مدرسان شریف

فصل اول

«الگوریتم (مشخصات، تجزیه و تحلیل)»

مقدمه

در این فصل به مطالعه الگوریتم‌ها می‌پردازیم و مطالبی را در مورد مشخصات یک الگوریتم و نحوهٔ تجزیه و تحلیل الگوریتم‌ها، بیان می‌کنیم. برای حل بسیاری از مسائل کامپیوتری، الگوریتم‌هایی وجود دارد. طراحی یک الگوریتم کارآمد، نقش مهمی در توسعهٔ سیستم‌های کامپیوتری دارد. بنابراین قبل از شروع، باید مفاهیمی را به طور کامل شرح دهیم.

❖ تعریف: الگوریتم، (Algorithm) مجموعه‌ای متناهی از دستورالعمل‌ها است که اگر دنبال و اجرا شوند، هدف خاصی را برآورده می‌کند. به علاوه هر الگوریتم باید دارای مشخصات زیر باشد:

۱ـ ورودی : هر الگوریتم می‌تواند هیچ یا چندین کمیت به عنوان ورودی داشته باشد که از محیط خارج تأمین می‌شود.

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

۳ـ قطعیت : هر دستورالعمل باید واضح و بدون ابهام باشد.

۴ـ محدود بودن : الگوریتم باید پس از طی مراحل معینی خاتمه یابد.

۵ـ کارآیی : هر دستورالعمل باید به گونه‌ای باشد که شخص با استفاده از قلم و کاغذ بتواند آن را به طور دستی اجرا کند. در واقع فقط قطعیت کافی نیست، بلکه اجرای هر دستورالعمل باید انجام‌پذیر باشد.

در علم کامپیوتر بین الگوریتم و برنامه تفاوت‌هایی وجود دارد، مثلاً یک برنامه لزوماً پایان‌پذیر نیست (شرط چهارم بالا). به عنوان نمونه، سیستم‌عامل، برنامه‌ای است که دائماً در یک حلقه انتظار است تا برنامهٔ بعدی وارد شود. این‌گونه برنامه‌ها تنها در صورتی به پایان می‌رسند که یا سیستم از کار بیفتد و یا دچار خرابی شود.

تحلیل الگوریتم‌ها

برای یک مسئله خاص، الگوریتم‌های مختلفی وجود دارد که باید با توجه به منابع موجود، مناسب‌ترین آن‌ها انتخاب شود. هنگامی که سخن از تحلیل الگوریتم‌ها به میان می‌آید، باید حیطه بحث مشخص شود. کارایی هر الگوریتم از جهات متفاوتی قابل بررسی است.

برای ارزیابی کارایی یک الگوریتم، دو معیار دارای اهمیت است :

۱) زمان اجرای الگوریتم ۲) حافظه لازم در زمان اجرای الگوریتم

« در بسیاری مواقع، معیار اصلی در انتخاب الگوریتم‌ها برای حل مسأله‌ای واحد، مدت زمان اجرای آن‌ها است.»

زمان مورد نظر به معنای تعداد ثانیه (یا هر واحد زمانی دیگر) نیست، بلکه تقریبی است از تعداد عملیاتی که یک الگوریتم در هنگام اجرا انجام می‌دهد. تعداد عملیات الگوریتم، رابطه مستقیم با زمان اجرای آن دارد. بنابراین، از زمان برای بیان پیچیدگی یک الگوریتم استفاده می‌شود.

محاسبه تعداد ثانیه‌هایی که یک الگوریتم برای اجرا بر روی یک کامپیوتر نیاز دارد، سود چندانی برای ما ندارد؛ زیرا مدت زمان واقعی اجرای یک الگوریتم در هر کامپیوتر، به عوامل گوناگونی مانند معماری ماشین، کامپایلر، سرعت پردازشگر و موارد نظیر آن بستگی دارد، در حالی که ما نیازمند معیاری مستقل از جزئیات سخت‌افزار و نرم‌افزار کامپیوتر اجراکننده الگوریتم هستیم.

❖ تعریف: **عمل مبنایی:** قسمتی از الگوریتم که عمدهٔ زمان اجرای الگوریتم را به خود اختصاص می‌دهد عمل مبنایی گفته می‌شود.

❖ تعریف: **نرخ رشد الگوریتم :** « میزان افزایش تعداد عملیات الگوریتم بر اثر افزایش اندازه ورودی »

مدت زمانی که الگوریتم‌ها برای اجرا نیاز دارند، در اکثر موارد به اندازه ورودی وابسته است. برای نمونه، مرتب‌سازی 2000 عنصر، در مقایسه با مرتب‌سازی 20 عنصر به زمان بیشتری نیاز دارد. بنابراین می‌توان انتظار داشت که زمان اجرای یک الگوریتم، تابعی از اندازه داده‌ی ورودی باشد. اما به دست‌آوردن چنین تابعی کار ساده‌ای نیست، زیرا محاسبه دقیق تعداد عملیات انجام شده در یک الگوریتم، به ازای اندازه‌ی ورودی، فرایند پیچیده‌ای است. به همین دلیل، در محاسبه‌ی زمان اجرای الگوریتم‌ها، از نوعی تقریب استفاده می‌شود که با استفاده از این تقریب در نهایت تخمینی از تعداد عملیات انجام شده به ازای اندازه ورودی تابع، به دست می‌آید.

