۱], [٥] => [٥, ۱۱]

سپس [۴, ۹] و [۲, ۷] و [۵, ۱۱] را ادغام می‌کنیم [۲, ٤, ٥, ۷, ۹, ۱۱] :

بخش دوم:

- [۱], [٦] => [۱, ٦]

- [۸], [۱۰] => [۸, ۱۰]

- [۳], [۱۲] => [۳, ۱۲]

سپس [۱, ۶] و [۸, ۱۰] و [۳, ۱۲] را ادغام می‌کنیم [۱, ۳, ٦, ۸, ۱۰, ۱۲] :

۴.ادغام نهایی:

حالا باید دو نیمه اصلی را ادغام کنیم:

[۲, ٤, ٥, ۷, ۹, ۱۱] -و [۱, ۳, ٦, ۸, ۱۰, ۱۲]

سپس این دو نیمه را ادغام می‌کنیم [۱, ۲, ۳, ٤, ٥, ٦, ۷, ۸, ۹, ۱۰, ۱۱, ۱۲] :

پس از انجام این مراحل، آرایه داده شده به صورت مرتب شده به شکل زیر خواهد بود:

[۱, ۲, ۳, ٤, ٥, ٦, ۷, ۸, ۹, ۱۰, ۱۱, ۱۲]

مراحل : Quick Sort

آرایه داده‌شده: [۹, ۴, ۲, ۷, ۱۱, ۵, ۱, ۶, ۸, ۱۰, ۳, ۱۲]

۱. انتخاب محور:

برای این آرایه، محور را می‌توان به صورت دلخواه انتخاب کرد. معمولاً انتخاب اولین یا آخرین عنصر به عنوان محور رایج است. در اینجا، ما اولین عنصر را به عنوان محور انتخاب می‌کنیم. بنابراین محور ما عدد ۹ است.

۲. تقسیم:

آرایه را بر اساس محور به دو بخش تقسیم می‌کنیم. عناصر کوچکتر از محور در یک بخش و عناصر بزرگتر در بخش دیگر قرار می‌گیرند.

- بخش کوچکتر از محور: [۴, ۲, ۷, ۵, ۱, ۶, ۸, ۳]

- بخش بزرگتر از محور: [۱۱, ۱۰, ۱۲]

۳. تقسیم بازگشتی:

هر دو بخش را به صورت بازگشتی مرتب می‌کنیم.

بخش کوچکتر از محور:

- [۴, ۲, ۷, ۵, ۱, ۶, ۸, ۳] => [۴, ۲, ۷, ۵, ۱, ۶, ۳, ۸] => [۳, ۲, ۱, ۴, ۷, ۶, ۵, ۸]

- بخش کوچکتر از محور به صورت بازگشتی مرتب شد.

بخش بزرگتر از محور:

- [۱۱, ۱۰, ۱۲] => [۱۰, ۱۱, ۱۲]

- بخش بزرگتر از محور به صورت بازگشتی مرتب شد.

۴. ادغام:

عناصر مرتب شده را با توجه به محور ادغام می‌کنیم.

- بخش کوچکتر از محور: [۱, ۲, ۳, ۴, ۵, ۶, ۷, ۸]

- محور: ۹

- بخش بزرگتر از محور: [۱۰, ۱۱, ۱۲]

۵. ادغام نهایی:

در نهایت، بخش‌های کوچکتر و بزرگتر از محور را به هم ادغام می‌کنیم.

- [۱, ۲, ۳, ۴, ۵, ۶, ۷, ۸, ۹, ۱۰, ۱۱, ۱۲]

پس از انجام این مراحل، آرایه داده شده به صورت مرتب شده به شکل زیر خواهد بود:

[۱, ۲, ۳, ۴, ۵, ۶, ۷, ۸, ۹, ۱۰, ۱۱, ۱۲]

سوال ۲:

برای مجموعه ای از اعداد، میانه به صورتی تعریف میشود که نیمی از داده ها از آن کوچکتر و نیمی دیگر از آن بزرگتر باشند. برای پیدا کردن میانه یک آرایه می توان آن را مرتب نمود و سپس میانه را مشخص نمود که این الگوریتم از اردر o(nlogn) بهره می برد. یک الگوریتم مبتنی بر رویکرد تقسیم و غلبه ارائه دهید که میانه یک آرایه را در زمان خطی محاسبه نماید.

