طراحی و تحلیل الگوریتم‌ها

مبحث دوم

تحلیل الگوریتم‌ها

Analysis of Algorithms

کاظم فولادی
دانشکده مهندسی برق و کامپیوتر
دانشگاه تهران

http://courses.fouladi.ir/algorithm

تحلیل الگوریتم‌ها

۱

مقدمه

تحلیل الگوریتم‌ها

زمان

زمان مصرفی برای اجرای الگوریتم

فضا

فضای مصرفی برای اجرای الگوریتم

تحلیل الگوریتم برحسب:

- ورودی
- اندازه‌ی ورودی

تحلیل الگوریتم بر اساس یک عمل خاص

مانند: مقایسه، انتساب، تعویض، جمع، ضرب

زمان اجرای یک الگوریتم

چه میزان **زمان** طول می‌کشد تا یک الگوریتم برای یک ورودی خاص **خاتمه** پیدا کند؟

زمان اجرای یک الگوریتم

زمان اجرای یک الگوریتم،
تعداد گام‌های آن در فرآیند اجرا
تا رسیدن به پاسخ است.

توابع زمان اجرا

توابع زمان اجرای الگوریتم	
زمان اجرا برای همه‌ی حالات	$T(n)$
• زمان اجرا برای همه‌ی حالات	
زمان اجرا در بدترین حالت	$W(n)$: *worst-case*
• بیانگر حداکثر دفعاتی که عمل اصلی اجرا می‌شود.	
زمان اجرا در حالت میانگین	$A(n)$: *average-case*
• میانگین (امیدریاضی) تعداد دفعاتی که عمل اصلی انجام می‌شود.	
زمان اجرا در بهترین حالت	$B(n)$: *best-case*
• بیانگر حداقل دفعاتی که عمل اصلی اجرا می‌شود.	

۲

زمان اجرا برای ساختارهای کنترل برنامه

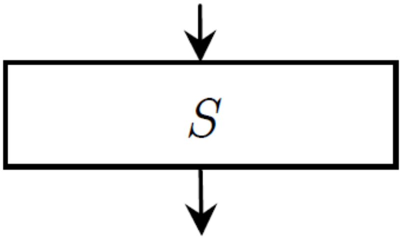
زمان اجرا و ساختارهای ترتیب، تصمیم، تکرار

ساختارهای کنترلی	
ترتیب	*Order*
• دستور • دستور مرکب	
تصمیم	*Decision*
• دستور شرطی if-then • دستور شرطی if-then-else	
تکرار	*Repeat*
• دستور تکرار for • دستور تکرار while-do • دستور تکرار do-while • دستور تکرار repeat-until	

زمان اجرای دستور

STATEMENT EXECUTION TIME

دستور را با S نمایش می‌دهیم. یک دستور، کوچکترین واحد الگوریتمی است.

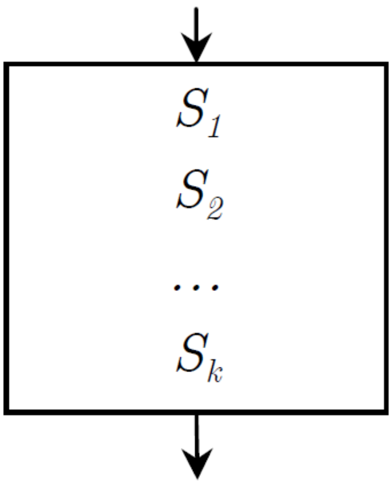


$$T(n)$$

زمان اجرای دستور مرکب

Compound Statement

دنباله‌ای از دو یا چند دستور، دستور مرکب نام دارد.
به عبارت دیگر دنباله‌ی چند دستور نیز یک دستور است که دستور مرکب نام دارد.



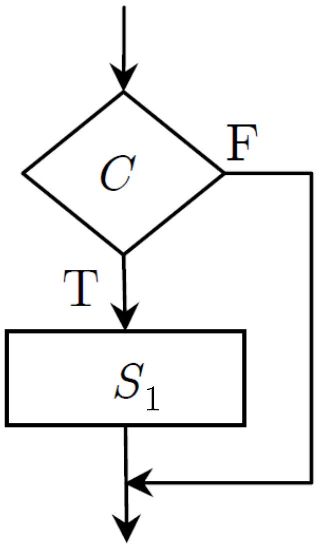
$$S_1 ; S_2 ; \ldots ; S_k$$

$$T(n) = \sum_{i=1}^{k} T_i(n)$$

زمان اجرای دستور شرطی

If-Then Statement

اگر C یک شرط و S یک دستور باشد، ساختار زیر نیز یک دستور است،
به طوری که اگر شرط درست باشد، دستور اجرا می‌شود:



$$\textbf{if } C \textbf{ then } S$$

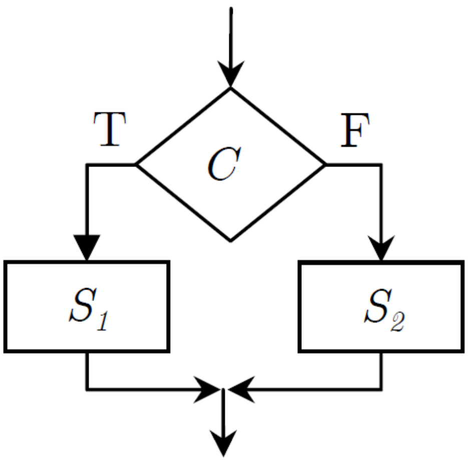
$$B(n) = T_C(n)$$
$$W(n) = T_C(n) + T_1(n)$$
$$A(n) = T_C(n) + p_1(n)T_1(n)$$

احتمال اجرای S_1

زمان اجرای دستور شرطی

If-Then-Else Statement

اگر C یک شرط و S_1 و S_2 دستور باشد، ساختار زیر نیز یک دستور است، به طوری که اگر شرط درست باشد، دستور S_1 و اگر شرط نادرست باشد، دستور S_2 اجرا می‌شود.



if C **then** S_1 **else** S_2

$$B(n) = T_C(n) + \min\{T_1(n), T_2(n)\}$$

$$W(n) = T_C(n) + \max\{T_1(n), T_2(n)\}$$

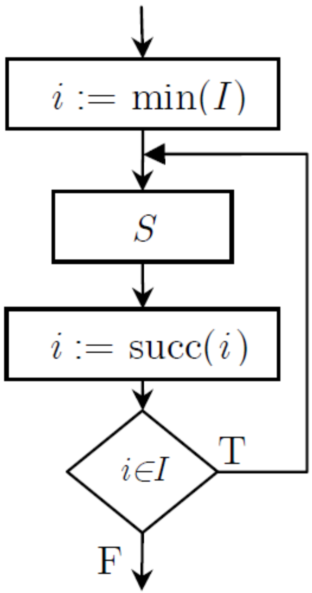
$$A(n) = T_C(n) + p_1(n)T_1(n) + p_2(n)T_2(n)$$

$$\boxed{p_1 + p_2 = 1}$$

زمان اجرای دستور تکرار ساده

For Statement

اگر یک مجموعه‌ی اندیس‌گذار (مجموعه‌ی مرتب و گسسته) باشد،
ساختار زیر یک دستور است که دستور S را به ازای هر $i \in I$ اجرا می‌کند.



