مرور فصل ۴

الگوی تقسیم و حل را به خاطر بیاورید که برای merge sort استفاده کردیم:

- **تقسیم**: مسئله را به یک یا چند زیرمسئله که نمونه‌های کوچکتری از مسئله اصلی هستند، تقسیم کنید.
- **حل**: زیرمسئله‌ها را به روش بازگشتی حل کنید.
    - به زیرمسئله به اندازه کافی کوچک باشد، که بتوان به آسانی و بطور مستقیم آن را حل کرد، حالت پایه (Base case) گفته می‌شود.
- **ترکیب**: راه‌حل‌های زیرمسئله‌ها را برای تشکیل راه‌حل مسئله اصلی(در صورت نیاز) ترکیب کنید.

در این فصل، ما به دو الگوریتم برای ضرب ماتریس‌های مربعی بر اساس تقسیم و حل نگاه می‌کنیم.

تحلیل الگوریتم‌های تقسیم و حل

از یک رابطه بازگشتی برای مشخص کردن زمان اجرای یک الگوریتم تقسیم و حل استفاده کنید. که حل آن رابطه بازگشتی به ما زمان اجرای مجانبی را می‌دهد. رابطه بازگشتی () که برای توصیف پیچیدگی کاربرد دارد، بر حسب موارد زیر تعریف می‌شود:

- یک یا چند base case
- و فراخوانی دوباره خود، البته با آرگومان‌های کوچکتر.

بنابراین، یک رابطه بازگشتی در حقیقت یک تابع است، که از صفر، یک و یا چند تابع دیگر تشکیل شده است. اگر رابطه بازگشتی تنها از یک تابع تشکیل شده باشد به آن **خوش تعریف** گفته می‌شود. در غیر این صورت رابطه خوش تعریف نخواهد بود.

روابط بازگشتی الگوریتمی

ما به روابط بازگشتی که زمان اجرای الگوریتم‌ها را توصیف می‌کنند، علاقه‌مندیم و آنها را **الگوریتمی** می‌نامیم. یک رابطه بازگشتی T(N) الگورماتیک algorithmic است اگر برای هر ثابت آستانه به اندازه کافی بزرگ ($n_0 > 0$) موارد زیر را داشته باشیم:

۱. برای همه $n < n_0$, داشته باشیم $T(n) = \Theta(1)$. [می‌توان زمان اجرا را برای اندازه‌های کوچک مسئله ثابت در نظر گرفت.]
۲. برای همه $n \geq n_0$, هر مسیر بازگشت به یک حالت پایه base case تعریف شده، و با تعداد محدودی از فراخوانی‌های بازگشتی خاتمه می‌یابد. [بطور ساده: الگوریتم بازگشتی خاتمه پذیر باشد.]

نمونه‌هایی از روابط بازگشتی ناشی از الگوریتم‌های تقسیم و حل

- ضرب ماتریس $n \times n$ با تقسیم به ۸ زیرمسئله اندازه $n/2 \times n/2$:

$$T(n) = 8T(n/2) + \Theta(1)$$

راه‌حل:

$$T(n) = \Theta(n^3)$$

- الگوریتم استراسن برای ضرب ماتریس $n \times n$ با تقسیم به ۷ زیرمسئله اندازه$n/2 \times n/2$

$$T(n) = 7T(n/2) + \Theta(1)$$

راه‌حل:

$$T(n) = \Theta\left(n^{\lg 7}\right) = O(n^{2.81})$$

- الگوریتمی که یک مسئله به اندازه n را به یک زیرمسئله اندازه $n/3$ و یک زیرمسئله دیگر به اندازه $2n/3$تقسیم می‌کند، با زمان $\Theta(n)$ برای تقسیم و ترکیب:

$$T(n) = T(n/3) + T(2n/3) + \Theta(n)$$

راه‌حل:

$$T(n) = \Theta(nlgn)$$

- الگوریتمی که یک به مسئله اندازه nرا به یک زیرمسئله اندازه $n/5$ و دیگری اندازه $7n/10$ تقسیم می‌کند، با زمان $\Theta(n)$ برای تقسیم و ترکیب:

$$T(n) = T(n/5) + T(7n/10) + \Theta(n)$$

راه‌حل:

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

- زیرمسئله‌ها همیشه نباید کسری ثابتی از اندازه مسئله اصلی باشند. مثال: جستجوی خطی بازگشتی یک زیرمسئله ایجاد می‌کند ، که زیر مسئله یک عنصر کمتر از مسئله اصلی دارد. زمان تقسیم و ترکیب$\Theta(1)$است، که می‌دهد

$$T(n) = T(n-1) + \Theta(1)$$

راه‌حل:

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

روش‌های حل روابط بازگشتی

این فصل شامل چهار روش برای حل روابط بازگشتی است. هر یک کران‌های مجانبی را برای مات فراهم می‌کنند.