اگر n اندازه ورودی باشد، زمان اجرای الگوریتم بر حسب n را معمولاً با T(n) نمایش می‌دهیم.

تحلیل پیچیدگی زمانی تعیین گام‌های یک برنامه

یک گام از برنامه (program step) از نظر دستوری یا معنایی، قسمت با معنی یک برنامه است که دارای زمان اجرایی مستقل از مشخصه‌های موردی است. تعداد گام‌های هر دستور از برنامه، به ماهیت آن دستور باز می‌گردد. در این بحث سعی ما بر این است که انواع مختلفی از دستور را که در هر برنامه می‌تواند وجود داشته باشد، بررسی کنیم.

۱ـ عبارات توضیحی (Comments) : توضیحات، عبارات غیراجرایی هستند و تعداد گام‌های آن صفر است.

۲ـ عبارت تعیین نوع (Declarative statements) : این طبقه شامل عباراتی نظیر معرفی متغیرها و ثابت‌ها و انواع جدید و... می‌باشد. تعداد گام‌های این عبارات صفر است، چون اجرایی نیستند.

۳ـ عبارات و دستورات انتساب (Expressions and assignment statements) : تعداد تکرار بیشتر دستورات، یک است. به جز عباراتی که شامل فراخوانی توابع می‌باشد. در این حالت باید هزینهٔ لازم برای اجرای توابع را محاسبه کنیم.

۴ـ عبارات تکرار (Iteration statements) : این دسته از عبارات، شامل عبارت for، while است. شمارش گام را فقط برای قسمت کنترل این عبارت در نظر می‌گیریم.

به طور کلی حلقه‌های تکرار به دو دسته تقسیم می‌شوند:

الف) حلقه‌های تکراری که ابتدا شرط را بررسی کرده و سپس دستورات را اجرا می‌کنند؛ در این مورد اگر تعداد تکرار دستورات k بار باشد، تعداد تکرار قسمت کنترل حلقه k+۱ بار خواهد بود.

ب) حلقه‌های تکراری که ابتدا دستورات را اجرا کرده و در آخر شرط را بررسی می‌کنند؛ در این مورد اگر تعداد تکرار دستورات k بار باشد، تعداد تکرار قسمت کنترل حلقه نیز k بار خواهد بود.

۵ـ عبارات { و } : تعداد گام هر عبارت {و} صفر است.

۶ـ عبارت نام تابع: تعداد گام این عبارت صفر است.

۷ـ عبارت return: تعداد گام این عبارت یک است.

با انتساب مقادیر فوق به شمارش گام عبارات، می‌توان تعداد گام‌های یک برنامهٔ خاص را تعیین کرد. برای تعیین شمارش گام برنامه باید کل تعداد گام‌های هر عبارات را لیست کنیم. در این حالت، ابتدا تعداد گام‌ها را در هر اجرای عبارت و تعداد کل دفعات اجرای هر عبارت را تعیین می‌کنیم. با ترکیب این دو مقدار، کل زمان اجرای هر عبارت به دست می‌آید. با جمع کردن این مقادیر برای همهٔ عبارات، شمارش گام برنامه به دست خواهد آمد.

مثال۱ : تعداد گام‌های قطعه برنامه زیر کدام است؟

```
شمارهٔ خط
1     float sum (int  n)
2     {
3     float    S= o ;
4     int    i;
5     for (i = 1  ; i < =n  ; ++ i )
6     S + = i  ;
7     Return   S ;
8     }
```

۱) 2n+1

۲) 2n+2

۳) 2n+3

۴) 2n

☑ **پاسخ: گزینه «۳»** در جدول زیر تعداد گام‌های هر اجرا (step per execution) s/e و تکرار هر دستور تابع sum آورده شده است، تعداد کل گام‌ها $2n+3$ می‌باشد. نکته‌ای که باید بدان توجه کنید، آن است که تکرار دستور خط 5، $n+1$ مرتبه است نه n مرتبه. زیرا در خاتمه حلقهٔ for، مقدار نهـایی i ، $n+1$ است. و گام دستور 3، یک است، زیرا در این خط یک دستور انتساب وجود دارد.

