

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ



طراحی و تحلیل الگوریتم‌ها

مبحث دوم

تحلیل الگوریتم‌ها

Analysis of Algorithms

کاظم فولادی قلعه
دانشکده مهندسی، دانشکدگان فارابی
دانشگاه تهران

<http://courses.fouladi.ir/algorithm>

تحلیل الگوریتمها



مقدمه

تحلیل الگوریتم‌ها

فضا

فضای مصرفی برای اجرای الگوریتم

زمان

زمان مصرفی برای اجرای الگوریتم

تحلیل الگوریتم بر اساس یک عمل خاص

مانند: مقایسه، انتساب، تعویض، جمع، ضرب

تحلیل الگوریتم برحسب:

- ورودی

- اندازه‌ی ورودی

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	
$T(n)$	زمان اجرا برای همه‌ی حالات <ul style="list-style-type: none"> • زمان اجرا برای همه‌ی حالات
$W(n): \textit{worst-case}$	زمان اجرا در بدترین حالت <ul style="list-style-type: none"> • بیانگر حداکثر دفعاتی که عمل اصلی اجرا می‌شود.
$A(n): \textit{average-case}$	زمان اجرا در حالت میانگین <ul style="list-style-type: none"> • میانگین (امیدریاضی) تعداد دفعاتی که عمل اصلی انجام می‌شود.
$B(n): \textit{best-case}$	زمان اجرا در بهترین حالت <ul style="list-style-type: none"> • بیانگر حداقل دفعاتی که عمل اصلی اجرا می‌شود.

تحلیل الگوریتمها

۲

زمان اجرا
برای
ساختارهای
کنترل برنامه

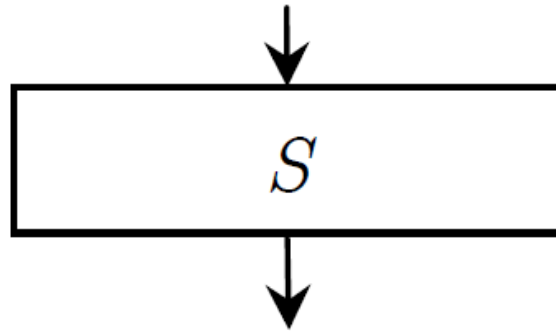
زمان اجرا و ساختارهای ترتیب، تصمیم، تکرار

ساختارهای کنترلی	
Order	ترتیب
	<ul style="list-style-type: none"> • دستور • دستور مرکب
Decision	تصمیم
	<ul style="list-style-type: none"> • دستور شرطی if-then • دستور شرطی if-then-else
Repeat	تکرار
	<ul style="list-style-type: none"> • 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.

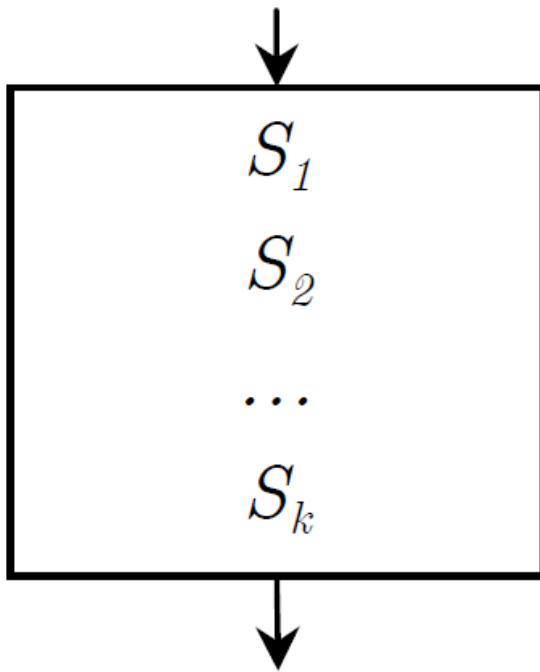


$$T(n)$$

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.



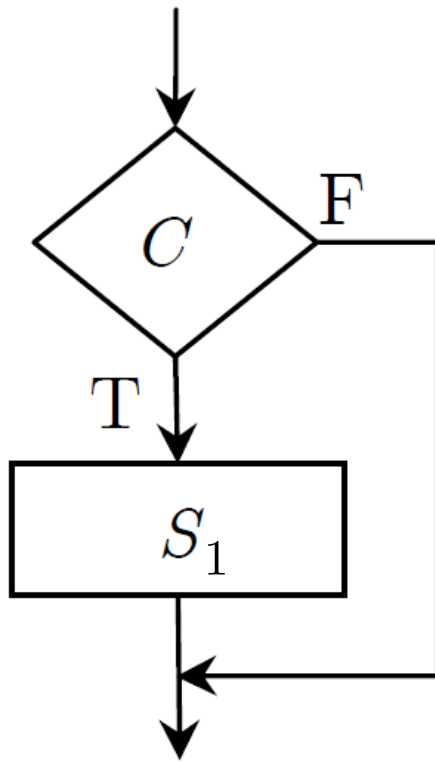
$$S_1; S_2; \dots; S_k$$

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

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:



if C then S

$$B(n) = T_C(n)$$

$$W(n) = T_C(n) + T_1(n)$$

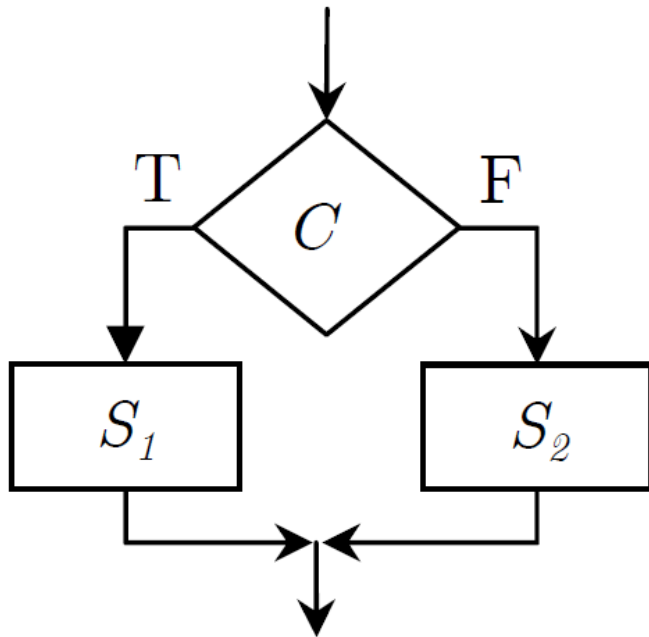
$$A(n) = T_C(n) + p_1(n)T_1(n)$$

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

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

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

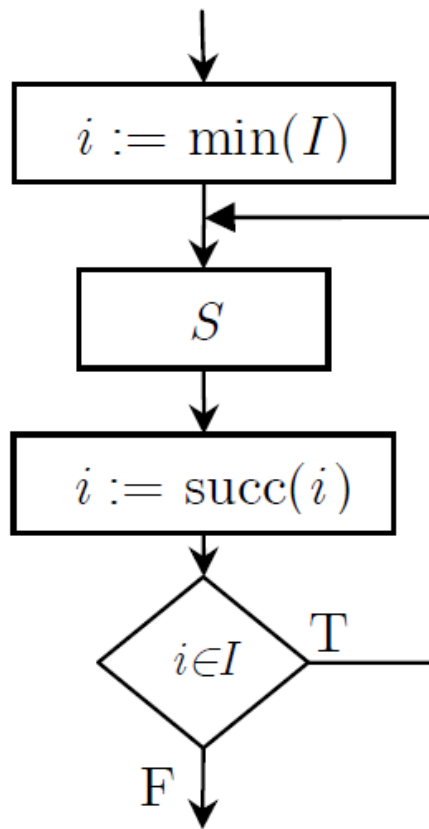
$$A(n) = T_C(n) + p_1(n)T_1(n) + p_2(n)T_2(n)$$

$$p_1 + p_2 = 1$$

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.



