دانشکده‌ی علوم ریاضی

ساختمان داده — ۲۶ شهریور ۹۲

جلسه‌ی ۲: مرتب‌سازی درجی

مدرّس: دکتر شهرام خزائی — نگارنده: حسن بندبنی و مصطفی کریمی

۱ مسأله مرتب‌سازی

همان‌طور که در جلسه قبل مطرح شد، مسأله‌ی مرتب‌سازی آرایه‌ای از ورودی‌ها را دریافت و مرتب شده‌ی آن‌ها را به عنوان خروجی باز می‌گرداند. در ادامه منظور از آرایه‌ی مرتب، آرایه‌ای است که صعودی باشد.

تعریف ۱ (مسأله مرتب‌سازی) *مسأله‌ی مرتب‌سازی با ورودی‌ها و خروجی‌های به صورت زیر تعریف می‌شود:*

- **ورودی:** *دنباله‌ای از اعداد مانند* $\langle a_1, a_2, \ldots, a_n \rangle$
- **خروجی:** *یک جایگشت* $\langle a'_1, a'_2, \ldots, a'_n \rangle$ *از دنباله‌ی ورودی به طوری که* $a'_i \leq a'_j$ *به ازای هر* $1 \leq i < j \leq n$.

۲ مرتب‌سازی درجی

به‌عنوان اولین روش برای حل مسأله مرتب‌سازی، مرتب‌سازی درجی را معرفی می‌کنیم. مرتب‌سازی درجی همانند روشی است که برای مرتب کردن دست خود در بازی ورق استفاده می‌کنیم. در حالی که کارت‌ها به پشت روی میز هستند در ابتدا کارت اول را برداشته و در دستمان می‌گیریم، سپس کارت دوم را برداشته و با کارت اول مقایسه می‌کنیم و این کارت جدید را در جای درستش قرار می‌دهیم. به همین ترتیب هر کارتی را که بر می‌داریم با کارت‌های موجود در دستمان مقایسه می‌نماییم و سپس آن را در جای درست خود قرار می‌دهیم. با تکرار این عمل پس از چند مرحله کارت‌ها به صورت مرتب در دستمان قرار می‌گیرند.

ایده‌ی اصلی مرتب‌سازی درجی این است که در هر مرحله با اضافه کردن یک عضو جدید به زیر آرایه مرتب شده آن را به آرایه‌ای بزرگتر تبدیل کنیم به صورتی که مرتب باقی بماند. برای این کار در هر مرحله، یک عضو از آرایه را در نظر می‌گیریم و با اعضای زیر آرایه مرتب شده قبلی مقایسه کرده و در جایی قرار می‌دهیم که خاصیت مرتب بودن زیرآرایه را حفظ کند. این کار را تا زمانی که تمامی اعضای آرایه اصلی را به این زیر آرایه اضافه کنیم ادامه می‌دهیم. شبه کد InsertionSort روند این کار را نشان می‌دهد.

Algorithm 1 InsertionSort

```
function InsertionSort(array A[1 : n])
    for j = 2 to n do
        key ← A[j]
        #Insert A[j] into the sorted sequence A[1 ··· j − 1]
        i ← j − 1
        while i > 0 and A[i] > key do
            A[i + 1] ← A[i]
            i ← i − 1
        A[i + 1] ← key
```

مثال ۱ *می‌خواهیم آرایه زیر را به روش درجی مرتب نماییم.*

5	2	4	6	1	3

*در اولین اجرای حلقه **for** مقدار $j = ۲$ می‌باشد. همچنین قبل از اجرای حلقه **while** داریم $key = ۲$ و $i = ۱$.*

	j↓				
5	2	4	6	1	3

*چون شرط حلقه **while** برقرار است با اجرای بدنه حلقه $A[۲] = ۵$ و $i = ۰$ می‌شود. سپس چون شرط حلقه، دیگر برقرار نیست خط آخر اجرا می‌شود که با اجرای آن $A[۱] = ۲$ می‌شود. بدین ترتیب آرایه پس از اولین تکرار به صورت زیر در می‌آید:*

		j↓			
2	5	4	6	1	3

*در دومین تکرار حلقه **for**، $j = ۳$ و در نتیجه مقادیر key و i به ترتیب ۴ و ۲ می‌شود. همچنین حلقه **while** نیز به ازای مقدار $i = ۲$ اجرا شده و جای عددهای ۴ و ۵ در آرایه را عوض می‌کند ولی به ازای $i = ۱$ به دلیل برقرار نبودن شرط $A[i] > key$ از حلقه خارج می‌شود. در پایان این تکرار، با اجرای خط آخر الگوریتم مقدار ۴ در خانه‌ی $A[۲]$ قرار داده خواهد شد. روند اجرای الگوریتم را تا مرتب‌سازی کامل آرایه در زیر مشاهده می‌کنید.*

			j↓		
2	4	5	6	1	3

				j↓	
2	4	5	6	1	3

					j↓
1	2	4	5	6	3

در پایان خروجی الگوریتم دنباله‌ی زیر خواهد بود.

