مدرسان شریف

فصل اول

«پیچیدگی زمانی الگوریتم‌ها»

مقدمه

می‌توان گفت بخش گسترده‌ای از مباحث موجود در علم کامپیوتر بر پایهٔ مطالعه و بررسی الگوریتم‌ها بنا شده است، در نوشتار پیش رو سعی بر این است که به مطالعه، بررسی و مقایسه روش‌های طراحی و آنالیز الگوریتم‌ها پرداخته شود. در زیر تعریف الگوریتم بیان شده است (واژه الگوریتم از نام ریاضی‌دان مشهور ایرانی «الخوارزمی» اقتباس شده است.)

❖ تعریف: الگوریتم مجموعه‌ای از دستورالعمل‌هایی است که اگر با ترتیب خاصی اجرا شود، هدف مشخصی را برآورده می‌کند، یک الگوریتم، دارای پنج خصوصیت زیر می‌باشد.

۱) **ورودی:** یک الگوریتم می‌تواند چندین کمیت را به عنوان ورودی داشته باشد و یا هیچ ورودی نداشته باشد. (یعنی از دنیای خارج هیچ ورودی دریافت نکند.)

۲) **خروجی:** یک الگوریتم باید حداقل یک کمیت به عنوان خروجی داشته باشد.

۳) **عدم ابهام (قطعیت):** هر دستور باید بدون ابهام باشد.

۴) **خاتمه‌پذیری:** هر الگوریتم باید پس از طی مراحل متناهی به پایان برسد.

۵) **قابل اجرا بودن:** هر دستورالعمل موجود در یک الگوریتم باید قابل اجرا باشد. (بتوان آن را با استفاده از کاغذ و قلم و به صورت دستی اجرا کرد.)

نکته۱: تفاوت اصلی الگوریتم با برنامه در این است که الگوریتم باید پایان‌پذیر باشد اما برنامه ممکن است پایان‌پذیر نباشد.

بدست آوردن مرتبه اجرایی الگوریتم

در یک تعریف کلی می‌توان یک الگوریتم را ایده‌ای برای حل یک مسئله دانست، اما برای حل یک مسئله خاص ایده‌های مختلفی وجود دارد به عنوان مثال برای مسئله مرتب‌سازی (Sorting Problem) الگوریتم‌های متفاوتی مانند مرتب‌سازی سریع (Quick Sort)، مرتب‌سازی درجی (Insertion Sort)، مرتب‌سازی حبابی (Bubble Sort)، مرتب‌سازی هرمی (Heap Sort)، مرتب‌سازی درختی (Tree Sort)، مرتب‌سازی ادغامی (Merge Sort)، مرتب‌سازی مبنایی (Radix Sort)، مرتب‌سازی پوسته‌ای (Shell Sort)، مرتب‌سازی شمارشی (Counting Sort)، مرتب‌سازی پیمانه‌ای (Bucket Sort)، مرتب‌سازی پن کیک (Panckake Sort) و ... وجود دارد. حال اگر بخواهیم در یک مسئله که زمان در آن نقش تعیین‌کننده‌ای دارد از یک الگوریتم مرتب‌سازی استفاده کنیم منطقی است که از بین الگوریتم‌های موجود سریعترین را انتخاب نماییم (البته در بسیاری از موارد عملی، مقدار حافظه مورد نیاز الگوریتم نیز اهمیت خاص خود را دارد)، در نتیجه برای مقایسهٔ الگوریتم‌های موجود باید معیارهایی داشته باشیم که مستقل از سخت‌افزار یا نرم‌افزار مورد استفاده، بتواند تعیین کند که در کل کدام الگوریتم سریع‌تر می‌باشد، هدف این بخش بدست آوردن ابزاری برای انجام این کار است.

لازم به ذکر است که الگوریتم‌های مختلف مقادیر مختلفی از حافظه را نیاز دارند و میزان حافظه مورد نیاز نیز یک عامل مهم در کارایی الگوریتم‌ها می‌باشد اما در مقابله بین حافظه و زمان، عامل زمانی تأثیر بیشتری دارد و از اهمیت بالاتری برخوردار است. (هر جا که دو الگوریتم مقایسه می‌شوند در حقیقت از نظر سرعت آنها را مقایسه می‌نماییم، مگر این که صراحتاً چیزی غیر از آن ذکر شود.)

تابع پیچیدگی و گام محاسباتی:

در این قسمت، هدف تعیین معیارهایی است که بتوان با استفاده از آن‌ها، الگوریتم‌ها را مستقل از نرم‌افزار و سخت‌افزار از نظر زمانی تحلیل نمود.

❖ تعریف: کوچکترین واحد زمانی برای اجرای الگوریتم‌ها را یک **step** یا یک **گام محاسباتی** می‌گوییم.

در زیر انواع عباراتی که در یک الگوریتم می‌تواند وجود داشته باشد به همراه تعداد گام‌های لازم برای اجرای آنها بیان شده است:

۱) **Comments** : توضیحاتی می‌باشند که برای فهم بهتر الگوریتم اضافه شده است و هیچ گام زمانی ندارد بنابراین step لازم برای اجرای آنها صفر است.

۲) **Declarations**: عبارات تعیین نوع(type) می‌باشند (نظیر const، char ، int ، . . .). تعداد stepهای لازم برای اجرای این دستورات نیز صفر است.

۳) **Assignments**: این دستورات یک مقدار را به یک متغیر نسبت می‌دهند و شکل کلی آنها به صورت y=f می‌باشد که f می‌تواند یک تابع، یک روال و یا یک عبارت محاسباتی باشد بنابراین تعداد step های آن به f بستگی دارد.

۴) **Iteration statements**: این دستورات شامل حلقه‌های تکرار می‌باشد مانند حلقه‌های زیر:

1) for i =< exp r1 > to < exp r2 > do{body} 2) for j =< exp r1 > down to < exp r2 > do{body}

3) while < exp r1 > do{body} 4) until < exp r1 > Re pat{body}

5) do {body} while < exp r >

تعداد step های لازم برای هر یک از این عبارات به تعداد دفعات تکرار حلقه‌ها و گام‌های بدنه بستگی دارد.

