عنوان بخش

طراحی الگوریتم

کارایی، تحلیل و مرتبه الگوریتم‌ها– بازگشتی

■ اگر الگوریتمی با پیچیدگی زمانی $T_1(n) = 100n$ داشته باشیم و الگوریتم دیگری با پیچیدگی زمانی $T_2(n) = 0/01n^2$ داشته باشیم. الگوریتم اول به‌ازای nهای خیلی بزرگ از الگوریتم دوم سریع‌تر است. به‌عبارت دقیق‌تر به ازای $n > 10000$ الگوریتم اول از دوم سریع‌تر است:

$$100n < 0/01n^2 \rightarrow n > 10000$$

■ گفته می‌شود رشد زمانی الگوریتم ۲ از ۱ بیشتر است. می‌نویسیم: $T_1(n) = \theta(n)$ و $T_2(n) = \theta(n^2)$.

به این معنی که زمان الگوریتم ۱ به n واسته است (خطی) و زمان الگوریتم ۲ به n^2 وابسته است. حال اگر الگوریتمی داشته باشیم که زمان آن از رابطه $2n^2 + 10n + 20$ به‌دست آید، این الگوریتم از مرتبه $\theta(n^2)$ است زیرا به ازای nهای بزرگ، جملات $10n + 20$ در مقابل n^2 قابل صرف‌نظر کردن هستند. برای بیان رشد زمان اجرای الگوریتم‌ها از نمادهای O, θ و ω, o, Ω استفاده می‌شود.

نماد O (big-O):

■ **گوییم تابع** $f(n)$ **از مرتبه** $O(g(n))$ **است و می‌نویسیم** $f(n) = O(g(n))$ **یا** $f(n) \in O(g(n))$ **اگر و فقط اگر:** $\exists c, n_0 > 0, \forall n \geq n_0 : 0 \leq f(n) \leq cg(n)$

✓ **مثال: آیا تابع** $f(n) = 2n^2 + 3n + 4$ **از مرتبه** $O(n^3)$ **هست؟**

✓ **پاسخ:** بله به ازای همان c و n_0 مثال نامساوی $2n^2 + 3n + 4 \leq 3n^2$ صحیح است. می‌توان نشان داد که تابع $f(n) = 2n^2 + 3n + 4$ از مرتبه $O(n)$ نیست.

■ **نتیجه:** نمادO، کران بالای تابع را مشخص می‌کند و معمولاً برای بیان بدترین حالت زمان اجرا، استفاده می‌شود.

نماد Ω:

■ **گوییم تابع** $f(n)$ **از مرتبه** $\Omega(g(n))$ **است و می‌نویسیم** $f(n) = \Omega(g(n))$ **یا** $f(n) \in \Omega(g(n))$ **اگر و فقط اگر:** $\exists c, n_0 > 0, \forall n \geq n_0 : 0 \leq cg(n) \leq f(n)$

✓ **مثال: آیا تابع** $f(n) = 2n^2 + 3n + 4$ **از مرتبه** $\Omega(n)$ **هست؟**

✓ **پاسخ:** بله. به‌ازای $c = 1$ و $n_0 = 1$ مثال قبل نامساوی $cg(n) \leq f(n)$ صحیح است. می‌توان نشان داد تابع $f(n) = 2n^2 + 3n + 4$ از مرتبه $\Omega(n^3)$ نیست.

✓ **نتیجه:** نماد Ω کران پایین تابع را مشخص می‌کند و معمولاً برای بیان بهترین حالت زمان اجرا، استفاده می‌شود.