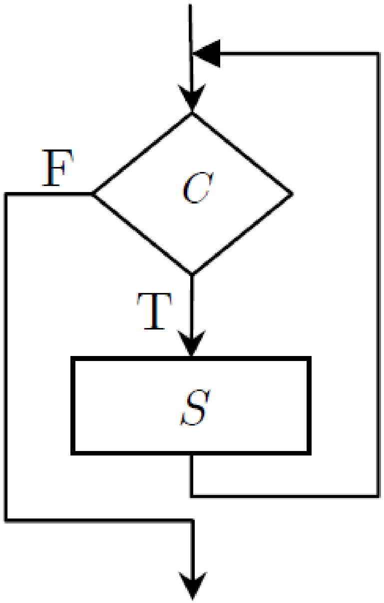
$$\textbf{for } i \in I \textbf{ do } S$$

$$T(n) = \sum_{i \in I} T_S(i)$$

زمان اجرای دستور تکرار شرطی

WHILE-DO STATEMENT

اگر C یک شرط و S یک دستور باشد، ساختار زیر نیز یک دستور است،
به طوری که تا وقتی که شرط درست باشد، دستور اجرا می‌شود.
(ابتدا شرط بررسی می‌شود)



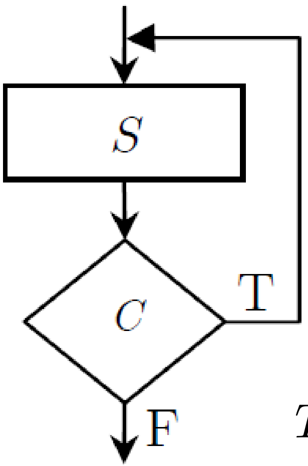
$$\textbf{while } C \textbf{ do } S$$

$$T(n) = T_C(n) + \sum_{C \equiv True} \left(T_S(n) + T_C(n)\right)$$

زمان اجرای دستور تکرار شرطی

Do-While Statement

اگر C یک شرط و S یک دستور باشد، ساختار زیر نیز یک دستور است،
به طوری که تا وقتی که شرط درست باشد، دستور اجرا می‌شود.
(انتها شرط بررسی می‌شود)



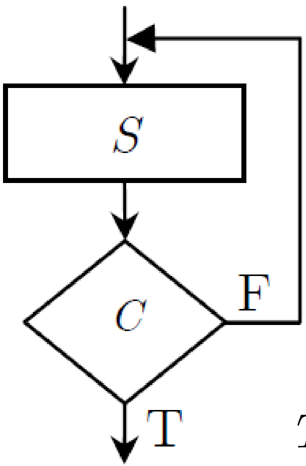
$$\textbf{do } S \textbf{ while } C$$

$$T(n) = T_S(n) + \sum_{C \equiv True} \left(T_C(n) + T_S(n)\right) + \underbrace{T_C(n)}_{C \equiv False}$$

دستور تکرار شرطی

Repeat-Until Statement

اگر C یک شرط و S یک دستور باشد، ساختار زیر نیز یک دستور است،
به طوری که تا وقتی که شرط درست باشد، دستور اجرا می‌شود
(انتها شرط بررسی می‌شود)



repeat S **until** C

$$T(n) = T_S(n) + \sum_{C \equiv False} (T_C(n) + T_S(n)) + \underbrace{T_C(n)}_{C \equiv True}$$

۳

مرور ریاضی

عملگر مجموع‌یابی

SUMMATION OPERATOR

$$\sum_{i=1}^{n} f(i) = f(1) + f(2) + \cdots + f(n)$$

$$\begin{cases} \displaystyle\sum_{i=m}^{n} f(i) = f(m) + f(m+1) + \cdots + f(n) & , m \leq n \\ \displaystyle\sum_{i=m}^{n} f(i) = 0 & , m > n \end{cases}$$

$$\sum_{i=m}^{n} f(i) = \sum_{i \in [m..n]} f(i)$$

عملگر مجموع‌یابی: خاصیت‌ها

مرور ریاضی

$$\sum_i \alpha f(i) = \alpha \sum_i f(i)$$

$$\sum_i \{f(i) + g(i)\} = \sum_i f(i) + \sum_i g(i)$$

$$\sum_i \{\alpha f(i) + \beta g(i)\} = \alpha \sum_i f(i) + \beta \sum_i g(i)$$

مرور ریاضی

عملگر مجموع‌یابی: خاصیت‌ها

$$\sum_{i=m}^{n} f(i) = \sum_{i=m+\eta}^{n+\eta} f(i-\eta)$$

$$\sum_{i=m}^{n} f(i) = \sum_{i=m}^{\eta} f(i) + \sum_{i=\eta+1}^{n} f(i) \quad , m \leq \eta \leq n$$

عملگر مجموع‌یابی: خاصیت‌ها

قاعده‌ی تلسکوپی برای مجموع‌یابی:

$$\sum_{i=m}^{n} \{f(i) - f(i-1)\} = f(n) - f(m-1)$$

عملگر مجموع‌یابی: خاصیت‌ها

$$\sum_{i=1}^{n} 1 = n$$

$$\sum_{i=1}^{n} i = \frac{1}{2} n(n+1)$$

$$\sum_{i=1}^{n} i^2 = \frac{1}{6} n(n+1)(2n+1)$$

$$\sum_{i=1}^{n} i^3 = \frac{1}{4} n^2 (n+1)^2$$

$$\sum_{i=1}^{n} i^4 = \frac{1}{30} n(n+1)(6n^3 + 9n^2 + n - 1)$$

$$\sum_{i=1}^{n} x^i = x \frac{1-x^n}{1-x}$$

$$\sum_{i=1}^{\infty} x^i = \frac{x}{1-x}, |x| < 1$$

عملگر مجموع‌یابی: خاصیت‌ها

تقریب انتگرال برای محاسبه‌ی مجموع: اگر f صعودی باشد:

$$\sum_{i=m}^{n} f(i) \approx \int_{m}^{n} f(x)dx$$

عملگر ضرب

مرور ریاضی

Product Operator

$$\prod_{i=1}^{n} f(i) = f(1) \times f(2) \times \cdots \times f(n)$$

$$\begin{cases} \prod_{i=m}^{n} f(i) = f(m) \times f(m+1) \times \cdots \times f(n) & , m \leq n \\ \prod_{i=m}^{n} f(i) = 1 & , m > n \end{cases}$$

$$\prod_{i=m}^{n} f(i) = \prod_{i \in [m..n]} f(i)$$

عملگر ضرب: خاصیت‌ها

$$\prod_{i=1}^{n} \alpha = \alpha^n$$

$$\prod_{i=1}^{n} i = n!$$

$$\prod_{i=m}^{n} \frac{f(i)}{f(i-1)} = \frac{f(n)}{f(m-1)}$$

توابع روی اعداد صحیح

توابع اعداد صحیح و روابط آنها	
تابع کف $\lfloor x \rfloor$	***Floor***
• بزرگ‌ترین عدد صحیح کوچک‌تر یا مساوی با x	
تابع سقف $\lceil x \rceil$	***Ceiling***
• کوچک‌ترین عدد صحیح بزرگ‌تر یا مساوی با x	
تابع برش $[[x]]$	***Truncate***
• بخش صحیح عدد x	
تابع باقیمانده $a \bmod b$	***Modulus***
• باقیمانده‌ی تقسیم a بر b	
تابع فاکتوریل $n!$	***Factorial***
• حاصل‌ضرب اعداد 1 تا n	

