5. الگوریتم‌های مراتب سازی

5. مرتب سازی

الگوریتم های مرتب سازی را می توان به چند روش دسته بندی کرد.

- مقایسه ای و غیر مقایسه ای

مبتنی بر مقایسه : در مرتب سازی مقایسه ای عناصر موجود مستقیما با یکدیگر مقایسه می شوند و این مقایسه تعیین کننده مکان آنهاست و با مقایسه دو عنصر جابجایی عناصر صورت می گیرد.

غیر مبتنی بر مقایسه: هیچ مقایسه ای میان عناصر صورت نمی گیرد و روشهای دیگری مکان نهایی عناصر را تعیین می کنند.

پیچیدگی زمانی الگوریتم های مرتب سازی مقایسه ای در بدترین حالت از $O(n \log n)$ بهتر نمی شود درحالی که الگوریم مرتب سازی غیر مقایسه ای می تواند پیچیدگی بدترین حالت در $O(n)$ نیز داشته باشند.

- **پایدار (stable) و ناپایدار(unstable)**

الگوریتم مرتب سازی پایدار است اگر هنگام مرتب سازی موقعیت نسبی عناصر با کلید برابر را نسبت به هم تغییر ندهد و ترتیب نسبی عناصر مساوی قبل و بعد از مرتب سازی یکسان باشد.

- **خارجی (external) و داخلی(internal)**

در مرتب سازی داخلی تنها از حافظه اصلی استفاده میشود و در صورتی قابل استفاده است که تعداد وحجم عناصر بیشتر از ظرفیت حافظه اصلی نباشد در صورتی که نتوان مرتب سازی را به صورت داخلی انجام داد از مرتب سازی خارجی استفاده میشود که از حافظه جانبی بهره می برد.

- **درجا (in-place) و برون جا(out-place)**

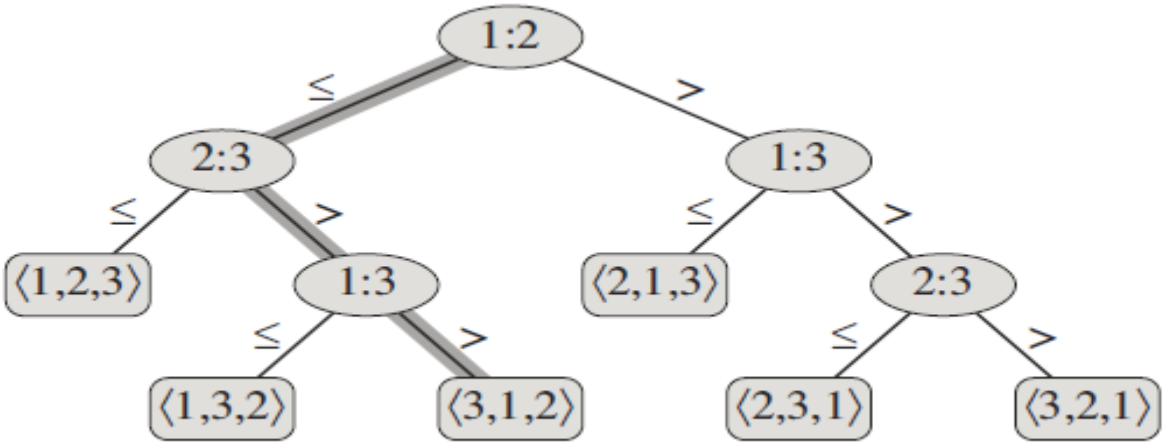
الگوریتم مرتب سازی را درجا میگویند، زمانی که فضای موقت و مورد استفاده ان با اندازه ورودی ارتباط نداشته باشد. مرتب سازی درجا عناصر را با پیچیدگی حافظه $O(1)$ مرتب می کند.

از سوی دیگر اندازه فضای اضافی موقتی که الگوریتم های مرتب سازی برون جا نیاز دارند با تعداد داده های ورودی متناسب است.

درخت تصمیم Decision Tree

درخت تصمیم یک روش برای معادل سازی تصمیم گیری است. مساله ی تصمیم گیری به هر مساله گفته میشود که از بین k انتخاب یکی از تصمیم ها را بگیرد. در درخت تصمیم برگ ها انتخاب های ممکن ما هستند و گره های داخلی شرط های ما می باشند. (برای اینکه کل انتخاب های ممکن را پوشش دهیم، باید حداقل به آن تعداد برگ داشته باشیم.)

