تمرین اول طراحی الگوریتم‌ها

اشکان شکیبا (۹۹۳۱۰۳۰)

سوال اول

معیارهای Theta، Big Omega و Big O در تحلیل الگوریتم‌ها برای بررسی حداقل، حداکثر و حد میانی زمان اجرای الگوریتم‌ها به کار می‌روند. هر یک از این معیارها در شرایط مختلفی برای مقایسه الگوریتم‌ها مناسب هستند، اما اغلب در مقایسه الگوریتم‌ها از معیار Big O استفاده می‌شود.

معیار Theta برای تحلیل بهترین و بدترین حالت‌های زمانی الگوریتم استفاده می‌شود. به عبارت دیگر، Theta برای بررسی تعداد مراحل الگوریتم در هر دو حالت بهترین و بدترین حالت مناسب است. معیار Theta معمولاً در الگوریتم‌هایی با کارایی ثابت و زمان اجرای تقریبا یکسان در هر حالت استفاده می‌شود.

معیار Big Omega برای تحلیل حداقل زمان اجرای الگوریتم در هر شرایطی مناسب است. این معیار برای بررسی کمترین تعداد مراحلی که الگوریتم می‌تواند انجام دهد، به کار می‌رود. به عبارت دیگر، این معیار به ما اطلاع می‌دهد که الگوریتم در بهترین حالت، حداقل چه تعداد مراحل را برای انجام یک وظیفه نیاز دارد.

معیار Big O برای تحلیل بدترین حالت زمانی الگوریتم مناسب است. این معیار به ما اجازه می‌دهد تا بدترین حالت اجرای الگوریتم را محاسبه کنیم و تضمین کنیم که الگوریتم به هیچ شکلی بیشتر از زمان محاسبه شده در این حالت به طور مداوم نیاز نداشته باشد.

سوال دوم

علاوه بر معیارهای Theta، Big Omega و Big O، برای مقایسه پیچیدگی الگوریتم‌ها معیارهای دیگری نیز وجود دارند.

معیار پیچیدگی محاسباتی:

این معیار برای تحلیل پیچیدگی الگوریتم‌هایی با توجه به تعداد عملیات محاسباتی که الگوریتم انجام می‌دهد، استفاده می‌شود. این معیار به عنوان یک معیار دقیق تر برای تحلیل الگوریتم‌هایی با تعداد عملیات محاسباتی بالا مفید است.

معیار پیچیدگی خارجی:

این معیار برای تحلیل پیچیدگی الگوریتم‌هایی که وابستگی به داده‌های ورودی دارند، استفاده می‌شود. این معیار برای تحلیل پیچیدگی الگوریتم‌هایی که تعداد داده‌های ورودی متفاوت را پردازش می‌کنند، مناسب است.

معیار پیچیدگی تئوری اطلاعات:

این معیار برای تحلیل الگوریتم‌هایی که با داده‌هایی با فراوانی های مختلف کار می‌کنند، مناسب است. این معیار از فرمول‌های پیچیدگی تئوری اطلاعات استفاده می‌کند.

به علاوه، پیچیدگی مکانی (حافظه) در برخی مسائل از اهمیت بسیاری برخوردار است، به‌طوری‌که الگوریتمی که از حافظه کمتری استفاده می‌کند به‌طور کلی بهینه‌تر است. برای مثال، در مسائلی که به ذخیره داده‌های بزرگ نیاز دارند، الگوریتم‌هایی که از حافظه کمتری استفاده می‌کنند، مزیت بیشتری دارند.

سوال سوم

برای مقایسه بهتر، می‌توان نتیجه‌گیری کرد که الگوریتم دوم متناسب با $40n^2$ عمل جمع است.

الگوریتم اول مسئله‌ای به اندازه ۱۰۰ را در واحد زمان حل کرده، پس در هر واحد زمان می‌توان $4(100^4)+2(100^2)=400020000$ عمل جمع انجام داد.

حال برای بررسی الگوریتم دوم، معادله $40n^2=400020000$ باید حل شود که به پاسخ $n=3162$ می‌رسد. بنابراین الگوریتم دوم در واحد زمانی مشابه می‌تواند مسئله‌ای به اندازه ۳۱۶۲ را حل کند.

بنابراین می‌توان نتیجه گرفت الگوریتم دوم برای حل این مسئله مناسب‌تر است، چرا که می‌تواند در زمانی یکسان مسئله‌ای با اندازه بسیار بزرگ‌تر را حل کند، یا به زبان ساده سریع‌تر است.

سوال چهارم

a) الگوریتم: شروع به پیمایش بر روی همه اعداد کرده و هر عدد را با خواسته مسئله مقایسه می‌کند و هر جا که به عددی برابر با پاسخ رسید، ایندکس آن را بازگردانی می‌کند.

پیچیدگی بهترین حالت: $O(1)$

پیچیدگی حالت کلی: $O(n)$

پیچیدگی بدترین حالت: $O(n)$

b) الگوریتم: ابتدا با مرتب‌سازی ادغامی اعداد را مرتب کرده و سپس ایندکس k لیست حاصل را بازگردانی می‌کند.

پیچیدگی بهترین حالت: $O(n \log n)$

پیچیدگی حالت کلی: $O(n \log n)$

پیچیدگی بدترین حالت: $O(n \log n)$

c) الگوریتم: بر روی لیست پیمایش کرده و در هر عدد، تعداد تکرارهای آن را در یک دیکشنری ثبت و یا بروزرسانی می‌کند و در نهایت بر روی مقادیر دیکشنری پیمایش کرده و کلید مربوط به بیشترین مقدار را به عنوان پرتکرارترین عدد لیست بازگردانی می‌کند.