توابع جزء صحیح: ویژگی‌ها

$$x-1 < \lfloor x \rfloor \le x \le \lceil x \rceil < x+1 \quad , x \in \mathbb{R}$$

$$\boxed{\left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{n}{2} \right\rfloor = n \quad , n \in \mathbb{Z}}$$

$$\left\lfloor \frac{\lfloor n/a \rfloor}{b} \right\rfloor = \left\lfloor \frac{n}{ab} \right\rfloor \qquad \left\lceil \frac{\lceil n/a \rceil}{b} \right\rceil = \left\lceil \frac{n}{ab} \right\rceil$$

$$\boxed{\left\lceil \frac{a}{b} \right\rceil \le \frac{a+b-1}{b} \qquad \left\lfloor \frac{a}{b} \right\rfloor \ge \frac{a-b+1}{b}}$$

$$\lfloor x+n \rfloor = \lfloor x \rfloor + n \quad , n \in \mathbb{Z}, x \in \mathbb{R}$$

$$\lceil x+n \rceil = \lceil x \rceil + n \quad , n \in \mathbb{Z}, x \in \mathbb{R}$$

باقیمانده

$$a \bmod b = a - \left\lfloor \frac{a}{b} \right\rfloor b \quad , a \geq 0, b > 0$$

الگوریتم تقسیم:

$$\forall a \forall b (a, b \in \mathbb{Z}, b \neq 0 \Rightarrow \exists q \exists r (q \in \mathbb{Z}, r \in \mathbb{Z}, a = bq + r)$$

فاکتوریل: ویژگی‌ها

$$n! = 1 \times 2 \times \cdots \times (n-1) \times n$$

$$\begin{cases} n! = n(n-1)! \\ 0! = 1 \end{cases}$$

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

تقریب استرلینگ

ضریب دوجمله‌ای

$$\binom{n}{k}$$

ضریب دوجمله‌ای = ترکیب k از n:

تعداد راه‌های انتخاب k شیئ از n شیئ وقتی ترتیب مهم نیست.

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n \times (n-1) \times \cdots \times (n-k+1)}{1 \times 2 \times \cdots \times k}$$

ضریب دوجمله‌ای: ویژگی‌ها

$$\binom{n}{1} = n$$

$$\binom{n}{0} = \binom{n}{n} = 1$$

$$\binom{n}{k} = \binom{n}{n-k}$$

$$\binom{n}{k} = \binom{n}{k-1} + \binom{n-1}{k-1}$$

$$\binom{n}{k}^k \le \binom{n}{k} \le \frac{n^k}{k!} \qquad ,0 \le k \le n$$

۴

مثال‌هایی از محاسبه‌ی زمان اجرا

مثال

زمان اجرای قطعه الگوریتم‌های زیر را محاسبه کنید:

۱

$$\textbf{for } i \leftarrow a \textbf{ to } b \textbf{ do}$$
$$\quad x \leftarrow x + 1$$

۳

$$\textbf{for } i \leftarrow a \textbf{ downto } b \textbf{ do}$$
$$\quad x \leftarrow x + 1$$

۲

$$\textbf{for } i \leftarrow 1 \textbf{ to } n \textbf{ do}$$
$$\quad x \leftarrow x + 1$$

۴

$$\textbf{for } i \leftarrow n \textbf{ downto } 1 \textbf{ do}$$
$$\quad x \leftarrow x + 1$$

مثال

زمان اجرای قطعه الگوریتم زیر را محاسبه کنید:

$$
\begin{aligned}
&\textbf{for } i \leftarrow 1 \textbf{ to } n \textbf{ do}\\
&\quad \textbf{for } j \leftarrow 1 \textbf{ to } n \textbf{ do}\\
&\quad\quad x \leftarrow x \,/\, 2
\end{aligned}
$$

۵

مثال

زمان اجرای قطعه الگوریتم زیر را محاسبه کنید:

$$
\begin{array}{l}
\textbf{for } i \leftarrow 1 \textbf{ to } n \textbf{ do} \\
\quad \textbf{for } j \leftarrow 1 \textbf{ to } n \textbf{ do} \\
\quad\quad \textbf{for } k \leftarrow j \textbf{ to } n \textbf{ do} \\
\quad\quad\quad x \leftarrow x \,/\, 2
\end{array}
$$

۶

مثال

زمان اجرای قطعه الگوریتم زیر را محاسبه کنید:

$$
\begin{array}{l}
\textbf{for } i \leftarrow 1 \textbf{ to } n \textbf{ do} \\
\quad \textbf{for } j \leftarrow i \textbf{ to } n \textbf{ do} \\
\quad\quad \textbf{print } i + j
\end{array}
$$

۷

مثال

زمان اجرای قطعه الگوریتم زیر را محاسبه کنید:

$i \leftarrow n$

$x \leftarrow k$

$\textbf{while } i > 0 \textbf{ do}$

$\quad x \leftarrow x + 1$

$\quad i \leftarrow i - 1$

۸

مثال

زمان اجرای قطعه الگوریتم زیر را محاسبه کنید:

$$i \leftarrow n$$
$$\textbf{while } i \geq 1 \textbf{ do}$$
$$\quad i \leftarrow i \,/\, 2$$

۹

۵

بازگشت:

تکرار روند جاری

بازگشت

تکرار روند جاری

RECURRENCE

چرا برنامه‌های بازگشتی؟

ساختار بعضی مسئله‌ها طبیعت بازگشتی دارد!

از بازگشت برای تعریف مفاهیمی که بر اساس خودشان تعریف می‌شوند استفاده می‌شود.