۵) فراخوانی روال و تابع (Call): دستوری که شامل یک فراخوانی است گامی را لازم ندارد اما در عوض باید تعداد گام‌های مورد نیاز برای تابع یا روال فراخوانی شده محاسبه گردد.

❖ **تعریف: تابع پیچیدگی زمانی (Complexity Function)** برای یک الگوریتم تابعی مانند $f(n)$ است که: $f : N \longrightarrow R \geq 0$

که در آن n اندازهٔ ورودی و $f(n)$ برابر با تعداد گام‌های زمانی مورد نیاز الگوریتم برحسب n می‌باشد.

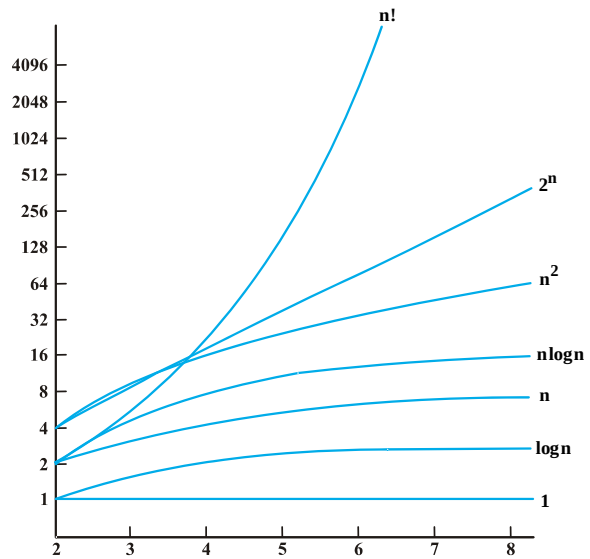
به همین طریق می‌توان تابع پیچیدگی حافظه یک الگوریتم را یک تابع مانند $g(n)$ در نظر گرفت که n اندازهٔ ورودی است و $g(n)$ نشان‌دهندهٔ مقدار حافظه‌ای است که الگوریتم مصرف می‌کند.

در حالت کلی، مقایسه دو الگوریتم به ازای ورودی‌های کوچک چندان معنادار نیست و مقایسه به ازای ورودی‌های به اندازه کافی بزرگ (در حد بی‌نهایت) انجام می‌پذیرد، دلیل این نوع مقایسه را می‌توان در این دانست که تفاوت توابع پیچیدگی به ازای ورودی‌های بزرگ، تعیین‌کننده و حیاتی است. به عنوان مثال در جدول زیر شش تابع پیچیدگی به ازای چند ورودی مختلف مقایسه شده‌اند.

اندازه ورودی	$\lg n$	n	$n \lg n$	n^2	n^3	2^n
1	0.0	1.0	0.0	1.0	1.0	2.0
2	1.0	2.0	2.0	4.0	8.0	4.0
5	2.3	5.0	11.6	25.0	125.5	32.0
10	3.3	10.0	33.2	100.0	1000.0	1024.0
15	3.9	15.0	58.6	225.0	3375.0	32768.0
20	4.3	20.0	86.4	400.0	8000.0	1048576.0
30	4.9	30.0	147.2	900.0	27000.0	1073741824.0
40	5.3	40.0	212.9	1600.0	64000.0	1099511627776.0
50	5.6	50.0	282.2	2500.0	125000.0	1125899906842620.0
60	5.9	60.0	354.4	3600.0	216000.0	1152921504606850000.0
70	6.1	70.0	429.0	4900.0	343000.0	1180591620717410000000.0
80	6.3	80.0	505.8	6400.0	512000.0	1208925819611463000000000.0
90	6.5	90.0	584.3	8100.0	729000.0	1237940039285380000000000000.0
100	6.6	100.0	664.4	10000.0	1000000.0	1267650600228230000000000000000.0

همان‌گونه که مشخص است با بزرگ شدن اندازه ورودی تفاوت بین توابع بیشتر می‌گردد و بنابراین به ازای ورودی‌های بزرگ، انتخاب الگوریتم مناسب برای یک مسئله خاص از اهمیت بسزایی برخوردار است.

شکل زیر، نمودار برخی از توابع پیچیدگی را نشان می‌دهد:



دقت کنید که این شکل فقط به ازای ورودی‌های کوچک ترسیم گردیده اما باز هم تفاوت رشد توابع مشخص است، حال اگر ورودی‌های بزرگ در نظر گرفته شوند آنگاه حتی در صورتی که توابع با رشد کم مانند $n\log n$ در ضرایب بزرگی نیز ضرب شوند باز هم رشد آن‌ها کمتر از توابعی مانند 2^n یا $n!$ خواهد بود.

اگر فرض کنیم توابع بررسی شده در شکل فوق، مربوط به الگوریتم‌های موجود برای یک مسئله خاص باشند و بر روی یک کامپیوتر سریع اجرا شوند به طوری که این کامپیوتر در هر ثانیه یک میلیارد عملیات پایه مربوط به مسئله مورد بررسی را انجام دهد در این صورت زمان مورد نیاز برای اجرای الگوریتم‌های مختلف این مسئله بر روی کامپیوتر بیان شده به صورت زیر خواهد بود:

اندازه ورودی	تابع پیچیدگی					
n	$\text{Log } n$	n	$n \text{ Log } n$	n^2	2^n	$n!$
10	$3\times10^{-9}s$	$10^{-8}s$	$3\times10^{-8}s$	$10^{-7}s$	$10^{-6}s$	$3\times10^{-3}s$
10^2	$7\times10^{-9}s$	$10^{-7}s$	$7\times10^{-7}s$	$10^{-5}s$	بیش از صد میلیارد قرن	بیش از صد هزار میلیارد قرن
10^3	$1/0\times10^{-8}s$	$10^{-6}s$	$1\times10^{-5}s$	$10^{-3}s$	بیش از صد هزار میلیارد قرن	
10^4	$1/3\times10^{-8}s$	$10^{-5}s$	$1\times10^{-4}s$	$10^{-1}s$		
10^5	$1/7\times10^{-8}s$	$10^{-4}s$	$2\times10^{-3}s$	$10s$		
10^6	$2\times10^{-8}s$	$10^{-3}s$	$2\times10^{-2}s$	$17\,min$		

همان‌گونه که می‌بینید، مدت زمان مورد نیاز برای توابع نمایی و فاکتوریل حتی به ازای ورودی‌های نه چندان بزرگ چند هزار برابر عمر کره زمین می‌باشد! (عمر کره زمین برابر ۴/۵۴ میلیارد سال و عمر حیات حدود یک میلیارد سال تخمین زده می‌شود). اگر طی سال‌های آینده سرعت کامپیوترها چند هزار برابر نیز شود باز هم توابع نمایی و فاکتوریل، توابعی مهار نشدنی می‌باشند و می‌توان گفت الگوریتم‌های چند جمله‌ای بیشترین سود را از پیشرفت در سرعت سخت‌افزار خواهند برد.