1	*2*	*3*	*4*	*5*	*6*

۱.۲ ناوردایی حلقه

قبل از تعریف ناوردایی حلقه[۱] بهتر است مروری بر استقرا داشته باشیم. در جلسات بعدی خواهید دید که استقرا و دید استقرایی نقش مهمی در طراحی الگوریتم‌ها و اثبات درستی آن‌ها دارد.

اثبات استقرایی. همان‌طور که به یاد دارید در اثبات به روش استقرایی بر روی پارامتری نظیر n، با گذاشتن یک پایه مشخص و سپس اثبات گام استقرا گزاره را برای تمامی مقادیر n که در شرط گام صدق کنند اثبات می‌کنیم. فرض کنید $P(n)$ حکمی در مورد اعداد طبیعی باشد. برای اثبات صحت حکم برای تمام مقادیر طبیعی n، یک اثبات استقرایی صحت گزاره‌های زیر را تایید می‌نماید:

- **پایه:** $P(۱)$ درست است.
- **گام استقرا:** به ازای عدد طبیعی مانند k، اگر $P(k)$ درست باشد، $P(k+۱)$ نیز درست است.

با تایید مراحل پایه و گذار استقرا، صحت حکم به استقرا ثابت می‌شود.

اثبات ناوردایی. برای اثبات الگوریتم‌ها علاوه بر اثبات از طریق استقرا، با استفاده از ایده‌ای مشابه، می‌توانیم از خاصیتی که در ابتدا وجود دارد و در حین اجرای الگوریتم نیز ثابت باقی می‌ماند استفاده کنیم. به چنین خاصیتی ناوردا[۲] گفته می‌شود. همچنین با توجه به نقش حلقه‌ها، نیاز داریم ثابت کنیم که حلقه‌ی مورد نظر در زمان تکرار، چنین خاصیتی را که آن را ناوردای حلقه می‌نامند حفظ می‌کند.
برای بررسی صحت الگوریتم‌ها به روش ناوردایی حلقه سه مرحله باید بررسی شود:

- **آغاز**[۳]: در ابتدا همانند بررسی پایه برای اثبات استقرایی، نشان می‌دهیم که ناوردای مورد نظر قبل از اجرای حلقه برقرار است.
- **نگهداری**[۴]: این مرحله همانند گام در استقرا می‌باشد. در این مرحله ثابت می‌کنیم که با تکرار حلقه، خاصیت ناوردایی آن حفظ می‌شود.
- **پایان**[۵]: در پایان نشان می‌دهیم که پس از خارج شدن الگوریتم از حلقه، ناوردایی دارای ویژگی مطلوبی است که می‌توان از آن صحت الگوریتم را استنتاج کرد.

[۱] loop invariant
[۲] invariant
[۳] Initialization
[۴] Maintenance
[۵] Termination

۲.۲ صحت مرتب‌سازی درجی

برای اثبات درستی الگوریتم مرتب‌سازی درجی، ناوردای حلقه را به صورت زیر در نظر می‌گیریم:

ناوردایی حلقه: درست پس از مقداردهی متغیر حلقه **for** با مقدار j، $j - ۱$ عضو اول آرایه مرتب شده هستند؛ یعنی زیر آرایه $A[۱...j - ۱]$ مرتب است.

حال نشان می‌دهیم که ناوردایی فوق در اولین تکرار برقرار است، بعد از هر تکرار نیز حفظ می‌گردد، و پس از آخرین تکرار می‌توان صحت الگوریتم را از آن استنتاج نمود.

- **آغاز:** اثبات را با نشان دادن این که ناوردایی حلقه درست بعد از اولین مقداردهی j در حلقه برقرار است آغاز می‌کنیم. در ابتدا $j = ۲$ است و در نتیجه $j - ۱ = ۱$ می‌باشد. پس زیرآرایه تنها شامل یک عضو می‌باشد و هر آرایه یک عضوی، مرتب شده است پس شرط ناوردایی حلقه برقرار است.