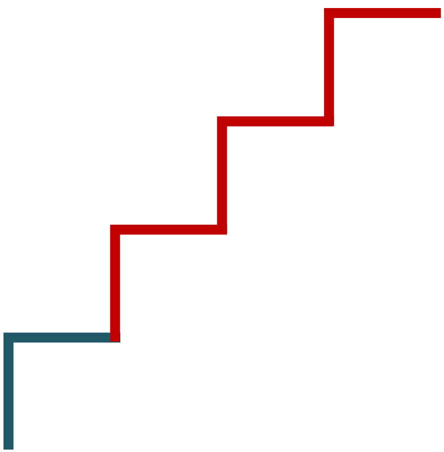
بازگشت

تکرار روند جاری

RECURRENCE

مثال

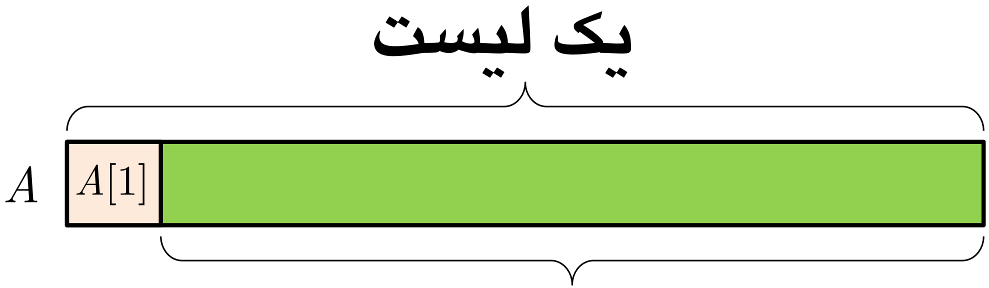
چگونه از یک پلکان بالا برویم؟



تعریف بالا رفتن از پلکان بر اساس بالا رفتن از پلکان!

- [شرط اولیه] بالا رفتن از یک پله را بلدیم!
- [بازگشت] بالا از رفتن از یک پلکان (با n پله) عبارت است از
  - بالا رفتن از یک پله +
  - بالا رفتن از یک پلکان (با n-1 پله)

مثال



تعریف لیست به صورت بازگشتی:

- [شرط اولیه] یک عنصر تکی یک **لیست** است.
- [بازگشت] یک **لیست**، یک عنصر است که به دنبال آن یک **لیست** می‌آید.

برنامه‌نویسی بازگشتی

اگر برای یک مسئله الگویی بازگشتی داشته باشیم، معمولاً می‌توانیم راه حل آن را در قالب یک تابع بازگشتی پیاده‌سازی کنیم.

کافی است تابعی نوشته شود که خودش را در بدنه‌ی خودش فراخوانی کند.

برنامه‌نویسی بازگشتی

مثال (الگوی ریاضی بازگشتی)

برنامه‌ای برای محاسبه‌ی مجموع اعداد 1 تا n

$$\underbrace{\underbrace{1+2+3+4+\cdots+(n-1)}_{S(n-1)}+n}_{S(n)}$$

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

$$n>0$$

برنامه‌نویسی بازگشتی

مثال (برنامه‌ی کامپیوتری)

برنامه‌ای برای محاسبه‌ی مجموع اعداد 1 تا n

$$\underbrace{\underbrace{1+2+3+4+\cdots+(n-1)}_{S(n-1)}+n}_{S(n)}$$

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

$$n > 0$$

```
int sum (int n)
{
    if (n == 1) return 1;
    return sum(n-1) + n;
}
```

برنامه‌نویسی بازگشتی

مثال (اجرای برنامه)

برنامه‌ای برای محاسبه‌ی مجموع اعداد 1 تا n

```
int sum (int n)
{
    if (n == 1) return 1;
    return sum(n-1) + n;
}
```

$$
\begin{array}{ll}
\texttt{sum(4)} & = 6 + 4 = 10 \\
\downarrow & \uparrow \\
\texttt{sum(3)} & = 3 + 3 = 6 \\
\downarrow & \uparrow \\
\texttt{sum(2)} & = 1 + 2 = 3 \\
\downarrow & \uparrow \\
\texttt{sum(1)} \Rightarrow & 1
\end{array}
$$

هشدار: حلقه‌های نامتناهی!

معمولاً شرط اولیه باید پیش از فراخوانی بازگشتی بررسی شود.

در غیر این صورت ممکن است گرفتار حلقه‌های نامتناهی شوید.

برنامه‌نویسی بازگشتی

اجتناب از حلقه‌های نامتناهی : مثال

معمولاً شرط اولیه باید پیش از فراخوانی بازگشتی بررسی شود.

در غیر این صورت ممکن است گرفتار حلقه‌های نامتناهی شوید.

```
int sum (int n)
{
    int s;
    s = sum(n-1) + n;
    if (n == 1) s = 1;
    return s;
}
```

```
sum(3)
  ↓
sum(2)
  ↓
sum(1)
  ↓
sum(0)
  ↓
 ...
```

برنامه‌نویسی بازگشتی

قالب عمومی یک تابع بازگشتی در C++

قالب عمومی یک تابع بازگشتی در C++

```
out_type f(in_type in)
{
   if (some easily-solved condition)     // base case

       solution statement

   else                                  // general case

       recursive function call
}
```

برنامه‌نویسی بازگشتی

مثال : تابع بازگشتی با خروجی void

این برنامه چه کاری انجام می‌دهد؟

```
void f (int n)
//   Prints n asterisks, one to a line
//   Precondition:  n is assigned
//   Postcondition:
//           IF n > 0, n stars have been printed, one to a line
//           ELSE no action has taken place
{
     if ( n <= 0 )    // base case
                      // do nothing
     else             // general case
     {
         cout   <<  ‘*’  <<  “ “ ;
         f (n - 1) ;
     }
}
```

برنامه‌نویسی بازگشتی

مثال : تابع بازگشتی با خروجی void

```
            *
          * *
        * * *
      * * * *
    * * * * *
  * * * * * *
* * * * * * *
```

برنامه‌نویسی بازگشتی

مثال : تابع بازگشتی با خروجی void

یک برنامه‌ی بازگشتی بنویسید که خروجی زیر را تولید کند:

```
void g (int n)
{
//precondition: n > 0

      if ( n == 1 )     // base case
          cout << '*';
      else              // general case

      {
          g (n - 1);
          cout << endl;
          f (n);
          cout << endl;
      }
}
```

```
            *
          * *
        * * *
      * * * *
    * * * * *
  * * * * * *
* * * * * * *
```

g (7)

تعاریف بازگشتی

RECURSIVE DEFINITION

از بازگشت برای تعریف مفاهیمی که بر اساس خودشان تعریف می‌شوند استفاده می‌شود،

مثلاً:

- **عبارت ریاضی**
  - *(شرط اولیه)* a یک **عبارت ریاضی** است
  - اگر E و F **عبارت‌های ریاضی** باشند، آنگاه $E + F$ هم **عبارت ریاضی** است.
- **لیست**
  - *(شرط اولیه)* یک عنصر تکی یک **لیست** است.
  - یک **لیست**، یک عنصر است که می‌تواند به دنبال آن یک **لیست** دیگر بیاید.