نماد O (Oی بزرگ یا Big O)

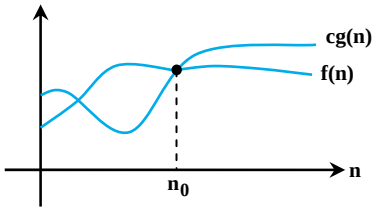
❖ **تعریف:** اگر $f(n)$ و $g(n)$ دو تابع پیچیدگی باشد آنگاه می‌نویسیم $f(n)\in O(g(n))$ (می‌خوانیم $f(n)$ عضو اوی بزرگ $g(n)$ است) هرگاه داشته باشیم:

$$\exists c>0, \exists n_0\geq 0 \ \text{ s.t. } (\forall n\geq n_0 : f(n)\leq cg(n)) \ ; \ c\in \mathbb{R}^+, n_0\in\mathbb{N}$$

در حقیقت $g(n)$ یک حد مجانبی بالا روی $f(n)$ می‌گذارد، به این معنی که برای n های به اندازهٔ کافی بزرگ $f(n)$ همواره از $cg(n)$ کوچکتر است و بنابراین می‌توانیم بگوییم در این حالت رشد $g(n)$ بیشتر از $f(n)$ خواهد بود و در نتیجه $f(n)$ سریع‌تر از $g(n)$ می‌باشد.

تذکر۱: در برخی منابع رابطه $f(n)\in O(g(n))$ به صورت $f(n)=O(g(n))$ استفاده می‌گردد.

نکته۲: در حقیقت $O(g(n))$ مجموعهٔ تمام توابعی مانند $f(n)$ است که $f(n)\in O(g(n))$ شکل روبرو رابطهٔ بین $f(n)$, $g(n)$ را در حالتی که $f(n)\in O(g(n))$ باشد نمایش می‌دهد.



مثال۱: اگر $f(n)=3n+14$ و $g(n)=n+\log n$ آنگاه: $f(n)\in O(g(n))$

اگر داشته باشیم $f(n)\in O(g(n))$ آنگاه یکی از دو حالت زیر اتفاق می‌افتد.

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\alpha>0 \qquad \alpha\in R \qquad \text{یا} \qquad \lim_{n\to\infty}\frac{f(n)}{g(n)}=0$$

مثال۲: اگر $g(n)=2^n$ و $f(n)=n^3$ آنگاه: $f(n)\in O(g(n))$

اما $g(n)\notin O(f(n))$ زیرا داریم:

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

نماد Ω (امگای بزرگ)

همان‌گونه که بیان شد با استفاده از نماد "O" یک حد بالا برای رشد تابع مورد بررسی تعیین می‌گردد، با استفاده از نماد Ω، یک حد پایین بر روی رشد تابع مورد نظر قرار داده می‌شود.

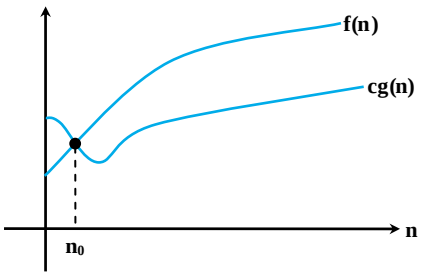
❖ **تعریف:** می‌نویسیم $f(n)\in\Omega(g(n))$ (می‌خوانیم $f(n)$ امگای بزرگ $g(n)$ است) هرگاه داشته باشیم:

$$\exists c>0\ ,\ \exists n_0>0 \quad (\ \forall n\geq n_0:\ f(n)\geq cg(n))\ ;\ c\in\mathbb{R}^+,\ n_0\in\mathbb{N}$$

در حقیقت $g(n)$ یک حد مجانبی پایین روی $f(n)$ می‌گذارد به این معنی که برای nهای به اندازهٔ کافی بزرگ $g(n)$ همواره از $f(n)$ کوچکتر است و بنابراین می‌توانیم بگوییم در این حالت رشد $g(n)$ از $f(n)$ کندتر است و در نتیجه $g(n)$ سریع‌تر از $f(n)$ خواهد بود.

تذکر۲: به جای $f(n)\in\Omega(g(n))$ گاهی از رابطه $f(n)=\Omega(g(n))$ استفاده می‌شود.

نکته۳: در حقیقت $\Omega(g(n))$ مجموعه تمام توابعی مانند $f(n)$ است که $f(n)\in\Omega(g(n))$ شکل روبرو رابطهٔ بین $g(n)$ و $f(n)$ را در حالتی که $f(n)\in\Omega(g(n))$ است نشان می‌دهد.



اگر داشته باشیم $f(n)\in\Omega(g(n))$ آنگاه یکی از دو حالت زیر رخ می‌دهد.

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\infty \qquad \text{یا} \qquad \lim_{n\to\infty}\frac{f(n)}{g(n)}=\alpha>0 \qquad \alpha\in R$$

مثال۳: اگر $f(n)=n^3$ و $g(n)=n^2+n$ آنگاه داریم: $\mathbf{f(n)\in\Omega(g(n))}$

اما $\mathbf{g(n)\notin\Omega(f(n))}$ زیرا: $\mathbf{\lim_{n\to\infty}\frac{g(n)}{f(n)}=0}$

مثال۴: اگر $f(n)=3n+14$, $g(n)=n+\log n$ آنگاه: $\mathbf{g(n)\in\Omega(f(n))}$ و همچنین $\mathbf{f(n)\in\Omega(g(n))}$

نماد θ

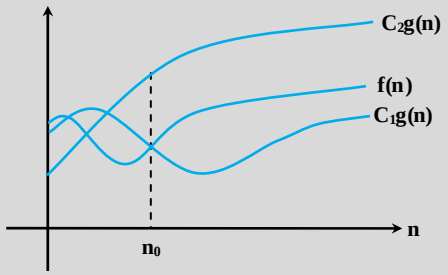
نماد مجانبی θ نسبت به دو نماد O و Ω می‌تواند دقیق‌تر رشد کلی یک تابع را تعیین نماید.