به عنوان مثال شکل زیر را در نظر بگیرید:



در این شکل میخواهیم اعداد ۱، ۲ ، ۳ را مرتب کنیم. در کل $3!=6$ حالت برای چینش این سه عدد در کنار یکدیگر داریم. بنابراین تعداد برگ ها حداقل ۶ تا باید باشد. هر گره ی داخلی بیانگر یک شرط است برای مثال در گره ی ریشه، عناصر ۱ و ۲ با هم مقایسه شده اند و اگر ۱ از ۲ زود تر آمده باشد، به زیر درخت سمت چپ رفته و اگر ۲ زودتر از ۱ آمده باشد به زیر درخت سمت راست میرویم.

در این رابطه مساله ی مرتب سازی مبتنی بر مقایسه یک مساله ی تصمیم گیری می باشد چرا که از بین تمام جایگشت های قرارگیری n عدد در کنار هم (کل n! حالت ممکن)، میخواهیم یک جایگشت را انتخاب کنیم.

در این درخت تصمیم، تعداد برگ ها بزرگتر مساوی n! است. از طرفی میدانیم ارتفاع هر درخت دودویی بزرگتر مساوی (تعداد برگهای آن)log می باشد و هزینه ی زمانی پیدا کردن جواب مطلوب در درخت تصمیم برابر ارتفاع درخت می باشد (چرا که در هر سطر یک مقایسه لازم است) بنابرین:

h >= log(Tedad Bargha) >= log(n!) = nlogn

➔ h >= nlogn

1.5. مرتب سازی شمارشی

در مرتب‌سازی شمارشی فرض بر این است که هر یک از n عنصر ورودی، یک مقدار صحیح بین 0 تا k به ازای یک مقدار صحیح k می باشد. هنگامی که k=O(n) باشد، مرتب‌سازی در زمان $\theta(n)$ اجرا می شود.

ایده اساسی مرتب‌سازی شمارشی این است که به ازای هر عنصر ورودیx، تعداد عناصر کوچکتر از x تعیین شود. با این اطلاعات می توان مکان عنصر x در آرایه خروجی را به طور مستقیم تعیین کرد. برای مثال اگر 17 عنصر کوچکتر از x داریم، x متعلق به خانه شماره 18 از رشته خروجی می‌باشد. این طرح، باید اندکی تغییر کند تا در شرایطی که عناصر مختلف دارای مقادیر یکسان وجود دارند نیز قابل اعمال باشد. چرا که نمی خواهیم عناصر دارای مقدار مساوی را در یک مکان قرار دهیم.

در کد مرتب‌سازی شمارشی، فرض می‌کنیم که ورودی، آرایهA[1…n] می باشد که length[A]=n است. به دو آرایه دیگر نیز نیاز داریم. یکی آرایهB[1…N] که خروجی مرتب شده را در خود نگه دارد و یکی آرایهC[0…k] که فضای کاری موقتی را برای ما فراهم می کند.

```
Counting–Sort ( A, B, K)
1       for  i ← 0   to   k
2                do C[i]    o
```

3 for j ← 1 to length[A]

4 do C[A[j]]= C[A[j]]+1

5 C[i] now contains the number of elements equal to i

6 for i ←1 to k

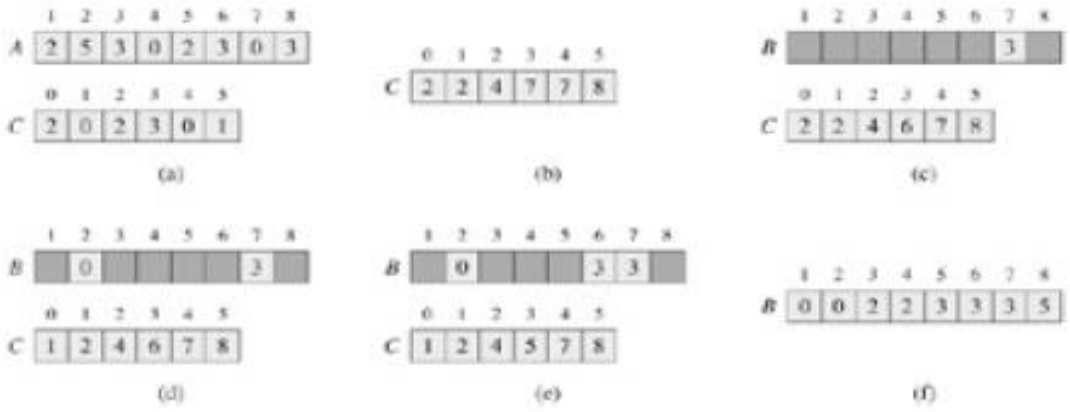
7 do C[i] ← C[i] +C[i-1]