- **نگهداری:** در این مرحله باید نشان دهیم که در هر تکرار، ناوردایی حلقه برقرار باقی می‌ماند. معادلاً باید ثابت کنیم که اگر در تکرار kام درست پس از اجرای حلقه **for** که مقدار $j = k + ۱$ است، زیر آرایه $A[۱...j - ۱]$ مرتب باشد؛ آنگاه پس از اجرای حلقه زیر آرایه $A[۱...j]$ مرتب خواهد بود. الگوریتم در هر تکرار حلقه عنصر jام را در key قرار می‌دهد. سپس در حلقه‌ی **while**، با مقایسه شدن مقدار key با اعضای زیر آرایه، عضو جدید در جای درست خود قرار می‌گیرد. در نتیجه زیر آرایه $A[۱...j]$ نیز به صورت مرتب در می آید. دقت کنید اثبات دقیق‌تر نیز می‌توانست با گرفتن ناوردایی مناسب برای حلقه داخلی **while** انجام شود.

- **پایان:** در پایان بررسی می‌کنیم که وقتی حلقه به پایان می‌رسد چه روی می‌دهد. شرط به پایان رسیدن حلقه این است که مقدار j از طول آرایه بیشتر شود. چون در هر تکرار یک واحد به j افزوده می‌شود برای به پایان رسیدن حلقه باید $j = n + ۱$ باشد. در تکرار $(n + ۱)$ام زیرآرایه $A[۱ \cdots n]$ شامل همه‌ی اعضای آرایه اصلی اما به صورت مرتب شده است. به وضوح ناوردایی حلقه صحت الگوریتم را نتیجه می‌دهد.

۳.۲ زمان اجرا

برای مقایسه زمان اجرای الگوریتم‌ها، از تابعی بر حسب اندازه ورودی که آن را پیچیدگی زمانی الگوریتم می‌نامند استفاده می‌کنیم. برای اندازه ورودی یکسان هرچه مقدار پیچیدگی زمانی الگوریتم کمتر باشد، الگوریتم با سرعت بیشتری اجرا شده و بهتر می‌باشد.
در ادامه حالت‌های مختلف بررسی زمان اجرای یک الگوریتم را معرفی می‌کنیم.

۱.۳.۲ زمان اجرای دقیق

در حالت نخست می‌خواهیم زمان اجرای دقیق الگوریتم مرتب‌سازی درجی را به دست آوریم. برای محاسبه آن از جدول زیر استفاده می‌نماییم:

for $j = 2$ **to** n	c_1	n
$key \leftarrow A[j]$	c_2	$n-1$
#Comment	c_3	$n-1$
$i \leftarrow j-1$	c_4	$n-1$
while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
$A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n}(t_j-1)$
$i \leftarrow i-1$	c_7	$\sum_{j=2}^{n} t_j$
$A[i+1] \leftarrow key$	c_8	$n-1$

در جدول فوق، ستون نخست دستورات هر خط از الگوریتم، ستون دوم زمان اجرای دقیق هر دستور و ستون سوم تعداد دفعات تکرار آن می‌باشد.
برای محاسبه زمان اجرای دقیق الگوریتم، پارامتر جدیدی به‌نام t_j را معرفی می‌کنیم که برابر تعداد دفعات تکرار دستور **while** در تکراری از حلقه‌ی **for** با مقدار j است.
در نهایت برای محاسبه زمان اجرای دقیق الگوریتم، عناصر ستون دوم جدول را در ستون سوم ضرب می‌کنیم که زمان اجرای هر دستور در کل الگوریتم به دست می‌آید. سپس با جمع کردن تمام این مقادیر زمان اجرای دقیق الگوریتم به‌دست می‌آید که به صورت زیر است:

$$\begin{aligned} T(n) &= c_۱ n \\ &\quad +(n-۱)(c_۲+c_۴-c_۶-c_۷+c_۸) \\ &\quad +(c_۵+c_۶+c_۷)\sum_{j=۲}^{n} t_j \end{aligned}$$

۲.۳.۲ بهترین حالت زمان اجرا

بهترین حالت زمانی رخ می‌دهد که آرایه ورودی مرتب باشد. در این حالت خواهیم داشت: $t_j = ۱$
پس زمان اجرا برابر خواهد بود با:

$$\begin{aligned} T_۱(n) &= (c_۱+c_۲+c_۴+c_۵+c_۸)n \\ &\quad -(c_۲+c_۴+c_۵+c_۸) \end{aligned}$$

حال اگر به جای $c_۱+c_۲+c_۴+c_۵+c_۸$ متغیر a و به جای $c_۲+c_۴+c_۵+c_۸$ متغیر b را قرار دهیم، خواهیم داشت:

$$T_۱(n) = an + b$$

برای مقادیر بزرگ n مقدار b ناچیز و قابل صرف‌نظر کردن خواهد بود. پس می‌توان زمان اجرا را با تقریب خوبی به صورت زیر نوشت:

$$\begin{aligned} T_۱(n) &\sim an \\ &= \Theta(n) \end{aligned}$$

یعنی زمان اجرا از مرتبه n است.