- روش جایگزینی: ابتدا راه حل را حدس بزنید، سپس از استقرا برای اثبات درستی آن استفاده کنید.
- روش درخت بازگشتی: یک درخت بازگشتی رسم کنید، هزینه‌ها را در هر سطح تعیین کنید و آنها را جمع بزنید. برای ایجاد حدس برای روش جایگزینی مفید است.
- روش اصلی Master method : یک روش برای روابط بازگشتی به شکل $T(n) = aT(n/b) + f(n)$ ، که در آن $a > 0$ و $b > 1$ ثابت هستند، تحت شرایط خاصی. نیاز به حفظ کردن سه حالت دارد، اما برای بسیاری از الگوریتم‌های تقسیم و حل اعمال می‌شود.

مسئله ضرب ماتریس‌های مربعی

ورودی : سه ماتریس $n \times n$ (مربعی)، $A = (a_{ij})$، $B = (b_{ij})$ و $C = (c_{ij})$

نتیجه(خروجی): حاصلضرب ماتریس $A \cdot B$ در ماتریس C قرار می‌گیرد

به طوری که

$$c_{ij} = c_{ij} + \sum_{k=1}^{n} a_{ik} \cdot b_{kj} \quad \text{برای} \quad i, j = 1,2, \dots, n.$$

تمام درایه‌های C را از قبل صفر کنید.

روش مستقیم

```
MATRIX-MULTIPLY(A, B, C, n)
  for i = 1 TO n do
    for j = 1 to n do
      for k = 1 to n do
        c_ij = c_ij + a_ik · b_kj
      end for
    end for
  end for
```

پیچیدگی زمانی این روش حل $\Theta(n^3)$ است.

یک الگوریتم تقسیم و حل برای مسئله ماتریس‌های مربعی

برای سادگی، فرض کنید C با صفر مقداردهی اولیه شده است، بنابراین $C = A \cdot B$ محاسبه می‌شود.

اگر $n > 1$، هر یک از A, B, C را به چهار ماتریس $n/2 \times n/2$ تقسیم کنید:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}.$$

نوشتن

$$C = A \cdot B$$

به صورت:

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix},$$

چهار معادله زیر را می‌دهد:

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21},$$
$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22},$$
$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21},$$
$$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}.$$

هر یک از این معادلات دو ماتریس $n/2 \times n/2$ را ضرب می‌کنند و سپس حاصلضرب‌های $n/2 \times n/2$ آنها را جمع می‌کنند. فرض کنید nتوان دقیقی از ۲ است، بنابراین ابعاد زیرماتریس‌ها همیشه صحیح هستند. از این معادلات برای به دست آوردن یک الگوریتم تقسیم و حل استفاده کنید:

MATRIX-MULTIPLY-RECURSIVE(A, B, C, n)
if $n == 1$ **then**
  // Base case.
  $c_{11} = c_{11} + a_{11} \cdot b_{11}$
  return
end if
// Divide.
//partition A, B, and C into $n/2 \times n/2$ submatrices
// $A_{11}, A_{12}, A_{21}, A_{22}; B_{11}, B_{12}, B_{21}, B_{22};$
//and $C_{11}, C_{12}, C_{21}, C_{22}$; respectively
// Conquer.
MATRIX-MULTIPLY-RECURSIVE$(A_{11}, B_{11}, C_{11}, n/2)$
MATRIX-MULTIPLY-RECURSIVE$(A_{11}, B_{12}, C_{12}, n/2)$
MATRIX-MULTIPLY-RECURSIVE$(A_{21}, B_{11}, C_{21}, n/2)$
MATRIX-MULTIPLY-RECURSIVE$(A_{21}, B_{12}, C_{22}, n/2)$
MATRIX-MULTIPLY-RECURSIVE$(A_{12}, B_{21}, C_{11}, n/2)$
MATRIX-MULTIPLY-RECURSIVE$(A_{12}, B_{22}, C_{12}, n/2)$
MATRIX-MULTIPLY-RECURSIVE$(A_{22}, B_{21}, C_{21}, n/2)$
MATRIX-MULTIPLY-RECURSIVE$(A_{22}, B_{22}, C_{22}, n/2)$