8 C[i] now contains the number of elements less than or equal to i.

9 for j← length [A] downto 1

B [C[A[j]]] ← A[j]

C[A[j]] ← C[A[j]]-1



شکل 8. 2 : عملیات مرتب‌سازی شمارشی بر روی آرایه ورودی A[1…8] که هر عنصر آن دارای مقداری صحیح و نامنفی و کوچکتر یا مساوی 5 است.

a) آرایهA و آرایه کمکیC بعد از خط 4

b) آرایه C بعد از خط 7

c-e) آرایه خروجیB و آرایه کمکیC پس از یک ، دو و سه بارتکرار حلقه خطوط 9 تا 11 تنها خانه‌های روشن در آرایهB پر شده‌اند.

f) آرایه مرتب‌شده خروجی B

شکل 8. 2 روندکار مرتب‌سازی شمارشی را نمایش می‌دهد. پس از مقدار دهی اولیه در حلقه for در خطهای 1 و 2، در حلقه for در خط‌های 3 و 4 به تک تک عناصر ورودی رسیدگی می کنیم. اگر مقدار یک عنصر ورودی i بود، C[i] را یک واحد افزایش می‌دهیم. لذا پس از خط 4، C[i] تعداد عناصر ورودی مساوی با i در ورودی را به ازای k و ...،i=0،2،1 نگهداری می‌کند. در خطوط 6 و 7، به ازای i=0،1،...،k ، تعداد عناصر کوچکتر مساویi در آرایه ورودی را در C[i] ذخیره می‌کنیم.

در انتها، در حلقه for خطوط 9 تا 11، هر عنصر A[j] را در مکان صحیح خود در آرایه خروجیB قرار می‌دهیم.

اگر هر n عنصر متمایز باشند، هنگامی که وارد خط 9 می‌شویم، برای هر A[j]، مقدار C[A[j]] تعداد عناصر کوچکتر یا مساوی A[j] است. از آنجا که ممکن است عناصر متمایز نباشند، هر بار که A[j] را در آرایه B قرار می دهیم، از C[A[j]] یکی کم میکنیم که سبب می‌شود عنصر ورودی بعدی، با مقداری برابر A[j]، اگر موجود است، به خانهٔ درست قبل از A[j] در آرایه خروجی برود.

مرتب‌سازی شمارشی به چه زمانی احتیاج دارد؟ حلقه for 1 و 2، به اندازه $\theta(k)$ زمان می‌گیرند. حلقه for خطوط 3 و 4 به اندازه $\theta(n)$ و حلقه for در خطوط 6و7، به اندازه $\theta(k)$ و حلقه for در خطوط 9 تا 11 به اندازه $\theta(n)$ زمان می گیرند. لذا روی هم رفته به اندازه $\theta(n+k)$ زمان صرف می‌شود. در عمل، هنگامی از مرتب‌سازی شمارشی استفاده می کنیم که k=O(n) باشد که در این شرایط هزینه زمانی از $\theta(n)$ خواهد شد.

مرتب‌سازی شمارشی، بر کران پائین $\Omega(n.\log n)$ غلبه می‌کند، چرا که این یک روش مرتب‌سازی مقایسه‌ای نیست. در واقع هیچ مقایسه‌ای بین عناصر ورودی صورت نمی‌گیرد. در عوض، مرتب‌سازی شمارشی از مقادیر واقعی عناصر برای آدرس‌دهی در آرایه استفاده می‌کند.

تمرین. ۸ عدد داریم که می‌دانیم همگی اعداد صحیحی بین ۱ تا ۶ میباشند، اعداد را با روش مراتب کنید

آرایه اعداد

آرایه تعداد تکرار

آرایه تجمعی تعداد تکرار

مراحل سرت

.....

