مرور فصل ۳

- در این فصل راهکارهایی برای توصیف رفتار توابع پیچیدگی (بخصوص در هنگامی که اندازه مسئله بسیار بزرگ است) را ارائه خواهیم کرد. مطالعه رفتار تابع بوسیله مجانب ها (asymptotic) انجام می‌شود.
- در ریاضیات، **مجانب** (Asymptotic)به بررسی **رفتار یک تابع هنگامی که ورودی آن به بینهایت نزدیک می‌شود** می‌پردازد. **تحلیل مجانبی** روشی برای **توصیف رفتار الگوریتم‌ها هنگامی که اندازه ورودی بسیار بزرگ می‌شود** است.
  - توصیف رشد (growth) توابع
  - تمرکز بر روی آنچه مهم است با حذف عبارات مرتبه پایین و عوامل ثابت.
  - نحوه نشان دادن زمان‌های اجرای الگوریتم‌ها.
  - روشی برای مقایسه اندازه توابع:

$$O \approx \leq$$
$$\Omega \approx \geq$$
$$\Theta \approx =$$
$$o \approx <$$
$$\omega \approx >$$

<u>تابع پیچیدگی یا تابع زمان اجراء</u>، به تابعی گفته شده برحسب اندازه مسئله، زمان اجرای مسئله (تعداد دستورات لازم) را مشخص می‌کند. مانند توابع زیر

n^2 n^3 $\frac{n(n-1)}{2}$

تابع پیچیدگی همواره **<u>مثبت</u>** و **<u>غیرنزولی</u>** است. با توجه به این نکته آیا توابع ریاضی زیر می‌توانند یک تابع پیچیدگی باشند؟

1) $5n^2 - 100n$ 2) $3n+7$ 3) $30 - n^3$

پاسخ

۱) برای $n>20$ تابع پیچیدگی است زیرا اگر n کمتر باشد تابع منفی خواهد بود
۲) درست است
۳) چون تابع نزولی است پس تابع پیچیدگی نیست

O-notation، Ω – notation و Θ – notation

O-notation

یک کران بالا برای رفتار مجانبی یک تابع مشخص می‌کند: می‌گوید که یک تابع سریعتر از یک نرخ معین رشد نمی‌کند. این نرخ بر اساس جمله با بالاترین مرتبه است.

۱. مثال عملی: کدام جمله واقعاً اهمیت دارد؟

فرض کنید تابع پیچیدگی یک الگوریتم $f(n) = n^2 + 100n + 5000$ باشد. این تابع می‌تواند نشان‌دهنده تعداد کل عملیات‌ها (مثلاً مقایسه‌ها یا جمع‌ها) باشد که الگوریتم برای ورودی با اندازه n انجام می‌دهد.

- n^2 ممکن است از یک حلقه تودرتو بیاید.
- $100n$ ممکن است از یک حلقه ساده دیگر بیاید.
- 5000 ممکن است هزینه‌های ثابت راه‌اندازی (setup cost) باشد.

حالا بیایید ببینیم وقتی n (اندازه ورودی) بزرگ می‌شود، سهم هر یک از این جملات چقدر است.

اندازه ورودی (n)	جمله n^2	جمله $100n$	جمله 5000	$f(n)$ (مجموع)	درصد سهم n^2
۱۰	۱۰۰	۱,۰۰۰	۵,۰۰۰	۶,۱۰۰	۱.۶٪
۱۰۰	۱۰,۰۰۰	۱۰,۰۰۰	۵,۰۰۰	۲۵,۰۰۰	۴۰٪
۱,۰۰۰	۱,۰۰۰,۰۰۰	۱۰۰,۰۰۰	۵,۰۰۰	۱,۱۰۵,۰۰۰	۹۰.۵٪
۱۰,۰۰۰	۱۰۰,۰۰۰,۰۰۰	۱,۰۰۰,۰۰۰	۵,۰۰۰	۱۰۱,۰۰۵,۰۰۰	۹۹.۰٪
۱۰۰,۰۰۰	۱۰,۰۰۰,۰۰۰,۰۰۰	۱۰,۰۰۰,۰۰۰	۵,۰۰۰	۱۰,۰۱۰,۰۰۵,۰۰۰	۹۹.۹٪

همانطور که می‌بینید، وقتی n کوچک است (مثل ۱۰)، جمله ثابت 5000 و جمله خطی $100n$ بخش بزرگی از کل را تشکیل می دهد اما به محض اینکه n به ۱۰۰۰ یا ۱۰,۰۰۰ می‌رسد، جمله n^2 به تنهایی تقریباً **تمام** زمان اجرا را به خود اختصاص می‌دهد.

این همان چیزی است که در تحلیل مجانبی به دنبال آن هستیم. وقتی $n \to \infty$ (به سمت بی‌نهایت میل می‌کند)، جملات $100n$ و 5000 در مقایسه با n^2 آنقدر ناچیز می‌شوند که می‌توانیم آن‌ها را نادیده بگیریم.

این مشاهده ما را مستقیم به مورد بعدی می‌برد:

برای مثال،

$$f(n) = 7n^3 + 100n^2 - 20n + 6$$

برابر با $O(n^3)$ است، زیرا بالاترین جمله مرتبه $7n^3$ است و بنابراین تابع سریعتر از n^3 رشد نمی‌کند.

تابع $f(n)$ همچنین $O(n^5)$، $O(n^6)$ و $O(n^c)$ برای هر ثابت $c \geq 3$ است.

notation-Ω (امگا بزرگ)

یک کران پایین برای رفتار مجانبی یک تابع را مشخص می‌کند: می‌گوید که یک تابع حداقل به اندازه یک نرخ معین رشد می‌کند. در این جا نیز جمله با بالاترین مرتبه در نظر گرفته می‌شود.

