دانشکده علوم ریاضی و آمار

مدرس: دکتر مجتبی رفیعی — نیمسال اول ۱۴۰۰-۱۴۰۱

ساختمان داده‌ها و الگوریتم‌ها

جلسه ۱۴

نگارنده: مهدی بیدرام

۲۵ مهر ۱۴۰۰

فهرست مطالب

۱ تحلیل کارایی الگوریتم مرتب‌سازی حبابی

در جلسات قبل دیدیم که هزینه هر خط از الگوریتم از آنجایی که یک عدد ثابت است، تاثیری در تحلیل کارایی در حالت کلی ندارد. بنابراین هزینه هر خط از الگوریتم را $\mathcal{O}(1)$ در نظر می‌گیریم.

شمارنده i	هزینه	توضیحات
$i=1$	$3(n-1)$	$j=n,...,2, n-2+1=n-1$
$i=2$	$3(n-2)$	$j=n,...,3, n-3+1=n-2$
.	.	.
.	.	.
.	.	.
$i=n-1$	$3(2)$	$j=n,...,n-1, n-n+1=1$

$$\boxed{T(n) = \sum_{i=1}^{n-1} 3(n-i) = \theta(n*n)}$$

سوال: پیچیدگی الگوریتم مرتب‌سازی حبابی در حالت متوسط، بدترین حالت و بهترین حالت چیست؟ همانطور که از الگوریتم مرتب‌سازی حبابی قابل مشاهده است این الگوریتم بدون توجه به محتوای آرایه عمل می‌کند. بنابراین هر سه حالت $\theta(n*n)$ است.

یادآوری

پیچیدگی زمانی الگوریتم مرتب‌سازی درجی به صورت زیر است:

- حالت متوسط: $\theta(n*n)$
- بهترین حالت: $\theta(n)$
- بدترین حالت: $\theta(n*n)$

شهود برای زمان مورد نیاز برای اجرا در $\theta(n*n)$ و $\theta(nlogn)$:
فرض کنید $n = 10^6$ و هر عمل 10^{-6} ثانیه نیاز دارد.

$$n^2 : (10^6)^2 * 10^{-6} = 10^6 s = 12 \text{ days}$$
$$n \log n : 10^6 \log 10^6 * 10^{-6} = 6s$$

۲ روش تقسیم و حل (Divide and Conquer)

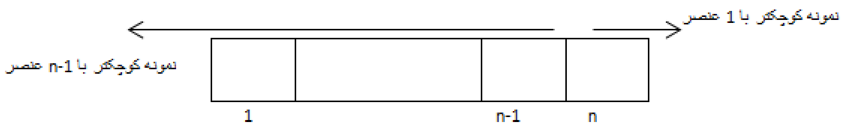
ساختار بسیاری از الگوریتم‌ها بازگشتی هستند، بدین معنا که در درون روال حل مساله، خود را فراخوانی می‌کند. این الگوریتم‌ها درواقع از روش تقسیم و حل پیروی می‌کند که شامل سه مرحله زیر است:

- تقسیم (Divide): مساله به تعدادی زیر مساله تقسیم می‌شود،
- حل (Conquer): زیرمساله‌ها به صورت بازگشتی حل می‌شوند،
- ترکیب (Combine): جواب زیرمساله‌ها باهم ترکیب و مساله اولیه حل می‌شود.

نکاتی در رابطه با گام تقسیم:

- تقسیم را تا کجا ادامه دهیم؟ تا رسیدن به نمونه‌های کوچک که به راحتی برای ما قابل حل باشد، مثلا $n = 1$, $n < 10$. در مقابل، نمونه های بزرگ نیز به دسته‌های k تایی مثلا $k = 2, 3, 4, \ldots$ تقسیم می‌شوند.
- الگوریتم‌های ارایه شده با روش تقسیم و غلبه معمولا زمانی بهترین پیچیدگی زمانی را دارند که نمونه های بزرگ به نمونه های کوچکتر با اندازه تقریبا یکسان تقسیم شوند.

مثال مرتب سازی درجی Insertion-Sort: آرایه n عنصری A را در نظر بگیرید:



واضح است در هر مرحله ۲ تقسیم انجام می شود:

- یک تقسیم با $n-1$ عنصر،
- یک تقسیم با 1 عنصر.

۳ الگوریتم مرتب‌سازی درجی

دو نسخه تکراری و بازگشتی را می‌توان برای الگوریتم مرتب‌سازی درجی در نظر گرفت:

- تکراری (Iterative): این نسخه از الگوریتم مرتب‌سازی درجی در جلسات قبلی تشریح شد.
- بازگشتی یا (Recursive): با الگو گرفتن از ایده مثال قبلی، نسخه بازگشتی الگوریتم مرتب‌سازی درجی به صورت زیر است.

Algorithm 1 R-Insertion-Sort($A[1 \ldots n],\ m$)

1: **if** $(m <= 1)$ **then**
2:     Return
3: R-Insertion-Sort($A[1 \ldots n],\ m-1$)
4: $key \leftarrow A[m]$
5: $i \leftarrow m$
6: **while** $i > 1$ and $A[i-1] > key$ **do**
7:     $A[i] \leftarrow A[i-1]$
8:     $i \leftarrow i-1$
9: $A[i] \leftarrow key$

سوال

اثبات درستی الگوریتم فوق به چه نحو قابل انجام است؟

اثبات درستی الگوریتم مرتب سازی درجی-نسخه بازگشتی

- خط ۶ تا ۸ الگوریتم شبیه قبل قابل با استفاده از مفهوم ناوردایی حلقه قابل اثبات است.
- برای کل الگوریتم هم از اثبات استقرایی استفاده می کنیم :
  - **پایه استقرا:** آرایه تک عنصری یک آرایه مرتب است،
  - **گزار استقرا:** $k-1$ عنصر اول آرایه مرتب باشد و از درستی خط ۶ تا ۸ داریم که k عنصر اول مرتب است.

سوال

تحلیل پیچیدگی الگوریتم فوق به چه نحو قابل محاسبه است ؟

فرم کلی برای محاسبه پیچیدگی زمانی الگوریتم های بازگشتی

$$T(n) = aT(n/b) + D(n) + C(n), \;\; T(1) = c$$

جاییکه:

- D(n): زمان اجرای مرحله تقسیم برای n عنصر است،
- C(n): زمان اجرای مرحله ترکیب برای n عنصر است،
- a: بیانگر تعداد زیرمساله‌هاست که باید حل شود،
- 1/b: بیانگر آن است که اندازه زیرمساله، $\frac{1}{b}$ اندازه مساله اصلی است.

بنابراین مطابق فرمول بالا تحلیل پیچدگی الگوریتم مرتب‌سازی درجی–نسخه بازگشتی به صورت زیر است:

$$T(n) = aT(n/b) + D(n)/O(1) + C(n)/O(n)$$
$$T(n) = T(n-1) + O(n), \quad T(n) = n * O(n) = O(n * n)$$