خط	s/e	تکرار	کل گام‌ها
1	0	1	0
2	0	1	0
3	1	1	1
4	0	1	0
5	1	n+1	n+1
6	1	n	n
7	1	1	1
8	0	1	0
		مجموع همهٔ گام‌ها	2n+3

مثال ۲: تعداد گام‌های قطعه برنامهٔ زیر کدام است؟

```
شماره خط
1     int   test(int   n)
2     {
3     int  i , j , a=5 , b=10 , c=a+ b ;
4     for (i=1 ; i< =n ; ++i)
5     for(j=1,j<=n;++j)
6     {
7     a+=1;
8     b+=2 ;
9     c+=a+b;
10    }
11    Return    c  ;
12    }
```

۱) $4n^2+2n+2$ ۲) $4n^2+2n+5$ ۳) $4n^2+n+5$ ۴) $4n^2+n+2$

☑ **پاسخ:** گزینه «۲» در جدول زیر تعداد گام‌های هر اجرا (s/e) و تکرار هر دستور تابع Test آورده شده است. تعداد کل گام‌ها $4n^2+2n+5$ است.

خط	s/e	تکرار	کل گام‌ها
1	0	1	0
2	0	1	0
3	3	1	3
4	1	n+1	n +1
5	1	n(n+1)	n (n+1)
6	0	n × n	0
7	1	n × n	n^2
8	1	n × n	n^2
9	1	n × n	n^2
10	0	n × n	0
11	1	1	1
12	0	1	0
		مجموع همهٔ گام‌ها	$4n^2+2n+5$

حالت‌های ورودی

در اکثر الگوریتم‌ها، تعداد دفعات تکرار حلقه‌ها و در نتیجه تعداد کل عملیات آن‌ها، برای ورودی‌های مختلف با اندازه‌های یکسان، الزاماً برابر نیست. به عبارت دیگر، چگونگی قرار گرفتن عناصر ورودی، در مدت زمان اجرای الگوریتم تأثیر می‌گذارد. بنابراین، در تحلیل زمان اجرای الگوریتم‌ها، می‌بایست انواع حالت‌های ورودی را نیز مد نظر قرار داد. برای این منظور، سه حالت مختلف در تحلیل الگوریتم‌ها تعریف می‌شود:

۱ـ بهترین حالت (Best case): تحلیل بهترین حالت، مربوط به ورودی‌هایی است که سبب می‌شوند الگوریتم کم‌ترین زمان اجرا را داشته باشد.

۲ـ بدترین حالت (Worst case) : تحلیل بدترین حالت، حداکثر زمان مورد نیاز یک الگوریتم برای اندازه ورودی خاص را نشان می‌دهد.

۳ـ حالت میانگین (Average case): در حالت میانگین فرض می‌شود که همه‌ی حالت‌های مختلف داده‌های ورودی، احتمال برابر دارند و میانگین پیچیدگی زمان الگوریتم در همه‌ی این حالت‌ها محاسبه می‌شود.

نکته۱: به طور کلی پیچیدگی زمانی حالت میانگین با استفاده از رابطه‌ی زیر قابل محاسبه است:

$$T_{Average}(n) = \sum_{i=1}^{m} p_i \; t_i$$

که در آن، n نشان‌دهنده اندازه ورودی، P_i احتمال رخداد دسته‌ی i ام و t_i تعداد عملیات زمان مورد نیاز دسته i ام است.

* به منظور ساده شدن محاسبات، انواع حالت‌های ورودی به گونه‌ای دسته‌بندی می‌شوند که حالت‌های قرار گرفته در یک دسته، به مدت زمان یکسانی برای اجرا نیاز داشته باشند. پس از دسته‌بندی ورودی‌ها، باید احتمال رخداد هر یک از دسته‌ها را بررسی کرد.

مثال۳: الگوریتم جستجوی ترتیبی یا خطی عنصر x در آرایه S[1..n] را به صورت زیر در نظر بگیرید:

```
void linear search (int  n ,
                    const key type   s[] ,
                    key type   x ,
                    index  δ location )
{
  index   i ;
location = 0;
for(i=1 ; i<=n ; ++i)
if (x== S [i]){
                location = i ;
                break ;
                }
}
```

(الگوریتم جستجوی خطی)

تحلیل پیچیدگی زمانی بدترین حالت الگوریتم جستجوی ترتیبی

عمل مبنایی: مقایسه یک عنصر آرایه با x **اندازه ورودی:** n، تعداد عناصر آرایه

عمل مبنایی، حداکثر n مرتبه انجام شده است و این حالتی است که x در آرایه وجود ندارد و یا x آخرین عنصر آرایه است.

بنابراین:

$$w(n) = n$$

تحلیل پیچیدگی زمانی حالت میانی الگوریتم جستجوی ترتیبی

عمل مبنایی : مقایسه یک عنصر آرایه با x

اندازه ورودی : n، تعداد عناصر موجود در آرایه

مسئله را در حالتی تحلیل می‌کنیم که می‌دانیم عنصر x حتماً در آرایه وجود دارد. همه عناصر آرایهٔ S به صورت مجزا هستند و هیچ دلیلی وجود ندارد که احتمال x در یک اندیس آرایه را بیشتر از اندیس دیگر بدانیم. بر اساس این اطلاعات، برای $1 < K < n$، احتمال اینکه x در اندیس Kام باشد برابر $\frac{1}{n}$ است. اگر x در اندیس Kام باشد، تعداد دفعاتی که عمل مبنایی باید اجرا شود تا محل x در آرایه مشخص شود (و از حلقه خارج شود) برابر K است و این موضوع بدین معناست که پیچیدگی زمانی حالت میانی برابر است با:

$$A(n) = \sum_{K=1}^{n} (K \times \frac{1}{n}) = \frac{1}{n} \times \sum_{K=1}^{n} K = \frac{1}{n} \times \frac{n(n+1)}{2} = \frac{n+1}{2}$$