۳.۳.۲ بدترین حالت زمان اجرا

بدترین حالت زمانی رخ می‌دهد که آرایه ورودی نزولی باشد. در این حالت در حلقه‌ی **for** به ازای تمامی مقادیر j، زمانی که $t_j = j$ است، باید $A[j]$ را با تمام اعضای زیر آرایه مرتب شده $A[۱ \cdots j-۱]$ مقایسه کنیم. زمان اجرای

بدترین حالت برابر خواهد بود با:

$$\begin{aligned} T_۲(n) &= c_۱n + c_۲(n-۱) + c_۴(n-۱) + c_۵(\frac{n(n+۱)}{۲} - ۱) \\ &\quad + c_۶(\frac{n(n-۱)}{۲}) + c_۷(\frac{n(n-۱)}{۲}) + c_۸(n-۱) \\ &= (\frac{c_۵}{۲} + \frac{c_۶}{۲} + \frac{c_۷}{۲})n^۲ \\ &\quad + (c_۱ + c_۲ + c_۴ + \frac{c_۵}{۲} - \frac{c_۶}{۲} - \frac{c_۷}{۲} + c_۸)n \\ &\quad - (c_۲ + c_۴ + c_۵ + c_۸) \end{aligned}$$

بار دیگر عبارت‌های طولانی و ثابت را با متغیرهای جدید جایگزین می‌کنیم و خواهیم داشت:

$$T_۲(n) = an^۲ + bn + c$$

وقتی n بسیار بزرگ می‌شود مقدار $bn + c$ در مقابل $an^۲$ قابل صرف‌نظر کردن می‌باشند، پس می‌توان نوشت:

$$T_۲(n) = an^۲ \Rightarrow \Theta(n^۲)$$

یعنی زمان اجرا از مرتبه $n^۲$ است.

۴.۳.۲ میانگین زمان اجرا

حالت متوسط حالتی است که در آن به طور متوسط نیمی از اعضای $A[۱ \cdots j-۱]$ کمتر از $A[j]$ و نیمی دیگر بیش‌تر از $A[j]$ هستند. بنابراین ما به طور متوسط نیمی از زیر آرایه $A[۱ \cdots j-۱]$ را بررسی می‌نماییم، در نتیجه خواهیم داشت:

$$\mathrm{E}[t_j] = \frac{(j+۱)}{۲}$$

به دلیل اینکه الگوریتم بر روی ورودی‌هایی مختلف اجرا می‌شود، در تحلیل الگوریتم‌ها اغلب بدترین حالت را در نظر می‌گیریم؛ یعنی شرایطی که الگوریتم بیشترین زمان را برای اجرا نیاز دارد. با این کار حد بالایی از زمان مورد نیاز اجرای الگوریتم بدست می‌آید. همچنین در بیشتر موارد امید ریاضی حالت متوسط دارای پیچیدگی معادل بدترین حالت می‌باشد. برای مثال در الگوریتم مرتب‌سازی درجی حالت متوسط همانند بدترین حالت تابعی از درجه دوم می‌باشد.

۴.۲ روش بازگشتی

برای بررسی زمان اجرای یک الگوریتم می‌توانیم به صورت بازگشتی نیز عمل کنیم. فرض کنید تابع پیچیدگی بدترین زمان اجرای الگوریتم مرتب‌سازی درجی را به ازای ورودی از اندازه n با $T(n)$ نشان دهیم. داریم:

$$\begin{cases} T(n) = T(n-۱) + an + b \\ T(۱) = c \end{cases}$$

در ادامه درس با نماد Θ آشنا خواهیم شد و چنین رابطه بازگشتی را به صورت زیر ساده خواهیم کرد:

$$\begin{cases} T(n) = T(n-۱) + \Theta(n) \\ T(۱) = \Theta(۱) \end{cases}$$

با استفاده از ابزارهایی که برای حل چنین معادلات بازگشتی یاد خواهیم گرفت، خواهیم دید که جواب معادله فوق به صورت زیر است:

$$T(n) = \Theta(n^۲)$$