الگوریتم تقسیم و غلبه برای محاسبه میانه‌ی یک آرایه به این صورت عمل می‌کند:

۱. ابتدا آرایه را مرتب می‌کنیم.

۲. سپس اگر طول آرایه فرد باشد، میانه آرایه این عنصر است که در میانه‌ی آرایه قرار دارد.

۳. اگر طول آرایه زوج باشد، میانه آرایه بین دو عنصر در وسط آرایه است.

۴. برای محاسبه میانه به صورت بازگشتی، آرایه را به دو بخش تقسیم می‌کنیم و بررسی می‌کنیم که عنصر میانی در کدام بخش قرار دارد. سپس با توجه به اینکه میانه آرایه در کدام بخش قرار دارد، ادامه‌ی عملیات را در بخش مربوطه انجام می‌دهیم.

۵. عملیات بازگشتی را تا زمانی که طول بخش‌ها به یک عدد معین (مثلا ۱ یا ۲) برسد انجام می‌دهیم. در این حالت، میانه برابر با مقدار معین شده است.

۶. در انتها، میانه آرایه را با توجه به میانه بین دو بخش محاسبه می‌کنیم و آن را به عنوان جواب نهایی برمی‌گردانیم.

این الگوریتم به صورت بازگشتی و با استفاده از تقسیم و حاکمیت عمل می‌کند که زمان اجرای آن به صورت متوسط به ازای یک آرایه با اندازه n، O(n) است.

سوال ۳:

یک الگوریتم مبتنی بر رویکرد تقسیم و غلبه ارائه دهید که در یک آرایه مرتب شده که دارای مقادیر تکراری است جایگاه اولین و آخرین خانه برابر با مقدار مورد نظر را برگرداند. سپس پیچیدگی این الگوریتم را با پیچیدگی الگوریتمی که بر اساس این رویکرد نیست مقایسه کنید.

Array = [۱, ۱ ,۱ , ۳ , ٤ , ٥]   Algo(۱) = [۰ , ۲]   Algo(۲) = [-۱,-۱]

برای حل این مسئله، می‌توان از رویکرد تقسیم و غلبه استفاده کرد. ابتدا می‌توان با استفاده از الگوریتم تقسیم و غلبه، مکان اولین و آخرین خانه‌هایی که مقدار مورد نظر (مقدار داده شده) در آنها قرار دارد را پیدا کرد. سپس با ترکیب این دو مکان، بازه مطلوب را به دست می‌آوریم.

۱. ابتدا از یک الگوریتم جستجوی دودویی برای پیدا کردن اولین خانه با مقدار داده شده استفاده می‌کنیم. اگر مقدار داده شده یافت نشد، خروجی -۱ برگردانده می‌شود.

۲. سپس از یک الگوریتم جستجوی دودویی دیگر برای پیدا کردن آخرین خانه با مقدار داده شده استفاده می‌کنیم. اگر مقدار داده شده یافت نشد، خروجی -۱ برگردانده می‌شود.

۳. در نهایت، مکان اولین و آخرین خانه‌هایی که مقدار مورد نظر در آنها قرار دارد را به عنوان خروجی الگوریتم برمی‌گردانیم.

حالا پیچیدگی این الگوریتم را با یک الگوریتمی که بر اساس روش تقسیم و غلبه نیست مقایسه می‌کنیم.