تحلیل پیچیدگی زمانی بهترین حالت الگوریتم جستجوی ترتیبی

عمل مبنایی : مقایسه یک عنصر آرایه با x

اندازه ورودی : n، تعداد عناصر آرایه

از آنجایی که $n>1$ است، لذا باید حداقل یک گذر از حلقه وجود داشته باشد و اگر $x=S[1]$ باشد، بدون توجه به مقدار اندازه ورودی n، تنها یک گذر از حلقه وجود خواهد داشت.

بنابراین: $B(n)=1$

مثال۴: الگوریتم جستجوی دودویی عنصر x در آرایه‌ی مرتب S[1..n] را به صورت زیر در نظر بگیرید:

```
void binary Search(int n,
                   Const Keytype s[],
                   Keytype x,
                   index & location)
{
index low, high, mid;
low = 1; high = n;
location = 0;
while(low <= high && location == 0){
  mid = ⌊(low + high) / 2⌋;
  if (x == s[mid])
    location = mid;
  else if (x < s[mid])
    high = mid – 1;
  else
    low = mid + 1;}
}
```

(الگوریتم جستجوی دودویی)

در الگوریتم جستجوی دودویی اگر x عنصر وسط آرایه باشد، عمل مبنایی تنها یک مرتبه انجام می‌شود، در حالی که اگر x در آرایه وجود نداشته باشد، عمل مبنایی $\lfloor \log_2^n \rfloor + 1$ مرتبه اجرا می‌گردد.

- آرایه **a[1..7] به صورت روبرو را در نظر بگیرید:** **a[1..7] = [11 15 20 30 40 50 55]**

مثال۵: برای جستجوی x = 50 در این آرایه به روش جستجوی دودویی چند مقایسه لازم است؟

۱) 1 ۲) 2 ۳) 3 ۴) 4

☑ **پاسخ:** گزینه «۲» با در نظر گرفتن $low=1$ و $high=7$ و بررسی گام به گام الگوریتم جستجوی دودویی داریم:

گام ۱:
$$\begin{cases} low=1, high=7 \\ mid=[\frac{1+7}{2}]=4 \\ x>a[mid] \rightarrow low=mid+1=4+1=5 \end{cases}$$

گام ۲:
$$\begin{cases} low=5, high=7 \\ mid=[\frac{5+7}{2}]=6 \\ x==a[mid] \rightarrow x \text{ در آرایه } a \text{ پیدا شد} \end{cases}$$

بنابراین برای جستجوی x در آرایه a دو مقایسه لازم است.

مثال ۶: برای جستجوی $x=20$ در آرایه $a[1..7]$ فوق به روش جستجوی دودویی چند مقایسه لازم است؟

۱) 1 ۲) 2 ۳) 3 ۴) 4

پاسخ: گزینه «۳» مانند مثال قبلی و با در نظر گرفتن $low=1$ و $high=7$ و بررسی گام به گام الگوریتم جستجوی دودویی داریم:

گام ۱:
$$\begin{cases} low=1, high=7 \\ mid=[\frac{1+7}{2}]=4 \\ x<a[mid] \rightarrow high=mid-1=4-1=3 \end{cases}$$

گام ۲:
$$\begin{cases} low=1, high=3 \\ mid=[\frac{1+3}{2}]=2 \\ x>a[mid] \rightarrow low=mid+1=2+1=3 \end{cases}$$

گام ۳:
$$\begin{cases} low=3, high=3 \\ mid=[\frac{3+3}{2}]=3 \\ x==a[mid] \rightarrow \text{ } x \text{ در آرایه } a \text{ پیدا شد} \end{cases}$$

بنابراین برای جستجوی x در آرایه a، $[\log_2^7]+1=2+1=3$ مقایسه لازم است.

مثال ۷: برای جستجوی $x=33$ در آرایه $a[1..7]$ فوق به روش جستجوی دودویی چند مقایسه لازم است؟

۱) 1 ۲) 2 ۳) 3 ۴) 4

پاسخ: گزینه «۳» مانند مثال‌های قبلی و با در نظر گرفتن $Low=1$ و $high=7$ و بررسی گام به گام الگوریتم جستجوی دودویی داریم:

گام ۱:
$$\begin{cases} low=1, high=7 \\ mid=[\frac{1+7}{2}]=4 \\ x>a[mid] \rightarrow low=mid+1=4+1=5 \end{cases}$$

گام ۲:
$$\begin{cases} low=5, high=7 \\ mid=[\frac{5+7}{2}]=6 \\ x<a[mid] \rightarrow high=mid-1=6-1=5 \end{cases}$$

گام ۳:
$$\begin{cases} low=5, high=5 \\ mid=[\frac{5+5}{2}]=5 \\ x<a[mid] \rightarrow high=mid-1=5-1=4, low<\neq high \rightarrow \text{ } x \text{ در آرایه } a \text{ پیدا نشد} \end{cases}$$

بنابراین برای آن که مشخص شود x در آرایه a نمی‌باشد، $[\log_2^7]+1=2+1=3$ مقایسه لازم است.