ویژگی های مرتب سازی شمارشی :

غیر مقایسه ای : این الگوریتم غیر مقایسه ای است چرا که هیچ مقایسه ای بین مقدار عناصر صورت نمیگیرد.

برون جا : برای مرتب کردن n عنصر نیاز به آرایه ای کمکی به اندازه ی $O(k)$ داریم و برای نگه داشتن پاسخ نیز به آرایه ای به اندازه ی $O(n)$ نیاز است.

پایدار : در خط ۹ در الگوریتم اگر عناصر را از اول به آخر بررسی میکردیم الگوریتم درست کار میکرد ولی دیگر پایدار نمی ماند. پس چون از آخر به اول بررسی کردیم، الگوریتم پایدار است.

هزینه ی زمانی الگوریتم فوق همانطور که محاسبه شد $\theta(n+k)$ است.

یک ویژگی مهم مرتب‌سازی شمارشی، پایدار بودن آن است: اعداد با مقادیر یکسان، در آرایه خروجی با همان ترتیب موجود در آرایه ورودی ظاهر می‌شوند و این به معنای این قانون است که بین عناصر یکسان هر عنصری که در آرایه ورودی، زودتر ظاهر می‌شود، در آرایه خروجی نیز زودتر ظاهر خواهد شد. معمولاً ویژگی پایدار بودن، تنها زمانی اهمیت دارد که اطلاعات دیگری نیز در ارتباط با هر عنصر مورد نظر در مرتب سازی وجود دارد. پایداری مرتب‌سازی شمارشی، از یک لحاظ دیگر نیز دارای اهمیت است و آن اینکه روش مرتب‌سازی شمارشی اغلب به عنوان یک زیرروال در مرتب‌سازی مبنایی بکار می‌رود.

همانطور که در بخش بعد خواهیم دید، پایداری مرتب سازی شمارشی در صحت مرتب‌سازی مبنایی، بسیار حائز اهمیت است.

2.5. مرتب سازی مبنایی

مبنای مرتب سازی مبنایی، مرتب سازی شارشی است.

مرتب سازی مبنایی، الگوریتمی است که مورد استفاده ماشین های مرتب سازی کارتی بوده که هم اکنون تنها در موزه ها پیدا می شوند. در این روش، کارت ها در هشتاد ستون قرار می گیرند که در هر ستون یک سوراخ می تواند در 12 حالت ایجاد شود. دستگاه مرتب ساز می تواند به صورت مکانیکی برنامه ریزی شود تا یک ستون داده شده از هر کارت در یک دسته را بررسی کند و کارت ها را در یکی از 12 سطل بسته به اینکه در کجا سوراخ شده توزیع کند. یک اپراتور می تواند کارت ها را سطل به سطل جمع آوری کند، به صورتی که کارت هایی که در اولین جای ممکن سوراخ شده اند بر روی آن هایی قرار بگیرد که در محل دوم سوراخ شده اند، الی آخر.

برای ارقام دهدهی، تنها 10 قسمت در هر ستون استفاده می شود. (دو محل دیگر برای کد کردن کاراکتر های غیر عددی به کار می روند.) یک عدد d-رقمی می تواند بخش d ستونی ای را پر کند. از آنجایی که مرتب ساز کارت ها در هر زمان می تواند تنها یک ستون را بررسی کند، مسئله مرتب سازی n کارت در یک عدد d-رقمی نیازمند یک الگوریتم مرتب سازی است.

شکل 8.3 عملکرد مرتب سازی مبنایی بر روی هفت عدد سه رقمی . سمت چپ ترین ستون نشان دهنده ورودی است بقیه ستون ها نشان دهنده اعداد بعد از انجام یک حلقه هستند. سایه ها نشان دهنده ستون هایی هستند که تفاوت هر لیست با لیست قبل از مرتب کردن آنها ناشی می شود.

بنابراین، یک شخص ممکن است بخواهد تا اعداد را بر حسب با ارزش ترین رقمشان مرتب بکند، و سپس دسته ها را به ترتیب با هم ترکیب کند. متأسفانه، از آنجایی که کارت ها در 9 سطل از 10 سطل برای مرتب سازی هر سطل باید کنار گذاشته شوند، این رویه تعداد زیادی پشته های واسط از کارت ها را ایجاد می کند که باید جای هر کدام مشخص باشد.