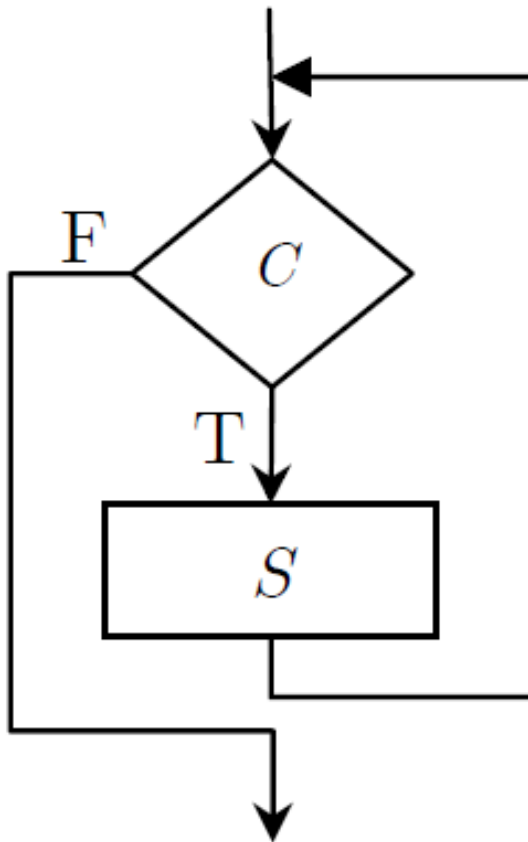
for $i \in I$ **do** S

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

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)



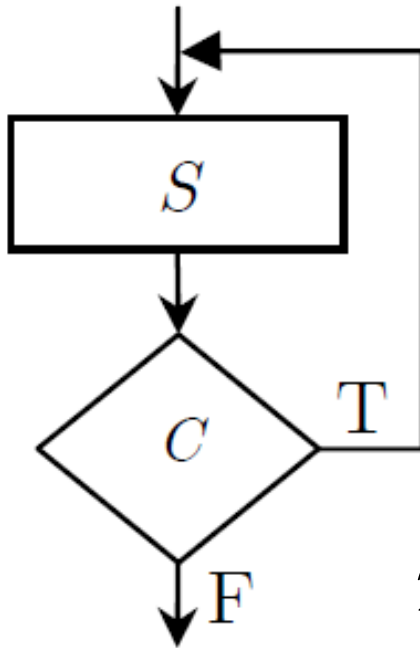
while C do S

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

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)



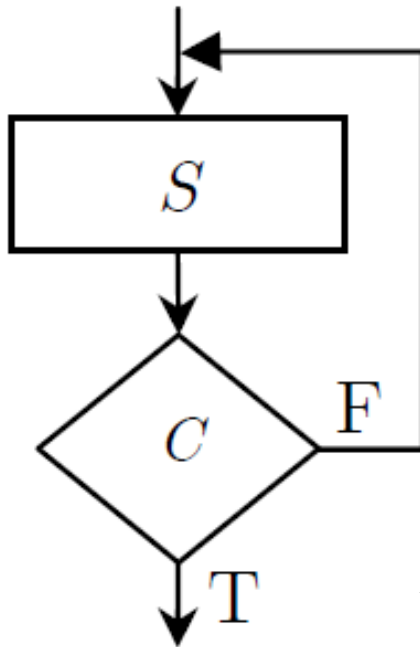
do S while C

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

زمان اجرای 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

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

تحلیل الگوریتمها

۳

مرور
ریاضی

SUMMATION OPERATOR

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

$$\left\{ \begin{array}{l} \sum_{i=m}^n f(i) = f(m) + f(m+1) + \cdots + f(n) \quad , m \leq n \\ \sum_{i=m}^n f(i) = 0 \quad , m > n \end{array} \right.$$

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

عملگر مجموعیابی: خاصیت‌ها

مردور ریاضی

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

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

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

عملگر مجموعیابی: خاصیت‌ها

مردود ریاضی

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

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

عملگر مجموعیابی: خاصیت‌ها

قاعده‌ی تلسکوپی برای مجموعیابی:

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

عملگر مجموعیابی: خاصیت‌ها

مرور ریاضی

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

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

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

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

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

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

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

عملگر مجموع‌یابی: خاصیت‌ها

تقریب انتگرال برای محاسبه‌ی مجموع: اگر f صعودی باشد:

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

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

مرور ریاضی

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

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

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

توابع روی اعداد صحیح

توابع اعداد صحیح و روابط آنها

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

توابع جزء صحیح: ویژگی‌ها

مرور ریاضی

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

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

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

$$\left\lfloor \frac{a}{b} \right\rfloor \leq \frac{a+b-1}{b} \quad \left\lceil \frac{a}{b} \right\rceil \geq \frac{a-b+1}{b}$$

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

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

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

الگوریتم تقسیم:

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

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

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

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

تقریب استرلینگ

ضریب دو جمله‌ای = ترکیب k از n :
 تعداد راه‌های انتخاب k شیء از n شیء وقتی ترتیب مهم نیست.

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

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

ضریب دو جمله‌ای: ویژگی‌ها

مرور ریاضی

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

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

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

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

$$\binom{n}{k}^k \leq \binom{n}{k} \leq \frac{n^k}{k!}, \quad 0 \leq k \leq n$$

تحلیل الگوریتمها

۴

مثالهایی از
محاسبه‌ی
زمان اجرا

مثال

زمان اجرای قطعه الگوریتم‌های زیر را محاسبه کنید:

for $i \leftarrow a$ **to** b **do**

$x \leftarrow x + 1$



for $i \leftarrow a$ **downto** b **do**

$x \leftarrow x + 1$



for $i \leftarrow 1$ **to** n **do**

$x \leftarrow x + 1$



for $i \leftarrow n$ **downto** 1 **do**

$x \leftarrow x + 1$



مثال

زمان اجرای قطعه الگوریتم زیر را محاسبه کنید:

```
for  $i \leftarrow 1$  to  $n$  do  
  for  $j \leftarrow 1$  to  $n$  do  
     $x \leftarrow x / 2$ 
```



مثال

زمان اجرای قطعه الگوریتم زیر را محاسبه کنید:

```
for  $i \leftarrow 1$  to  $n$  do  
  for  $j \leftarrow 1$  to  $n$  do  
    for  $k \leftarrow j$  to  $n$  do  
       $x \leftarrow x / 2$ 
```



مثال

زمان اجرای قطعه الگوریتم زیر را محاسبه کنید:

```
for  $i \leftarrow 1$  to  $n$  do  
  for  $j \leftarrow i$  to  $n$  do  
    print  $i + j$ 
```



مثال

زمان اجرای قطعه الگوریتم زیر را محاسبه کنید:

```

$$i \leftarrow n$$

$$x \leftarrow k$$
while  $i > 0$  do
$$x \leftarrow x + 1$$

$$i \leftarrow i - 1$$

```



مثال

زمان اجرای قطعه الگوریتم زیر را محاسبه کنید:

```

$$i \leftarrow n$$
while  $i \geq 1$  do  
     $i \leftarrow i / 2$ 
```