[کتاب به طور خلاصه در مورد چگونگی اجتناب از کپی کردن درای‌ه‌ها هنگام تقسیم ماتریس ها بحث می‌کند. می‌توان ماتریس ها را بدون کپی کردن درایه‌ها با استفاده از محاسبات اندیس تقسیم کرد. یک زیرماتریس را با محدوده‌ای از ردیف ها و ستون های ماتریس اصلی شناسایی کنید. در نهایت یت زیرماتریس را متفاوت از نحوه نمایش ماتریس اصلی نمایش می‌دهید. مزیت اجتناب از کپی کردن این است که تقسیم فقط زمان ثابت می‌گیرد، به جای $\Theta(n2)$ زمان. نتیجه تحلیل مجانبی تغییر نمی‌کند، اما استفاده از محاسبات اندیس برای اجتناب از کپی کردن عوامل ثابت بهتری می‌دهد]

تحلیل پیچیدگی الگوریتم ضرب ماتریس‌های مربعی با روش تقسیم و حل

فرض کنید$T(n)$ زمان ضرب دو ماتریس$n \times n$ باشد. ما باید دو حالت را بررسی کنیم. این دو حالت عبارتند از حالت پایه و برای حالت n>n0 که بازگشت اتفاق می‌افتد.

- در حالت پایه یک ضرب اسکالر انجام می شود پس زمان مورد نیاز $\Theta(1)$ است
- حالت بازگشتی $(n > 1)$
  - تقسیم کردن زمان $\Theta(1)$ را به خود اختصاص می‌دهد، البته با استفاده از محاسبات اندیس. [در غیر این صورت،$\Theta(n^2)$ زمان.]
  - حل کردن ۸ فراخوانی بازگشتی انجام می‌دهد، هر یک ماتریس‌های$n/2 \times n/2$ را ضرب می‌کنند ⇐ $8T(n/2)$
  - هیچ مرحله ترکیبی وجود ندارد، زیرا C در جا به‌روزرسانی می‌شود.

رابطه بازگشتی (حذف base case) بصورت زیر بدست خواهد آمد

$$T(n) = 8T(n/2) + \Theta(1)$$

می‌توان از روش اصلی استفاده کرد تا نشان دهیم که راه‌حل $T(n) = \Theta(n^3)$ است.

واضح است که این زمان بهترین زمان ممکن نیست

نکته (بررسی ظاهر درخت بازگشتی حاصل از این الگوریتم) : این رابطه بازگشتی را با رابطه بازگشتی merge-sort $T(n) = 2T(n/2) + \Theta(n)$ مقایسه کنید. اگر درخت‌های بازگشتی را رسم کنیم، ضریب ۲ در رابطه بازگشتی merge-sort می‌گوید که هر گره غیربرگ ۲ فرزند دارد. اما ضریب ۸ در رابطه بازگشتی برای Matrix-Multiply-Recursive می‌گوید که هر گره غیربرگ ۸ فرزند دارد. یک درخت با شاخه های زیاد و با برگ‌های بسیار بیشتر به دست می‌آید، حتی اگر گره‌های داخلی هزینه کمتری داشته باشند.

الگوریتم استراسن

ایده اصلی: سعی می شود درخت بازگشت انبوهی کمتری داشته باشد.

این روش فقط ۷ ضرب بازگشتی ماتریس‌های $n/2 \times n/2$ را انجام می‌دهد، نه ۸. هزینه چند جمع/تفریق ماتریس‌های $n/2 \times n/2$ است.

یعنی تلاش می‌شود از تعداد ضرب ها کاسته شود و به جای آنها از جمع و تفریق استفاده کنیم

از آنجا که تفریق یک "جمع با علامت منفی" است، فقط به همه جمع‌ها و تفریق‌ها به عنوان جمع اشاره می‌کنیم.

مثال: با داشتن x و y، می خواهیم $x^2 - y^2$ را محاسبه کنیم. روش معمولی از ۲ ضرب و یک تفریق استفاده می‌کند. اما مشاهده کنید:

$$x^2 - y^2 = x^2 - xy + xy - y^2 = x(x-y) + y(x-y) = (x+y)(x-y),$$

بنابراین به قیمت یک جمع اضافی، می‌توان با فقط ۱ ضرب انجام داد. اگر x , y اسکالر باشند مهم نیست، اما اگر ماتریس باشند می‌تواند تفاوت ایجاد کند.

الگوریتم:

۱. زمانی که $n = 1$ است حالت پایه رخ می دهد

۲. وقتی $n > 1$، همانند روش بازگشتی، هر یک از ماتریس‌ها را به چهار زیرماتریس $n/2 \times n/2$ تقسیم کنید. زمان: $\Theta(1)$، با استفاده از محاسبات اندیس.