برای مثال،

$$f(n) = 7n^3 + 100n^2 - 20n + 6$$

برابر با $\Omega(n^3)$ است، زیرا بالاترین جمله مرتبه، n^3 ، حداقل به اندازه n^3 رشد می‌کند.

تابع $f(n)$ همچنین $\Omega(n^2)$، $\Omega(n)$ و $\Omega(n^c)$ برای هر ثابت $c \leq 3$ است.

notation-Θ

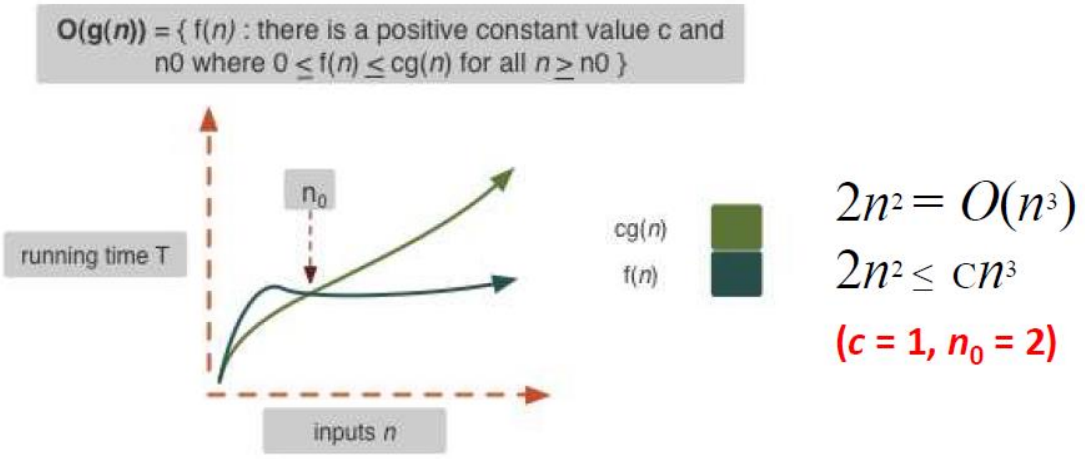
بطور همزمان یک کران بالا و پایین برای رفتار مجانبی یک تابع را مشخص می‌کند. مشخص کننده یک حد و مرز نزدیک از رفتار مجانبی سیستم است. این نماد می‌گوید که یک تابع دقیقاً با یک نرخ معین رشد می‌کند، و مانند نمادهای قبلی بر اساس بالاترین جمله مرتبه خواهد بود.

اگر $O(f(n))$ و هم $\Omega(f(n))$ تابع یکسانی باشد، آنگاه آن تابع، $\Theta(f(n))$ نیز خواهد بود.

بیان فرمول برای مجانب‌ها

نماد O بزرگ

- در صورتی که تابع f(n) و g(n) داده شده باشد، می گوییم که f(n) از مرتبه O(g(n)) است($f(n) = O(g(n))$) اگر ثابت های مثبت c و n_0ای وجود داشته باشد به صورتی که:
- $f(n) \leq cg(n)$ for $n > n_0$



$g(n)$ یک کران بالای مجانبی برای $f(n)$ است. اگر $f(n) \in O(g(n))$، می‌نویسیم

$$f(n) = O(g(n))$$

(به زودی دقیقاً توضیح خواهیم داد که این علامت تساوی به معنی تعلق است).

مثال : چند نمونه از توابع که دارای پیچیدگی $O(n^2)$ هستند

n^2

$n^2 + n$

$n^2 + 1000n$

$1000n^2 + 1000n$

n
$n/1000$
$n^{1.99999}$
$n^2/\lg\lg\lg n$

مثال) نشان دهید تابع پیچیدگی زمانی *2n-9* از مرتبه $O(n^2)$ است؟

پاسخ : این کار را در دو مرحله انجام می‌دهیم نخست پیدا کردن n_0 و سپس پیدا یک ثابت C بطوریکه که برای $n \geq n_0$ داشته باشیم $2n - 9 \leq Cn^2$

ما می‌دانیم که تابع *2n-9* باید مثبت باشد بنابراین کمترین مقدار n_0 که می‌توانیم داشته باشیم حداقل *5* است اگر فرض کنیم n_0=5 است آن‌گاه با توجه به این که $C \geq \frac{2n-9}{n^2}$ پس می‌توان نتیجه گرفت که C باید بزرگتر از $\frac{1}{25}$ باشد

تمرین) نشان دهید تابع پیچیدگی زمانی $13n^2 + 7n$ از مرتبه $O(n^2)$ است؟

پاسخ : ما می‌توانیم مقدار n_0=1 در نظر بگیریم همچنین $13\, n^2 + 7n \leq Cn^2 \;\rightarrow\; 13 + \frac{7}{n} \leq C$ بنابراین مقدار C می‌تواند بزرگتر و مساوی ۲۰ باشد.