RECURSIVE ALGORITHMS

الگوریتم **بازگشتی**، الگوریتمی است که در بدنه‌ی خود، خودش را فراخوانی کند.

انواع الگوریتم بازگشتی	
بازگشتی مستقیم	***Direct Recursive***
• فراخوانی بی‌واسطه $A \rightarrow A$	
بازگشتی غیرمستقیم	***Indirect Recursive***
• فراخوانی با واسطه $A \rightarrow B \rightarrow A$	

طراحی الگوریتم بازگشتی زمانی مناسب است که
خود مسئله به صورت بازگشتی تعریف شده باشد.

توابع بازگشتی

RECURSIVE FUNCTIONS

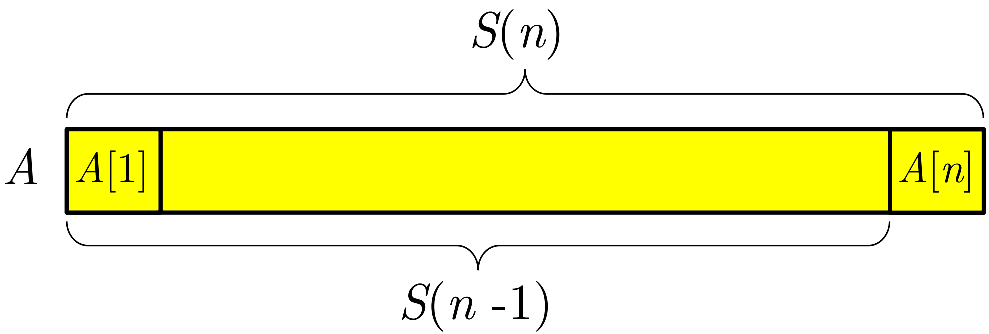
الگوریتم‌های بازگشتی روشی مستقیم برای محاسبه‌ی توابع بازگشتی است:

- **تابع برای انجام عملیات خود، خودش را فراخوانی می‌کند.**
- **برای خاتمه‌ی فرآیند بازگشت باید شرایط اولیه وجود داشته باشد.**

توابع بازگشتی

مثال

تابعی بازگشتی برای محاسبه‌ی مجموع یک آرایه‌ی n عضوی

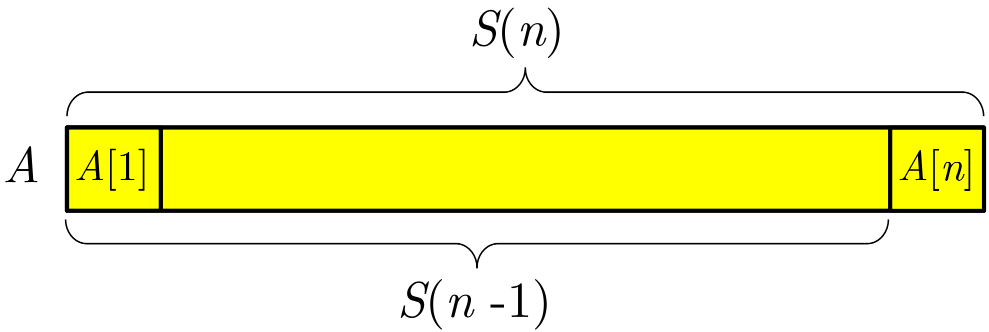


$$S(n) = \begin{cases} S(n-1) + A[n] & , n > 1 \\ A[n] & , n = 1 \end{cases}$$

الگوریتم‌های بازگشتی

مثال

الگوریتم بازگشتی برای محاسبه‌ی مجموع یک آرایه‌ی nعضوی



$\textsc{Recursive-Sum}(A, n)$
    if $n = 1$ **then**
        return $A[n]$
    else
        return $\textsc{Recursive-Sum}(A, n-1) + A[n]$

الگوریتم‌های بازگشتی

محاسبه‌ی زمان اجرا

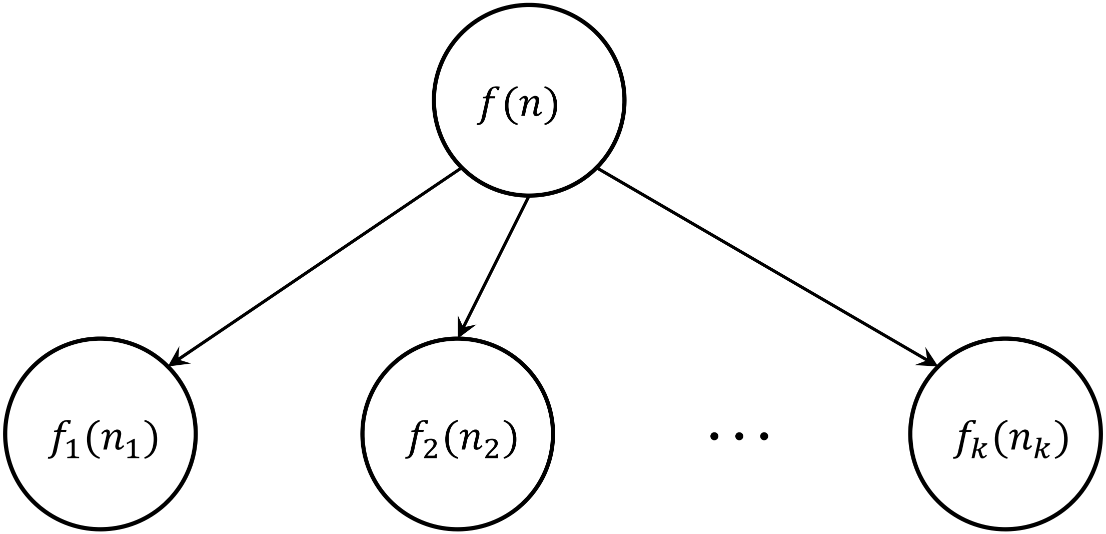
زمان اجرای الگوریتم بازگشتی زیر را محاسبه کنید:

$\boxed{T(n)}$

```
RECURSIVE-SUM(A,n)
    if n = 1 then
        return A[n]
    else                                  T(n -1)
        return RECURSIVE-SUM(A, n -1) + A[n]
```