❖ **تعریف:** می‌نویسیم $f(n)\in\theta(g(n))$ (می‌خوانیم $f(n)$ تتای $g(n)$ است) هرگاه داشته باشیم:

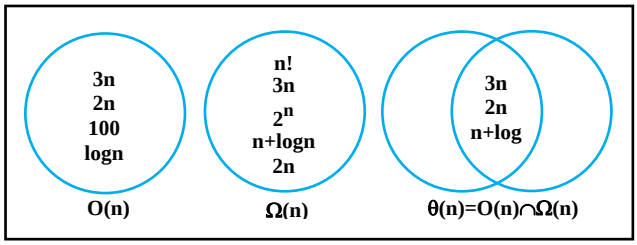
$$\exists c_1>0,\exists c_2>0\ ,\exists n_0>0\ (\forall n\geq n_0: c_1g(n)\leq f(n)\leq c_2g(n))\ ;\ c_1,c_2\in\mathbb{R}^+,n_0\in\mathbb{N}$$

در حقیقت در این حالت رشد $f(n)$ و $g(n)$ یکسان است.

نکته۴: می‌توان گفت $\theta(g(n))$ مجموعه تمام توابع مانند $f(n)$ می‌باشد که $f(n)\in\theta(g(n))$ ، شکل روبه‌رو رابطهٔ بین $f(n)$ و $g(n)$ را در حالتی که $f(n)\in\theta(g(n))$ باشد نمایش می‌دهد.



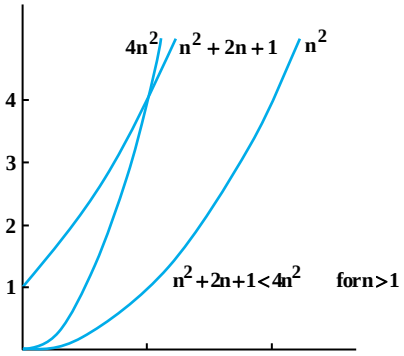
در شکل زیر چند تابع نمونه که در $\theta(n)$، $O(n)$ و $\Omega(n)$ قرار می‌گیرند قابل مشاهده است:



نکته۵: اگر داشته باشیم $f(n)\in\theta(g(n))$ آنگاه خواهیم داشت: $\lim_{n\to\infty}\frac{f(n)}{g(n)}=\alpha$

که α یک عدد ثابت است.

به عنوان مثال، شکل زیر نشان‌دهنده این است که:



با فرض $c_1=1$ و $c_2=4$ و $n_\circ=2$

داریم: $n^2 \le n^2+2n+1 \le 4n^2$

$n^2+2n+1 \in \theta(n^2)$

مثال۵: کدام گزینه درست است؟

۱) $n\log n \in \theta(n\log^2 n)$ ۲) $\frac{n}{\log n} \in O(\log n)$ ۳) $\frac{n}{\log n} \in \theta(n)$ ۴) $\frac{n}{\log n} \in O(n)$

☑ **پاسخ:** گزینه «۴» دقت کنید که:

$$\lim_{n\to\infty} \frac{\frac{n}{\log n}}{n} = 0 \Rightarrow \frac{n}{\log n} \in O(n)\,,\ \frac{n}{\log n} \notin \theta(n)$$

و همچنین:

$$\lim_{n\to\infty} \frac{n\log n}{n\log^2 n} = 0 \Rightarrow n\log n \in O(n\log^2 n)\,,\ n\log n \notin \theta(n\log^2 n)$$

نکته۶: اگر داشته باشیم $f(n) \in \theta(g(n))$ آنگاه $g(n) \in \theta(f(n))$ نیز برقرار است.

ممکن است $f(n) \in O(g(n))$ باشد اما $g(n) \notin O(f(n))$

ممکن است $f(n) \in \Omega(g(n))$ باشد اما $g(n) \notin \Omega(f(n))$

با توجه به تعریف نمادهای مجانبی نتیجه می‌گیریم که $\theta(g(n)) = O(g(n)) \cap \Omega(g(n))$ بنابراین داریم:

$$f(n) \in \theta(g(n)) \Leftrightarrow f(n) \in O(g(n)) \ \&\ f(n) \in \Omega(g(n))$$

نکته۷: برای دو تابع $f(n)$ و $g(n)$ خواهیم داشت:

$$f(n) \in \Omega(g(n)) \Leftrightarrow g(n) \in O(f(n))$$

مثال۶: اگر f و g دو تابع دلخواه به صورت $f,g : N \to R^+$ باشند به‌طوریکه $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ ، آنگاه کدام‌یک از گزینه‌های زیر درست است؟

۱) $f(n) \in O(g(n)), g(n) \notin \Omega(f(n))$ ۲) $g(n) \in O(f(n)), f(n) \in O(g(n))$

۳) $f(n) \in \Omega(g(n)), f(n) \notin \theta(g(n))$ ۴) $f(n) \in \theta(g(n)), g(n) \notin \Omega(f(n))$

☑ **پاسخ:** گزینه «۳» با توجه به این‌که حد خارج قسمت دو تابع برابر بی‌نهایت است، بنابراین بین این دو تابع، رابطه θ برقرار نیست و با توجه به حد بدست آمده خواهیم داشت:

$$f(n) \in \Omega(g(n))\,,\ g(n) \in O(f(n)) \quad ؛ \quad f(n) \notin \theta(g(n))\,,\ g(n) \notin \theta(f(n))$$

مثال۷: کدام گزینه زیر غلط است؟

۱) $10^n + n^{20} \notin \theta(n^n)$ ۲) $\log_2^n \in \theta(\log_{10}^n)$ ۳) $(\log_2^n)! \in \Omega(n!)$ ۴) $4n^3+5n^2+7n \in \Omega(\log_2^n)$

☑ **پاسخ:** گزینه «۳» شکل صحیح گزینه سوم به صورت $(\log_2^n)! \in O(n!)$ و یا $n! \in \Omega(\log_2^n)!)$ می‌باشد.