الگوریتمی که مستقیماً از رویکرد تقسیم و غلبه استفاده نمی‌کند، ممکن است نیاز به پیمایش کل آرایه داشته باشد تا اولین و آخرین خانه‌هایی با مقدار مورد نظر را پیدا کند. این الگوریتم ممکن است با استفاده از حلقه‌ها یا دیگر روش‌های مستقیم جستجوی خطی انجام شود. در نتیجه، پیچیدگی زمانی این الگوریتم به طور مستقیم به اندازه طول آرایه می‌پردازد، یعنی (O(n)، که در آن (n) طول آرایه است.

اما الگوریتم مبتنی بر تقسیم و غلبه، با استفاده از جستجوی دودویی، پیچیدگی زمانی بهتری دارد. اگر طول آرایه را با (n) نشان دهیم، هر دو جستجوی دودویی که در این الگوریتم استفاده می‌شود، پیچیدگی زمانی (O(log n)

دارند. بنابراین پیچیدگی زمانی کل الگوریتم، با در نظر گرفتن هر دو جستجوی دودویی، (O(log n)) خواهد بود که به طور معمول به مراتب بهتر از روش خطی است.

الگوریتم تقسیم و غلبه برای حل این مسئله به این صورت است:

۱. مرحله تقسیم:

– ابتدا آرایه را به دو نیمه تقسیم می‌کنیم.

– سپس به طور بازگشتی بر روی هر نیمه عملیات تقسیم را انجام می‌دهیم تا به آرایه‌هایی با طول یک عنصر برسیم.

۲. مرحله فرمان:

– در این مرحله، از پایه یا جزئیات حالت‌های خاص استفاده می‌کنیم تا مکان اولین و آخرین خانه‌هایی که مقدار مورد نظر در آنها قرار دارد را پیدا کنیم.

– برای این کار، به ازای هر زیرآرایه با طول یک عنصر، مکان آن عنصر را بررسی می‌کنیم و اگر مقدار آن با مقدار داده شده برابر بود، آن عنصر مکان مورد نظر است. اگر چنین عنصری یافت نشد، خروجی برابر –۱ است.

– سپس این مکان‌ها را بازگردانده و به صورت بازگشتی بر روی زیرآرایه‌های بزرگتر و کوچکتر از مقدار داده شده عملیات تقسیم و فرمان را انجام می‌دهیم تا به جواب نهایی برسیم.

به این ترتیب، الگوریتم تقسیم و غلبه برای حل این مسئله از دو مرحله تقسیم و فرمان استفاده می‌کند. در مرحله تقسیم، آرایه را به زیرآرایه‌های کوچکتر تقسیم می‌کند، و در مرحله فرمان، با استفاده از جستجوی دودویی مکان اولین و آخرین خانه‌هایی که مقدار مورد نظر در آنها قرار دارد را پیدا می‌کند. سپس این دو مکان به عنوان خروجی برگردانده می‌شود

سوال ٤:

زمستان در راه است! اولین کار شما طراحی یک نوع بخاری استاندارد با شعاع گرم ثابت برای گرم کردن تمام خانه ها است. موقعیت خانه ها و بخاری ها در دو آرایه به شما داده شده است. یک الگوریتم مبتنی بر رویکرد تقسیم و غلبه طراحی کنید که کمترین شعاع بخاری ممکن برای گرم کردن خانه ها را بیابد.

مثال:

Houses = [۱ , ۲ , ۳] , Heaters = [۲] Answer = ۱

از آنجایی که یک بخاری در خانه شماره ۲ وجود دارد با انتخاب شعاع بخاری برابر با ۱ تمامی خانه ها پوشش داده میشود.

Houses = [۱ , ۲ , ۳ , ٤ , ٥] , Heaters = [۱ , ۲] Answer = ۳

از آنجایی که نزدیک ترین بخاری به خانه شماره ٥ در خانه شماره ۲ قرار دارد پس نیاز است تا بخاری ها حداقل تا شعاع ۳ خانه را پوشش دهند.

Houses = [۱ , ۲ , ۳ , ٤ , ٥] , Heaters = [۳] Answer = ۲

از انجایی که فقط یک بخاری در خانه ۳ موجود است پس برای گرم کردن خانه های ۱ و ٥ به بخاری با شعاع ۲ نیازمندیم.