نماد θ:

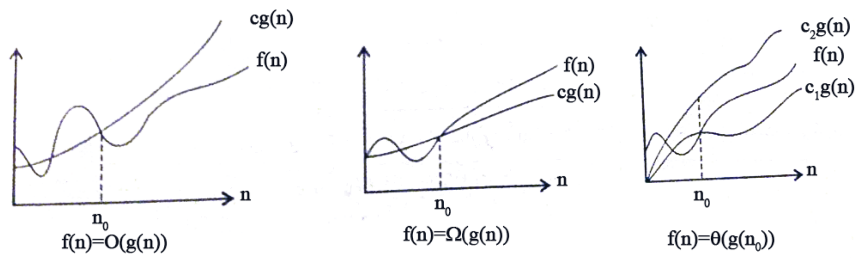
✓ **گوییم تابع** $f(n)$ **از مرتبه** $\theta(g(n))$ **است و می‌نویسیم** $f(n)=\theta(g(n))$ **یا** $f(n)\in\theta(g(n))$ **اگر و فقط اگر:** $\exists c_۱, c_۲\ n_۰>۰, \forall n\geq n_۰ : ۰\leq c_۱g(n)\leq f(n)\leq c_۲g(n)$

✓ **مثال: می‌توان نشان داد تابع** $f(n)=۲n^۲+۳n+۴$ **از مرتبه** $f(n)=O(g(n))$ **هست ولی از مرتبه** $\theta(n)$ **و** $\theta(n^۳)$ **نیست.**

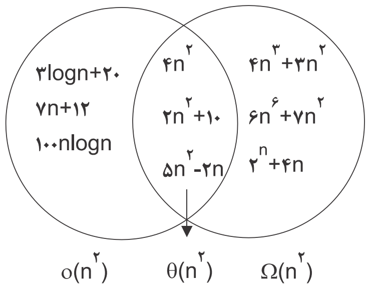
✓ **نتیجه:** نماد θ درجه خود تابع را مشخص می‌کند و معمولاً برای بیان حالت متوسط زمان اجرا، استفاده می‌شود.

✓ **تذکر:** در بسیاری از متون، از نماد O به معنی θ استفاده می‌شود.

■ **شکل زیر نمادها را با هم مقایسه کرده است:**



■ **شکل زیر کلاس مختلف نمادها را با هم مقایسه کرده است:**



☑ **قضـــیه:** بـــرای توابـــع $f(n)$ و $g(n)$. $f(n)=\theta(g(n))$ اگـــر و فقـــط اگـــر $f(n)=O(g(n))$ و $f(n)=\Omega(g(n))$.

نماد o (Small-o)**:**

■ **گوییم تابع** $f(n)$ **از مرتبه** $o(g(n))$ **است و می‌نویسیم** $f(n)=o(g(n))$ **یا** $f(n)\in o(g(n))$ **اگر و فقط اگر:** $\forall c>۰, \exists n_۰>۰, \forall n>n_۰ : ۰\leq f(n)<cg(n)$

✓ **به‌عنوان مثال** $۲n=o(n^۲)$ **ولی** $۲n^۲\neq o(n^۲)$

نکات برتر

نماد **o** مشابه **O** می‌باشد. تفاوت اصلی این است که وقتی می‌نویسـیم $f(n)=O(g(n))$، آنگـاه نامساوی $۰\leq f(n)\leq cg(n)$ برای برخـی مقـادیر ثابـت $c>۰$ صـادق اسـت. ولـی وقتـی کـه می‌نویسیم $۰\leq f(n)<cg(n)$ برای تمام مقادیر ثابت $c>۰$ صادق است.

نماد ω:

■ **گوییم تابع $f(n)$ از مرتبه $\omega(g(n))$ است اگر:**

$$\forall c > 0, \exists n_0 > 0, \forall n \geq n_0 : 0 \leq cg(n) < f(n)$$

به‌عنوان مثال $\frac{n^2}{2} = \omega(n)$ ولی $\frac{n^2}{2} \neq \omega(n^2)$.