تمرین) آیا تابع پیچیدگی زمانی $3n^3 + 7n^2$ از مرتبه $O(n^2)$ است؟

پاسخ)

$$3n^3 + 7n^2 \leq Cn^2$$

$$3n + 7 \leq C$$

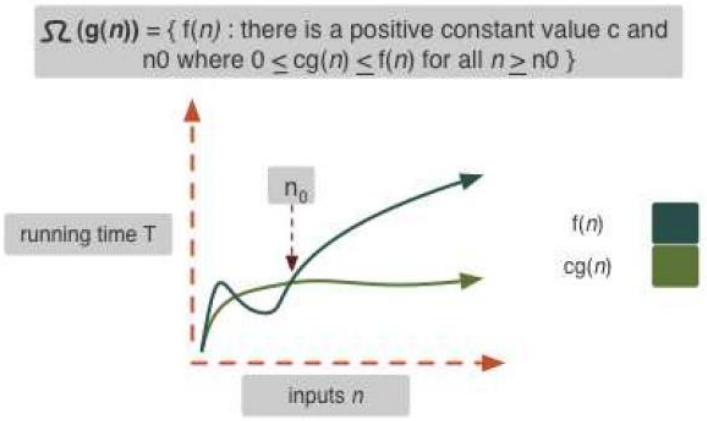
بنابراین ما نمی‌توانیم یک n_0 را طوری پیدا کنیم که برای nهای بزرگتر از n_0 رابطه فوق برقرار باشد.

با توجه به این که مقدار n می‌تواند تا بی‌نهایت ادامه پیدا کند بنابراین ما نمی‌توانیم مقدار ثابتی مانند C پیدا کنیم که رابطه بالا برای آن بر قرار باشد.

نماد امگا (Ω)

در صورتی که تابع f(n) و g(n) داده شده باشد، می گوییم که f(n) از مرتبه $\Omega(g(n))$ است($f(n) = \Omega(g(n))$) اگر ثابت های مثبت c و n_0ای وجود داشته باشد به صورتی که:

$cg(n) \leq f(n)$ for $n > n_0$



$\sqrt{n} = \Omega(\log_2 n)$

$C(\log_2 n) < \sqrt{n}$

$(c = 1, n_0 = 16)$

g(n) یک کران پایین مجانبی برای f(n) است

Example

$\sqrt{n} = \Omega(\lg n)$, with $c = 1$ and $n_0 = 16$.

Examples of functions in $\Omega(n^2)$:

n^2
$n^2 + n$
$n^2 - n$
$1000n^2 + 1000n$
$1000n^2 - 1000n$

Also,

n^3
$n^{2.00001}$
$n^2 \lg \lg \lg n$
2^{2^n}

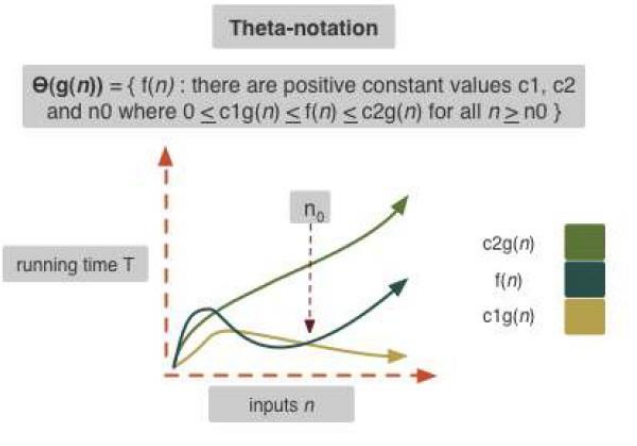
مثال) آیا تابع پیچیدگی زمانی $5n^2 - 11n^2$ از مرتبه Ω (n) است؟

پاسخ) اگر $n_0=1$ باشد، آن‌گاه تابع منفی خواهد شد و این ممکن نیست، بنابراین کوچکترین n_0 که می‌توان در نظر گرفت ۳ است. اگر $n_0=3$ در نظر بگیریم با توجه به این که $C \leq 5n - 11$ باشد پس می‌توان گفت که ثابت C باید کوچکتر یا مساوی از ۴ باشد.

نماد تتا (Θ)

در صورتی که تابع $f(n)$ و $g(n)$ داده شده باشد، می گوییم که $f(n)$ از مرتبه$\Theta(g(n))$ است($f(n) = \Theta(g(n))$) اگر ثابت های مثبت c_1 و c_2 و n_0ای وجود داشته باشد به صورتی که:

$c_1 g(n) \leq f(n) \leq c_2 g(n)$ for $n > n_0$



به عبارت دیگر $\Theta(n)$ مجموعه‌ای از توابع پیچیدگی مانند $f(n)$ است که دارای شرایط زیر باشند:

$$\Theta(g(n)) = f(n) | \exists C_1, C_2, n_0 > 0, C_1 g(n) \leq f(n) \leq C_2 g(n) \forall n \geq n_0$$

$$n^2 + 5n + 7 = \Theta(n^2)$$

$C_1 n^2 \leq n^2+5n+7 \leq C_2 n^2$

$C_1 n^2 \leq n^2+5n+7$

$C_1 = 1$, $n_0=1$

$n^2+5n+7 \leq C_2 n^2$

$C_2 = 13$

قضیه

$f(n) = \Theta(g(n))$ اگر و فقط اگر $f = O(g(n))$ و $f = \Omega(g(n))$.
ضرایب پیشرو و عبارات مرتبه پایین مهم نیستند.

قضیه: اگر $f(n) = a_m n^m + \ldots + a_1 n + n_0$ باشد آنگاه $f(n) = O(n^m)$ خواهد بود.

قضیه: اگر $f(n) = a_m n^m + \ldots + a_1 n + n_0$ باشد آنگاه $f(n) = \Omega(n^m)$ خواهد بود.

قضیه: اگر $f(n) = a_m n^m + \ldots + a_1 n + n_0$ باشد آنگاه $f(n) = \Theta(n^m)$ خواهد بود.