۳. ۱۰ ماتریس $S_1, S_2, \ldots, S_{10}$ ایجاد کنید. هر یک $n/2 \times n/2$ است و جمع یا تفاوت دو ماتریس ایجاد شده در مرحله قبل است. زمان: $\Theta(n^2)$ برای ایجاد همه ۱۰ ماتریس.

۴. ۷ ماتریس $P_1, P_2, \ldots, P_7$ ایجاد کنید و درایه‌های آنها را صفر کنید، هر یک $n/2 \times n/2$. زمان: $\Theta(n^2)$.

۵. با استفاده از زیرماتریس‌های A و B و ماتریس‌های $S_1, S_2, \ldots, S_{10}$، $P_1, P_2, \ldots, P_7$ را به صورت بازگشتی محاسبه کنید. زمان: $7T(n/2)$.

۶. چهار زیرماتریس $C_{11}, C_{12}, C_{21}, C_{22}$ از C را با جمع و تفریق ترکیبات مختلف P_i به‌روزرسانی کنید. زمان: $\Theta(n^2)$.

تحلیل

رابطه بازگشتی $T(n) = 7T(n/2) + \Theta(n^2)$ خواهد بود. با روش اصلی، راه‌حل $T(n) = \Theta(n^{\lg 7})$ است. از آنجا که $\lg 7 < 2.81$، زمان اجرا $O(n^{2.81})$ است، که از روش‌های $\Theta(n^3)$time- بهتر است.

جزئیات

مرحله ۲: ایجاد ۱۰ ماتریس

$$
\begin{aligned}
S_1 &= B_{12} - B_{22},\\
S_2 &= A_{11} + A_{12},\\
S_3 &= A_{21} + A_{22},\\
S_4 &= B_{21} - B_{11},\\
S_5 &= A_{11} + A_{22},\\
S_6 &= B_{11} + B_{22},\\
S_7 &= A_{12} - A_{22},\\
S_8 &= B_{21} + B_{22},\\
S_9 &= A_{11} - A_{21},\\
S_{10} &= B_{11} + B_{12}.
\end{aligned}
$$

جمع یا تفریق ماتریس‌های $n/2 \times n/2$ ۱۰ بار $\Rightarrow$ زمان $\Theta(n^2)$ است.

مرحله ۴: محاسبه ۷ ماتریس

$$
\begin{aligned}
P_1 &= A_{11} \cdot S_1 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22},\\
P_2 &= S_2 \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22},\\
P_3 &= S_3 \cdot B_{11} = A_{21} \cdot B_{11} + A_{22} \cdot B_{11},\\
P_4 &= A_{22} \cdot S_4 = A_{22} \cdot B_{21} - A_{22} \cdot B_{11},\\
P_5 &= S_5 \cdot S_6 = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22},\\
P_6 &= S_7 \cdot S_8 = A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22},\\
P_7 &= S_9 \cdot S_{10} = A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}.
\end{aligned}
$$

تنها ضرب‌های مورد نیاز در ستون وسط هستند؛ ستون سمت راست فقط حاصلضرب‌ها را بر حسب زیرماتریس‌های اصلی A و B نشان می‌دهد.

مرحله ۵: جمع و تفریق P_i برای ساخت زیرماتریس‌های C:

$$
\begin{aligned}
C_{11} &= P_5 + P_4 - P_2 + P_6,\\
C_{12} &= P_1 + P_2,\\
C_{21} &= P_3 + P_4,\\
C_{22} &= P_5 + P_1 - P_3 - P_7.
\end{aligned}
$$

نکات نظری و عملی

الگوریتم استراسن اولین الگوریتمی بود که از زمان $\Theta(n^3)$ بهتر شد، اما از نظر مجانبی سریع‌ترین الگوریتم شناخته شده نیست. یک روش توسط کاپرسمیث و وینوگراد در زمان $O(n^{2.376})$ اجرا می‌شود. بهترین کران مجانبی فعلی (عملی نیست) $O(n^{2.37286})$ است.

مسائل عملی علیه الگوریتم استراسن:

- ضریب ثابت بالاتر از روش معمولی $\Theta(n^3)$ دارد.
- برای ماتریس‌های پراکنده خوب نیست.
- از نظر عددی پایدار نیست: خطاهای بزرگتری نسبت به روش معمولی جمع می‌شوند.
- زیرماتریس‌ها فضا مصرف می‌کنند، به ویژه اگر کپی شوند.