نکات برتر:

✓ اگر درجه $f(n)$ را **a** و درجه $g(n)$ را **b** فرض کنیم:

$$f(n) = \theta(g(n)) \leftrightarrow a = b$$

$$f(n) = O(g(n)) \leftrightarrow a \leq b$$

$$f(n) = \Omega(g(n)) \leftrightarrow a \geq b$$

$$f(n) = o(g(n)) \leftrightarrow a < b$$

$$f(n) = \omega(g(n)) \leftrightarrow a > b$$

✓ خواص زیر برای نمادهای مجانبی برقرار است:

a) تعدی **(transitivity)**:

$$f(n) = \theta(g(n)) \text{ and } g(n) = \theta(h(n)) \Rightarrow f(n) = \theta(h(n))$$

$$f(n) = O(g(n)) \text{ and } g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$$

$$f(n) = \Omega(g(n)) \text{ and } g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$$

$$f(n) = o(g(n)) \text{ and } g(n) = o(h(n)) \Rightarrow f(n) = o(h(n))$$

$$f(n) = \omega(g(n)) \text{ and } g(n) = \omega(h(n)) \Rightarrow f(n) = \omega(h(n))$$

b) بازتابی **(Reflexivity)**:

$$f(n) = \theta(f(n))$$

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

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

c) تقارنی **(Symmetry)**: $f(n) = \theta(g(n)) \Leftrightarrow g(n) = \theta(f(n))$

d) تقارنی ترانهاده **(Transpose Symmetry)**:

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

$$f(n) = o(g(n)) \Leftrightarrow g(n) = \omega(f(n))$$

✓ می‌دانیم به ازای هر دو عدد حقیقی **a** و **b**، یکی از شرایط $a > b$ و $a = b$ و $a < b$ برقرار است. ولی برای دو تابع $f(n)$ و $g(n)$ ممکن است نه $f(n) = O(g(n))$ و نه $f(n) = \Omega(g(n))$ هیچ‌یک برقرار نباشد مثلاً توابع $f(n) = n$ و $g(n) = n^{1+\sin n}$. زیرا $1 + \text{Sinn}$ بین ۰ و ۲ نوسان می‌کند.

✓ زمان اجرای الگوریتمی $\theta(g(n))$ است اگر و فقط اگر زمان اجرا در بدترین حالت $O(g(n))$ و زمان اجرا در بهترین حالت $\Omega(g(n))$ باشد.

✓ اگر $\lim_{n\to\infty}\frac{f(n)}{g(n)} = ۰$ آنگاه $f(n)=o(g(n))$ و برعکس

✓ اگر $\lim_{n\to\infty}\frac{f(n)}{g(n)} = \infty$ آنگاه $f(n)=\omega(g(n))$ و برعکس

✓ اگر $\lim_{n\to\infty}\frac{f(n)}{g(n)} = c \neq ۰$ آنگاه $f(n)=\theta(g(n))$ و برعکس

✓ می‌توان ثابت کرد $Logn! = \theta(nLogn)$

✓ می‌توان ثابت کرد که $a,b>۱, Log_a^n = \theta(Log_b^n)$

✓ $\sum_{i=۱}^{n} i^p = \theta(n^{p+۱})$

بازگشتی :

■ الگوریتم بازگشتی، الگوریتمی است که خودش را فراخوانی کند. بـه‌عنـوان مثـال محاسـبه $f(n)=n!$ را می‌توان به‌صورت بازگشتی نوشت:

$$f(n)=\begin{cases} ۱ & ۰\le n\le ۱ \\ n*f(n-۱) & n>۱ \end{cases}$$

■ اگر $T(n)$ را زمان اجرای $f(n)$ فرض کنیم واضح است که $T(۱)=۱$ زیرا به ازای $n=۱$ فقط یک بـار تابع f اجرا می‌شود. $T(۲)=۲$ زیرا به ازای $n=۲$ دو بـار تـابع f اجـرا مـی‌شـود ($f(۲)$ و بعـد $f(۱)$) و می‌توان نشان داد که $T(n)=n$، یعنی زمان تابع بازگشتی فاکتوریل از مرتبه $\theta(n)$ است.

✓ **مثال: تابع بازگشتی محاسبه مجموع $f(n)=۱+۲+...+n$ به شکل زیر ا ست:**

$$f(n)=\begin{cases} n+f(n-۱) & n>۱ \\ ۱ & n=۱ \end{cases}$$

زمان اجرای این تابع از چه مرتبه‌ای است.