بررسی خواص هم ارزی روی نمادهای مذکور

خاصیت بازتابی :

$$f(n) \in \Theta(f(n))$$

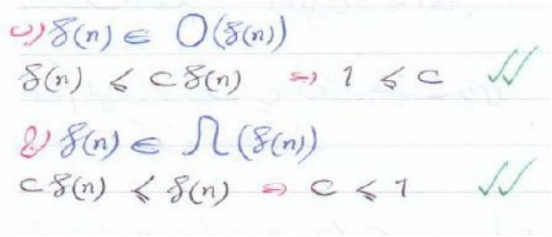
$$C_1 f(n) \leq f(n) \leq C_2 f(n) \rightarrow C_1 \leq 1 \leq C_2$$

بنابراین تعیین C_1 و C_2 مستقل از n_0 است و می توانیم مثلا $C_1=1$ و C_2 را هر عدد بزرگتر یا مساوی ۱ قرار دهیم.

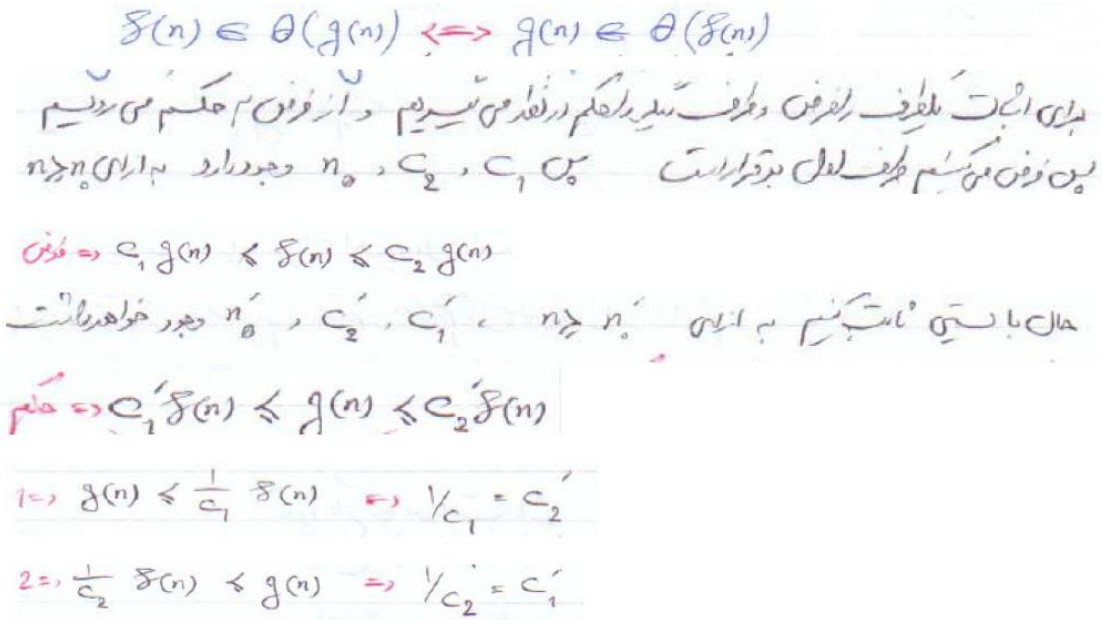
نکته:

$$f(n) \in O(f(n))$$

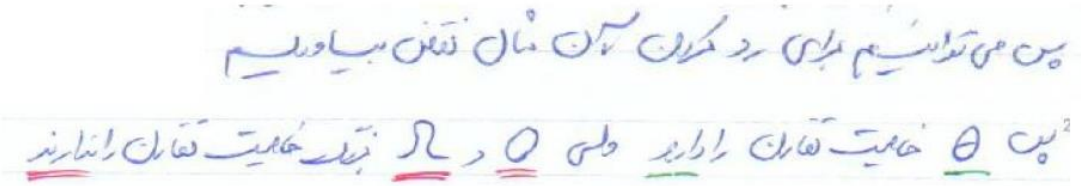
$$f(n) \in \Omega(f(n))$$



خاصیت تقارن

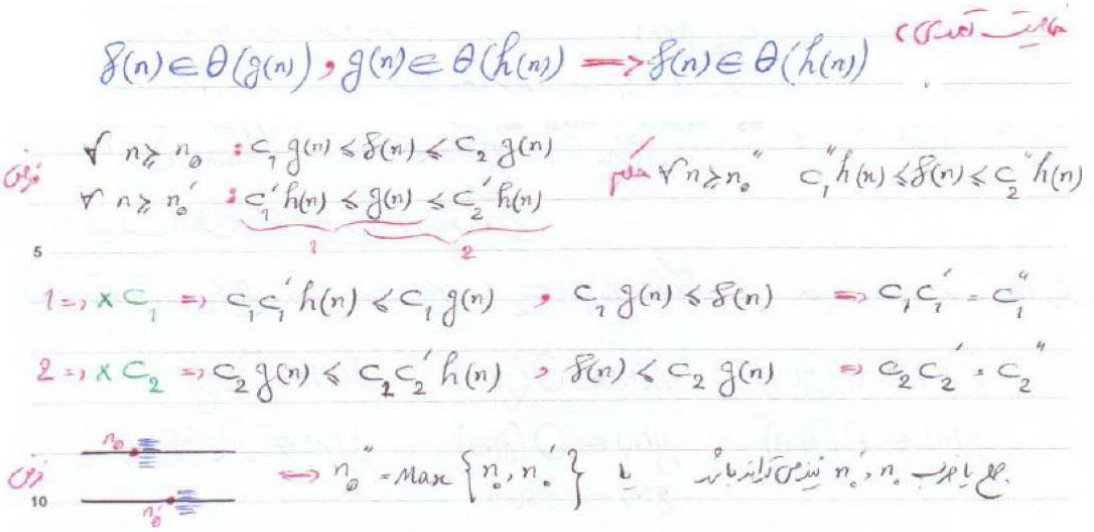


تمرین) آیا $\mathbf{O}$ و $\mathbf{\Omega}$ خاصیت تقارنی دارند؟



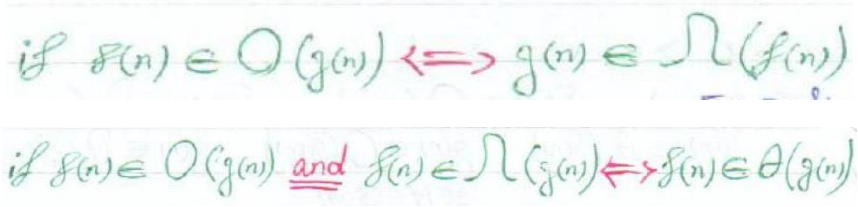
خاصیت تعدی

$$f(n) \in \theta\big(g(n)\big), g(n) \in \theta\big(h(n)\big) \rightarrow f(n) \in \theta\big(h(n)\big)$$



اثبات خاصیت تعدی برای $\mathbf{O}$ و $\mathbf{\Omega}$ نیز به راحتی امکان پذیر است.

نکته :



نمادهای مجانبی و زمان اجرا

باید در اســـتفاده از نماد مجانبی برای مشـــخص کردن زمان اجرا دقت کرد. نماد مجانبی توابع را توصـــیف می‌کند که به نوبه خود زمان‌های اجرا را توصیف می‌کنند. باید دقت کرد که کدام زمان اجرا مشخص شود.

برای مثال، زمان اجرای بدترین حالت برایinsertion sort برابر با$O(n^2)$،$\Omega(n^2)$ و$\Theta(n^2)$ است؛ همه صحیح هستند. ترجیحاً از$\Theta(n^2)$ اســـتفاده می‌کنیم چون دقیق‌ترین اســـت. زمان اجرای بهترین حللت برای insertion sort برابر با $O(n)$،$\Omega(n)$ و$\Theta(n)$ است؛ ترجیحاً$\Theta(n)$.

اما نمی‌توان گفت که زمان اجرای insertion sort برابر با$\Theta(n^2)$ اســت، بدون ذکر **"بدترین حالت"**. حذف این مورد به معنای یک بیان کلی است که همه موارد را پوشش می‌دهد، و insertion sort در همه موارد در زمان$\Theta(n^2)$ اجرا نمی‌شود.

می‌توان این بیان کلی را داشـت که زمان اجرای insertion sort برابر با$O(n^2)$ اسـت، یا اینکه$\Omega(n)$ اسـت، زیرا این زمان‌های اجرای مجانبی برای همه موارد صحیح هستند.

برایmerge sort، زمان اجرای آن در همه موارد برابر با$\Theta(nlgn)$ است، بنابراین می‌توان از ذکر مورد خاص صرف نظر کرد.

خطای رایج: اشـتباه گرفتن نماد O با نمادΘ با اسـتفاده از نماد Θ برای نشـان دادن یک کران مجانبی تنگ. نمادO فقط یک کران بالای مجانبی می‌دهد.

یک الگوریتم که در زمان$\Theta(n)$ اجرا می‌شود، در زمان$O(n^2)$ نیز اجرا می‌شود.

اگر واقعاً منظور یک کران مجانبی تنگ اســت، باید از نمادΘ اســتفاده کرد. از ســاده‌ترین و دقیق‌ترین نماد مجانبی که قابل اعمال اســت اســتفاده کنید. فرض کنید زمان اجرای یک الگوریتم$\frac{3n^2+20n}{4}$ باشـد. بهترین کار این اســت که بگوییم$\Theta(n^2)$. همچنین می‌توان گفت$O(n^3)$ ، اما این کمتر دقیق است. می‌توان گفت$\Theta(3n^2 + 20n)$ ، اما این ترتیب رشد را مبهم می‌کند.

نماد o- کوچک

- در صورتی که تابع f(n) و g(n) داده شده باشد، می گوییم که f(n) از مرتبه o(g(n)) است($f(n) = o(g(n))$) اگر ثابت های مثبت c و n_0ای وجود داشته باشد به صورتی که:

$f(n) < cg(n)$ for $n>n_0$

$o(g(n)) = f(n)$:for all constants $c > 0$,there exists a constant $n_0 > 0$ such that $0 \leq f(n) < cg(n)$

نماد ω (امگا کوچک)

- در صورتی که تابع f(n) و g(n) داده شده باشد، می گوییم که f(n) از مرتبهw(g(n)) است($f(n) = w(g(n))$) اگر ثابت های مثبت c و n_0ای وجود داشته باشد به صورتی که:

$cg(n) < f(n)$ for $n>n_0$

$\omega(g(n)) = f(n)$:for all constants $c > 0$,there exists a constant $n_0 > 0$ such that $0 \leq cg(n) < f(n)$

بررسی مجانب‌ها با استفاده از حد

استفاده از حد (limit) روشی عالی و دقیق برای رسمی کردن ایده‌ی "رفتار مجانبی" است.

ایده اصلی این است: وقتی می‌خواهیم دو الگوریتم را مقایسه کنیم، به رفتار آن‌ها برای ورودی‌های **بسیار بزرگ** (یعنی $n \to \infty$) اهمیت می‌دهیم. حد به ما کمک می‌کند تا ببینیم در این مقیاس، کدام بخش از تابع هزینه (مثلاً n^2 یا100n)" برنده" می‌شود و رشد کلی را تعیین می‌کند.

برای شروع، ما معمولاً دو تابع را با استفاده از حدِ تقسیم آن‌ها مقایسه می‌کنیم. برای دو تابع $f(n)$ (الگوریتم ما) و $g(n)$ (یک تابع محک شناخته شده)، ما به $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ نگاه می‌کنیم.