الگوریتم مبتنی بر تقسیم و غلبه برای حل این مسئله به شرح زیر است:

۱. مرحله تقسیم:

- ابتدا آرایه‌های خانه‌ها و بخاری‌ها را به دو نیمه تقسیم می‌کنیم.

– سپس به طور بازگشتی بر روی هر نیمه عملیات تقسیم را انجام می‌دهیم تا به زیرآرایه‌هایی با تعداد کمتری خانه و بخاری برسیم.

۲. مرحله غلبه:

– در این مرحله، برای هر زیرآرایه از خانه‌ها و بخاری‌ها، فاصله از هر خانه تا نزدیک‌ترین بخاری را محاسبه می‌کنیم.

– برای این کار، برای هر خانه در زیرآرایه خانه‌ها، فاصله آن را تا نزدیک‌ترین بخاری را محاسبه می‌کنیم.

– سپس بزرگترین این فواصل را بین زیرآرایه‌ها پیدا کرده و آن را به عنوان جواب نهایی انتخاب می‌کنیم.

برای روشن شدن مراحل این الگوریتم، با مثال زیر آن را توضیح می‌دهیم:

فرض کنید آرایه‌های خانه‌ها و بخاری‌ها به ترتیب به صورت زیر باشند:

Houses = [۱, ۲, ۳, ۴, ۵]

Heaters = [۱, ۲]

۱. مرحله تقسیم:

- ابتدا آرایه خانه‌ها و بخاری‌ها را به دو نیمه تقسیم می‌کنیم.
- در این مثال، زیرآرایه‌های خانه‌ها به **[۱, ۲, ۳]** و **[٤, ٥]** تقسیم می‌شوند.
- زیرآرایه‌های بخاری‌ها نیز به **[۱]** و **[۲]** تقسیم می‌شوند.

۲. مرحله غلبه:

- در این مرحله، برای هر زیرآرایه از خانه‌ها و بخاری‌ها، فاصله از هر خانه تا نزدیک‌ترین بخاری را محاسبه می‌کنیم.
- برای زیرآرایه **[۱, ۲, ۳]**، باید فاصله هر خانه از نزدیک‌ترین بخاری را محاسبه کنیم. در اینجا، نزدیک‌ترین بخاری به هر خانه باید بخاری با شعاع ۲ باشد. پس فواصل مربوط به هر خانه به ترتیب برابر با **[۱, ۰, ۱]** است.

- برای زیرآرایه **[٤, ٥]**، نزدیک‌ترین بخاری به هر خانه باید بخاری با شعاع ۲ باشد. پس فواصل مربوط به هر خانه به ترتیب برابر با **[۱, ۰]**است.
- بزرگترین این فواصل بین زیرآرایه‌ها برابر با ۲ است، بنابراین جواب نهایی نیز برابر با ۳ خواهد بود.

سوال ۵:

یک الگوریتم مبتنی بر رویکرد تقسیم و غلبه ارائه دهید که در یک رشته بزرگترین زیر رشته ای که حداقل دوبار تکرار شده است را پیدا کند . سپس پیچیدگی این الگوریتم را با پیچیدگی الگوریتمی که بر اساس این رویکرد نیست مقایسه کنید.

مثال:

String = ‘banana’ Answer = ‘ana’
String = ‘abcabcab’ Answer = ‘abcab’

برای حل این مسئله با استفاده از الگوریتم تقسیم و غلبه می‌توانیم به شرح زیر عمل کنیم:

۱. **مرحله تقسیم:**

- ابتدا رشته را به دو نیمه تقسیم می‌کنیم.
- سپس به طور بازگشتی بر روی هر نیمه عملیات تقسیم را انجام می‌دهیم تا به رشته‌هایی با طول کوچکتر برسیم.