نکته۸: در بررسی توابع پیچیدگی در مورد توابع لگاریتمی پایهٔ لگاریتم اهمیتی ندارد. $\log_a^n \in \theta(\log_b^n)\ ;\ a>1\,,\ b>1$

مثال۸: از بین چهار مورد داده شده کدام‌یک صحیح می‌باشند؟

الف ـ $n! = O(n^n)$ ب ـ $\frac{n^2}{\log n} = \theta(n^2)$ ج ـ $n^2\log n = \theta(n^2)$ د ـ $n^{2^n} + 6(2^n) = \theta(n^{2^n})$

۱) الف و ب ۲) همه موارد ۳) الف و د ۴) الف

☑ **پاسخ:** گزینه «۳» توابع داده شده را به ترتیب رشد می‌توانیم به صورت زیر دسته‌بندی کنیم:

$$\frac{n^2}{\log n} < n^2 < n^2\log n < n! < n^n < \begin{matrix} n^{2^n} \\ n^{2^n}+6(2^n) \end{matrix}$$

نکته۹: برای هر $a>0$, $b>1$ داریم: $n^a \in O(b^n)$

اگر $b>a>1$ آنگاه: $a^n \in O(b^n)$

نکته۱۰: اگر $f(n)$ یک چند جمله‌ای از درجهٔ k به صورت روبرو باشد: $f(n)=a_k n^k + a_{k-1}n^{k-1}+\cdots+a_1 n+a_\circ$

آنگاه خواهیم داشت: $f(n)\in\theta(n^k)$

تذکر۳: پیچیدگی تمام توابعی که دارای تعداد ثابتی گام باشند با $O(1)$ نمایش داده می‌شود.

توجه: اگر زمان اجرای یک الگوریتم از مرتبه $\theta(f(n))$ باشد آنگاه زمان اجرای آن الگوریتم در بدترین حالت از مرتبه $O(f(n))$ و در بهترین حالت از مرتبه $\Omega(f(n))$ می‌باشد.

با توجه به تقریب استرلینگ برای $n!$ خواهیم داشت: $(\log n!)\in\theta(n\log n)$

نکته۱۱: ممکن است به ازای دو تابع $f(n)$ و $g(n)$ هیچ یک از سه حالت $f(n)\in O(g(n))$ یا $f(n)\in\Omega(g(n))$ یا $f(n)\in\theta(g(n))$ برقرار نباشد.

به عنوان مثال اگر $g(n)=\log n$, $f(n)=\begin{cases} n & \text{به ازای } n \text{ زوج} \\ 1 & \text{به ازای } n \text{ فرد} \end{cases}$ باشد، در نتیجه $f(n)\notin O(g(n))$ و $f(n)\notin\Omega(g(n))$ و $f(n)\notin\theta(g(n))$.

مثال۹: کدام گزینه درست است؟ (ε یک عدد مثبت کوچک می‌باشد.)

۱) $n^3\log n = O(n^{3+\varepsilon})$ ۲) $\sqrt{n}=O(\log n)$ ۳) $n^{1+\varepsilon}=O(n\log n)$ ۴) $n^2=O(\frac{n^2}{\log n})$

☑ **پاسخ:** گزینه «۱» دقت کنید که هر توان n مانند n^ε که $\varepsilon>0$ رشد بیشتری نسبت به $\log n$ دارد.

مثال۱۰: کدام گزینه درست است؟

۱) $n^{\frac{1}{10}}\in\Omega(\log n)$ ۲) $\log(n!)\in O(\log n)$

۳) $n^2+4n-10\in O(n\log n)$ ۴) $1^n+2^n+3^n+\cdots+n^n\in O(n^n)$

☑ **پاسخ:** گزینه «۱» همان‌گونه که قبلاً بیان شد، هر توانی از n مانند n^ε (که $\varepsilon>0$) نسبت به توابع لگاریتمی از رشد بیشتری برخوردار است.

مثال۱۱: در کدام گزینه توابع از چپ به راست براساس ترتیب رشد نوشته شده‌اند؟

۱) $n\log^n,(1.006)^n,n^2$ ۲) $n(\log n)^2,(1.006)^n,n^2$ ۳) $n^2,n(\log n)^2,(1.006)^n$ ۴) $n(\log n)^2,n^2,(1.006)^n$

☑ **پاسخ:** گزینه «۴» دقت کنید که رشد توابع نمایی از تمام توابع لگاریتمی و چندجمله‌ای و حاصل‌ضرب آن‌ها بیشتر است.

نماد o (o کوچک)

اگر $f(n)$ و $g(n)$ دو تابع پیچیدگی باشند آنگاه $f(n)\in o(g(n))$، هرگاه داشته باشیم:

$$\forall c>0 \quad \exists n_0 \; s\cdot t\cdot \; (\forall n\geq n_0 : f(n)\leq cg(n)\; ; c\in R\; , n_0\in N)$$

دقت کنید تفاوت تعریف o کوچک و O بزرگ در این است که ثابت c در تعریف Oی بزرگ دارای سور وجودی می‌باشد (به این معنی که کافیست نامساوی مورد نظر به ازای یک ثابت دلخواه برقرار باشد)؛ اما در تعریف oی کوچک، ثابت c دارای سور عمومی است؛ یعنی نامساوی مربوطه باید به ازای هر ثابت دلخواه c برقرار باشد.

به صورت غیردقیق می‌توانیم Oی بزرگ و oی کوچک را به شکل زیر بیان کنیم:

$f(n) \in O(g(n)) \Leftrightarrow f(n) \le g(n)$ ؛ $f(n) \in o(g(n)) \Leftrightarrow f(n) < g(n)$

نکته۱۲: اگر داشته باشیم $f(n) \in o(g(n))$، آنگاه خواهیم داشت: $f(n) \notin \Omega(g(n))$ اما $f(n) \in O(g(n))$

مثال۱۲: کدامیک از گزینه‌های زیر رابطه بین دو تابع را با استفاده از نماد oی کوچک به درستی نشان می‌دهد؟

۱) $n^2 + n \in o(\frac{n^2}{5})$

۲) $n^2 \log n + n^2 \in o(n \log^2 n)$

۳) $n^2 \log^2 n + n^2 \in o(\frac{n^3}{5})$

۴) هیچ‌کدام

☑ **پاسخ:** گزینه «۳» گزینه‌های اول و دوم نادرست می‌باشند.