مقایسه f(n) و g(n) با استفاده از حد

ابزار اصلی ما، همانطور که اشاره کردید، محاسبه‌ی حدِ **نسبت** این دو تابع است، وقتی n به سمت بی‌نهایت می‌رود:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)}$$

در اینجا $f(n)$ تابع پیچیدگی الگوریتم ما است و $g(n)$ یک تابع محک شناخته شده (مثل n یا n^2 یا $\log n$) است که می‌خواهیم $f(n)$ را با آن مقایسه کنیم.

نتیجه‌ی این حد می‌تواند سه حالت کلی داشته باشد. بیایید با مهم‌ترین حالت شروع کنیم:

حالت ۱: حد، یک عدد ثابت مثبت است

فرض کنید حد را محاسبه می‌کنیم و به یک عدد ثابت c می‌رسیم که c بزرگتر از صفر و کوچکتر از بی‌نهایت است.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \quad (\text{بطوری که } 0 < c < \infty)$$

- **معنی این چیست؟** این یعنی وقتی n خیلی بزرگ می‌شود، $f(n)$ تقریباً c برابر $g(n)$ است (مثلاً $f(n) \approx 3g(n)$).

به نظر شما، این در مورد **نرخ رشد** (Rate of Growth) $f(n)$ در مقایسه با $g(n)$ چه می‌گوید؟ آیا یکی بسیار سریع‌تر از دیگری رشد می‌کند، یا اساساً با یک سرعت رشد می‌کنند؟

اینکه حد برابر با یک عدد ثابت (مثل $c = 3$) می‌شود، به ما نمی‌گوید که $f(n)$ **همیشه** کمتر از $3g(n)$ است، بلکه می‌گوید وقتی n **بسیار بسیار بزرگ** می‌شود، نسبت $\frac{f(n)}{g(n)}$ به عدد c *میل می‌کند* (نزدیک و نزدیک‌تر می‌شود).

اما مهم‌ترین نتیجه‌گیری این است: **هر دو تابع با یک نرخ یکسان رشد می‌کنند.**

وقتی n بزرگ است، $f(n)$ تقریباً c برابر $g(n)$ است. در تحلیل مجانبی، ما به ضریب ثابت c اهمیتی نمی‌دهیم (چون سرعت کامپیوترها می‌تواند این ضریب را جبران کند)، اما به این واقعیت که نرخ رشد آن‌ها یکی است، *بسیار* اهمیت می‌دهیم.

وقتی این حالت پیش می‌آید (یعنی حد، یک عدد ثابت **مثبت** است)، ما از نماد Θ (تتا) استفاده می‌کنیم و می‌گوییم:

$$f(n) = \Theta(g(n))$$

(بخوانید: "اِف اِن از مرتبه تِتا-ی جی اِن است").

این نماد Θ به معنای یک **"کران محکم" (Tight Bound)** است. یعنی $g(n)$ هم یک کران بالا و هم یک کران پایین برای $f(n)$ است. به زبان ساده:

- $f(n)$ سریع‌تر از $g(n)$ رشد **نمی‌کند**.
- $f(n)$ کندتر از $g(n)$ رشد **نمی‌کند**.
- آن‌ها با هم، با یک نرخ، رشد می‌کنند.

حالت دوم

وقتی حدِ کسر $\frac{f(n)}{g(n)}$ برابر **صفر** می‌شود، به این معنی است که مخرج ($g(n)$) بسیار بسیار سریع‌تر از صورت ($f(n)$) در حال رشد است. $f(n)$ در مقایسه با $g(n)$ آنقدر کند رشد می‌کند که در بی‌نهایت، نسبت آن‌ها به صفر میل می‌کند.

وقتی این حالت پیش می‌آید (یعنی حد، **صفر** است)، ما می‌گوییم:

$$f(n) = O(g(n))$$

(بخوانید: "اِف اِن از مرتبه اُ-ی جی اِن است").

این نماد O به معنای یک **"کران بالا" (Upper Bound)** است. این می‌گوید که $f(n)$ **حداکثر** به سرعت $g(n)$ رشد می‌کند (و در این مورد خاص که حد صفر شد، می‌دانیم که *قطعاً کندتر* رشد می‌کند).

حالت سوم

اگر حدِ کسر $\frac{f(n)}{g(n)}$ برابر **بی‌نهایت (∞)** شود، به این معنی است که صورت ($f(n)$) بسیار بسیار سریع‌تر از مخرج ($g(n)$) رشد می‌کند.

وقتی این حالت پیش می‌آید (یعنی حد، **بی‌نهایت** است)، ما می‌گوییم:

$$f(n) = \Omega(g(n))$$

(بخوانید: "اِف اِن از مرتبه اُمگا-ی جی اِن است").

این نماد Ω به معنای یک **"کران پایین" (Lower Bound)** است. این می‌گوید که $f(n)$ **حداقل** به سرعت $g(n)$ رشد می‌کند (و در این مورد خاص که حد بی‌نهایت شد، می‌دانیم که *قطعاً سریع‌تر* رشد می‌کند).

خلاصه ۳ حالت مقایسه با حد

بیایید هر سه حالتی را که بررسی کردیم در یک جدول خلاصه کنیم:

نتیجه $\lim_{n\to\infty} \frac{f(n)}{g(n)}$	معنی	نمادگذاری	اصطلاح
0 **(صفر)**	$f(n)$ **کندتر** از $g(n)$ رشد می‌کند.	$f(n) = O(g(n))$	کران بالا
c **(ثابت مثبت)**	$f(n)$ **هم‌نرخ** با $g(n)$ رشد می‌کند.	$f(n) = \Theta(g(n))$	کران محکم
∞ **(بی‌نهایت)**	$f(n)$ **سریع‌تر** از $g(n)$ رشد می‌کند.	$f(n) = \Omega(g(n))$	کران پایین

Export to Sheets

این سه ابزار (O, Θ, Ω) اساس تحلیل مجانبی با استفاده از حد هستند.