مشکل پایداری عددی به اندازه گذشته بد نیست. و می‌توان از محاسبات اندیس برای کاهش نیاز به فضا استفاده کرد.

محققان مختلف سعی کرده‌اند نقطه تقاطع را پیدا کنند، جایی که الگوریتم استراسن سریع‌تر از روش معمولی $\Theta(n^3)$ time-اجرا می‌شود. پاسخ‌ها متفاوت است.

۱. پیدا کردن یک حدس خوب برای پیچیدگی زمانی

۲. گام استقرا: از استقرا برای یافتن ثوابت و نشان دادن کارایی جواب استفاده کنید.

معمولاً از روش جایگزینی برای ایجاد کران بالا (O-bound) یا کران پایین (Ω-bound) استفاده می‌شود.

گام نخست: پیدا کردن یک حدس مناسب

چند راه رایج برای پیدا کردن یک حدس خوب وجود دارد:

روش تکرار (Iteration Method): 🔁

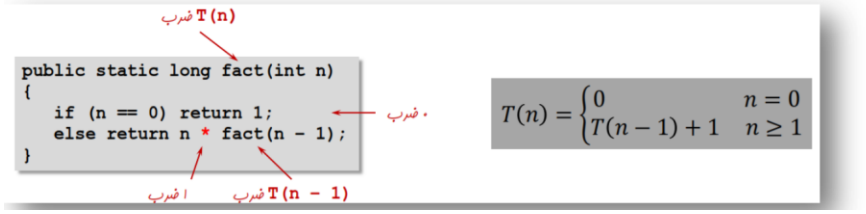
- این شبیه درخت بازگشتی است، اما جبری‌تر است.
- ما معادله را باز می‌کنیم (جایگذاری می‌کنیم) و آن را بارها و بارها تکرار می‌کنیم تا یک الگو پیدا شود.

روش درخت بازگشتی (Recursion Tree): 🌳

- این یک روش بصری عالی است. ما ساختار بازگشتی را به شکل یک درخت رسم می‌کنیم و هزینه‌ی هر سطح را محاسبه می‌کنیم.
- با جمع زدن هزینه‌های تمام سطوح، یک فرم کلی برای $T(n)$ به دست می‌آوریم که همان حدس ما می‌شود.

استفاده از قضیه اصلی (Master Theorem): 📜

- قضیه اصلی (Master Theorem) یک ابزار بسیار قدرتمند است که پاسخ را مستقیماً برای دسته‌ی بزرگی از معادلات بازگشتی (به شکل $T(n) = aT(n/b) + f(n)$) به ما می‌دهد.
- در عمل، خیلی‌ها اول با قضیه اصلی جواب را پیدا می‌کنند و سپس اگر نیاز به اثبات رسمی باشد، از روش جایگذاری (و استقراء) برای اثبات همان جواب استفاده می‌کنند.



$$T(n) = \begin{cases} 0 & n = 0 \\ T(n-1) + 1 & n \geq 1 \end{cases}$$

حدس

```
T(1) = T(0) + 1 = 0 + 1 = 1
T(2) = T(1) + 1 = 1 + 1 = 2
T(3) = T(2) + 1 = 2 + 1 = 3
T(4) = T(3) + 1 = 3 + 1 = 4
...
T(n) = n
```

مثال:

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T(n-1) + 1 & n > 1 \end{cases}$$

حدس

```
T(2) = 2T(1) + 1 =  2 + 1 =  3
T(3) = 2T(2) + 1 =  6 + 1 =  7
T(4) = 2T(3) + 1 = 14 + 1 = 15
T(5) = 2T(4) + 1 = 30 + 1 = 31
...
T(n) = 2^n - 1
```

مثال:

$$T(n) = \begin{cases} 0 & n = 1 \\ 2T(n/2) + n - 1 & n > 1, n = 2^k \end{cases}$$

- حدس
  - حدس جواب با استفاده از چند جمله‌ی اولیه

حدس

```
T(2)  = 2T(1) + 1  =  0 +  1 =  1
T(4)  = 2T(2) + 3  =  2 +  3 =  5
T(8)  = 2T(4) + 7  = 10 +  7 = 17
T(16) = 2T(8) + 15 = 34 + 15 = 49
...
T(n)  = ?
```

حدس راه‌حل همیشه امکان‌پذیر نیست!

حدس زدن به کمک جایگذاری مکرر

مثال: ما می خواهیم جواب $T(n) = 2T(n/2) + cn$ را با جایگذاری مکرر بدست آوریم.