درخت فراخوانی

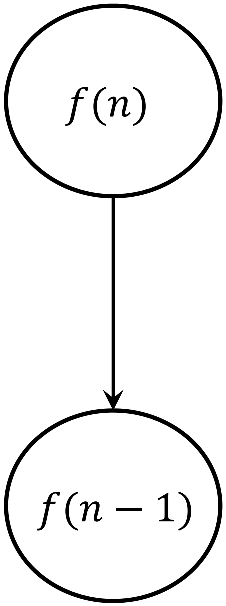
CALLING TREE



$$f(n) = \mathcal{F}(f_1(n_1), f_2(n_2), \ldots, f_k(n_k))$$

درخت فراخوانی

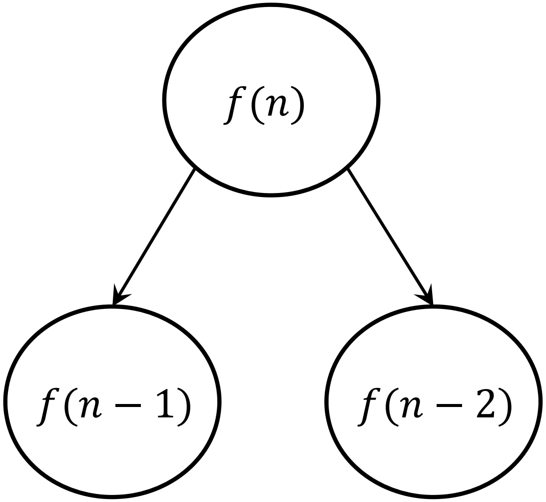
مثال



$$f(n) = \mathcal{F}(f(n-1))$$

درخت فراخوانی

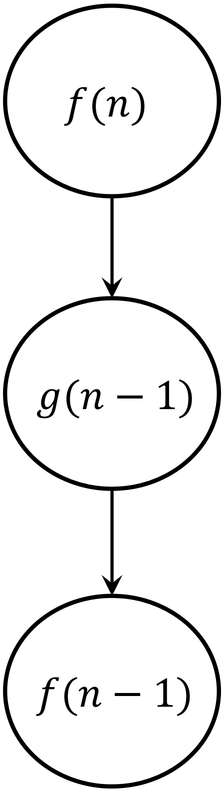
مثال



$$f(n) = \mathcal{F}(f(n-1), f(n-2))$$

درخت فراخوانی

مثال



$$f(n) = \mathcal{F}(g(n-1))$$
$$g(n) = \mathcal{G}(f(n))$$

بازگشتی غیرمستقیم

الگوریتم‌های بازگشتی

مثال

یک الگوریتم بازگشتی برای محاسبه‌ی توان x^n بنویسید
و زمان اجرای آن را محاسبه کنید.

حل معادلات بازگشتی

معادلات خطی مرتبه اول

$$f(n) = p(n)f(n-1) + q(n)$$

حل معادلات بازگشتی

معادلات خطی مرتبه kام همگن با ضرایب ثابت

$$f(n) = A_1 f(n-1) + A_2 f(n-2) + \cdots + A_k f(n-k)$$

حل معادلات بازگشتی

معادلات خطی مرتبه kام ناهمگن با ضرایب ثابت

$$f(n) = A_1 f(n-1) + A_2 f(n-2) + \cdots + A_k f(n-k) + g(n)$$

حل معادلات بازگشتی

مرور ریاضی

معادلات تقسیم و غلبه

$$f(n) = p(n)f\left(\frac{n}{b}\right) + q(n)$$

$$\boxed{n = b^k}$$

حل معادلات بازگشتی: حالات خاص

معادلات مرتبه اول

$f(n) = f(n-1) + \alpha$	$\Rightarrow$	$f(n) = f(0) + n\alpha$
$f(n) = f(n-1) + \alpha n$	$\Rightarrow$	$f(n) = f(0) + \alpha \frac{n(n+1)}{2}$
$f(n) = \alpha f(n-1)$	$\Rightarrow$	$f(n) = f(0)\alpha^n$
$f(n) = p(n) f(n-1)$	$\Rightarrow$	$f(n) = f(0)\prod_{i=1}^{n} p(i)$

الگوریتم‌های بازگشتی

محاسبه‌ی زمان اجرا : مثال

تعداد عمل ضرب در اجرای الگوریتم زیر را محاسبه کنید :

$$
\begin{array}{l}
\textbf{for } i \leftarrow 1 \textbf{ to } n \textbf{ do} \\
\quad \textbf{for } j \leftarrow n \textbf{ to } i \textbf{ do} \\
\quad\quad \textbf{print } f(i)
\end{array}
$$

$$
\begin{array}{l}
f(n) \\
\quad \textbf{if } n = 0 \textbf{ then} \\
\quad\quad \textbf{return } 1 \\
\quad \textbf{else} \\
\quad\quad \textbf{return } f(n\text{ -}1) * n
\end{array}
$$

الگوریتم‌های بازگشتی

محاسبه‌ی زمان اجرا : مثال

دستور **print** در قطعه کد زیر، چند مرتبه اجرا می‌شود؟

$f(n)$
    if $n = 1$ **then**
        return 1
    else
        print $f(n/2)$
        print $f(n/2)$
    return 0

۶

تحلیل مجانبی زمان اجرا

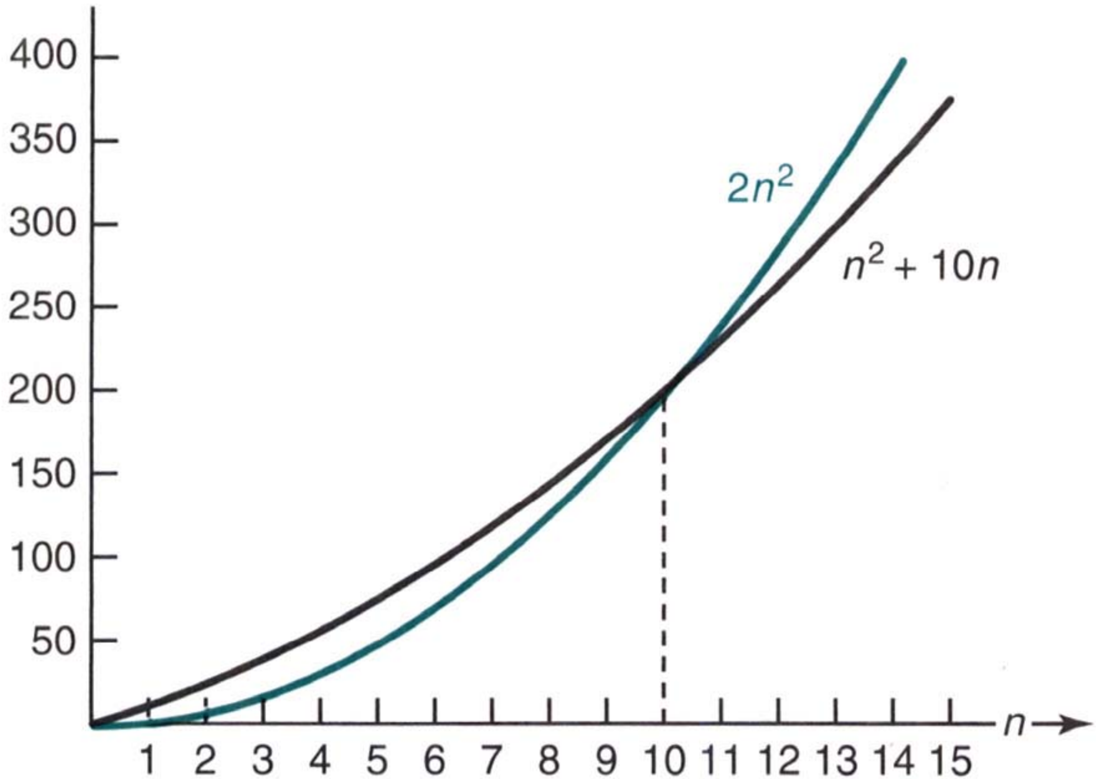
تحلیل مجانبی زمان اجرا

اگرچه **تحلیل زمانی الگوریتم‌ها** به طور دقیق مفید است، اما در عمل به دست آوردن **تقریبی** از آن نیز کافی است.