مثال ۸: تعداد متوسط مقایسه برای جستجوی موفق x با احتمال مساوی در آرایه $a[1..7]$ فوق به روش جستجوی دودویی کدام است؟

۱) 2 ۲) $\frac{17}{7}$ ۳) $\frac{16}{7}$ ۴) $\frac{18}{7}$

پاسخ: گزینه «۲» جستجوی موفق x، یعنی آن که x حتماً در آرایه باشد که در این صورت در آرایه $a[1..7]$، به طور کلی هفت حالت وجود دارد. بنابراین برای به دست آوردن تعداد متوسط مقایسه، باید برای هر حالتی تعداد مقایسات را به دست آورد و بر کل حالات که هفت حالت است، تقسیم کنیم.

در ذیل تعداد مقایسات برای هر حالتی آورده شده است:

$$a[1..7]=[\underset{3}{11} \quad \underset{2}{15} \quad \underset{3}{20} \quad \underset{1}{30} \quad \underset{3}{40} \quad \underset{2}{50} \quad \underset{3}{55}]$$

$$\text{تعداد متوسط مقایسه} = A(7)=\frac{3+2+3+1+3+2+3}{7}=\frac{17}{7}$$

نکته ۲: با توجه به مطالب فوق نتیجه می‌گیریم که مرتبه اجرایی الگوریتم جستجوی دودویی در بهترین حالت $B(n)=1$ و در بدترین حالت $W(n)=[\log_2^n]+1$ است.

نمادهای مجانبی یا حدی

انگیزه‌ی ما برای تعیین شمار گام‌ها و تعداد عملیات الگوریتم برای اندازه‌ی ورودی، توانایی مقایسه پیچیدگی‌های زمانی دو برنامه‌ای است که یک عمل را انجام می‌دهند و نیز پیش‌بینی رشد زمان اجرا با تغییر مشخصه‌های موردی است.

در عمل، محاسبات دقیق فوق و یا تعیین شمار دقیق گام‌های (بدترین حالت یا حالت میانگین) یک برنامه، کار بسیار مشکلی است.

تلاش و کوشش زیاد برای تعیین دقیق شمار گام‌ها، مقرون به صرفه نیست. از طرف دیگر، هدف از تحلیل یک الگوریتم، به دست آوردن تقریبی از رابطه‌ی آن است که محاسبه آن ساده باشد و همچنین این تقریب بیانگر میزان کارایی الگوریتم بوده و شامل اطلاعات اضافی نباشد.

الگوریتم‌ها اغلب براساس نرخ رشد تابع تعداد عملیات، دسته‌بندی و مقایسه می‌شوند و زمانی که ورودی تابع به اندازه کافی بزرگ باشد، جملاتی که در بزرگ شدن اندازه‌ی تابع نقش زیادی ندارند قابل چشم پوشی بوده و حذف می‌گردند. به عبارت دیگر، پیچیدگی یک الگوریتم به ازای مقادیر به اندازه کافی بزرگ ورودی، به مقدار واقعی تعداد عملیات انجام شده در الگوریتم به ازای آن ورودی نزدیک است. به مثال زیر توجه کنید:

مثال ۹ : کدام یک از جملات رابطه زیر، به ازای مقادیر بزرگ ورودی، بیشترین نقش را در رشد تابع دارد؟

$$t(n) = n^3 + 50n \log n + 1000n + 5000$$

اگر مقدار تابع t را به ازای مقادیر بزرگ ورودی n حساب کنیم، n^3 بزرگ‌ترین جمله بوده و به دلیل نرخ رشد بیشتری که نسبت به سه جمله دیگر دارد، بزرگ‌ترین جمله باقی می‌ماند. به عنوان نمونه، هنگامی که $n = 100$ شود، جمله اول در مقایسه با سه جمله دیگر، به حدی بزرگ می‌شود که می‌توان از سایر جملات تابع صرف‌نظر کرد و بنابراین رشد تابع t تقریباً وابسته به رشد جمله n^3 است.

برای نمایش نرخ رشد توابع، از نمادهای خاصی استفاده می‌شود که هر یک مفهوم و کاربرد خاص خود را دارند. این نمادها شامل O (Big-O) ، Ω (Big-Omega) ، θ (theta) ، o (small-o) ، ω (small-Omega) هستند که در ادامه معرفی خواهند شد.

نکته ۳: نمادهای مجانبی ، رفتار تابع را در مقادیر بزرگ ورودی در نظر می‌گیرند.