گام ۱: معادله اصلی

ما با این معادله شروع می‌کنیم:

$$T(n) = 2T(n/2) + cn$$

گام ۲: باز کردن (جایگذاری $T(n/2)$)

حالا، ما باید عبارت $T(n/2)$ را پیدا کنیم. برای این کار، کافی است در معادله اصلی، هر n را با $n/2$ جایگزین کنیم:

$$T(n/2) = 2T(n/4) + c(n/2)$$

حالا این عبارت جدید را در معادله اصلی (گام ۱) جایگذاری می‌کنیم:

$$T(n) = 2\left[2T(n/4) + c(n/2)\right] + cn$$

$$T(n) = 4T(n/4) + cn + cn$$

$$T(n) = 4T(n/4) + 2cn$$

گام ۳: باز کردن (جایگذاری $T(n/4)$)

بیایید یک بار دیگر این کار را تکرار کنیم. این بار $T(n/4)$ را پیدا می‌کنیم:

$$T(n/4) = 2T(n/8) + c(n/4)$$

و آن را در نتیجه‌ی گام ۲ (یعنی $T(n) = 4T(n/4) + 2cn$) جایگذاری می‌کنیم:

$$T(n) = 4\left[2T(n/8) + c(n/4)\right] + 2cn$$

$$T(n) = 8T(n/8) + cn + 2cn$$

$$T(n) = 8T(n/8) + 3cn$$

گام ۴: پیدا کردن الگو

بیایید نتایجی را که به دست آوردیم کنار هم بگذاریم:

- **بعد از ۱ بار باز کردن:** $T(n) = 2^1T(n/2^1) + 1cn$
- **بعد از ۲ بار باز کردن:** $T(n) = 2^2T(n/2^2) + 2cn$
- **بعد از ۳ بار باز کردن:** $T(n) = 2^3T(n/2^3) + 3cn$

$$T(n) = 2^i T(n/2^i) + icn$$

گام ۵: رسیدن به پایه (Base Case)

این فرآیند باز کردن و تکرار تا ابد ادامه پیدا نمی‌کند. در نهایت، ما به "شرایط پایه" (Base Case) می‌رسیم. در روابط بازگشتی، ما معمولاً فرض می‌کنیم که این تکرار زمانی متوقف می‌شود که اندازه ورودی به 1 برسد (یعنی $T(1)$).

ما می‌دانیم که $T(1)$ یک هزینه ثابت است (مثلاً $T(1) = 1$ یا $T(1) = d$).

حالا سوال کلیدی این است: **چند بار** باید این فرآیند را تکرار کنیم (یعنی i باید چقدر باشد) تا $n/2^i$ برابر با 1 شود؟

به عبارت دیگر، در معادله $n/2^i = 1$، مقدار i بر حسب n چه می‌شود؟

$$n/2^i = 1$$

اگر دو طرف تساوی را در 2^i ضرب کنیم، به این می‌رسیم:

$$n = 2^i$$

این معادله دقیقاً **تعریف لگاریتم** (در پایه ۲) است. این معادله می‌پرسد: «عدد ۲ باید به چه توانی (i) برسد تا برابر با n شود؟»

پاسخ این سؤال $i = \log_2 n$ (لگاریتم n در پایه ۲) است.

گام ۶: جایگذاری i و پیدا کردن حدس نهایی

حالا که هم الگوی کلی را داریم و هم مقدار i را می‌دانیم، می‌توانیم آن‌ها را با هم ترکیب کنیم.

- **الگوی کلی:** $T(n) = 2^i T(n/2^i) + icn$
- **مقدار i:** $i = \log_2 n$

بیایید این i را در الگوی کلی جایگذاری کنیم:

$$T(n) = 2^{\log_2 n} \cdot T(n/2^{\log_2 n}) + (\log_2 n) \cdot cn$$

این عبارت به نظر پیچیده می‌آید، اما می‌توانیم آن را بسیار ساده کنیم.

دو بخش کلیدی در این عبارت وجود دارد که باید آن‌ها را ساده کنیم:

1. $2^{\log_2 n}$
2. $T(n/2^{\log_2 n})$

گام ۷: ساده‌سازی حدس

ما به این رابطه رسیده بودیم:

$$T(n) = 2^{\log_2 n} \cdot T(n/2^{\log_2 n}) + (\log_2 n) \cdot cn$$

۱. **ساده‌سازی توان لگاریتمی:** 🔢 طبق تعریف لگاریتم (قاعده $b^{\log_b x} = x$):

- $2^{\log_2 n} = n$
- پس بخش اول عبارت ما می‌شود: $n \cdot T(\dots)$

۲. **ساده‌سازی آرگومان T:** ⏳ با توجه به نتیجه بالا، آرگومان T به شکل زیر ساده می‌شود:

- $T(n/2^{\log_2 n}) = T(n/n) = T(1)$
- $T(1)$ **شرط پایه** است و ما فرض می‌کنیم یک هزینه ثابت دارد. فرض کنیم $T(1) = d$ (که d یک ثابت است).