۹

تحلیل الگوریتمها

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

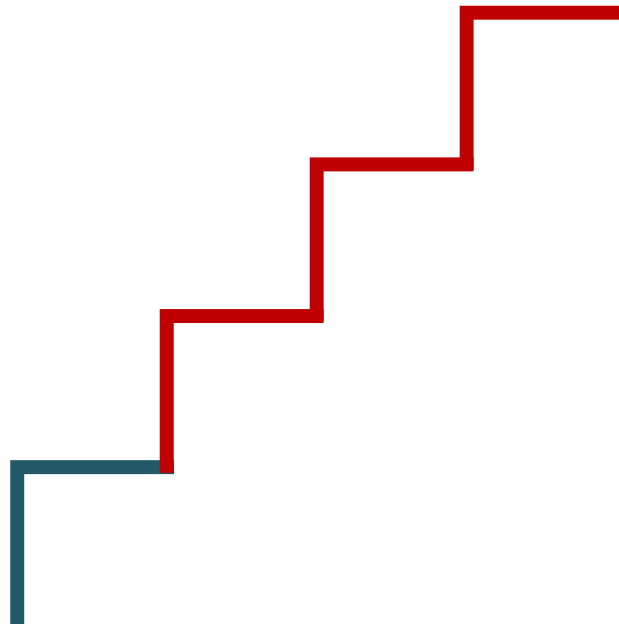
RECURRENCE

مثال

چگونه از یک پلکان بالا برویم؟

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

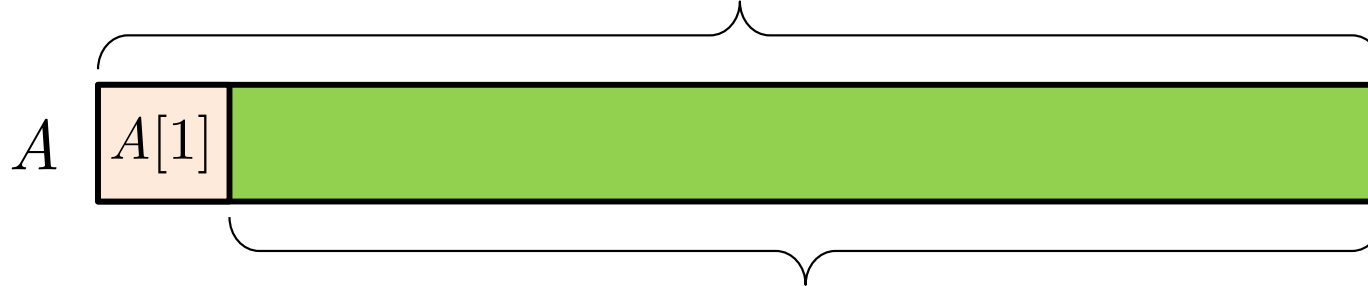
- [شرط اولیه] بالا رفتن از یک پله را بلدیم!
- [بازگشت] بالا از رفتن از یک پلکان (با n پله) عبارت است از
 - بالا رفتن از یک پله +
 - بالا رفتن از یک پلکان (با $n-1$ پله)



بازگشت

مثال

یک لیست



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

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

$$\underbrace{\hspace{10em}}_{S(n)}$$

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

$$n > 0$$

برنامه‌نویسی بازگشتی

مثال (برنامه‌ی کامپیوتری)

برنامه‌ای برای محاسبه‌ی مجموع اعداد 1 تا n

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

$$\underbrace{\hspace{10em}}_{S(n)}$$

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

$$n > 0$$

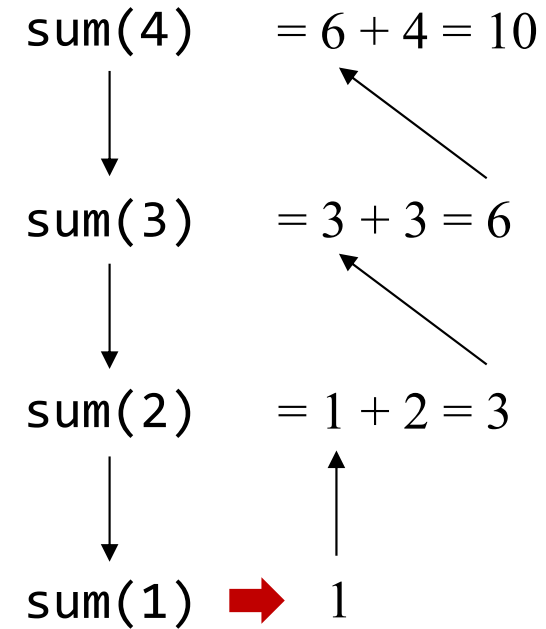
```
int sum (int n)
{
    if (n == 1) return 1;
    return sum(n-1) + n;
}
```

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;
}
```



برنامه‌نویسی بازگشتی

اجتناب از حلقه‌های نامتناهی

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:

```

*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
g (7)
void g (int n)
{
  //precondition: n > 0
  if ( n == 1 ) // base case
    cout << "*" << endl;
  else // general case
  {
    g (n - 1);
    cout << endl;
    f (n);
    cout << endl;
  }
}

```

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.

الگوریتم‌های بازگشتی

RECURSIVE ALGORITHMS

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

انواع الگوریتم بازگشتی	
<i>Direct Recursive</i>	بازگشتی مستقیم
	• فراخوانی بی‌واسطه $A \rightarrow A$
<i>Indirect Recursive</i>	بازگشتی غیرمستقیم
	• 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

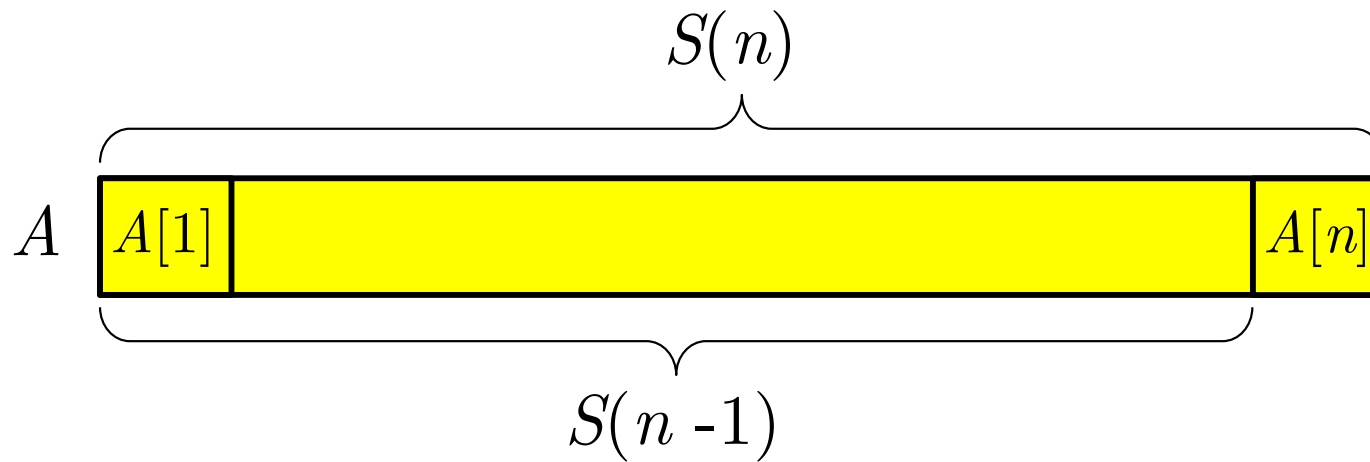
الگوریتم‌های بازگشتی روشی مستقیم برای محاسبه‌ی توابع بازگشتی است:

- 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

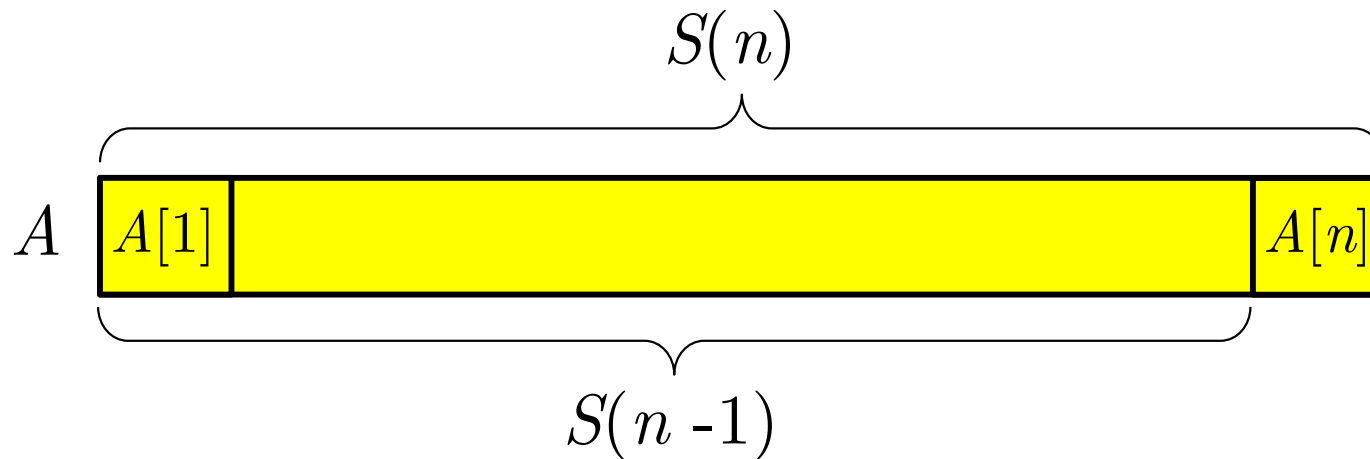


$$S(n) = \begin{cases} S(n-1) + A[n] & , n > 1 \\ A[n] & , n = 1 \end{cases}$$

الگوریتم‌های بازگشتی

مثال

الگوریتم بازگشتی برای محاسبه‌ی مجموع یک آرایه‌ی n عضوی



RECURSIVE-SUM(A, n)

if $n = 1$ **then**

return $A[n]$

else

return RECURSIVE-SUM($A, n - 1$) + $A[n]$

الگوریتم‌های بازگشتی

محاسبه‌ی زمان اجرا

زمان اجرای الگوریتم بازگشتی زیر را محاسبه کنید:

 $T(n)$

RECURSIVE-SUM(A, n)

if $n = 1$ **then**

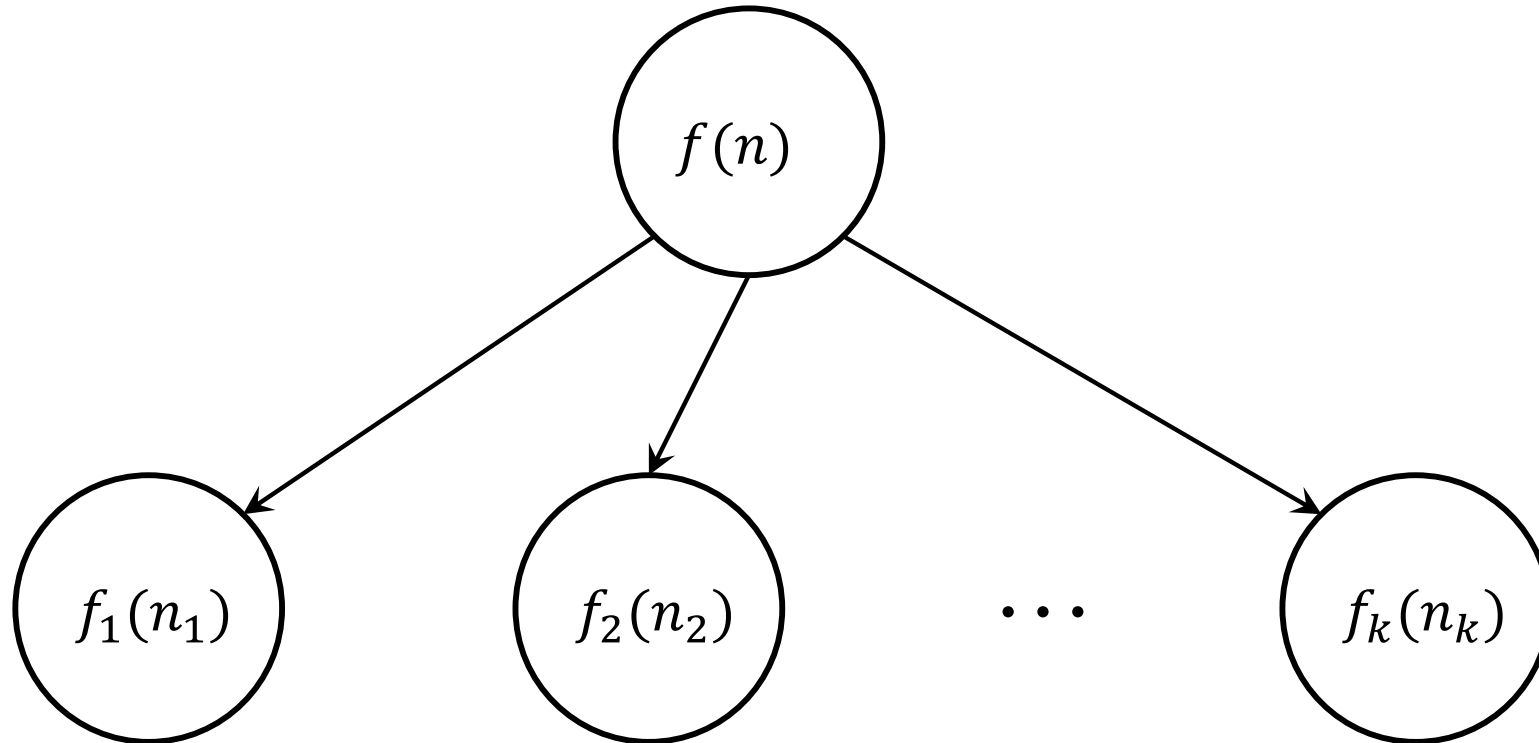
return $A[n]$

else

return RECURSIVE-SUM($A, n - 1$) + $A[n]$

 $T(n - 1)$
 $+$
 $A[n]$

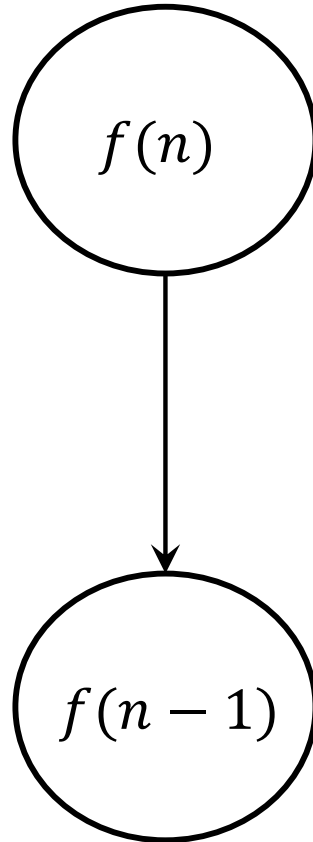
درخت فراخوانی

CALLING TREE

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

درخت فراخوانی

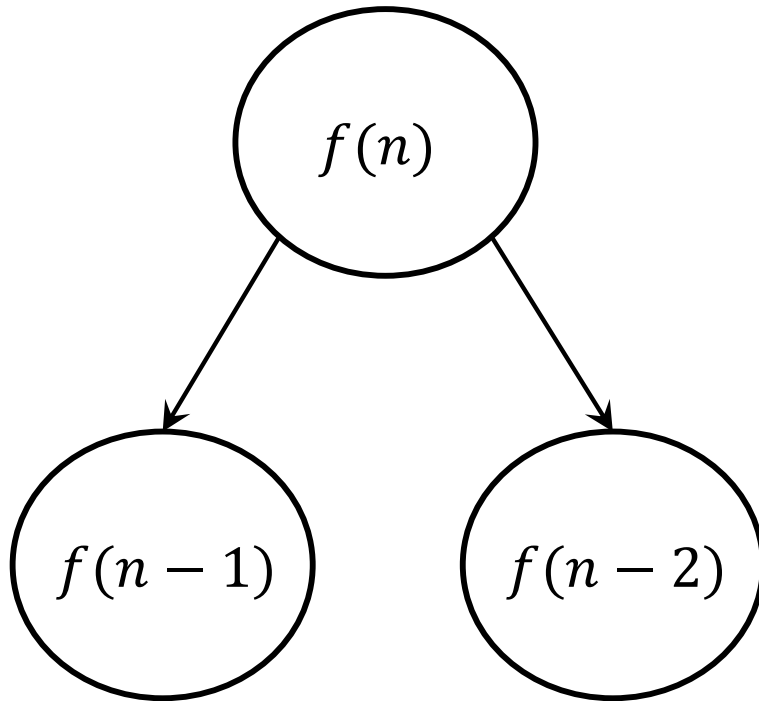
مثال



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

درخت فراخوانی

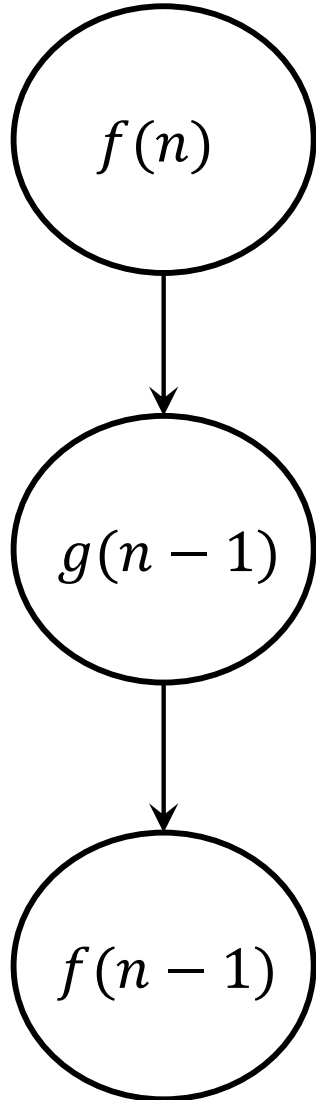
مثال



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

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.

$$n \in \mathbb{Z}^{\geq 0}$$

مثال: مناسبه توان x^n (عدد صحیح نامنفی n ، عدد حقیقی x)

$$x \in \mathbb{R}$$

روش اول:

تابع: $\text{power}(x, n) \triangleq x^n$

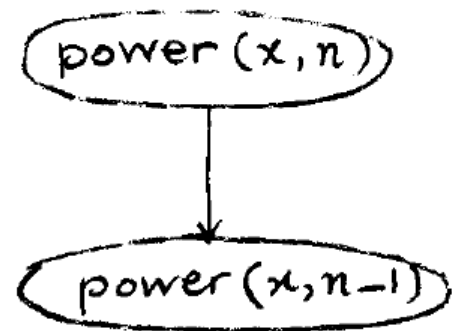
$$\begin{cases} x^n = x \cdot x^{n-1} & \Rightarrow \text{power}(x, n) = x \cdot \text{power}(x, n-1) \\ x^0 = 1 & \Rightarrow \text{power}(x, 0) = 1 \end{cases}$$

$\text{power}(x, n)$

if $n = 0$ then return 1 \triangleright شرط اولیه

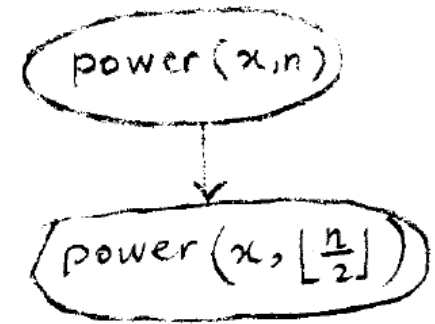
return $x * \text{power}(x, n-1)$

زمان اجرا بر حسب تعداد ضرب: $T(0) = 0, T(n) = T(n-1) + 1$



روش دوم:

$$\begin{cases} x^n = x^{\frac{n}{2}} x^{\frac{n}{2}} & , n \text{ زوج} \\ x^n = x^{\lfloor \frac{n}{2} \rfloor} x^{\lfloor \frac{n}{2} \rfloor} x & , n \text{ فرد} \\ x^0 = 1 \end{cases}$$



power(x, n)