توجه: مجموعه $o(g(n))$ و مجموعه $O(g(n)) - \Omega(g(n))$ با یکدیگر برابر نمی‌باشند بلکه رابطه زیر بین آنها برقرار است:

$o(g(n)) \subseteq O(g(n)) - \Omega(g(n))$

زیرا توابعی وجود دارند که در $O(g(n)) - \Omega(g(n))$ هستند، اما در $o(g(n))$ نیستند.

با توجه به تعریف نماد oی کوچک می‌توانیم بگوییم این نماد، یک حد مجانبی بالای اکید برای یک تابع پیچیدگی تعیین می‌کند.

$f(n) \in o(g(n))$ بدین معناست که تابع پیچیدگی $f(n)$ برای ورودی‌های به اندازه کافی بزرگ سریع‌تر از $g(n)$ می‌باشد.

اگر $f(n) \in o(g(n))$ آنگاه به صورت حدی رابطه روبرو بین آنها برقرار است: $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

نماد ω (امگای کوچک)

❖ **تعریف:** اگر $f(n)$ و $g(n)$ دو تابع پیچیدگی باشند، آنگاه می‌نویسیم $f(n) \in \omega(g(n))$ هرگاه داشته باشیم:

$\forall c > 0 \ \exists n_0 \ s.t. \ (\forall n \ge n_0 : f(n) \ge c g(n) \ ; \ c \in R, n_0 \in N)$

به صورت غیردقیق می‌توانیم ارتباط دو تابع با نمادهای Ω و ω را به صورت زیر بیان کنیم:

$f(n) \in \Omega(g(n)) \Leftrightarrow f(n) \ge g(n)$ ؛ $f(n) \in \omega(g(n)) \Leftrightarrow f(n) > g(n)$

اگر داشته باشیم $f(n) \in \omega(g(n))$، آنگاه: $f(n) \notin O(g(n))$ اما $f(n) \in \Omega(g(n))$

مثال۱۳: کدامیک از گزینه‌های زیر درست است؟

۱) $n^2 + n \in \omega(\frac{n^2}{5})$

۲) $n^2 \log n + n^2 \in \omega(n \log^2 n)$

۳) $n^2 \log n + n^2 \in \omega(n^3)$

۴) هیچ‌کدام

☑ **پاسخ:** گزینه «۲» زیرا داریم:

$n^2 \log n + n^2 > n \log^2 n \Rightarrow$ گزینه ۲ درست است.

$n^2 + n \cong n^2 \Rightarrow n^2 \in \theta(\frac{n^2}{5}), n^2 + n \notin \omega(\frac{n^2}{5})$

$n^2 \log n + n^2 < n^3 \Rightarrow n^2 \log n + n^2 \in o(n^3), n^2 \log + n^2 \notin \omega(n^3)$

توجه: بین نمادهای O, ω, Ω رابطه روبرو برقرار است: $\omega(g(n)) \subseteq \Omega(g(n)) - O(g(n))$

مثال۱۴: از بین روابط زیر چند رابطه درست است؟

۱- $n^2 \log n + n \in \omega(\log n)$

۲- $n \log^2 n + n^2 + n \in \omega(n^2 \log_5^n)$

۳- $n + \log n + 10 \in \omega(\log n)$

۱) ۱ ۲) ۲ ۳) ۳ ۴) هیچ‌کدام

☑ **پاسخ:** گزینه «۲» رابطه‌های ۱ و ۳ درست می‌باشند.

$n + \log n + 10 > \log n$ $n^2 \log n + n > \log n$

اما رابطه ۲ برقرار نیست زیرا داریم:

$n \log^2 n + n^2 + n < n^2 \log_5^n \Rightarrow n \log^2 n + n^2 + n \notin \omega(n^2 \log_5^n)$

توجه: تفاوتی ظریف (شبیه به تفاوت بین o‌ی کوچک و O‌ی بزرگ) بین Ω و ω وجود دارد؛ به این‌معنی که نامساوی موجود در رابطه ω باید به ازای هر ثابت c برقرار باشد، اما نامساوی موجود در رابطه Ω کافیست به ازای یک ثابت c برقرار باشد.

نماد ω برای قرار دادن یک حد مجانبی پایین اکید برای یک تابع پیچیدگی استفاده می‌شود.

نکته۱۳: اگر داشته باشیم $f(n)\in\omega(g(n))$ آنگاه رابطه حدی مقابل بین آنها برقرار است: $\lim_{n\to\infty}=\frac{f(n)}{g(n)}=\infty$

نکته۱۴: برای یک تابع پیچیدگی مانند $g(n)$ داریم: $o(g(n))\cap\omega(g(n))=\phi$

یعنی هیچ تابعی مانند $f(n)$ وجود ندارد که $f(n)\in\omega(g(n))$ و همچنین $f(n)\in o(g(n))$ باشد.

$f(n)\in\omega(g(n))$ بدین معناست که تابع پیچیدگی $f(n)$ به ازای ورودی‌های به اندازه کافی بزرگ بدتر از $g(n)$ می‌باشد.

نکته۱۵: برای دو تابع $f(n)$ و $g(n)$ با توجه به نمادهای o و ω رابطه مقابل برقرار است: $f(n)\in\omega(g(n))\Leftrightarrow g(n)\in o(f(n))$

نکات تکمیلی نمادهای مجانبی و مثال‌های بیشتر

تذکر۴: در تمام نکات زیر $f(n)$ و $g(n)$ توابع پیچیدگی در نظر گرفته می‌شوند.

نکته۱۶: بین $f(n)$ و $g(n)$ همواره روابط زیر برقرار است:

$i)f(n)+g(n)\in\theta(\max(f(n),g(n)))\quad;\quad ii)f(n)+g(n)\in O(\max(f(n),g(n)))\quad;\quad iii)f(n)+g(n)\in\Omega(\min(f(n),g(n)))$

$iv)\ f(n)+g(n)\in\Omega(\max(f(n),g(n)))$

مثال۱۵: اگر $g(n)=n+10$ و $f(n)=n^2\log n$ آنگاه درستی روابط بالا را در مورد این توابع بررسی کنید.