✓ **پاسخ:** اگر $T(n)$ را زمان اجرای تابع **f** درنظر بگیریم می‌توان نوشت:

$$T(n)=\begin{cases} T(n-۱)+۱ & n>۱ \\ ۱ & n=۱ \end{cases}$$

و بنابراین $T(n)=\theta(n)$.

قضیه Master**:**

■ **برای یافتن مرتبه اجرایی روابط بازگشتی به شکل** $T(n)=aT\left(\frac{n}{b}\right)+\theta(n^k$ **به شرطی کـه** $a\ge$ **و** $b>۱$ **ثابت هستند و درضمن** $\frac{n}{b}$ **ممکن است به صورت کف** $\left\lfloor\frac{n}{b}\right\rfloor$ **یا سقف** $\left\lceil\frac{n}{b}\right\rceil$ **ظاهر شود به شکل ذیل عمل می‌کنیم:**

$$T(n)=\begin{cases}\theta(n^k) & \text{if} \quad a<b^k\\ \theta(n^k\, \mathrm{Log}_۲^n) & \text{if} \quad a=b^k\\ \theta(n^{\mathrm{Log}_b^a}) & \text{if} \quad a>b^k\end{cases}$$

✓ **مثال: تعداد انتقال‌های مسئله برج‌های هانوی از رابطه** $T(n)=۲T(n-۱)+۱$ **به‌دست می‌آید که طبق نکته فوق** $T(n)=\theta\left(۲^{\frac{n}{۱}}\right)=\theta(۲^n)$.

نکات برتر

در رابطه بازگشتی $T(۲)=aT(n-۱)+bT(n-۲)$ با شرایط اولیه $T(۰)$ و $T(۱)$. می‌توان معادله مشخصه تشیکل داد که معادله مشخصه آن $x^۲-ax-b=۰$ می‌باشد اگر این معادله دو جواب متمایز $x_۱$ و $x_۲$ داشته باشد آنگاه $T(n)=c_۱x_۱^n+c_۲x_۲^n$ که با اعمال شرایط اولیه مقادیر $c_۱$ و $c_۲$ محاسبه می‌شوند. اگر معادله مشخصه جواب مضاعف $x_۱=x_۲$ داشته باشد آنگاه $T(n)=(c_۱+c_۲n)x_۱^n$ خواهد بود.

روش‌های تقسیم و غلبه

☑ **الگوریتم ۱: الگوریتم انتزاعی روش تقسیم و غلبه**

```
Algorithm DandC (Low , high)
    If small (low , high) then return g (Low , high)
        else {
            mid = Divide (Low , Hight)
            return combine (DandC (Low , mid) DandC (mid + 1 , high))
        }
```

در این الگوریتم تابع Small مشخص می‌کند آیا مسأله به اندازه کافی کوچک هست که بتوان آن را بدون شکستن حل کرد یا خیر. اگر جواب مثبت باشد، تابع g فراخوانی می‌شود در غیر این صورت عمل تقسیم انجام می‌شود. در ضمن اگر فرض کنیم ورودی در آرایه $A[۱..n]$ باشد، فراخوانی باید به صورت $\mathrm{DandC}(۱ \text{ و } n)$ صورت گیرد.

■ **اگر زمان اجرای الگوریتم combine و الگوریتم** $f(n).۱$ **فرض شود آنگاه زمان اجرای الگوریتم، T(n) برابر است با:**

$$T(n)=\begin{cases}۲T\left(\frac{n}{۲}\right)+f(n) & \text{برای } n \text{های بزرگ}\\ g(n) & \text{برای } n \text{های کوچک}\end{cases}$$

در $T(n)$ فرض شده است که n توانی از ۲ است و همچنین در هر بار تقسیم، آرایه دقیقاً نصف می‌شود و همچنین $g(n)$ زمان تابع g می‌باشد.