مقایسه‌ی دو تابع رشد درجه دوم

n	$0.1n^2$	$0.1n^2 + n + 100$
10	10	120
20	40	160
50	250	400
100	1000	1200
1000	100000	101100

مقایسه‌ی نموداری دو تابع درجه دوم

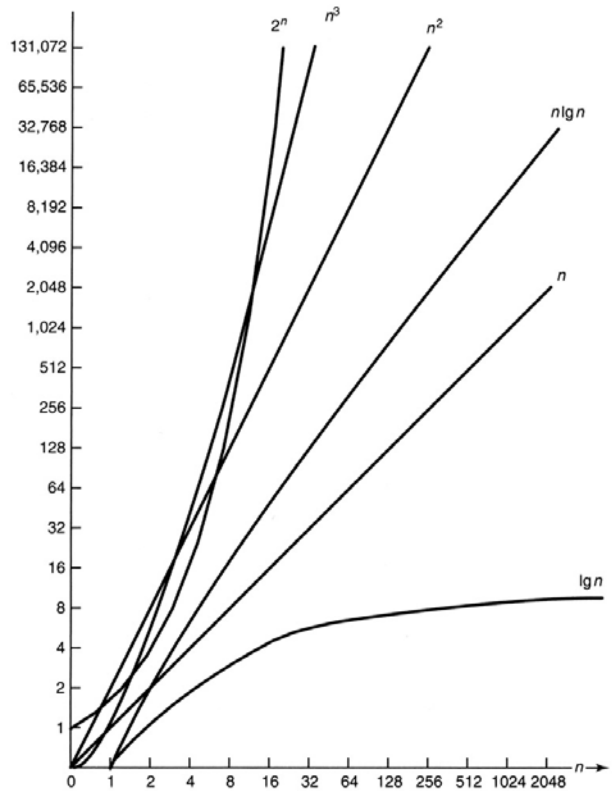


مقایسه‌ی توابع رشد

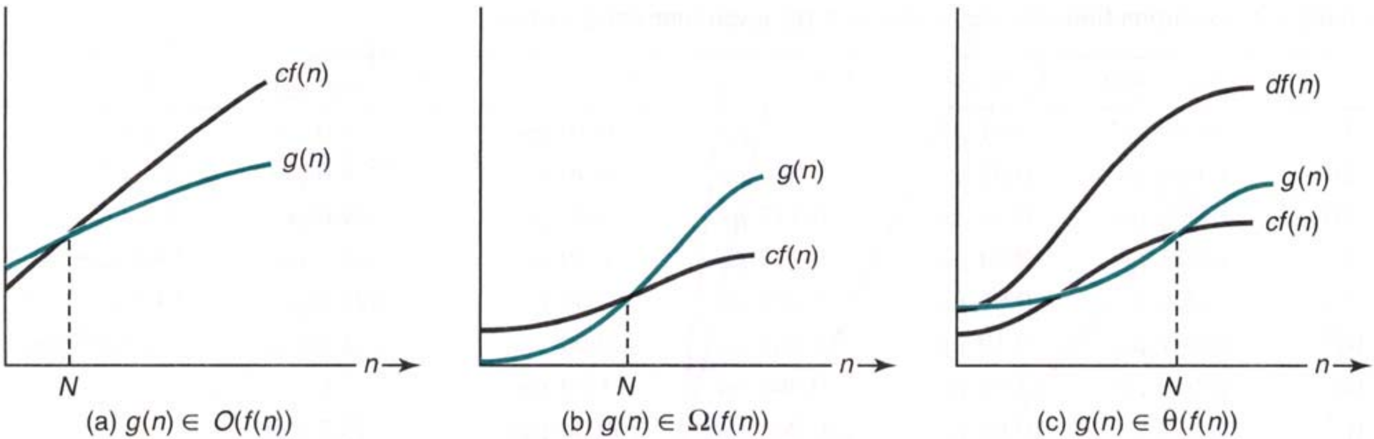
n	$f(n) = \lg n$	$f(n) = n$	$f(n) = n \lg n$	$f(n) = n^2$	$f(n) = n^3$	$f(n) = 2^n$
10	0.003 µs[*]	0.01 µs	0.033 µs	0.10 µs	1.0 µs	1 µs
20	0.004 µs	0.02 µs	0.086 µs	0.40 µs	8.0 µs	1 ms[†]
30	0.005 µs	0.03 µs	0.147 µs	0.90 µs	27.0 µs	1 s
40	0.005 µs	0.04 µs	0.213 µs	1.60 µs	64.0 µs	18.3 min
50	0.006 µs	0.05 µs	0.282 µs	2.50 µs	125.0 µs	13 days
10^2	0.007 µs	0.10 µs	0.664 µs	10.00 µs	1.0 ms	4×10^{13} years
10^3	0.010 µs	1.00 µs	9.966 µs	1.00 ms	1.0 s	
10^4	0.013 µs	10.00 µs	130.000 µs	100.00 ms	16.7 min	
10^5	0.017 µs	0.10 ms	1.670 ms	10.00 s	11.6 days	
10^6	0.020 µs	1.00 ms	19.930 ms	16.70 min	31.7 years	
10^7	0.023 µs	0.01 s	2.660 s	1.16 days	31,709 years	
10^8	0.027 µs	0.10 s	2.660 s	115.70 days	3.17×10^7 years	
10^9	0.030 µs	1.00 s	29.900 s	31.70 years		

[*] 1 ns $= 10^{-9}$ second.
[†] 1 ms $= 10^{-3}$ second.