نماد O :

نماد مجانبی O (Big-O) ، حد بالای نرخ رشد توابع را نشان می‌دهد. $f(n) \in O(g(n))$ (خوانده می‌شود f ، O بزرگ g ی n است) به این معنی است که نرخ رشد f، کوچک‌تر یا مساوی نرخ رشد تابع g است. تعریف ریاضی آن به صورت زیر می‌باشد:

$$f(n) \in O(g(n)) \longleftrightarrow \{\exists\, c > 0, n_\circ > 0 \mid \forall\, n \geq n_\circ \;\; ; \;\; f(n) \leq c.\, g(n)\}$$

طبق این تعریف تابع f متعلق به مجموعه $O(g(n))$ است، اگر و فقط اگر، ثابت‌های مثبتی مانند $c, n_\circ$ وجود داشته باشد به طوری که به ازای تمام nهای به اندازه کافی بزرگ $(n \geq n_\circ)$، مقدار $f(n)$ کوچک تر یا مساوی تابع $c.\, g(n)$ باشد.

می‌توان نشان داد که $5n^2 + 3n + 6 \in O(n^2)$ و یا $6n + 5 \in O(n^2)$ و به طور کلی $O(n^2)$ شامل همه توابعی است که نرخ رشد آن‌ها کمتر یا مساوی n^2 است.

تذکر۱: در برخی موارد $f(n) \in O(g(n))$ را به صورت $f(n) = O(g(n))$ نیز نمایش می‌دهند.

نماد Ω :

نماد مجانبی Ω (Big-Omega) ، حد پایین نرخ رشد توابع را نشان می‌دهد. $f(n) \in \Omega(g(n))$، (خوانده می‌شود f ، n اُمگای g ی n است) به این معنی است که رشد تابع f بزرگ‌تر یا مساوی نرخ رشد تابع g است. تعریف ریاضی آن به صورت زیر است:

$$f(n) \in \Omega(g(n)) \leftrightarrow \{\, \exists\, c > 0, n_\circ > 0 \mid \forall\, n \geq n_\circ \,;\, f(n) \geq c.g(n) \,\}$$

طبق این تعریف تابع f متعلق به مجموعه $\Omega(g(n))$ است، اگر و فقط اگر ثابت‌های مثبتی مانند $C, n_\circ$ وجود داشته باشد به طوری که به ازای n های به اندازه کافی بزرگ $(n \geq n_\circ)$ ، مقدار تابع $f(n)$ ، بزرگ‌تر یا مساوی $c.g(n)$ باشد.

می‌توان نشان داد که $7n^2 + n + 5 \in \Omega(n^2)$,$6n^4 + 2n^2 + 1 \in \Omega(n^2)$ و به طور کلی $\Omega(n^2)$ شامل همه توابعی است که رشد آن‌ها بیشتر یا مساوی n^2 است.

نماد θ :

نماد مجانبی θ، زمانی استفاده می‌شود که نرخ رشد دو تابع مساوی باشد. تعریف ریاضی آن به صورت زیر است:

$$f(n)\in\theta(g(n))\longleftrightarrow\{\exists\, c_1,c_2,n_\circ>0\,|\forall n>n_\circ;c_1.g(n)\le f(n)\le c_2.g(n)\}$$

طبق این تعریف تابع f متعلق به $\theta(g(n))$ است، اگر و فقط اگر ثابت‌های مثبتی مانند $n_\circ,c_2,c_1$ وجود داشته باشند، به طوری که به ازای nهای به اندازه کافی بزرگ $(n\ge n_\circ):c_1.g(n)\le f(n)\le c_2.g(n)$ باشد.

می‌توان نشان داد که $3n+3\in\theta(n)$ و یا $10n^2+4n+2\in\theta(n^2)$ و $10n^2+4n+2\notin\theta(n)$.

نکته۴: θ دقیق‌ترین بیان برای نرخ رشد یک تابع است.

نماد O :

نماد مجانبی o (small-o) همانند نماد $Big-O$ است با این تفاوت که حد بالای اکید (مطلق) را برای رشد توابع مشخص می‌کند. $f(n)\in o(g(n))$ به این معنی است که رشد $f(n)$ همیشه کوچک‌تر از رشد تابع $g(n)$ خواهد بود.

تعریف ریاضی آن به صورت زیر است :

$$f(n)\in o(g(n))\longleftrightarrow\{\forall c>0\ ,\exists n_\circ>\circ|\forall\, n>n_\circ\ ;\ 0\le f(n)<c.g(n)\}$$

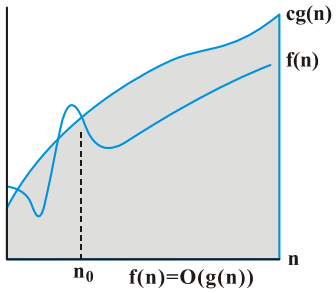
نماد ω :

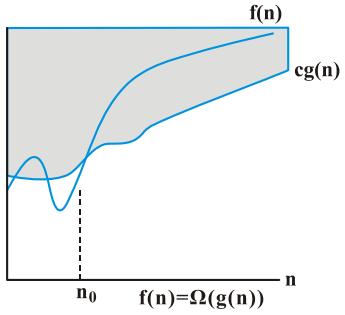
نماد مجانبی ω (small – omega) نیز همانند نماد $Big-Omega$ است و حد پایینی اکید (مطلق) را برای رشد تابع ارائه می‌دهد. $f(n)\in\omega(g(n))$ به این معنی است که رشد $f(n)$ ، همیشه بیشتر از رشد تابع $g(n)$ است.