if $n = 0$ then return 1 \triangleright شرط اولیه

else if n is even

$y \leftarrow \text{power}(x, \frac{n}{2})$

return $y * y$

else

$y \leftarrow \text{power}(x, \lfloor \frac{n}{2} \rfloor)$

return $y * y * x$

زمان اجرا بر حسب تعداد ضرب

$$T(0) = 0$$

$$T(n) = \begin{cases} T(\frac{n}{2}) + 1 & n \text{ زوج} \\ T(\lfloor \frac{n}{2} \rfloor) + 2 & n \text{ فرد} \end{cases}$$

معادلات خطی مرتبه اول

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

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

(۱) معادلات خطی مرتبه اول :

$$Q(n) = \prod_{i=1}^n p(i)$$

برای حل این معادله، ابتدا مقدار

را محاسبه کرده، طرفین معادله را بر این عبارت تقسیم می‌کنیم :

$$\frac{f(n)}{Q(n)} = \frac{f(n-1)}{Q(n-1)} + \frac{q(n)}{Q(n)}$$

$$\sum_{n'=1}^n \left\{ \frac{f(n')}{Q(n')} - \frac{f(n'-1)}{Q(n'-1)} \right\} = \sum_{n'=1}^n \underbrace{\left\{ \frac{q(n')}{Q(n')} \right\}}_{R(n')}$$

قاعده اعداد

$$\frac{f(n)}{Q(n)} - \frac{f(0)}{Q(0)} = \sum_{n'=1}^n R(n') \Rightarrow$$

$$f(n) = Q(n) \left(\frac{f(0)}{Q(0)} + \sum_{n'=1}^n R(n') \right)$$

مثال :

$$f(n) = 2f(n-1) + 1, \quad f(0) = 1$$

$$Q(n) = \prod_{n'=1}^n 2 = 2^n$$

$$\begin{aligned} \frac{f(n)}{2^n} &= \frac{f(n-1)}{2^{n-1}} + \frac{1}{2^n} \Rightarrow \sum_{n'=1}^n \left\{ \frac{f(n')}{2^{n'}} - \frac{f(n'-1)}{2^{n'-1}} \right\} = \sum_{n'=1}^n \left(\frac{1}{2} \right)^{n'} \\ &\Rightarrow \frac{f(n)}{2^n} - \frac{f(0)}{2^0} = \frac{1}{2} \frac{1 - (1/2)^n}{1 - 1/2} \\ &\Rightarrow f(n) = 2^n \left(1 - \left(\frac{1}{2} \right)^n \right) \\ &= 2^n - 1 \end{aligned}$$

حل معادلات بازگشتی

معادلات خطی مرتبه k ام همگن با ضرایب ثابت

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

معادله خطی مرتبه دوم ممکن با ضرایب ثابت :

با دو شرط اولیه

$$f(n) = A f(n-1) + B f(n-2)$$

معادله مشخصه $x^2 = Ax + B \Rightarrow x^2 - Ax - B = 0$

$$\Delta = A^2 + 4B$$

$$\Delta > 0 \Rightarrow x = x_1 \text{ و } x = x_2 \Rightarrow f(n) = C_1 x_1^n + C_2 x_2^n$$

$$\Delta = 0 \Rightarrow x_1 = x_2 = x \Rightarrow f(n) = (C_1 + C_2 x) x^n$$

$$\Delta < 0 \Rightarrow x_1, x_2 = \lambda \pm \mu i \Rightarrow f(n) = \dots ?$$

حل معادلات بازگشتی

مردور ریاضی

معادلات خطی مرتبه k ام ناهمگن با ضرایب ثابت

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

معادلات تقسیم و غلبه

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

$$n = b^k$$

معادلات تقسیم و علیهم :

$$f(n) = p(n) f\left(\frac{n}{b}\right) + q(n) \quad n \in \{1, b, b^2, b^3, \dots\}$$

تبدیل به معادله خطی مرتبه اول $\Rightarrow n = b^k$; تغییر متغیر $k \in \mathbb{Z}^{\geq 0}$

حل معادلات بازگشتی: حالات خاص

مردود ریاضی

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:

```
for  $i \leftarrow 1$  to  $n$  do
  for  $j \leftarrow n$  to  $i$  do
    print  $f(i)$ 
```

```
 $f(n)$ 
  if  $n = 0$  then
    return 1
  else
    return  $f(n - 1) * n$ 
```

```

for i := 1 to n do
  for j := n to i do
    print(f(i));

```

```

f(n)
{
  if (n = 0) then return 1
  return f(n-1).n
}

```

تعداد ضرب ها؟

$$\begin{aligned}
 T(n) &= \sum_{i=1}^n \sum_{j=n}^i T_f(i) \\
 &= \sum_{i=1}^n \sum_{j=n}^i i = \sum_{i=1}^n i \sum_{j=n}^i 1 \\
 &= \frac{n(n+1)}{2} \sum_{j=n}^n 1 = \frac{n(n+1)}{2}
 \end{aligned}$$

$$T_f(n) = 1 + T_f(n-1)$$

$$T_f(0) = 0$$

$$T_f(n) = n$$

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
```

مثال: در قطعه کد زیر، دستور print چند بار اجرا می شود؟
 ($n = 2^k$)

```
function f(n) : integer
{
    if (n = 1) then return 1;

    print(f(n/2));

    print(f(n/2));

    return 0;
}
```

$$T(n) = 2T\left(\frac{n}{2}\right) + 2$$

$$T(1) = 0$$

$$T(2^k) = 2T(2^{k-1}) + 2$$

$$\frac{T(2^k)}{2^k} = \frac{T(2^{k-1})}{2^{k-1}} + \left(\frac{1}{2}\right)^{k-1}$$

$$\sum_{i=1}^k \left\{ \frac{T(2^i)}{2^i} - \frac{T(2^{i-1})}{2^{i-1}} \right\} = 2 \sum_{i=1}^k \left(\frac{1}{2}\right)^i$$

$$\frac{T(2^k)}{2^k} - \frac{T(1)}{1} = 2 \times \frac{1}{2} \frac{1 - \left(\frac{1}{2}\right)^k}{1 - 1/2}$$