۲. **مرحله غلبه:**

- در این مرحله، برای هر رشته زیر، محاسبه زیررشته‌هایی که حداقل دوبار تکرار شده‌اند را انجام می‌دهیم.
- سپس بزرگترین این زیررشته‌ها را بین رشته‌های زیر پیدا کرده و آن را به عنوان جواب نهایی انتخاب می‌کنیم.

الگوریتم تقسیم و غلبه برای حل این مسئله از دو مرحله تقسیم و فرمان استفاده می‌کند. در مرحله تقسیم، رشته را به زیررشته‌های کوچکتر تقسیم می‌کند و سپس در مرحله فرمان، زیررشته‌هایی که حداقل دوبار تکرار شده‌اند را پیدا می‌کند. این الگوریتم به وضوح از الگوریتم تقسیم و غلبه استفاده می‌کند، زیرا مسئله را به زیرمسائل کوچک‌تر تقسیم می‌کند و سپس نتایج آنها را ترکیب می‌کند تا به جواب نهایی برسد.

حالا با مقایسه با یک الگوریتمی که بر اساس تقسیم و غلبه نیست، می‌توانیم مشاهده کنیم که الگوریتم تقسیم و غلبه کمترین پیچیدگی زمانی را دارد. به عنوان مثال، یک روش نادرست و زمان‌بر برای حل این مسئله می‌تواند شامل استفاده از دو حلقه تو در تو برای بررسی هر زیررشته و اعمال کلیتی بر روی همه زیررشته‌ها باشد، که پیچیدگی زمانی آن به صورت مربعی است. اما الگوریتم تقسیم و غلبه با استفاده از رویکردی بازگشتی و بهینه‌سازی‌های مناسب، می‌تواند با پیچیدگی زمانی بهتر این مسئله را حل کند.

سوال ٦:

در مورد الگوریتم ضرب ماتریسی استراسن تحقیق کرده و این روش را برای دو ماتریس ٣*٣ دلخواه پیاده سازی کرده و زیرماتریس ها و پیچیدگی این الگوریتم را محاسبه کرده و آنرا با روش ضرب ماتریسی عادی مقایسه کنید.

الگوریتم ضرب ماتریسی استراسن یک الگوریتم کارآمد برای ضرب ماتریس‌ها است که برای کاهش تعداد عملیات مورد نیاز برای ضرب دو ماتریس، از تقسیم و غلبه استفاده می‌کند. این الگوریتم به طور بازگشتی ماتریس‌ها را به چهار زیرماتریس تقسیم می‌کند، سپس عملیات ضرب را برای این زیرماتریس‌ها انجام می‌دهد و در نهایت نتایج را با یکدیگر ترکیب می‌کند.

الگوریتم ضرب ماتریسی استراسن به شکل زیر عمل می‌کند:

١. ابتدا ماتریس‌های A و B را به چهار زیرماتریس تقسیم می‌کنیم A١١, A١٢, A٢١, A٢٢ :و B١١, B١٢, B٢١, B٢٢.
٢. سپس ماتریس‌های P١ تا P٧ را محاسبه می‌کنیم:
   - P١ = A١١ * (B١٢ - B٢٢)
   - P٢ = (A١١ + A١٢) * B٢٢
   - P٣ = (A٢١ + A٢٢) * B١١
   - P٤ = A٢٢ * (B٢١ - B١١)
   - P٥ = (A١١ + A٢٢) * (B١١ + B٢٢)
   - P٦ = (A١٢ - A٢٢) * (B٢١ + B٢٢)
   - P٧ = (A١١ - A٢١) * (B١١ + B١٢)
٣. سپس ماتریس نتیجه C را برحسب زیرماتریس‌های P١ تا P٧ محاسبه می‌کنیم:
   - C١١ = P٥ + P٤ - P٢ + P٦
   - C١٢ = P١ + P٢