$\underbrace{n^2\log n}_{f(n)}+\underbrace{n+10}_{g(n)}\in O\ \underbrace{(n^2\log n)}_{\max(f(n),g(n))}\qquad \underbrace{n^2\log n}_{f(n)}+\underbrace{n+10}_{g(n)}\in\Omega\ \underbrace{(n+10)}_{\min(f(n),g(n)}\qquad \underbrace{n^2\log\ n}_{f(n)}+\underbrace{n+10}_{g(n)}\in\Omega\ \underbrace{(n^2\log n)}_{\max(f(n),g(n))}$

و همچنین:

$\underbrace{n^2\log n}_{f(n)}+\underbrace{n+10}_{g(n)}\in\theta\ \underbrace{(n^2\log n)}_{\max(f(n),g(n))}$

در این مثال همچنین می‌توان گفت:

$\underbrace{n^2\log n}_{f(n)}+\underbrace{n+10}_{g(n)}\in\omega\ \underbrace{(n+10)}_{\min(f(n),g(n))}\qquad \underbrace{n^2\log n}_{f(n)}+\underbrace{n+10}_{g(n)}\notin o\ \underbrace{(n^2\log n)}_{\max(f(n),g(x))}$

اما اگر قرار دهیم $g(n)=n\log n$ و $f(n)=3n\log n$ آنگاه داریم:

$\underbrace{3n\log n}_{f(n)}+\underbrace{n\log n}_{g(n)}\notin\omega\ \underbrace{(n\log n)}_{\min(f(n),g(n))}$

همان‌طور که قبلاً بیان شد، می‌توانیم نمادهای مجانبی را به صورت روابط منطقی موجود در اعداد حقیقی تفسیر کنیم. یعنی:

$f(n)\in O(g(n))\Leftrightarrow f(n)\le g(n)$

$f(n)\in o(g(n))\Leftrightarrow f(n)<g(n)$

$f(n)\in\Omega(g(n))\Leftrightarrow f(n)\ge g(n)$

$f(n)\in\omega(g(n))\Leftrightarrow f(n)>g(n)$

$f(n)\in\theta(g(n))\Leftrightarrow f(n)\cong g(n)$

بنابراین می‌توان استدلال‌های زیر را در مورد این نمادها بیان کرد:

$f(n)\in O(g(n)),g(n)\in O(h(n))\Rightarrow f(n)\in O(h(n))$

$f(n)\in o(g(n)),g(n)\in o(h(n))\Rightarrow f(n)\in o(h(n))$

$f(n)\in\Omega(g(n)),g(n)\in\Omega(h(n))\Rightarrow f(n)\in\Omega(h(n))$

$f(n)\in\omega(g(n)),g(n)\in\omega(h(n))\Rightarrow f(n)\in\omega(h(n))$

$f(n)\in\theta(g(n)),g(n)\in\theta(h(n))\Rightarrow f(n)\in\theta(h(n))$

مثال ۱۶: کدام یک از روابط زیر برقرار است؟

۱) $n^{O(1)} < n^5$ ۲) $\log^5 n > n^{0.4}$ ۳) $3^{\log n} < n^2 \log n$ ۴) $\sqrt{n} < \log^2 n$

☑ **پاسخ:** گزینه «۳» تابع $3^{\log n}$ از مرتبه $\theta(n^{\log_2^3})$ می‌باشد. در مورد گزینه اول دقت کنید که $O(1)$ می‌تواند شامل هر عدد ثابت بزرگتر از پنج نیز باشد و بنابراین گزینه اول نادرست است.

مثال ۱۷: از بین روابط زیر کدام موارد صحیح می‌باشد؟

الف) $1+2+3+\cdots+n \in \theta(n^2)$ ب) $1+4+9+\cdots+n^2 \in \theta(n^3)$ ج) $1+8+27+\cdots+n^3 \in \theta(n^5)$

۱) الف و ب ۲) الف و ج ۳) ب و ج ۴) فقط الف

☑ **پاسخ:** گزینه «۱» به روابط زیر دقت نمایید:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \in \theta(n^2) \qquad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \in \theta(n^3) \qquad \sum i^3 = [\frac{n(n+1)}{2}]^2 \in \theta(n^4)$$

نکته زیر حالت کلی مرتبه زمانی توابع مشابه توابع مثال فوق را نشان می‌دهد:

نکته ۱۷: در حالت کلی تابع $\sum_{i=1}^{n} (i)^{\ell}$ از مرتبه $\theta(n^{\ell+1})$ می‌باشد.

تعیین مرتبه پیچیدگی الگوریتم‌ها

هدف از بخش پیش‌رو، تعیین مرتبه زمانی مربوط به شبه کد یک الگوریتم می‌باشد. قسمت اصلی مرتبه زمانی هر قطعه برنامه، به حلقه‌های موجود در آن قطعه برنامه مربوط می‌شود؛ زیرا هزینه زمانی بقیه دستورات (به غیر از فراخوانی‌ها) یک مقدار ثابت است و بنابراین می‌تواند باعث ایجاد یک عدد ثابت در مرتبه زمانی کل الگوریتم شود. برای به دست آوردن مرتبه زمانی حلقه‌ها، حالت‌های زیر را در نظر می‌گیریم: (در تمام حالات زیر، عدد n به عنوان اندازه ورودی مسئله در نظر گرفته شده است).

i ـ حلقه‌های تکرار به فرم کلی زیر که در آنها حد پایین و بالای حلقه، مستقل از اندازه ورودی است.

```
For i := L1  TO  U1  DO
  Begin
    ⋮
    دستورات بدنه
    ⋮
  End
```

و یا

```
For i := U1  Down to  L1 DO
  Begin
    ⋮
    دستورات بدنه
    ⋮
  End
```

(U_1 و L_1 اعداد ثابت می‌باشند)

مرتبه زمانی این‌گونه حلقه‌ها یک مقدار ثابت است که در تعیین پیچیدگی تأثیری ندارد، بنابراین دستورات موجود در بدنه حلقه، مرتبه زمانی را تعیین می‌کنند. در نتیجه مرتبه اجرایی این گونه حلقه‌ها برابر است با $\theta(b)$ که b مرتبه اجرایی دستورات بدنه می‌باشد.

ii: حلقه‌های تکراری که تکرار آنها به اندازه ورودی وابسته است و به شکل کلی زیر می‌باشند.

```
For  i := L1 To   n    Do
  Begin
    ⋮
    دستورات بدنه
    ⋮
  End
```

یا

```
For  i := n   Downto   L1   Do
  Begin
    ⋮
    دستورات بدنه
    ⋮
  End
```

(L_1 یک عدد ثابت است)

مرتبه اجرایی این گونه حلقه‌ها برابر است با $\theta(n \times b)$ که b مرتبه اجرایی دستورات بدنه می‌باشد.