۳. **ترکیب نهایی:** حالا تمام اجزا را کنار هم می‌گذاریم:

$$T(n) = (n) \cdot T(1) + (\log_2 n) \cdot cn$$

$$T(n) = n \cdot d + cn \log_2 n$$

☀️ **حدس خوب (The Good Guess)**

در نهایت، از آنجایی که در تحلیل مجانبی، ثابت‌ها (مثل d و c) و جملات با رشد کندتر را نادیده می‌گیریم، **حدس خوب** ما عبارت است از:

$$T(n) = O(n \log n)$$

□ جایگذاری مکرر

$$T(n) = \begin{cases} 0 & n = 0 \\ T(n-1) + 1 & n \geq 1 \end{cases}$$

```
T(n) = T(n - 1) + 1
     = T(n - 2) + 1 + 1     = T(n - 2) + 2
     = T(n - 3) + 1 + 2     = T(n - 3) + 3
     = T(n - 4) + 1 + 3     = T(n - 4) + 4
                            ...
                            = T(n - i) + i
```

```
T(n - i) = T(0)      ⟹ n - i = 0
                     ⟹ i = n
T(n) = T(n - n) + n
     = T(0) + n
     = 0 + n
     = n
```

$$T(n) = \begin{cases} 1 & n = 1 \\ T(n/2) + 1 & n > 1, n = 2^k \end{cases}$$

```
T(n) = T(n/2¹) + 1                 = T(n/2¹) + 1
     = T(n/2²) + 1 + 1             = T(n/2²) + 2
     = T(n/2³) + 1 + 1 + 1         = T(n/2³) + 3
     = T(n/2⁴) + 1 + 1 + 1 + 1     = T(n/2⁴) + 4
                                     ...
                                   = T(n/2ⁱ) + i
```

```
T(n/2ⁱ) = T(1)      ⟹ n/2ⁱ = 1
                    ⟹ i = lg n

T(n) = T(1) + lg n
     =   1  + lg n
```

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T(n-1) + 1 & n > 1 \end{cases}$$

$$\begin{aligned}
T(n) &= 2T(n-1) + 1 && = 2^1T(n-1) + 2^1 - 1 \\
&= 2^1[2T(n-2) + 1] + 1 && = 2^2T(n-2) + 2^2 - 1 \\
&= 2^2[2T(n-3) + 1] + 3 && = 2^3T(n-3) + 2^3 - 1 \\
&= 2^3[2T(n-4) + 1] + 7 && = 2^4T(n-4) + 2^4 - 1 \\
&\quad \cdots \\
&= 2^iT(n-i) + 2^i - 1
\end{aligned}$$

$$\begin{aligned}
T(n-i) = T(1) &\Rightarrow n - i = 1 \\
&\Rightarrow \underline{i = n - 1} \\
\\
T(n) &= 2^{n-1}\,T(1) + 2^{n-1} - 1 \\
&= 2 \cdot 2^{n-1} - 1 \\
&= 2^n - 1
\end{aligned}$$

$$T(n) = \begin{cases} 0 & n = 1 \\ 2T(n/2) + n - 1 & n > 1, n = 2^k \end{cases}$$

$$\begin{aligned}
T(n) &= 2T(n/2) + n && = 2^1T(n/2^1) + 1 \cdot n \\
&= 2^1[2T(n/2^2) + n/2^1] + n && = 2^2T(n/2^2) + 2 \cdot n \\
&= 2^2[2T(n/2^3) + n/2^2] + 2n && = 2^3T(n/2^3) + 3 \cdot n \\
&= 2^3[2T(n/2^4) + n/2^3] + 3n && = 2^4T(n/2^4) + 4 \cdot n \\
&\quad \cdots \\
&= 2^iT(n/2^i) + i \cdot n
\end{aligned}$$

$$\begin{aligned}
T(n/2^i) = T(1) &\Rightarrow n/2^i = 1 \\
&\Rightarrow \underline{i = \lg n} \\
\\
T(n) &= 2^{\lg n}\,T(1) + n \lg n \\
&= n \cdot 0 + n \lg n \\
&= n \lg n
\end{aligned}$$

گام دوم : استفاده از استقراء برای نشان دادن درستی حدس

چرا اصلاً از استقراء استفاده می‌کنیم؟

ایده اصلی این است:

۱. **روابط بازگشتی** (Recurrences)ذاتاً بر اساس مقادیر **کوچکتر** خود تعریف می‌شوند. (مثلاً T(n) به T(n/2) یا T(n-1) بستگی دارد.)