- C۲۱ = P۳ + P٤
- C۲۲ = P۵ + P۱ - P۳ - P۷

حالا بیایید این الگوریتم را برای ضرب دو ماتریس ۳ x۳پیاده‌سازی کنیم و سپس پیچیدگی آن را محاسبه کنیم. سپس با روش ضرب ماتریسی عادی مقایسه خواهیم کرد. دو ماتریس A و B به شکل زیر در نظر بگیرید:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

$$B = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}$$

و الگوریتم ضرب ماتریسی استراسن به صورت زیر عمل می‌کند:

```python
def strassen_matrix_multiply(A, B):
    # Check if the matrices are 1x1
    if len(A) == 1:
        return [[A[0][0] * B[0][0]]]
    # Divide matrices A and B into submatrices
    n = len(A) // 2
    A11 = [row[:n] for row in A[:n]]
    A12 = [row[n:] for row in A[:n]]
    A21 = [row[:n] for row in A[n:]]
    A22 = [row[n:] for row in A[n:]]
    B11 = [row[:n] for row in B[:n]]
    B12 = [row[n:] for row in B[:n]]
    B21 = [row[:n] for row in B[n:]]
    B22 = [row[n:] for row in B[n:]]
    # Calculate products P1 to P7
    P1 = strassen_matrix_multiply(A11, matrix_subtract(B12, B22))
    P2 = strassen_matrix_multiply(matrix_add(A11, A12), B22)
    P3 = strassen_matrix_multiply(matrix_add(A21, A22), B11)
    P4 = strassen_matrix_multiply(A22, matrix_subtract(B21, B11))
    P5 = strassen_matrix_multiply(matrix_add(A11, A22), matrix_add(B11, B22))
    P6 = strassen_matrix_multiply(matrix_subtract(A12, A22), matrix_add(B21, B22))
    P7 = strassen_matrix_multiply(matrix_subtract(A11, A21), matrix_add(B11, B12))
    # Calculate result matrices C11, C12, C21, C22
    C11 = matrix_add(matrix_subtract(matrix_add(P5, P4), P2), P6)
    C12 = matrix_add(P1, P2)
    C21 = matrix_add(P3, P4)
    C22 = matrix_subtract(matrix_subtract(matrix_add(P5, P1), P3), P7)
    # Combine result matrices into single matrix
    C = []
    for i in range(n):
        C.append(C11[i] + C12[i])
    for i in range(n):
        C.append(C21[i] + C22[i])
    return C
def matrix_add(A, B):
    return [[A[i][j] + B[i][j] for j in range(len(A))] for i in range(len(A))]
def matrix_subtract(A, B):
    return [[A[i][j] - B[i][j] for j in range(len(A))] for i in range(len(A))]
# Example matrices
A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
B = [[9, 8, 7], [6, 5, 4], [3, 2, 1]]
# Calculate matrix multiplication using Strassen algorithm
result = strassen_matrix_multiply(A, B)
print("Result of matrix multiplication using Strassen algorithm:")
for row in result:
    print(row)
```

برای مقایسه پیچیدگی زمانی بین الگوریتم ضرب ماتریسی استراسن و الگوریتم ضرب ماتریسی عادی برای ماتریس‌های ۳×۳، باید تعداد کل عملیات مورد نیاز برای هر الگوریتم را محاسبه کنیم.

الگوریتم ضرب ماتریسی استراسن برای ماتریس‌های ۳×۳ دارای پیچیدگی زمانی O(۱) است. چون در هر مرحله تعداد محاسبات ثابت است و مستقل از اندازه ماتریس.

الگوریتم ضرب ماتریسی عادی برای ماتریس‌های ۳×۳ نیز دارای پیچیدگی زمانی O(۱) است. چرا که تعداد کل عملیات مورد نیاز نیز ثابت است و مستقل از اندازه ماتریس.

بنابراین، هر دو الگوریتم برای ماتریس‌های ۳×۳ دارای پیچیدگی زمانی ثابت O(۱) هستند. اما در کل، الگوریتم ضرب ماتریسی استراسن معمولاً برای ماتریس‌های بزرگ‌تر بهتر است زیرا دارای پیچیدگی زمانی بهتری است.