مقایسه‌ی نمودار توابع رشد



مقایسه‌ی توابع مرتبه



نسبت توابع مرتبه

O

Θ

Ω

اوی بزرگ

BIG-O

f حداکثر از مرتبه‌ی g است:

$$\boxed{f(n) \in O(g(n))}$$

$$\exists c \in \mathbb{R}^+ \, \exists N \in \mathbb{N} \, \forall n \in \mathbb{N} \left[(n \geq N) \Rightarrow |f(n)| \leq c|g(n)| \right]$$

اوی بزرگ

مثال

ثابت کنید:

$$n^2 \in O(n^2 + 10n)$$

$$n \in O(n^2)$$

امگای بزرگ

BIG-OMEGA

f حداقل از مرتبه‌ی g است

$$\boxed{f(n) \in \Omega(g(n))}$$

$$\exists c \in \mathbb{R}^{+}\, \exists N \in \mathbb{N}\, \forall n \in \mathbb{N} [(n \geq N) \Rightarrow |f(n)| \geq c|g(n)|]$$

امگای بزرگ

مثال

ثابت کنید:

$$5n^2 \in \Omega(n^2)$$

$$n^2 + 10n \in \Omega(n^2)$$

تتای بزرگ

BIG-THETA

f از مرتبه‌ی g است

$$\boxed{f(n) \in \Theta(g(n))}$$

$$\exists c_1, c_2 \in \mathbb{R}^+ \, \exists N \in \mathbb{N} \, \forall n \in \mathbb{N} \left[(n \geq N) \Rightarrow c_1 \left| g(n) \right| \leq \left| f(n) \right| \leq c_2 \left| g(n) \right| \right]$$

اوی کوچک

LITTLE-O

f به طور اکید حداکثر از مرتبه‌ی g است

$$\boxed{f(n) \in o(g(n))}$$

$$\exists c \in \mathbb{R}^{+} \, \exists N \in \mathbb{N} \, \forall n \in \mathbb{N} \left[(n \geq N) \Rightarrow \left| f(n) \right| < c \left| g(n) \right| \right]$$

امگای کوچک

LITTLE-OMEGA

f به طور اکید حداقل از مرتبه‌ی g است

$$\boxed{f(n) \in \omega(g(n))}$$

$$\exists c \in \mathbb{R}^+ \, \exists N \in \mathbb{N} \, \forall n \in \mathbb{N} \big[(n \geq N) \Rightarrow \big| f(n) \big| > c \big| g(n) \big| \big]$$

درجه‌ی پیچیدگی

notation	complexity degree
$f \in O(g)$	$F \leq G$
$f \in \Omega(g)$	$F \geq G$
$f \in \Theta(g)$	$F = G$
$f \in \omega(g)$	$F > G$
$f \in o(g)$	$F < G$

خواص بازتابی و ضد بازتابی

خاصیت‌های بازتابی

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

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

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

خاصیت‌های ضد بازتابی

$$f(n) \notin o(f(n))$$

$$f(n) \notin \omega(f(n))$$

خاصیت‌های تقارنی و برگردان تقارنی

خاصیت‌های تقارنی

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

خاصیت‌های برگردان تقارنی

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

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

خاصیت‌های تراگذری

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

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

$$f(n) \in \Theta(g(n)) \wedge g(n) \in \Theta(h(n)) \Rightarrow f(n) \in \Theta(h(n))$$

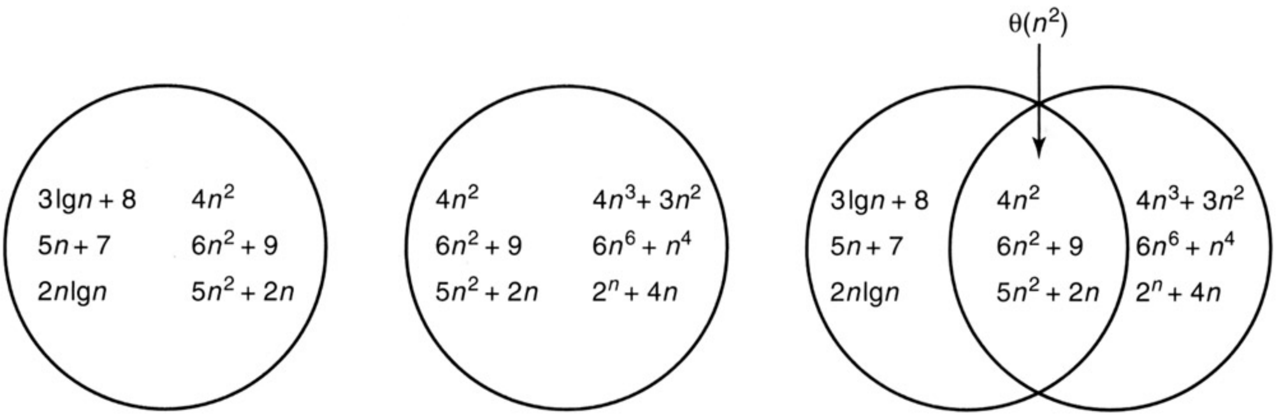
$$f(n) \in \omega(g(n)) \wedge g(n) \in \omega(h(n)) \Rightarrow f(n) \in \omega(h(n))$$

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

قضیه

$$\Theta(g(n)) = O(g(n)) \bigcap \Omega(g(n))$$

مثال



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

قضیه

$$\log_a n \in \Theta(\log_b n)$$

$$a > 1, b > 1$$

قضیه

$$p(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_0 \Rightarrow p(n) \in \Theta(n^k)$$

قضیه

$$a^n \in o(n!)$$

قضیه

اگر

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

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

آن‌گاه

$$c.g(n) + d.h(n) \in \Theta(f(n))$$

$$c,d \in \mathbb{R}^{+}$$

قضیه

$$f(n) \in o(g(n)) \Rightarrow f(n) \in O(g(n))$$

$$f(n) \in \omega(g(n)) \Rightarrow f(n) \in \Omega(g(n))$$

محاسبه‌ی مرتبه با استفاده از حد

$$\lim_{n\to\infty}\left|\frac{f(n)}{g(n)}\right| = \begin{cases} 0 & , f(n) \in o(g(n)) \\ c & , f(n) \in O(g(n)), c \geq 0 \\ c & , f(n) \in \Theta(g(n)), c > 0 \\ c & , f(n) \in \Omega(g(n)), 0 < c \leq \infty \\ \infty & , f(n) \in \omega(g(n)) \end{cases}$$

قاعده‌ی هوپیتال

قاعده‌ی هوپیتال

$$\begin{cases} \lim_{n\to\infty} f(n) = +\infty \\ \lim_{n\to\infty} g(n) = +\infty \end{cases} \Rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)}$$

محاسبه‌ی مرتبه با استفاده از حد

مثال

ثابت کنید:

$$n^2 / 2 \in O(n^2)$$

$$a^n \in o(b^n) \quad , 0 < a < b$$

$$\log n \in O(n)$$

$$\log_a n \in \Theta(\log_b n) \quad , a > 1, b > 1$$

$$\lim_{n\to\infty}\left|\frac{f(n)}{g(n)}\right| = \begin{cases} 0 & , f(n) \in o(g(n)) \\ c & , f(n) \in O(g(n)), c \geq 0 \\ c & , f(n) \in \Theta(g(n)), c > 0 \\ c & , f(n) \in \Omega(g(n)), 0 < c \leq \infty \\ \infty & , f(n) \in \omega(g(n)) \end{cases}$$

Prepared by Kazim Fouladi | Fall 2016 | 2nd Edition
کاظم فولادی