مرتب سازی مبنایی مشکل مرتب سازی کارت ها را به طور غیر ذاتی با مرتب کردن کم ارزش ترین رقم در ابتدا حل می کند. سپس کارت ها به ترتیب، به یک دسته ترکیب می شوند. به صورتی که پشت سر کارت های سطل صفرم کارت های سطل یکم قرار بگیرند که پس از آن ها کارت های سطل دو و... قرار بگیرند. سپس تمام دسته دوباره بر اساس دومین رقم بی ارزش ترشان مرتب سازی می شوند و به روش مشابهی با هم ترکیب می شوند. این روند تا زمانی که کارت ها بر اساس همه d رقمشان مرتب شوند ادامه پیدا می کند. جالب توجه است که، در این زمان کارت ها بر اساس همه d رقمشان مرتب خواهند بود. بنابراین، تنها d حرکت روی دسته لازم خواهد بود تا مرتب سازی کامل شود. شکل 8.3 نشان می دهد که مرتب سازی مبنایی روی یک دسته هفت عدد سه رقمی صورت می گیرد.

این نکته مهم است که رقم های مرتب شده در این الگوریتم پایدار باشند. مرتب سازی ای که با یک مرتب ساز کارت صورت می گیرد پایدار است، ولی اپراتور باید درباره تغییر ندادن ترتیب کارت ها هنگامی که از یک سطل

بیرون می آیند احتیاط کند، حتی اگر همه کارت های درون یک سطل رقم مشابهی در یک ستون انتخابی داشته باشند.

در یک کامپیوتر معمولی، که یک ماشین با دسترسی تصادفی متوالی است، مرتب سازی مبنایی گاهی برای مرتب سازی رکورد هایی از اطلاعات که با فیلد های متعددی کلید گذاری شده اند به کار می رود. به عنوان مثال، ما ممکن است بخواهیم روزها را با سه کلید مرتب کنیم : سال، ماه، و روز. ما می توانیم یک الگوریتم مرتب سازی با یک تابع مقایسه اجرا کنیم که دو روز را بگیرد و سال ها را مقایسه کند اگر شباهتی بود ، ماه ها را مقایسه کند و اگر شباهتی دیگر پیدا شد، روزها را مقایسه کند. متناوباً ما می توانیم اطلاعات را سه بار با مرتب سازی پایدار، مرتب کنیم : اول بر اساس روز، دوم بر اساس ماه، و در آخر بر اساس سال.

کد مربوط به مرتب سازی مبنایی بسیار سر راست است. رویه زیر فرض می کند که هر عنصر از یک آرایه n-عنصره A، d رقمه است، که در آن رقم اول کوچکترین مرتبه است و رقم d ام بزرگ ترین مرتبه.

```
RADIX_SORT (A, d)

for i 1 to d

    do use a stable sort to sort array A on digit i
```

n عدد شامل d رقم که هر رقم در آن می تواند k مقدار متفاوت بگیرد داده شده است، RADIX_SORT این اعداد را به درستی در زمان $\theta(d(n+k))$ مرتب می کند.

اثبات

درستی مرتب سازی مبنایی از مقدمه استقرا برای ستونی که مرتب شده می آید. تحلیل زمان اجرا بستگی به مرتب سازی پایداری دارد که توسط الگوریتم مرتب سازی واسط استفاده می شود. هنگامی که هر رقم در بازه 0 تا $k-1$ است (بنابراین می تواند k حالت ممکن داشته باشد)، و در آن k عدد زیاد بزرگی نیست، مرتب سازی شمارشی انتخاب مسلم است. بنابراین، هر حرکت بر روی n عدد d- رقمه زمان $\theta(n+k)$ خواهد گرفت. و برای d حرکت زمان کل مرتب سازی مبنایی $\theta(d(n+k))$ خواهد بود.

زمانی که d ثابت است و ، مرتب سازی مبنایی در زمان خطی اجرا می شود. به طور کلی تر، ما مقداری انعطاف در شکستن هر کلید به رقم ها داریم.

n عدد b-بیتی و هر عدد صحیح مثبت داده شده اند، RADIX_SORT این اعداد را در $\theta((b/r)(n+2^r))$ مرتب می کند.