. **یک مثال ساده:** بیایید تابع $f(n) = 3n^2 + 100n + 5$ را برداریم و با $g(n) = n^2$ مقایسه‌اش کنیم.

با توجه به جدولی که الان دیدیم، انتظار دارید حد $\lim_{n\to\infty} \frac{3n^2+100n+5}{n^2}$ چه بشود؟

محاسبه حد

بیایید ببینیم چطور ریاضیات، شهود ما را تأیید می‌کند. برای محاسبه این حد، کافی است هر جمله را بر n^2 تقسیم کنیم:

$$\lim_{n\to\infty} \frac{3n^2 + 100n + 5}{n^2} = \lim_{n\to\infty} \left(\frac{3n^2}{n^2} + \frac{100n}{n^2} + \frac{5}{n^2} \right)$$

$$\lim_{n\to\infty} \left(3 + \frac{100}{n} + \frac{5}{n^2} \right)$$

همانطور که n به سمت بی‌نهایت میل می‌کند:

- عبارت $\frac{100}{n}$ به **صفر** میل می‌کند.
- عبارت $\frac{5}{n^2}$ به **صفر** میل می‌کند.

بنابراین، حد نهایی می‌شود:

$$3 + 0 + 0 = 3$$

نتیجه‌گیری

- نتیجه‌ی حد یک عدد ثابت مثبت $(c = 3)$ شد.
- این به این معنی است که $f(n) = 3n^2 + 100n + 5$ و $g(n) = n^2$ دارای **نرخ رشد یکسان** هستند.
- ما به طور رسمی نتیجه می‌گیریم که پیچیدگی الگوریتم ما $\Theta(n^2)$ است.

مقایسه رشدهای مختلف: چرا n^2 بد است؟

ما می‌دانیم که $O(n)$ بهتر از $O(n^2)$ است، اما این تفاوت چقدر بزرگ است؟

بیایید فرض کنیم یک کامپیوتر می‌تواند در هر ثانیه ۱,۰۰۰,۰۰۰,۰۰۰ (یک میلیارد) عملیات انجام دهد و ما اندازه ورودی $n = 100,000$ را داریم.

مرتبه رشد $(g(n))$	تابع پیچیدگی (تقریبی)	تعداد عملیات (تقریبی)	زمان اجرای تخمینی
$O(\log n)$	≈ 17	۱۷ عملیات	بسیار ناچیز
$O(n)$	n	10^5	۰.۰۰۰۱ ثانیه
$O(n \log n)$	$n \log_2 n$	1.7×10^6	۰.۰۰۱۷ ثانیه
$O(n^2)$	n^2	10^{10}	**۱۰ ثانیه**
$O(2^n)$	2^n	10^{30103}	**بیشتر از عمر جهان!**

همانطور که می‌بینید، حتی یک تفاوت ظاهراً کوچک در توان (مثل رفتن از n به n^2) به یک تفاوت **بسیار زیاد** در زمان اجرا منجر می‌شود. این تأثیر به قدری قوی است که هر الگوریتم با مرتبه‌ی $O(n)$ برای n بزرگ، همیشه از هر الگوریتم با مرتبه‌ی $O(n^2)$ سریع‌تر خواهد بود، صرف نظر از ضریب ثابت یا سرعت پردازنده.

جمع‌بندی نهایی

تحلیل مجانبی با استفاده از حد، به ما این تضمین ریاضی را می‌دهد که برای مقیاس‌های بزرگ، فقط بالاترین توان (جمله غالب) مهم است، و این کار مقایسه الگوریتم‌ها را ممکن می‌سازد.

مثال: دو تابع $f(n) = 100n \log n$ و $g(n) = 2n^2$ را با هم مقایسه کنیم؟

گام ۱: تنظیم کسر و ساده‌سازی

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{100n \log n}{2n^2}$$

ابتدا اعداد ثابت و جملات مشترک را حذف می‌کنیم:

$$\lim_{n\to\infty} \frac{50 \log n}{n}$$

گام ۲: محاسبه حد نهایی

ما باید ببینیم وقتی n به بی‌نهایت می‌رود، $\frac{\log n}{n}$ چه اتفاقی برایش می‌افتد.

- صورت ($\log n$) به کندی به بی‌نهایت می‌رود.
- مخرج (n) با سرعت خطی (بسیار سریع‌تر) به بی‌نهایت می‌رود.

در ریاضیات و علوم کامپیوتر، یک اصل اساسی داریم: **رشد هر تابع چندجمله‌ای (Polynomial) همیشه سریع‌تر از رشد هر تابع لگاریتمی (Logarithmic) است.**

چون n یک تابع چندجمله‌ای (با توان ۱) است و $\log n$ یک تابع لگاریتمی است، $\frac{\log n}{n}$ به سمت **صفر** میل می‌کند:

$$\lim_{n\to\infty} \frac{50 \log n}{n} = 50 \times 0 = 0$$

نتیجه‌گیری نهایی

1. **نتیجه حد:** صفر (0) شد.
2. **تفسیر مجانبی:** طبق جدول مقایسه، وقتی حد برابر صفر می‌شود، یعنی $f(n)$ **کندتر** از $g(n)$ رشد می‌کند.
3. **نمادگذاری:** پس می‌توانیم بگوییم $f(n) = 100n \log n$ از مرتبه $O(n^2)$ است. این یعنی یک الگوریتم درجه دو (Quadratic) در نهایت همیشه کندتر از یک الگوریتم لگاریتم-خطی خواهد بود.

