دانشکده‌ی علوم ریاضی

داده‌ساختارها و الگوریتم‌ها — ۲۸ مهر ۹۲

جلسه‌ی ۱۰: الگوریتم مرتب‌سازی سریع

مدرّس: دکتر شهرام خزائی — *نگارنده: محمد امین ادریسی و سینا منصور لکورج*

۱ شرح الگوریتم

الگوریتم مرتب‌سازی سریع [۱] فرآیند تقسیم‌وحل را به کار می‌گیرد. سه مرحله‌ی مربوطه در زیر شرح داده شده‌اند:

- **تقسیم:** در این مرحله، آرایه‌ی $A[p..r]$، به سه زیرآرایه، به صورت $A[q+۱..r]$، $A[q]$ و $A[p..q-۱]$ تقسیم می‌شود، به گونه‌ای که زیرآرایه‌ی اول شامل اعضایی از آرایه‌ی اصلی است که از $A[q]$ بزرگ‌ترند و زیرآرایه‌ی سوم شامل اعضای کوچکتر یا مساوی با $A[q]$ است.
- **حل:** در مرحله‌ی حل، با فراخوانی بازگشتی تابع مرتب‌سازی سریع، دو زیرآرایه‌ی تولیدشده در مرحله تقسیم، یعنی $A[q+۱..r]$ و $A[p..q-۱]$ را مرتب می‌کنیم.
- **ترکیب:** چون که با فراخوانی بازگشتی تابع مرتب‌سازی سریع، زیرآرایه‌ها در نهایت به صورت مجموعه‌های تک‌عضوی مرتب‌شده‌ای نسبت به هم، تجزیه می‌شوند، پس نیازی به ترکیب‌کردن نتایج نداریم.

شبه‌کد مربوط به تابع مرتب‌سازی سریع در زیر آورده شده‌است:

a QUICKSORT

```
function QUICKSORT(A, p, r)
    if p < r then
        q ← PARTITION(A, p, r)
        QUICKSORT(A, p, q − 1)
        QUICKSORT(A, q + 1, r)
```

b PARTITION

```
function PARTITION(A, p, r)
    x = A[r]
    i ← p − 1
    for j = p to r − 1 do
        if A[j] ≤ x then
            i ← i + 1
            A[i] ↔ A[j]
    A[i + 1] ↔ A[r]
return i + 1
```

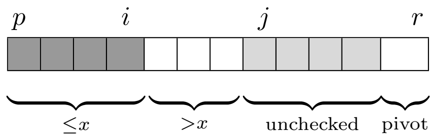
کار تقسیم‌بندی آرایه را در هر فراخوانی، تابع تقسیم‌بندی [۲] انجام می‌دهد که در بالا شبه‌کد آن نمایش داده شده است. خروجی تابع، اندیسی از آرایه است که کلیدهای مربوط به اندیس‌های بعد از آن بزرگ‌تر، و کلیدهای مربوط به

[۱] Quick sort
[۲] Partition

اندیس‌های قبل از آن، کوچکتر از کلید این اندیس هستند. این تابع برای مقایسه، همواره عنصر آخر آرایه‌ی ورودی را به عنوان محور[۳] انتخاب می‌کند. در هنگام اجرای این تابع، آرایه‌ی ورودی به چهار ناحیه تقسیم می‌شود که طبق شبه کد اگر j و i را در هر تکرار در نظر بگیریم، به صورت زیر هستند (در اینجا x همان $A[r]$ است):

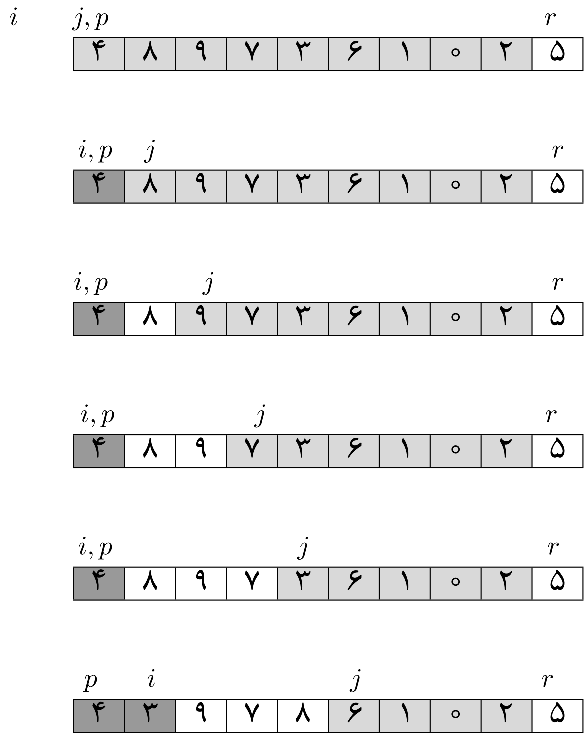
- برای اندیس‌هایی که $p \le k \le i$، داریم، $A[k] \le x$
- برای اندیس‌هایی که $i + ۱ \le k \le j - ۱$، داریم، $A[k] > x$
- اندیس‌هایی که از $j - ۱$ بزرگتر و از r کوچکترند نیز بررسی نشده‌اند
- ناحیه چهارم را نیز می‌توان به عنصر محور که همان $A[r] = x$ است، اختصاص داد