$$T(2^k) = 2^{k+1} \left(1 - \frac{1}{2^k}\right)$$

$$= 2^{k+1} - 2$$

$$T(n) = 2n - 2$$

تحلیل الگوریتمها

٦

تحلیل
مجانبي
زمان اجرا

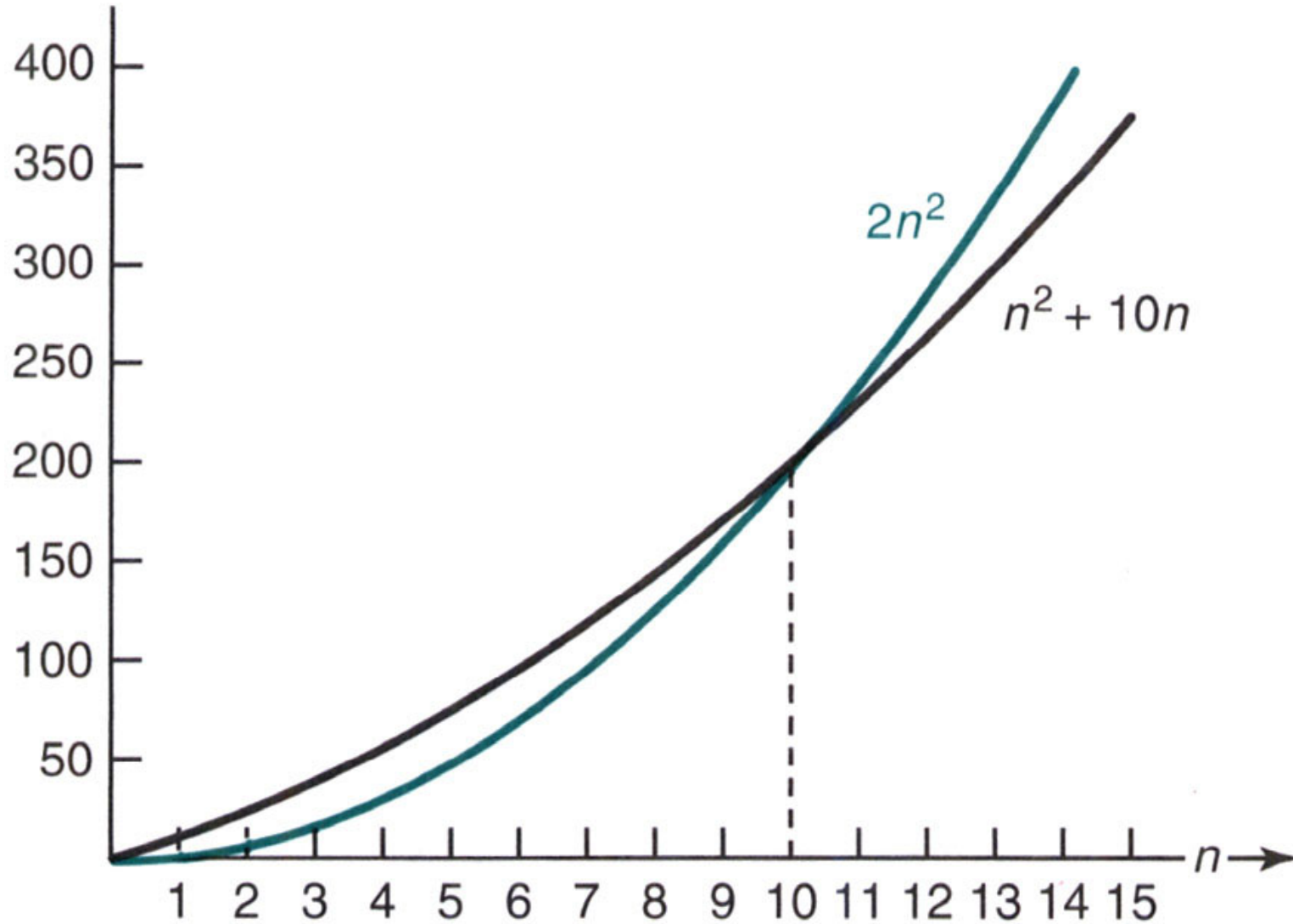
تحلیل مجانبی زمان اجرا

اگر چه 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

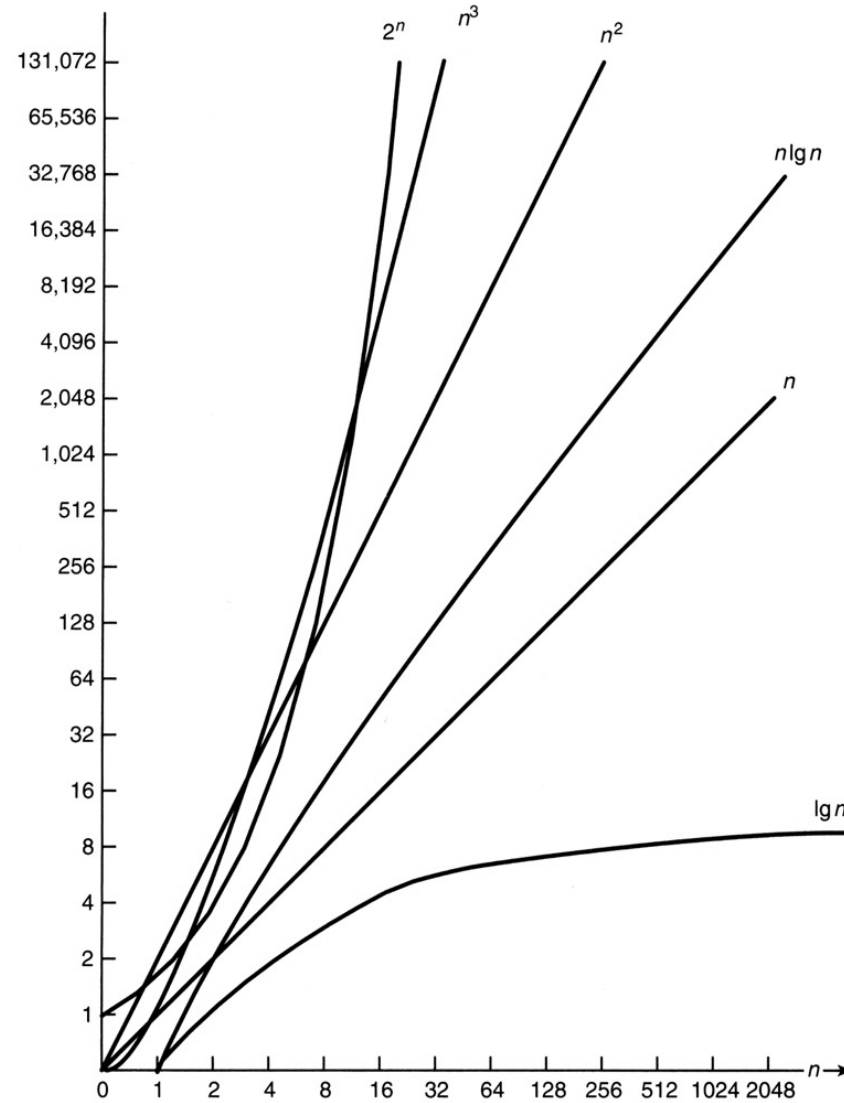


مقایسه‌ی توابع رشد

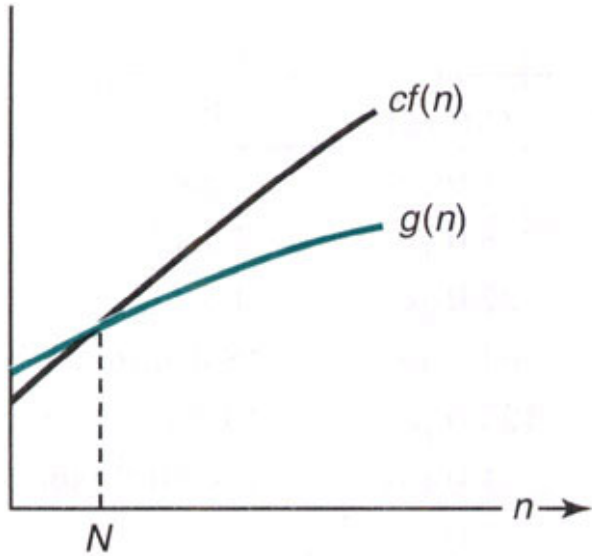
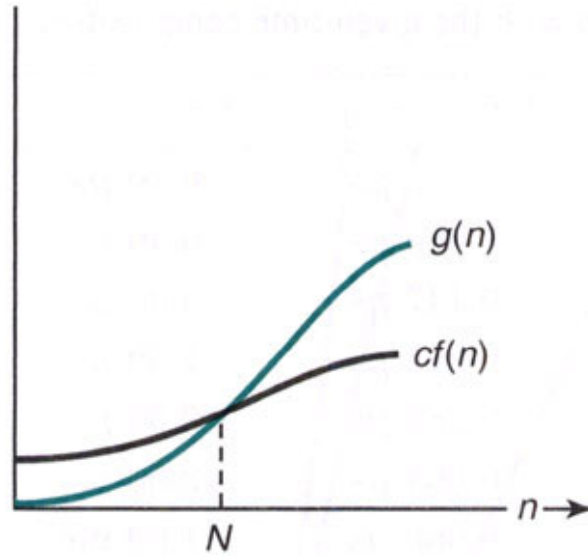
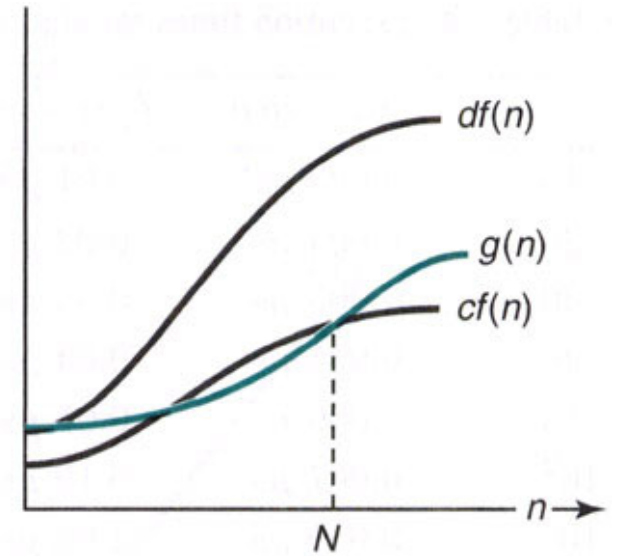
n	$f(n) = \lg n$	$f(n) = n$	$f(n) = n \lg n$	$f(n) = n^2$	$f(n) = n^3$	$f(n) = 2^n$
10	0.003 μs ^[*]	0.01 μs	0.033 μs	0.10 μs	1.0 μs	1 μs
20	0.004 μs	0.02 μs	0.086 μs	0.40 μs	8.0 μs	1 ms ^[†]
30	0.005 μs	0.03 μs	0.147 μs	0.90 μs	27.0 μs	1 s
40	0.005 μs	0.04 μs	0.213 μs	1.60 μs	64.0 μs	18.3 min
50	0.006 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

(a) $g(n) \in O(f(n))$ (b) $g(n) \in \Omega(f(n))$ (c) $g(n) \in \Theta(f(n))$

نسبت توابع مرتبه

Ω

اوی بزرگ

BIG-O

f حداکثر از مرتبه‌ی g است:

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

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

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

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

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

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

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

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

اوی کوچک

LITTLE-O

f به طور اکید حداکثر از مرتبه‌ی g است

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

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

امگای کوچک

LITTLE-OMEGA

f به طور اکید حداقل از مرتبه‌ی g است

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

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

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

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

خاصیت‌های بازتابی

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

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

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

خاصیت‌های ضد بازتابی

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

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

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

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

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

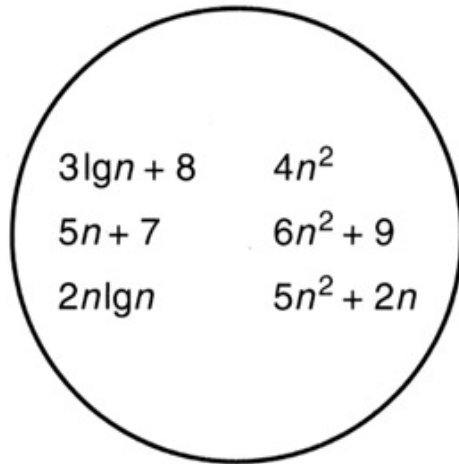
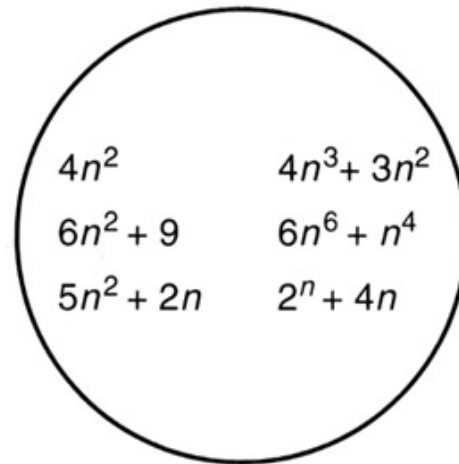
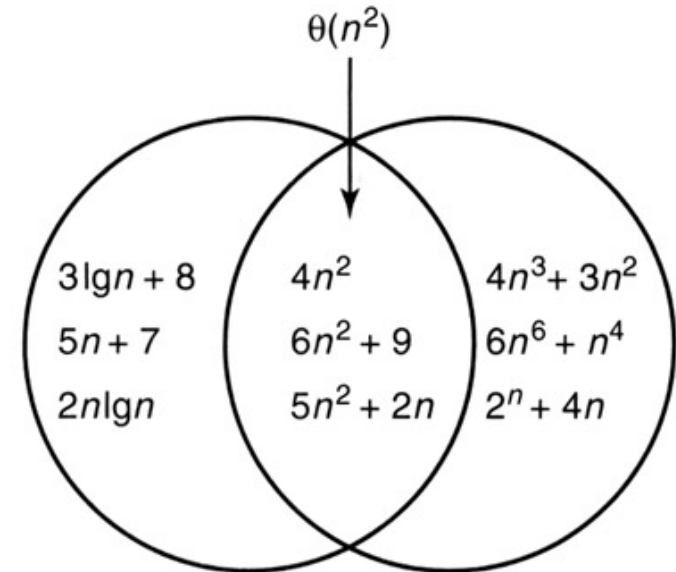
$$f(n) \in \omega(g(n)) \wedge g(n) \in \omega(h(n)) \Rightarrow f(n) \in \omega(h(n))$$

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

قضیه

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

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

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

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

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

قاعده‌ی هوییتال

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

محاسبه‌ی مرتبه با استفاده از حد

مثال

ثابت کنید:

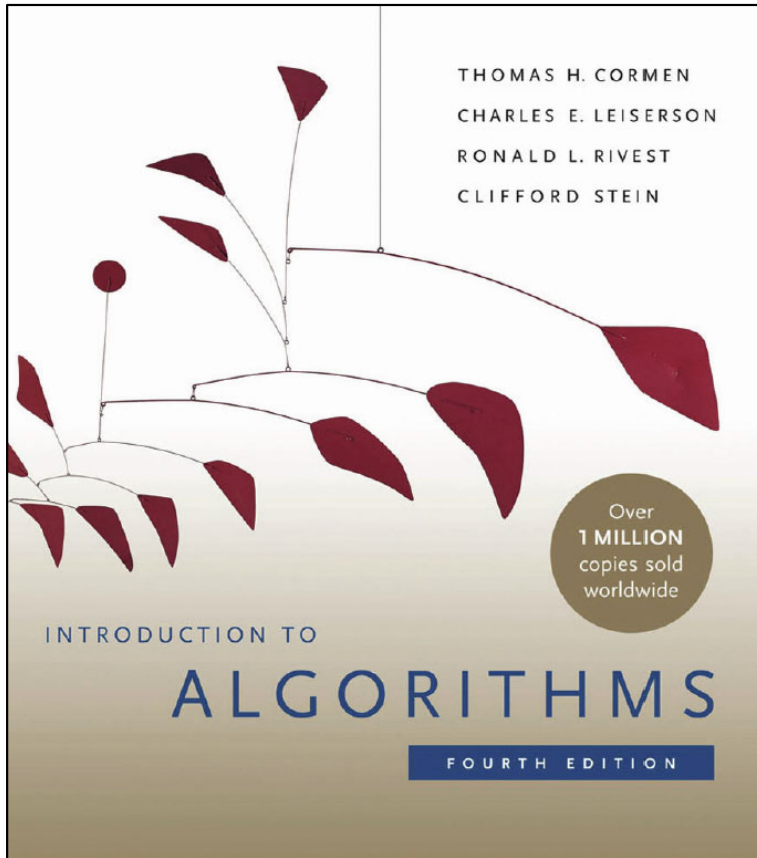
$$n^2 / 2 \in O(n^2)$$

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

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

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

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



T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein,
Introduction to Algorithms,
 4th Edition, MIT Press, 2022.

Chapter 1

1 The Role of Algorithms in Computing

What are algorithms? Why is the study of algorithms worthwhile? What is the role of algorithms relative to other technologies used in computers? This chapter will answer these questions.

1.1 Algorithms

Informally, an *algorithm* is any well-defined computational procedure that takes some value, or set of values, as *input* and produces some value, or set of values, as *output* in a finite amount of time. An algorithm is thus a sequence of computational steps that transform the input into the output.

You can also view an algorithm as a tool for solving a well-specified *computational problem*. The statement of the problem specifies in general terms the desired input/output relationship for problem instances, typically of arbitrarily large size. The algorithm describes a specific computational procedure for achieving that input/output relationship for all problem instances.

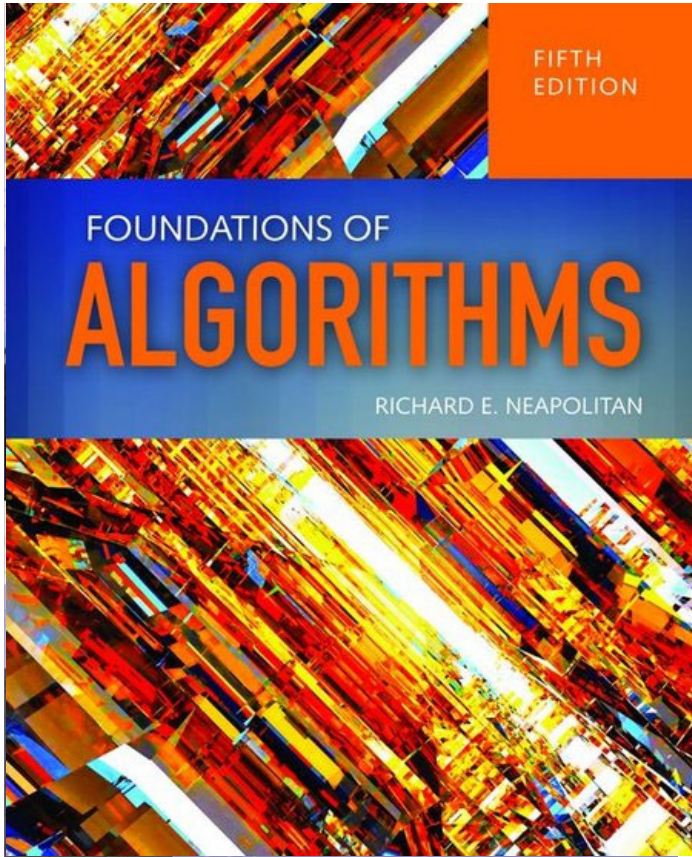
As an example, suppose that you need to sort a sequence of numbers into monotonically increasing order. This problem arises frequently in practice and provides fertile ground for introducing many standard design techniques and analysis tools. Here is how we formally define the *sorting problem*:

Input: A sequence of n numbers $\langle a_1, a_2, \dots, a_n \rangle$.

Output: A permutation (reordering) $\langle a'_1, a'_2, \dots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \dots \leq a'_n$.

Thus, given the input sequence $\langle 31, 41, 59, 26, 41, 58 \rangle$, a correct sorting algorithm returns as output the sequence $\langle 26, 31, 41, 41, 58, 59 \rangle$. Such an input sequence is

مرجع



R.E. Neapolitan,
Foundations of Algorithms,
5th Edition, Jones and Bartlett Publishers, 2015.

Chapter 1