تمرین : به کمک مرتب سازی مبنایی نشان دهید که چگونه میتوان n عدد صحیح بین ۰ تا n^3-1 را با هزینه ی زمانی O(n) مرتب کرد؟

پاسخ : عدد صحیح را عددی سه رقمی در مبنای n در نظر بگیرید. در این صورت این عدد سه رقمی ارقامش از ۰ تا n-1 تغییر میکنند. حال میتوانید این عدد سه رقمی را به روش مرتب سازی مبنایی، مرتب کنید.
پس باید ۳ بار تابع مرتب سازی شمارشی را صدا بزنید که در آنصورت هزینه ی زمانی الگوریتم فوق خواهد بود O(3n) که در کل هزینه ی زمانی O(n) خواهد شد.

ویژگی های مرتب سازی مبنایی

1) غیر مقایسه ای: در این الگوریتم نیز درست مشابه مرتب سازی شمارشی، هیچ مقایسه ای میان عناصر صورت نمی گیرد.

2) برون جا: در این الگوریتم در هر مرحله پخش و جمع کردن عناصر، به حافظه ای از مرتبه تعداد عناصر نیاز است.

3) پایدار: تمامی مراحل پخش و جمع کردن عناصر از ابتدا به انتهای لیست و خوشه ها صورت میگیرد، بنابراین اگر دو عنصر یکسان در لیست موجود باشند، مکان انها تغیر نمی کند.

4) پیچیدگی زمانی در همه حالت ها $\theta(kn+kr)$ است. که k طول رشته ها و R مبنای عناصر و n تعداد عناصر است.

2.5. مرتب سازی سطلی

هنگامی که داده های ورودی از توزیع یکنواختی برخوردار باشند، *مرتب سازی سطلی* در زمان خطی اجرا می شود. مشابه مرتب سازی شمارشی، مرتب سازی سطلی نیز سریع است، زیرا در مورد ورودی چیزی را فرض می کند. مرتب سازی شمارشی تصور می کند که ورودی ها شامل اعداد صحیح در بازۀ کوچکی هستند، در حالی که مرتب سازی سطلی چنین تصور می کند که ورودی توسط یک روند تصادفی ایجاد شده که در آن عناصر به طور یکسان در بازۀ [0, 1) قرار گرفته اند.

ایدۀ مرتب سازی سطلی تقسیم بازه های [0, 1) به n زیربازه ی هم اندازه، یا n سطل است. پس از آن n ورودی در سطل ها قرار داده میشوند. از آنجایی که ورودی ها به صورت یکنواخت روی بازۀ [0, 1) توزیع شده اند، ما انتظار نداریم که تعداد زیادی شماره در هر سطل قرار بگیرد. برای ایجاد خروجی، ما به سادگی، شماره های هر سطل را مرتب می کنیم و سپس به ترتیب از سطلی به سطل دیگر می رویم و عناصر موجود در هر یک را لیست می کنیم.

کد ما برای مرتب سازی سطلی، فرض می کند که ورودی یک آرایه n-عنصره است و هر عنصر آرایه به صورت $0 <= A[i] < 1$ است. این کد به یک آرایۀ کمکی $B[0..n-1]$ از لیست های پیوندی (سطل ها) احتیاج دارد و فرض می کند که مکانیسمی برای نگهداری چنین لیستی موجود است.

```
BUCKET_SORT (A)
1 n ← length [A]
2 for i ← 1 to n
3       do  insert A[i] into list B[[n A[i]]]
4 for I ← 0 to n – 1
5       do sort list B[i] with insertion sort
6 concatenate the lists B[0], B[1], …,B[n – 1]
```

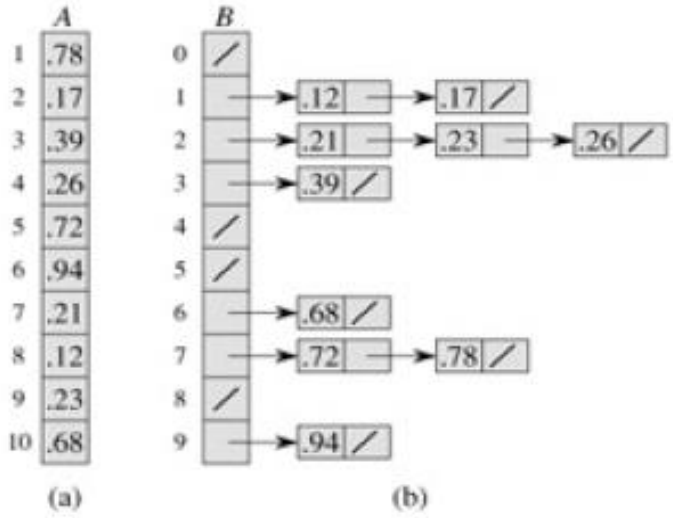
شکل 8.4 عملکرد مرتب سازی سطلی را بر روی آرایه ورودی از 10 عدد نشان می دهد.