در شکل زیر این چهار ناحیه نمایش داده‌شده‌اند:

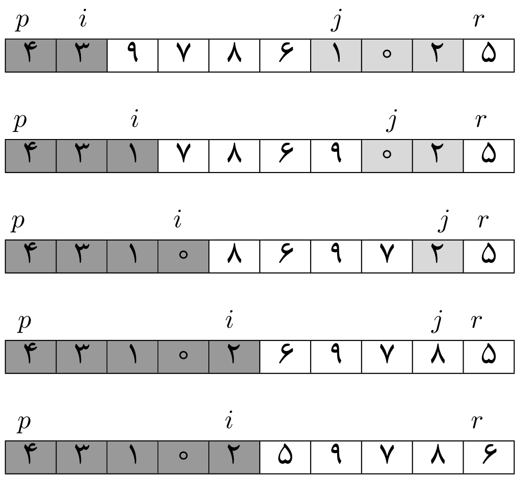


برای درک بهتر نحوه‌ی عملکرد این تابع آن را بر روی یک آرایه‌ی دلخواه اجرا می‌کنیم:

مثال ۱ *اجرای تابع تقسیم روی آرایه‌ی* $A[۴, ۸, ۹, ۷, ۳, ۶, ۱, ۰, ۲, ۵]$:



[۳] pivot



۲ صحت الگوریتم

برای نشان‌دادن صحت الگوریتم مرتب‌سازی سریع، باید نشان داد که تابع تقسیم به درستی کار می‌کند. ابتدا ناوردای حلقه را تعریف می‌کنیم:

ناوردا: در ابتدای اجرای حلقه‌ی **for** کلید مربوط به اندیس‌های بین p و i از $A[r]$ کوچکتر و کلید مربوط به اندیس‌های بین $i+۱$ و $j-۱$ نیز از $A[r]$ بزرگترند.

مراحل اثبات صحت الگوریتم به شرح زیر است:

- **آغاز:** قبل از اولین اجرای حلقه، چون $i = p - ۱$ است و $j = p$، پس زیرآرایه‌های $A[p..i]$ و $A[i+۱..j-۱]$ تهی هستند در نتیجه شرایط ناوردایی برقرار است.

- **نگهداری:** در این مرحله دو حالت را در نظر می‌گیریم. اگر $A[j] > x$ باشد که در این صورت، تنها کاری که انجام می‌شود این است که j یک واحد افزایش می‌یابد و باز ناوردایی حلقه برقرار است، چون در ابتدای شروع دوباره‌ی حلقه اندیس‌های $i+۱$ تا $j-۱$ همگی از x بزرگترند. حال اگر $A[j] \leq x$ باشد در این صورت i یک واحد افزایش یافته و جای $A[j]$ و $A[i]$ عوض می‌شود. در این حالت، همچنان اندیس‌های بین p و i از x کوچکترند، همچنین $A[i]$ (i جدید در اینجا منظور است) که از x بزرگتر است، در مکان j قرار می‌گیرد. با افزایش j در آغاز اجرای بعدی حلقه می‌توان گفت اعضای با اندیس‌های $i+۱$ تا $j-۱$ از x بزرگترند. پس ناوردایی برقرار است.

- **پایان:** در آخرین اجرای حلقه‌ی **for**، $j = r$ است پس از آنجایی که ناوردایی برقرار است، می‌توان گفت که زیرآرایه‌ی $A[p..i]$، شامل اعضای کوچکتر یا مساوی x است و زیرآرایه‌ی $A[i+۱..r-۱]$، شامل اعضای بزرگتر از x است و عضو با اندیس r نیز محور است؛ در واقع آرایه‌ی اصلی به سه زیرآرایه تقسیم شده‌است و نشان می‌دهد که تابع به درستی کار می‌کند.