پیچیدگی بهترین حالت: $O(n)$

پیچیدگی حالت کلی: $O(n)$

پیچیدگی بدترین حالت: $O(n)$

d) الگوریتم: مشابه الگوریتم بخش c، تعداد تکرارها را در یک دیکشنری ذخیره می‌کند، سپس با مرتب‌سازی نزولی مقادیر دیکشنری، کلید مربوط به kمین مقدار را بازگردانی می‌کند.

پیچیدگی بهترین حالت: $O(n)$

پیچیدگی حالت کلی: $O(n \log n)$

پیچیدگی بدترین حالت: $O(n \log n)$

سوال پنجم

a) درست

b) درست

c) درست

d) درست

سوال ششم

a) $T(n) = T(\sqrt{n}) + c$

فرض کنیم $m = \log n$، در این صورت $n = 2^m$ و داریم:

$T(2^m) = T(2^{m/2}) + c$

$S(m) = S(m/2) + c$

با فرض $a = 1$ و $b = 2$ داریم:

$m \wedge \log_b a = m^0 = 1$

بنابراین:

$S(m) \in \Theta(\log m)$

$T(n) \in \Theta(\log\log n)$

b) $T(n) = 2T(\sqrt{n}) + \log n$

فرض کنیم $m = \log n$، در این صورت $n = 2^m$ و داریم:

$T(2^m) = 2T(2^{m/2}) + m$

$S(m) = 2S(m/2) + m$

با فرض $a = 2$ و $b = 2$ داریم:

$m \wedge \log_b a = m^1 = m$

بنابراین:

$S(m) \in \Theta(m \log m)$

$T(n) \in \Theta(\log n \log\log n)$

c) $T(n) = 2T(\sqrt{n}) + \log n / \log\log n$

فرض کنیم $m = \log n$، در این صورت $n = 2^m$ و داریم:

$T(2^m) = 2T(2^{m/2}) + m / \log m$

$S(m) = 2S(m/2) + m / \log m$

$S(m) = \Sigma_{r=0}^{\log m} m(1 /(\log m - r)) = m(1 / \log m + 1 /(\log m - 1) +\ldots+ 1)$

که $1+1/2+1/3+\ldots+1/n = \log n$، بنابراین:

$S(m) \in \Theta(m \log\log m)$

$T(n) \in \Theta(\log n \log\log\log n)$

d) $T(n) = 3T(n/3) + n/\log n$

$T(n) = \Sigma_{k=0}^{\log3n-1} n / (\log_3 n - k)$

مشابه بخش c، حاصل این سری برابر $n \log\log n$ است و داریم:

$T(n) \in \Theta(n \log\log n)$

e) $T(n) = T(n/2) + T(n/4) + T(n/8) + n$

$T(n) = \sum_{k=0}^{\log n} (7/8)^k n$

می‌دانیم $\lim_{n\to\infty} \sum_{k=0}^{\log n} (7/8)^k n = 8n$، بنابراین:

$T(n) \in \Theta(n)$

f) $T(n) = \sqrt{n}\, T(\sqrt{n}) + n$

فرض کنیم $m = \log n$، در این صورت $n = 2^m$ و داریم:

$T(2^m) = 2^{m/2}\, T(2^{m/2}) + 2^m$

$S(m) = S(m/2) + 1$

با فرض $a = 1$ و $b = 2$ داریم:

$m \wedge \log_b a = m^0 = 1$

بنابراین:

$S(m) \in \Theta(\log m)$

$T(n) \in \Theta(n \log\log n)$

سوال هفتم

الگوریتم ادغام دو آرایه به شکل مرتب شده می‌تواند به این شکل باشد که در ابتدا برای هر آرایه یک اشاره‌گر تعریف شود که به ابتدای آن اشاره کند. سپس در هر مرحله مقادیر اشاره‌گرها را با یکدیگر مقایسه و عدد کوچک‌تر را به آرایه حاصل بیفزاید و اشاره‌گر مربوط به آن را افزایش دهد تا به عدد بعدی آرایه اشاره کند. هنگامی که همه اعداد یکی از آرایه‌ها به پایان رسید، تمام اعداد باقی‌مانده آرایه دیگر با همان ترتیب به انتهای آرایه حاصل افزوده می‌شوند.

پیچیدگی زمانی این الگوریتم O(n+m) است، چرا که بر روی اعداد هر دو آرایه پیمایش می‌کند. پیچیدگی حافظه‌ای آن نیز O(1) است، چرا که در هنگام محاسبه حاصل هیچ نیازی به حافظه‌ای مجزا ندارد.

سوال هشتم

a) الگوریتم: محدوده سطر و ستون مکان مناسب برای درج را با دو بار انجام جست‌وجوی باینری پیدا می‌کند.

پیچیدگی: پیچیدگی جست‌وجوی باینری بر روی آرایه یک بعدی n عضوی از مرتبه O(logn) است، بنابراین پیچیدگی الگوریتم گفته شده از مرتبه O(log(n+m)) خواهد بود.

b) الگوریتم: ابتدا به سراغ بالاترین درایه ستون چپ می‌رود و آن را بررسی می‌کند، در صورتی که خالی باشد به سراغ درایه‌های بعدی رفته و این روند تا یافتن عدد مورد نظر ادامه می‌یابد.

پیچیدگی: در بدترین حالت الگوریتم باید n*m درایه را بررسی کند، بنابراین از مرتبه O(nm) است.

c) الگوریتم: مشابه بخش a، با انجام جست‌وجوی باینری بر روی سطرها و نیز ستون‌ها، به دنبال عدد مورد نظر می‌گردد تا وجود یا عدم وجود آن را مشخص کند.

پیچیدگی: مشابه بخش a به دلیل استفاده از جست‌وجوی باینری، پیچیدگی الگوریتم از مرتبه O(log(n+m)) است.