برای اینکه ببینیم این الگوریتم چگونه کار می کند، دو عنصر A[i] و A[j] را در نظر بگیرید. فرض کنید بدون از دست دادن کلیت موضوع داریم A[i] =< A[j]. از آنجایی که (کف) [n A[i]] <= [n A[j]] عنصر A[i] یا در سطل مشابهی با A[j] قرار گرفته است و یا در سطلی با اندیس پایین تر. اگر A[i] و A[j] در سطل مشابهی باشند، حلقهٔ **for**در خط های 4و 5 آن ها را در ترتیب صحیح قرار می دهد. اگر این دو عنصر در سطل های متفاوتی باشند، خط 6 ام آن ها را در ترتیب صحیح قرار می دهد. بنابراین، مرتب سازی سطلی درست کار می کند.

برای تحلیل زمان اجرای این الگوریتم، توجه کنید که همهٔ خطوط به جز خط 5 در بدترین حالت زمان اجرای O(n) خواهند داشت. این موضوع در توازن با زمان کل n فراخوانی در مرتب سازی درجی در خط 5 است.

برای تحلیل هزینهٔ همهٔ فراخوانی ها در مرتب سازی درجی، تصور کنید ni متغیر تصادفی باشد که بیانگر تعداد عناصر قرار گرفته در سطل B[i] باشد. از آنجایی که مرتب سازی درجی در زمان درجه دویی اجرا می شود (بخش 2.2 را نگاه کنید)، زمان اجرای مرتب سازی سطلی عبارتست از

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2) .$$



عملکرد مرتب سازی سطلی

شکل بالا عملکرد مرتب سازی سطلی را نشان می دهد. (a) آرایهٔ ورودی A[1..10]. (b) آرایهٔ B[0..9] از لیست های مرتب شده (سطل ها) بعد از خط 5 ام الگوریتم. سطل i ام مقادیر موجود در نیم بازهٔ $[i/10, (i + 1)/10)$ را نگه می دارد. خروجی مرتب شده شامل به هم چسباندن به ترتیب لیست های B[0]، B[1]، ...، B[9] است

با گرفتن امید ریاضی از دو طرف و با توجه به خاصیت خطی بودن امید ریاضی و با توجه به تساوی C.21 داریم

$$\begin{aligned} \mathrm{E}[T(n)] &= \mathrm{E}\left[\Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)\right] \\ &= \Theta(n) + \sum_{i=0}^{n-1} \mathrm{E}[O(n_i^2)] \\ &= \Theta(n) + \sum_{i=0}^{n-1} O\left(\mathrm{E}[n_i^2]\right) \end{aligned}$$

ما ادعا می کنیم که برای $i = 0, 1, \ldots, n - 1$ داریم

$$\mathrm{E}[n_i^2] = 2 - 1/n$$

تعجبی در این نیست که هر سطل i ام شامل مقدار E[ni 2] است، زیرا هر مقدار در آرایهٔ ورودی A به طور مشابه وارد هر سطل می شود. برای اثبات تساوی (8.2)، ما متغیرهای تصادفی نشانگر را تعریف می کنیم