۳ پیچیدگی الگوریتم

می‌دانیم که تابع تقسیم به دلیل داشتن یک حلقه‌ی **for** دارای مرتبه‌ی زمانی $\Theta(n)$ است؛ زمان اجرا، در حقیقت به افرازهای حاصل از تابع تقسیم بستگی دارد. حال زمان اجرای الگوریتم را در بهترین و بدترین حالت محاسبه می‌کنیم:

۱.۳ بدترین حالت

بدترین حالت را می‌توان حالتی در نظر گرفت که در آن به ازای هر بار فراخوانی یکی از افرازهای حاصل از تقسیم، تهی باشد. در این صورت می‌توان زمان اجرا را به صورت بازگشتی زیر نوشت:

$$T(n) = T(۰) + T(n - ۱) + \Theta(n)$$

که در اینجا $\Theta(n)$ هزینه مرحله‌ی ترکیب است. با حل این رابطه‌ی بازگشتی درمی‌یابیم که $T(n) = \Theta(n^۲)$

۲.۳ بهترین حالت

بهترین حالت را می‌توان حالتی در نظر گرفت که برخلاف مورد قبل، در هر بار فراخوانی افرازها دارای طول برابر باشند. در این حالت داریم:

$$T(n) = ۲T(\frac{n-۱}{۲}) + \Theta(n)$$

می‌دانیم که وجود عدد ثابت $\frac{۱}{۲}$ در اینجا تأثیری در جواب این رابطه‌ی بازگشتی ندارد؛ در واقع این رابطه تفاوتی با رابطه‌ی $T(n) = ۲T(n/۲) + \Theta(n)$ ندارد که از قضیه‌ی اصلی می‌دانیم که دارای جواب $\Theta(n \log n)$ است.
در ادامه زمان اجرا را به طور میانگین محاسبه می‌کنیم و نشان داده می‌شود که این زمان اجرا به بهترین حالت نزدیک‌تر است تا بدترین حالت. برای داشتن شهودی نسبت به این مطلب، یعنی نزدیک بودن حالت میانگین به بهترین حالت، می‌توان فرض کرد که در هر بار اجرای تابع تقسیم، آرایه، به نسبت ۱ به ۹۹ تقسیم می‌شود؛ یعنی داریم:

$$T(n) = T(\frac{n}{۱۰۰}) + T(\frac{۹۹n}{۱۰۰}) + \Theta(n)$$

در اینجا می‌توان به روشی مشابه مثالی از جلسه که نسبت‌ها $\frac{۲}{۳}$ و $\frac{۱}{۳}$ بود، این رابطه را حل کرد که به جواب $\Theta(n \log n)$ می‌رسیم.

۴ مرتب‌سازی سریع تصادفی

ما تمایل داریم که این زمان اجرا وابستگی کمی به ورودی داشته باشد. برای این کار دو راه‌حل وجود دارد؛ یکی اینکه قبل از اجرای الگوریتم، ورودی را به صورت تصادفی مرتب کنیم، و دیگری اینکه در تابع تقسیم، محور را به طور تصادفی انتخاب کنیم که این کار منجر به ایجاد الگوریتم مرتب‌سازی سریع تصادفی[۴] می‌شود که در ادامه شبه کد آن آمده‌است:

[۴] Randomized Quick Sort

c RANDOMIZED-QUICKSORT

$$
\begin{array}{l}
\_)A, p, r(\\
\quad p < r \\
\qquad q \leftarrow \_(A, p, r) \\
\qquad \_(A, p, q-1) \\
\qquad \_(A, q+1, r)
\end{array}
$$

d RANDOMIZED-PARTITION

$$
\begin{array}{l}
\_)A, p, r(\\
\quad i \leftarrow (p, r) \\
\quad A[i] \leftrightarrow A[r] \\
\quad (A, p, r)
\end{array}
$$

۱.۴ محاسبه‌ی متوسط زمان اجرا

حال متوسط زمان اجرا را به دست می‌آوریم. در هر بار اجرای کل الگوریتم می‌توان گفت که حداکثر، n بار تابع تقسیم فراخوانی می‌شود. در هر بار فراخوانی این تابع نیز تعدادی زمان اجرا از مرتبه‌ی $\Theta(۱)$ داریم و حلقه‌ی **for**ای که زمان اجرای آن به تعداد مقایسه‌هایی بستگی دارد که در آن انجام می‌شود. از آنجایی که با الگوریتم مرتب‌سازی سریع تصادفی کار می‌کنیم، تعداد مقایسه‌ها در هر بار اجرای تابع تقسیم، تصادفی است؛ اگر تعداد کل مقایسه‌ها را با نماد X نشان دهیم، داریم:

$$T(n) = \Theta(n) + X$$

که چون برای متوسط زمان اجرا باید امید ریاضی را به‌دست آوریم، پس:

$$\mathbb{E}[T(n)] = \Theta(n) + \mathbb{E}[X]$$

۱.۱.۴ محاسبه‌ی امید ریاضی تعداد مقایسه‌ها

برای این کار ابتدا اعضای آرایه‌ی nتایی ورودی را به صورت $z_۱, z_۲, \cdots, z_n$ در نظر می‌گیریم و مجموعه‌ی $Z_{i,j} = \{z_i, z_{i+۱}, \cdots, z_j\}$ را به عنوان مجموعه‌ی اعضای بین i و j تعریف می‌کنیم. حال اگر پیشامد مقایسه‌شدن دو عضو را با $C_{i,j}$ نشان دهیم، متغیر شاخص $X_{i,j}$ را برای این پیشامد به صورت زیر می‌توان تعریف کرد:

$$X_{i,j} = \begin{cases} ۱ & \text{اگر پیشامد } C_{i,j} \text{ رخ دهد} \\ ۰ & \text{در غیر این صورت} \end{cases}$$

که آن را به اختصار به صورت $X_{i,j} = \mathrm{I}(C_{i,j})$ نشان می‌دهند.
از آنجایی مقایسه‌ی بین دو عنصر زمانی رخ می‌دهد که یکی از آن دو به عنوان محور انتخاب شده‌باشد و پس از مقایسه نیز دیگر امکان مقایسه‌ی دوباره وجود ندارد، پس می‌توان نوشت:

$$X = \sum_{i=۱}^{n-۱} \sum_{j=i+۱}^{n} X_{i,j}$$

در نتیجه برای محاسبه‌ی امید ریاضی X از خطی‌بودن امید ریاضی استفاده کرده و داریم:

$$\begin{aligned}\mathbb{E}[X] &= \mathbb{E}\left(\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{i,j}\right)\\ &= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \mathbb{E}[X_{i,j}]\\ &= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \Pr(C_{i,j})\end{aligned}$$

برای اینکه احتمال مقایسه‌شدن دو عضو z_j و z_i را محاسبه کنیم، فرض می‌کنیم که مجموعه‌ی ورودی به تابع تقسیم دارای زیرمجموعه‌ی $Z_{i,j}$ باشد؛ در این صورت اگر عضو محور، کوچکتر از z_i، یا بزرگتر از z_j باشد در این صورت نمی‌توان گفت که این دو مقایسه می‌شوند یا نه و باید مراحل بعدی را هم بررسی کرد. پس فرض می‌کنیم که به مرحله‌ای رسیده‌ایم که در آن خود مجموعه‌ی $Z_{i,j}$ ورودی تابع تقسیم است. در این حالت فقط در صورتی این دو با هم مقایسه می‌شوند که یکی از این دو به عنوان محور انتخاب شود و چون فرض کردیم اعضا به صورت یکنواخت انتخاب‌می‌شوند، می‌توان نوشت:

$$\begin{aligned}\Pr(C_{i,j}) &= \Pr(\text{انتخاب‌شدن } i \text{ به عنوان محور}) + \Pr(\text{انتخاب‌شدن } j \text{ به عنوان محور})\\ &= \frac{1}{j-i+1} + \frac{1}{j-i+1}\\ &= \frac{2}{j-i+1}\end{aligned}$$

با جایگذاری در $\mathbb{E}[X]$ داریم:

$$\begin{aligned}\mathbb{E}[X] &= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \frac{2}{j-i+1}\\ &= \sum_{i=1}^{n-1}\sum_{\mathrm{k}=1}^{n-i} \frac{2}{\mathrm{k}+1}\\ &< \sum_{i=1}^{n-1}\sum_{\mathrm{k}=1}^{n} \frac{2}{k}\\ &= \sum_{i=1}^{n-1} \mathcal{O}(\log n)\\ &= \mathcal{O}(n\log n)\end{aligned}$$

پس می‌توان نوشت:

$$\begin{aligned}\mathbb{E}[T(n)] &= \Theta(n) + \mathcal{O}(n\log n)\\ &= \mathcal{O}(n\log n)\end{aligned}$$

از طرفی چون بهترین حالت که به آن اشاره‌شد، دارای مرتبه‌ی زمانی $\mathcal{O}(n\log n)$ است، پس به طور قوی‌تر کران پایین آن به صورت $\Omega(n\log n)$ است که از این مطلب و اثبات بالا می‌توان نتیجه گرفت که:

$$\mathbb{E}[T(n)] = \Theta(n\log n)$$