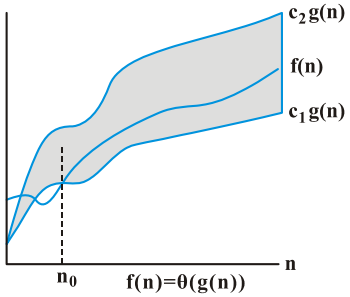
تعریف ریاضی آن به صورت زیر است :

$$f(n)\in\omega(g(n))\longleftrightarrow\{\forall c>0,\exists n_\circ>\circ|\forall n>n_\circ\ ;\ 0\le c.g(n)<f(n)\}$$

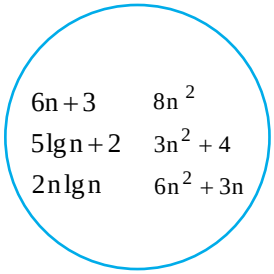
نمودارهای زیر مقایسه‌ای از تعاریف θ,Ω,O را نشان می‌دهند:



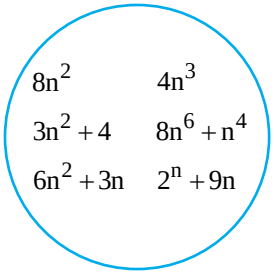




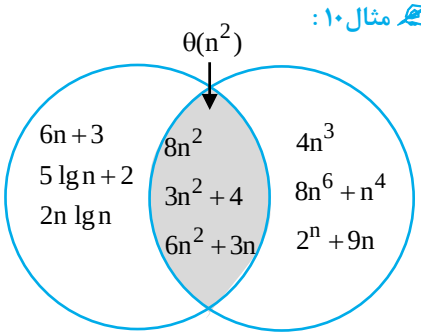
مثال۱۰ :



$O(n^2)$



$\Omega(n^2)$

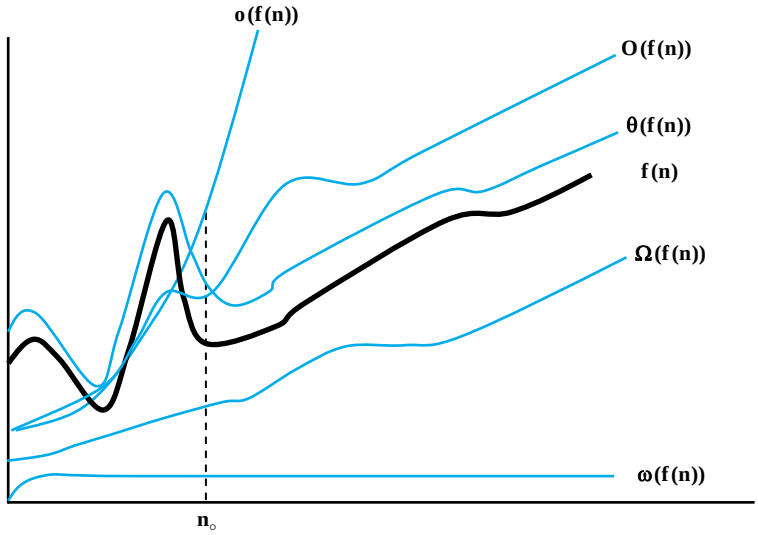


$\theta(n^2)=O(n^2)\cap\Omega(n^2)$

نکته۵:

رابطه‌های ریاضی	نمادهای مجانبی
$\leq$	O
$\geq$	Ω
$=$	θ
$<$	o
$>$	ω

شکل زیر مقایسه مناسبی از نمادهای مجانبی $\omega, o, \theta, \Omega, O$ را نشان می‌دهد:



خواص نمادهای مجانبی

۱ـ خاصیت بازتابی

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

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

$f(n) \in \theta(f(n))$

۲ـ خاصیت تعدی (تراگذری)

اگر $f(n) \in O(g(n)), g(n) \in O(h(n)) \longrightarrow f(n) \in O(h(n))$

اگر $f(n) \in \Omega(g(n)), g(n) \in \Omega(h(n)) \longrightarrow f(n) \in \Omega(h(n))$

اگر $f(n) \in \theta(g(n)), g(n) \in \theta(h(n)) \longrightarrow f(n) \in \theta(h(n))$

اگر $f(n) \in o(g(n)), g(n) \in o(h(n)) \longrightarrow f(n) \in o(h(n))$

اگر $f(n) \in \omega(g(n)), g(n) \in \omega(h(n)) \longrightarrow f(n) \in \omega(h(n))$

۳ـ خاصیت تقارنی

اگر $f(n) \in \theta(g(n)) \leftrightarrow g(n) \in \theta(f(n))$

نکته۶: توابع چند جمله‌ای که متعلق به θ یکدیگر هستند، بزرگ‌ترین توان آن‌ها با هم برابر است.

