به نام خداوند

ساختمان داده و الگوریتم

حل تمرین دوم

دکتر حاجی اسماعیلی

دانشکده ریاضی و علوم کامپیوتر

فروردین ماه ۱۴۰۲

---

۱ صرفا مقایسس

۱-۱

- $Bubble\ sort$

$Bubblesort$ عناصر کنار هم دیگه در یک آرایه را مقایسه می‌کند و اگر ترتیب اشتباهی داشته باشند، آن‌ها را تعویض می‌کند. این روند را تا زمانی که کل آرایه مرتب شود تکرار می کند.

- بهترین حالت

اگر آرایه از قبل مرتب شده باشد، $Bubblesort$ فقط باید یک بار از آرایه عبور می کند که زمان $O(n)$ طول می‌کشد. این به این دلیل است که هیچ تعویضی لازم نیست و الگوریتم می تواند زودتر خاتمه یابد.

- بدترین حالت

بدترین حالت: اگر آرایه به ترتیب معکوس باشد، که میشود $O(n^2)$

- $Insertion\ sort$

$InsertionSort$ هر عنصر از یک آرایه را در موقعیت صحیح خود در یک زیرآرایه مرتب شده قرار می دهد. این روند را تا زمانی که کل آرایه مرتب شود تکرار می کند.

- بهترین حالت

اگر آرایه از قبل مرتب شده باشد، مرتب سازی فقط نیاز به انجام یک جابه جایی در هر خانه را دارد که زمان $O(n)$ طول می کشد. این به این دلیل است که هیچ عنصری نیاز به جابجایی ندارد..

- بدترین حالت

اگر آرایه در جهت معکوس باشد، $InsertionSort$ باید $n-1$ جابه جایی برای هر $node$ انجام

دهد و هر عنصر را به موقعیت صحیح خود در زیرآرایه مرتب شده منتقل کند، که زمان $O(n^2)$ طول می کشد.

- $Merge\ sort$

$MergeSort$ یک آرایه را به دو نیمه تقسیم می کند به صورت بازگشتی، $Node$ های نهایی را مرتب می کند و سپس دو نیمه را در یک آرایه مرتب شده ادغام می کند.

- بهترین حالت

بهترین حالت پیچیدگی زمانی $MergeSort$ $O(nlogn)$ است. این زمانی اتفاق می افتد که آرایه از قبل مرتب شده باشد.

- بدترین حالت

بدترین حالت پیچیدگی زمانی $MergeSort$ نیز $O(nlogn)$ است. این زمانی اتفاق می افتد که هر فراخوانی بازگشتی آرایه را به دو نیمه با اندازه های مختلف تقسیم می کند و باعث می شود درخت فراخوانی های بازگشتی نامتعادل شود و یا اینکه آرایه به صورت کاهشی چیده شده باشد.

۲-۱

بر اساس تحلیل پیچیدگی زمانی، من استفاده از $Mergesort$ یا $Heapsort$ را برای مرتب‌سازی آرایه‌های بزرگ توصیه می‌کنم.ولی $Mergesort$ که برای آرایه‌های بزرگ کارآمد است به فضای بیشتری نیاز دارد، در حالی که $Heapsort$ به فضای اضافی نیاز ندارد و می‌تواند برای مرتب‌سازی $Inplace$ استفاده شود.

۲ $Mergesort$ عه بد

همانطور که میدانیم الگوریتم $MergSort$ با وجود عملکرد خوب زمانی ولی از دیدگاه مصرف حافظه عملکزد افتضاحی دارد و برای همین الگوریتمی وجود دارد به نام $Externalsort$ که ۲ یا ۳ برابر کند تر است ولی حافظه مصرفی از $O(n)$ به $O(1)$ میرسد.

$Externalsort$ یک الگوریتم مرتب‌سازی است که زمانی استفاده می‌شود که داده‌هایی که قرار است مرتب شوند نمی‌توانند یکباره با هم در حافظه قرار بگیرند. در این سناریو، داده ها به تکه های کوچکتری تقسیم می شوند که می توانند در حافظه قرار گیرند، به صورت جداگانه مرتب شده و سپس با هم ادغام می شوند.

$Externalsort$ معمولاً شامل دو مرحله است:

۱. مرحله مرتب‌سازی: در مرحله مرتب‌سازی، داده‌ها از فایل ورودی خوانده می‌شوند و به تکه‌های کوچک‌تری تقسیم می‌شوند که سپس با استفاده از یک الگوریتم در حافظه مرتب می‌شوند. هنگامی که تکه های کوچکتر مرتب شدند، به عنوان فایل های موقت بر روی دیسک نوشته می شوند.

۲. فاز $Merge$: در طول مرحله $Merge$، فایل های موقت مرتب شده در یک فایل خروجی مرتب شده ترکیب می شوند. این کار با ادغام مکرر جفت فایل های موقت تا زمانی که یک فایل مرتب شده تولید شود، انجام می شود.

برای تغییر $Mergesort$ به یک الگوریتم $Externalsort$، باید الگوریتم را طوری تغییر دهیم که داده‌هایی را که خیلی بزرگ هستند و نمی‌توانند یک‌باره در حافظه جای دهند، تغییر دهیم. این شامل تقسیم داده ها به قطعات کوچکتر، مرتب کردن آنها به صورت جداگانه در حافظه و سپس ادغام تکه های مرتب شده با هم به همان روشی است که در بالا توضیح داده شد. این فرآیند می تواند تا زمانی که کل مجموعه داده مرتب شود تکرار شود.

۳ سریع تر نمیشه

در DTM، فرض می‌کنیم که هر الگوریتم مرتب‌سازی مبتنی بر مقایسه را می‌توان به عنوان یک درخت دودویی نشان داد، که در آن هر گره داخلی مقایسه‌ای بین دو عنصر و هر گره برگ نشان‌دهنده یک دنباله مرتب‌شده است.

ارتفاع درخت تصمیم برای یک الگوریتم مرتب‌سازی نشان‌دهنده بدترین تعداد مقایسه‌های مورد نیاز برای مرتب‌سازی توالی ورودی است. بنابراین، کران پایین برای مسئله مرتب‌سازی حداقل ارتفاع درخت تصمیم است که می‌تواند تمام دنباله‌های ورودی ممکن به طول n را مرتب کند.

و اینکه حداقل ارتفاع درخت تصمیم برای مرتب سازی n عنصر $(nlogn)$ است. این بدان معناست که هر الگوریتم مرتب‌سازی مبتنی بر مقایسه باید حداقل $(nlogn)$ مقایسه را در بدترین حالت انجام دهد تا n عنصر را مرتب کند.