مقایسه رشد نمایی(Exponential vs. Polynomial)

تنظیم مسئله: غول‌ها در برابر هم

بیایید دو تابع پیچیدگی زیر را که شما به آن‌ها اشاره کردید، با استفاده از حد با یکدیگر مقایسه کنیم:

1. **تابع چندجمله‌ای (Polynomial):** $g(n) = n^{100}$
   - این یک رشد بسیار سریع است. اگر $n = 2$ باشد، $g(n)$ برابر با 2^{100} خواهد بود.
2. **تابع نمایی (Exponential):** $f(n) = 2^n$
   - این یک رشد نمایی است.

ما می‌خواهیم ببینیم وقتی $n \to \infty$، کدام یک از آن‌ها واقعاً "برنده" می‌شود. برای این کار، حد نسبت آن‌ها را محاسبه می‌کنیم:

$$\lim_{n \to \infty} \frac{n^{100}}{2^n}$$

نکته فریبنده: اگر $n = 1000$ باشد، n^{100} یک عدد با ۳۰۰ صفر است که بسیار بزرگ‌تر از 2^{1000} است. در ابتدا، تابع چندجمله‌ای (صورت کسر) برنده است!

$$\lim_{n \to \infty} \frac{n^{100}}{2^n} = 0$$

چرا رشد نمایی برنده است؟

این نتیجه، یکی از مهم‌ترین اصول در طراحی الگوریتم را تأیید می‌کند: **رشد نمایی (Exponential) همیشه، در نهایت، بر رشد هر تابع چندجمله‌ای (Polynomial) غالب است.**

برای اثبات رسمی این موضوع، می‌توان از **قاعده هوپیتال (L'Hôpital's Rule)** استفاده کرد.

1. **مشتق‌گیری مکرر:** ما باید ۱۰۰ بار از صورت (n^{100}) و مخرج (2^n) مشتق بگیریم (چون توان n برابر ۱۰۰ است).
2. **اتفاقی که برای صورت می‌افتد:** بعد از ۱۰۰ بار مشتق‌گیری، n^{100} به یک عدد ثابت بسیار بزرگ تبدیل می‌شود: $100!$ (فاکتوریل ۱۰۰).
3. **اتفاقی که برای مخرج می‌افتد:** مشتق تابع نمایی 2^n همیشه خودش (با ضرب در ضریب $\ln 2$) باقی می‌ماند. پس مخرج همچنان یک تابع نمایی ($C \cdot 2^n$) باقی می‌ماند.

حد نهایی به این شکل درمی‌آید:

$$\lim_{n\to\infty} \frac{\text{ثابت } (!100)}{C \cdot 2^n}$$

در این کسر، صورت یک عدد ثابت است، اما مخرج (2^n) به سمت **بی‌نهایت** میل می‌کند. بنابراین، کل کسر به **صفر** میل می‌کند.

نتیجه‌گیری نهایی

این بدان معناست که یک الگوریتم با پیچیدگی $O(n^{100})$ (چندجمله‌ای)، برای nهای به اندازه کافی بزرگ، **همیشه** بهتر و سریع‌تر از یک الگوریتم با پیچیدگی $O(2^n)$ (نمایی) خواهد بود.

چند نکته درباره استفاده از نمادهای مجانبی در معادلات

- اگر در معادلات علامت مجانبی یا همان کران استفاده شده باشد آنگاه علامت مساوی (=) که در معادلات استفاده می‌شود بیشتر به معنی تعلق داشتن است
- نمادهای مجانبی نشان دهنده یک مجموعه از تابع‌ها (تعدادی تابع) هستند. مثلا O(n) نماینده و نشان از توابعی است که پیچیدگی آنها خطی می‌باشد. پس اگر در یک معادله ظاهر شود، توابع خطی زیادی را می‌توان جایگزین آن کرد.

برای مثال $2n^2 + 3n + 1 = 2n^2 + \Theta(n)$یعنی تابع مانند f(n) وجود دارد بطوری که

$$2n^2 + 3n + 1 = 2n^2 + f(n)$$

حال ما می توانیم f(n) را هر تابع متعلق به O(n) مانند 3n+1 انتخاب کنیم.

مهم نیست که نماد مجانبی (که یک تابع ناشناس است) در سمت چپ چگونه انتخاب شده‌اند، راهی برای انتخاب یک تابع در سمت راست وجود دارد تا معادله معتبر شود. برای مثال:

$$2n^2 + \Theta(n) = \Theta(n^2)$$

این معادله را اینگونه تفسیر کنید که برای همه توابع

$$f(n) \in \Theta(n)$$

، تابعی

$$g(n) \in \Theta(n^2)$$

وجود دارد به طوری که

$$2n^2 + f(n) = g(n)$$

می‌توان به صورت زنجیره‌ای نوشت: $2n^2 + 3n + 1 = 2n^2 + \Theta(n) = \Theta(n^2)$

تفسیر بیشتر این که:

معادله اول: تابعی $f(n) \in \Theta(n)$ وجود دارد به طوری که $2n^2 + 3n + 1 = 2n^2 + f(n)$

معادله دوم: برای همه

$$g(n) \in \Theta(n)$$

(مانند $f(n)$ که برای معتبر کردن معادله اول استفاده شد)، تابعی $h(n) \in \Theta(n^2)$ وجود دارد به طوری که

$$2n^2 + g(n) = h(n)$$