Xij =I { A[i] در سطل ام قرار بگیرد }

برای $i = 0, 1, \ldots, n - 1$ و $j = 1, 2, \ldots, n$، بنابراین،

$$n_i = \sum_{j=1}^{n} X_{ij} .$$

برای محاسبهٔ E[ni 2]، ما عبارت توان دو را باز می کنیم و عبارات را دوباره دسته بندی می کنیم :

$$
\begin{aligned}
\mathrm{E}[n_i^2] &= \mathrm{E}\left[\left(\sum_{j=1}^{n} X_{ij}\right)^2\right] \\
&= \mathrm{E}\left[\sum_{j=1}^{n}\sum_{k=1}^{n} X_{ij}X_{ik}\right] \\
&= \mathrm{E}\left[\sum_{j=1}^{n} X_{ij}^2 + \sum_{1\le j\le n}\ \sum_{\substack{1\le k\le n \\ k\ne j}} X_{ij}X_{ik}\right] \\
&= \sum_{j=1}^{n}\mathrm{E}\left[X_{ij}^2\right] + \sum_{1\le j\le n}\ \sum_{\substack{1\le k\le n \\ k\ne j}} \mathrm{E}[X_{ij}X_{ik}]\ ,
\end{aligned}
$$

به طوری که خط آخر از خاصیت خطی بودن امید ریاضی به دست آمد. ما دو جمع را جدا حساب می کنیم.
نشانگر متغیر تصادفی Xij یک است با احتمال n/1 و 0 برای حالت های دیگر و بنابراین

$$
\begin{aligned}
\mathrm{E}\left[X_{ij}^2\right] &= 1\cdot\frac{1}{n} + 0\cdot\left(1-\frac{1}{n}\right) \\
&= \frac{1}{n}\ .
\end{aligned}
$$

هنگامی که j <> k متغیر های Xij و Xik مستقل از هم هستند و بنابراین

$$
\begin{aligned}
\mathrm{E}[X_{ij}X_{ik}] &= \mathrm{E}[X_{ij}]\mathrm{E}[X_{ik}] \\
&= \frac{1}{n}\cdot\frac{1}{n} \\
&= \frac{1}{n^2}\ .
\end{aligned}
$$

با جایگزینی این مقادیر امید ریاضی در تساوی(8.3) به دست می آوریم

$$
\begin{aligned}
\mathrm{E}[n_i^2] &= \sum_{j=1}^{n} \frac{1}{n} + \sum_{1 \le j \le n} \sum_{\substack{1 \le k \le n \\ k \ne j}} \frac{1}{n^2} \\
&= n \cdot \frac{1}{n} + n(n-1) \cdot \frac{1}{n^2} \\
&= 1 + \frac{n-1}{n} \\
&= 2 - \frac{1}{n},
\end{aligned}
$$

که تساوی (8.2) را ثابت می کند.

با استفادهٔ این امید های ریاضی در تساوی (8.1)، ما نتیجه می گیریم که زمان منتظرهٔ مرتب سازی سطلی Q (n) + n. O (2 – 1/n)) است. بنابراین، کل الگوریتم مرتب سازی سطلی در زمان خطی منتظره اجرا می شود.

حتی اگر ورودی از یک توزیع یکنواخت به دست نیامده باشد، مرتب سازی سطلی ممکن است در زمان خطی اجرا شود. تا زمانی که ورودی این خصوصیت که مجموع مربعات سایز سطل ها نسبت به تعداد کل عناصر خطی است، تساوی (8.1) به ما می گوید که مرتب سازی سطلی در زمان خطی اجرا می شود.

تمرین : توضیح دهید که چرا هزینه ی زمانی مرتب سازی سطلی در بدترین حالت تتای n^2 است؟ سپس پیشنهاد دهید که با چه تغییر ساده در الگوریتم میتوان هزینه ی زمانی حالت متوسط را همان مقدار قبلی نگه داشت ولی هزینه ی زمانی بدترین حالت بجای تتای n^2 تتای nlogn شود؟

پاسخ : بد ترین هزینه ی زمانی در مرتب سازی سطلی هنگامی اتفاق میفتد که بر خلاف فرض ما، داده ها بصورت منظم بین سطل ها پخش نشده باشند و برای مثال کل داده ها فقط در یک سطل بیفتند که در آن صورت در مرحله ای که مرتب سازی درجی اتفاق می افتد داده ها باید با O(n^2) مرتب شوند.
تغییر ساده ای در الگوریتم که میتواند هزینه ی زمانی حالت متوسط را نگه دارد و در عین حال هزینه ی بدترین حالت را کاهش دهد، این است که بجای مرتب سازی درجی که در بدترین حالت O(n^2) است از مرتب سازی ادغامی برای مرتب کردن سطل ها کمک بگیریم که در بدترین حالت O(nlogn) است.