قضیه ۱: اگر $g(n) \in o(f(n))$ باشد، در این صورت: $g(n) \in O(f(n)) - \Omega(f(n))$

ویژگی‌های ترتیب:

۱ـ $g(n) \in O(f(n))$ است اگر و فقط اگر $f(n) \in \Omega(g(n))$ باشد.

۲ـ اگر $a > 1, b > 1$ باشد، آنگاه $Log_a n \in \theta(Log_b n)$ است.

این ویژگی نشان می‌دهد که همه توابع پیچیدگی لگاریتمی در یک ردهٔ پیچیدگی قرار دارند.

ما این رده را با $\theta(Logn)$ نشان می‌دهیم.

۳) اگر $b > a > 0$ باشد، آنگاه $a^n \in o(b^n)$ است.

این ویژگی نشان می‌دهد که همه توابع پیچیدگی نمایی در یک ردهٔ پیچیدگی قرار ندارند.

۴) برای هر $a > 0$ ، $a^n \in o(n!)$ است.

این ویژگی نشان می‌دهد $n!$ از تمامی توابع پیچیدگی نمایی بدتر می‌باشد.

۵) به رده‌های پیچیدگی مقابل توجه کنید: $\theta(Logn)\ \theta(n)\ \theta(nLogn)\ \theta(n^2)\ \theta(n^i)\ \theta(n^k)\ \theta(a^n)\ \theta(b^n)\ \theta(n!)$

که در آن $k > i > 2$ و $b > a > 1$ می‌باشد. اگر یک تابع پیچیدگی $g(n)$ در رده سمت چپ رده شامل $f(n)$ باشد، آنگاه $g(n) \in o(f(n))$ خواهد بود.

۶) اگر $C \geq 0$, $d > 0$, $g(n) \in O(f(n))$, $h(n) \in \theta(f(n))$ باشد، آنگاه $c \times g(n) + d \times h(n) \in \theta(f(n))$

نکته۷: نرخ رشد برخی از توابع معروف، به صورت زیر است:

$$n^{\frac{1}{\lg n}} < \lg(\lg^* n) < \lg^* n \approx \lg^*(\lg n) < 2^{\lg^* n} < Ln\ Ln\ n < \sqrt{\lg n} < \lg n < \lg^2 n < \sqrt{2}^{\lg n} \approx \sqrt{n}$$
$$< \sqrt{n \lg n} < \sqrt{n}\ \lg n < n < n \lg n \approx \lg(n!) \approx \lg(n^n) < n^2 < 7^{\lg n} < n^3 < (\lg n)! < (\lg n)^{\lg n}$$
$$\approx n^{\lg\lg n} < (\frac{3}{2})^n < 2^n < n.2^n < e^n < (\lg n)^n < n! < (n+1)! < n^n < 2^{2^n} < 2^{n!} < n^{n!} < n^{n^n}$$

قضیه ۲: برای توابع پیچیدگی $f(n)$ و $g(n)$ داریم:

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \begin{cases} C \Rightarrow g(n) \in \theta(f(n)) & \text{اگر } C > 0 \\ 0 \Rightarrow g(n) \in o(f(n)) \\ \infty \Rightarrow f(n) \in o(g(n)) \end{cases}$$

مثال۱۱: کدام گزینه نادرست است؟

۱) $3n^2 + 3n + 1 \in O(n^2)$ ۲) $5n + 1 \in \Omega(n)$ ۳) $3n^3 + 5n + 1 \in \Omega(n^2)$ ۴) $3n^2 + 5n \in O(n)$

☑ **پاسخ:** گزینه «۴» هیچ‌گاه منحنی n از منحنی $3n^2 + 5n$ بالاتر قرار نمی‌گیرد و n حد بالایی تابع پیچیدگی $3n^2 + 5n$ نمی‌باشد.

مثال۱۲: تکه برنامهٔ زیر را در نظر بگیرید:

```
int  S = 0 , i =1 ;
while (i<=n)
{
S+=i ;
i* = 2;
}
```

پیچیدگی زمانی این برنامه در حالت معمول به صورت روبرو است: $T(n) = [\log_2^n] + 1 \in \theta(\log n)$

مثال۱۳: تکه برنامهٔ زیر را در نظر بگیرید:

```
int S =0 , i =n
while (i >1)
{
S + = i ;
i /= 2 ;
}
```

پیچیدگی زمانی این برنامه در حالت معمول به صورت روبرو است: $T(n) = [\log_2^n] \in \theta(\log n)$