۲. **اثبات استقرایی** (Induction)هم دقیقاً همین ساختار را دارد. ما یک "فرض" می‌کنیم که ادعای ما برای مقادیر **کوچکتر** (مثل $n_0 < k < n$)درست است، و سپس از این فرض استفاده می‌کنیم تا ثابت کنیم ادعای ما برای سایر مقادیر n هم درست است.

این دو (بازگشت و استقراء) مثل قفل و کلید برای هم ساخته شده‌اند.

مثال: می خواهیم یک کران بالا برای رابطه بازگشتی $T(n) = 2T(\lfloor n/2 \rfloor) + \Theta(n)$ پیدا کنیم.

پاسخ:

برای داشتن یک حدس خوب با کمی دقت متوجه میشویم که شبیه معداله پیچیدگی زمانی مرتب سازی ادغامی بوده و تنها در یک تابع کف تفاوت دارد. (این تابع جزء صحیح این اطمینان را می‌دهد که T(n) روی اعداد صحیح تعریف شده است.)

حدس می‌زنیم همان کران بالای مجانبی رابطه بازگشتی مرتب‌سازی ادغام در این جا هم درست باشد:

$$T(n) = O(n\lg n)$$

پس می خواهیم نشان دهیم که برای $n \geq n_0$ رابطه

$$T(n) \leq cn\lg n$$

برقرار است و ثابت‌های $c, n_0 > 0$ را پیدا کنیم. برای پیدا کردن این ثابت ها از استقراء استفاده خواهیم کرد.

استقرا:

با توجه به روش استقراء فرض کنید $T(n) \leq cn\lg n$ برای همه اعداد بزرگتر و مساوی از n_0 و کوچکتر از n برقرار باشد. در این صورت برای n/2 که عددی عددی بین n_0 و n است، رابطه برقرار خواهد بود.

اگر n/۲ عددی عددی بین n_0 و n است بنابراین رابطه برای آن بر قرار است. پس می توان گفت برای $n \geq 2n_0$ (یعنی n/2 را در رابطه جایگذار می کنیم) خواهیم داشت

$$\lfloor n/2 \rfloor \Rightarrow T(\lfloor n/2 \rfloor) \le c\ \lfloor n/2 \rfloor lg \lfloor n/2 \rfloor$$

حال می توانیم از درستی این عبارت ($T(\lfloor n/2 \rfloor) \le c\ \lfloor n/2 \rfloor lg \lfloor n/2 \rfloor$) ، درستی عبارت اصلی ($T(n) \le cn\lg n$) را برای اعداد بزرگتر و مساوی n را نتیجه بگیریم.

برای این کار رابطه $T(\lfloor n/2 \rfloor) \le c\ \lfloor n/2 \rfloor lg \lfloor n/2 \rfloor$ را در رابطه بازگشـــتی $T(n) = 2T(\lfloor n/2 \rfloor) + \Theta(n)$ جایگزین کنید:

$$T(n) = 2T(\lfloor n/2 \rfloor) + \Theta(n) \le 2\big(c\ \lfloor n/2 \rfloor lg(\lfloor n/2 \rfloor)\big) + \Theta(n)$$

$$\le 2\big(c(n/2)lg(n/2)\big) + \Theta(n)$$

$$= cn\lg(n/2) + \Theta(n)$$

$$= cn\lg n - cn\lg 2 + \Theta(n)$$

$$= cn\lg n - cn + \Theta(n)$$

$$\le cn\lg n.$$

آخرین گام اگر c, n_0 به اندازه کافی بزرگ باشند که برای $n \ge 2n_0$، cn بر جمله $\Theta(n)$ غالب باشد، برقرار است.

جمع‌بندی:داریم $T(n) \le cn\lg n$ برای همه $n \ge 2 \Rightarrow T(n) = O(n\lg n)$

حدس خوب زدن

هیچ روش کلی برای حدس خوب زدن وجود ندارد. تجربه کمک می‌کند. همچنین می‌توان درخت بازگشت را رسم کرد.

اگر رابطه بازگشتی شبیه رابطه‌ای باشد که قبلاً دیده‌اید، سعی کنید جوابی مشابه حدس بزنید.