مثال ۱۸: مرتبه اجرایی قطعه برنامه زیر چیست؟

```
For(i = 5; i < n; i++)
  For(j = 5; j < 100; j++)
    S = S + i * j;
```

۱) $\theta(n)$ ۲) $\theta(n \log \frac{n}{5})$ ۳) $\theta(n^2)$ ۴) هیچ کدام

☑ **پاسخ:** گزینه «۱» حلقه for خارجی، دارای مرتبه $\theta(n)$ و حلقه داخلی دارای مرتبه اجرایی $\theta(1)$ می‌باشد؛ زیرا حد بالا و پایین آن به ورودی وابسته نیست و تعداد دفعات تکرار آن باعث ایجاد یک عدد ثابت می‌شود.

مثال۱۹: مرتبه اجرایی قطعه برنامه زیر کدام است؟

```
For(i = 3; i < n^2; i++)
    For(j = 100; j < n; j++)
        For(k = 10; k < 20; k++);
```

۱) $\theta(n^2)$ ۲) $\theta(n)$ ۳) $\theta(n^3)$ ۴) $\theta(30n^2)$

پاسخ: گزینه «۳» حلقه For i دارای مرتبه $\theta(n^2)$ است و حلقه For j دارای مرتبه $\theta(n)$ می‌باشد؛ اما در حلقه For k حد بالا و پایین به n وابسته نمی‌باشد، در نتیجه مرتبه آن یک عدد ثابت خواهد بود. بنابراین مرتبه کل قطعه برنامه برابر است با: $\theta(n \times n^2) = \theta(n^3)$

iii: حلقه‌های تکرار تو در تو که شمارنده حلقه داخلی به حلقه خارجی وابسته است.

در این گونه حلقه‌ها اگر حد بالا یا پایین حلقه خارجی به اندازه ورودی وابسته باشد، آنگاه تعداد حلقه‌ها تعیین‌کننده پیچیدگی الگوریتم می‌باشد. اما اگر حلقه خارجی به اندازه ورودی وابسته نباشد، آنگاه مرتبه اجرایی یک مقدار ثابت خواهد شد (در برخی حالت‌ها نیاز به بررسی گام به گام حلقه‌ها می‌باشد).

مثال۲۰: مرتبه پیچیدگی قطعه برنامه زیر چیست؟

```
For(i = 100; i < 500; i++)
    For(j = n - 5; j > 10; j--)
        For(k = 10; k < j; k++)
            S = k + i + j * S;
```

۱) $\theta(n)$
۲) $\theta(n^2)$
۳) $\theta(n^3)$
۴) با توجه به ورودی هر یک می‌تواند درست باشد.

پاسخ: گزینه «۲» حلقه For i دارای زمان ثابت و حلقه For j دارای زمان $\theta(n)$ است. حلقه For k به حلقه For j وابسته می‌باشد؛ بنابراین زمان این دو حلقه برابر $\theta(n^2)$ خواهد بود.

مثال۲۱: مرتبه زمانی قطعه برنامه زیر کدام است؟

```
For(i = 5; i < n - 10; i++)
    For(j = i; j > 1; j--)
        For(k = 1; k < j; k++)
        {
            S = S + k + j;
            S = S * 2;
        }
```

۱) $\theta(n^2)$
۲) $\theta(n^3)$
۳) $\theta(n \log n)$
۴) $\theta(\frac{n(n+1)}{2})$

پاسخ: گزینه «۲» با توجه به توضیحات ارائه شده، مرتبه زمانی برابر $\theta(n \times n \times n) = \theta(n^3)$ می‌باشد.

iv- حلقه‌های تکراری که در آنها شمارنده در یک عدد ثابت ضرب می‌شود و یا بر یک عدد ثابت تقسیم می‌گردد.

اگر حلقه افزایشی باشد که در هر تکرار آن، شمارنده در یک عدد ثابت مانند c ضرب شود و یا اگر حلقه کاهشی باشد که شمارنده هر بار بر یک عدد ثابت مانند c تقسیم شود، آنگاه در صورتی که حد بالا و پایین این حلقه به ورودی وابسته باشد؛ مرتبه الگوریتم، لگاریتمی خواهد بود و مبنای لگاریتم عدد ثابت c می‌باشد یعنی از مرتبه $\theta(\log_c^n)$ خواهد بود.

مثال۲۲: مرتبه اجرایی قطعه برنامه زیر را به دست آورید.

```
For(i = 1; i < n; i++)
    For(j = 100; j < i; j++)
    {
        k = 1;
        while(k < n)
        {
            S = S + k;
            S = S * i * j;
            k = k * 2;
        }
    }
```

۱) $\theta(n^3)$
۲) $\theta(n^2)$
۳) $\theta(n^3 \log_2^n)$
۴) $\theta(n^2 \log_2^n)$

پاسخ: گزینه «۴» مرتبه اجرایی حلقه For i برابر $\theta(n)$ می‌باشد. با توجه به این‌که شمارنده حلقه داخلی به شمارنده حلقه خارجی وابسته است، بنابراین مرتبه این دو حلقه برابر $\theta(n^2)$ می‌باشد. اما حلقه while در داخل این دو حلقه قرار دارد و یک حلقه افزایشی می‌باشد که شمارنده آن هر بار در عدد 2 ضرب می‌شود، بنابراین مرتبه حلقه while برابر $\theta(\log_2^n)$ خواهد بود و در نتیجه مرتبه کل قطعه برنامه برابر است با: $\theta(n^2 \log_2^n)$