ساختمان‌های داده

Data Structures

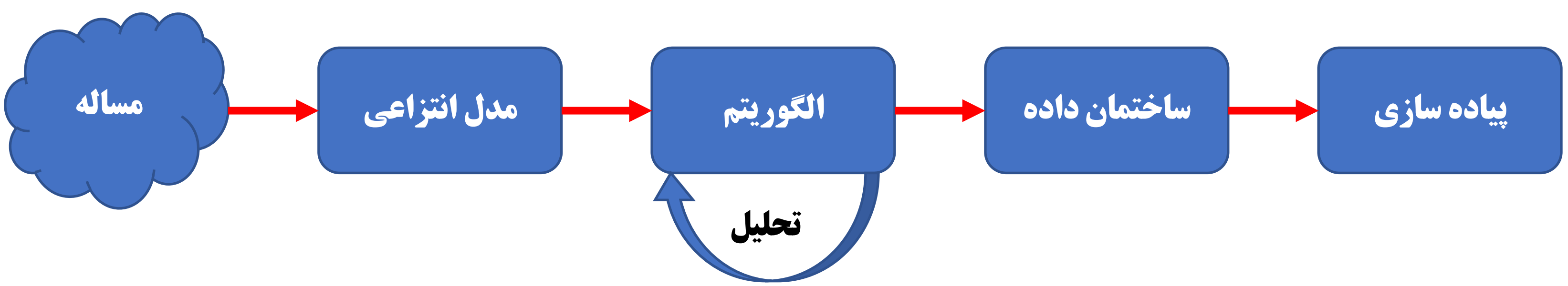
Parand Islamic Azad University (PIAU)
H.R. Imanikia

مقدمه

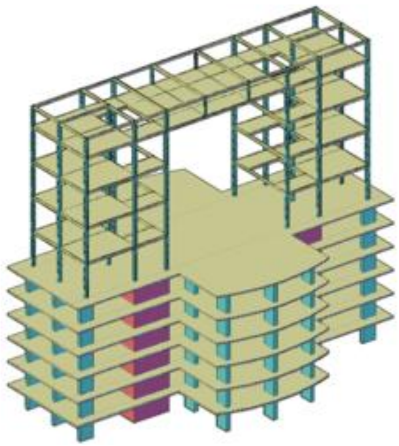
مقدمه

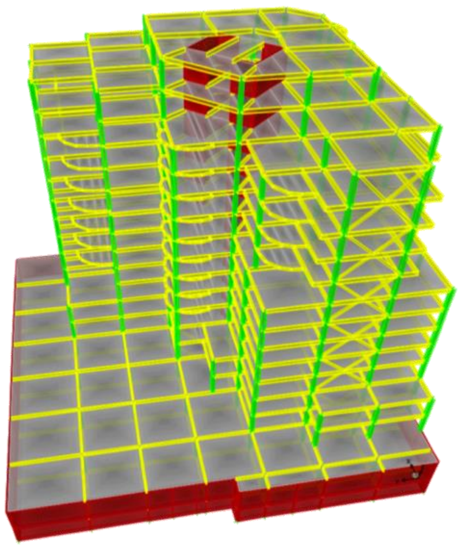
- مراحل حل مساله

1. ایجاد یک مدل انتزاعی از مساله – مدل
2. یافتن الگوریتم برای حل مدل – الگوریتم
3. تحلیل الگوریتم برای حل کارا – الگوریتم صحیح و کارا
4. تعریف ساختمان داده ها – ساختمان داده
5. پیاده سازی – برنامه قابل اجرا بر روی ماشین
6. ...



ساختمان داده ها





❖چند تعریف

❑به مدل ریاضی سازماندهی داده ها، ساختمان داده ها گفته می شود.

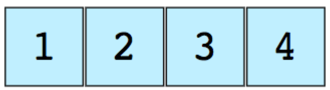
❑ به ساختارهایی که جهت ذخیره سازی، بازیابی و ... اطلاعات به کار می رود، ساختمان داده ها گفته می شود.

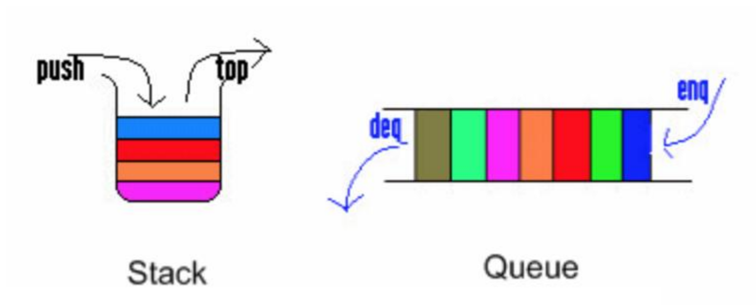
❑ به ساختارهایی که جهت دریافت داده های خام به شکل مناسب توسط کامپیوتر برای پیاده سازی و اجرای الگوریتم های مختلف مورد استفاده قرار می گیرد، ساختمان داده ها گفته می شود.

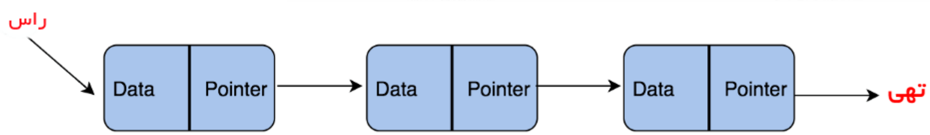
❑ساختمان داده‌ها روش‌های ذخیره داده‌ها در رایانه با هدف دسترسی آسان‌تر و بهینه تر است در حالی که الگوریتم روشی به منظور حل مسئله به وسیله کامپیوتر است.

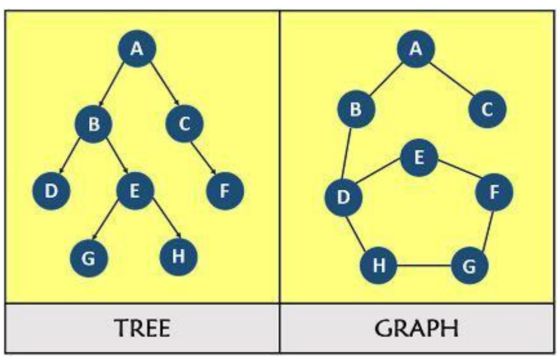
ساختمان داده های مرسوم

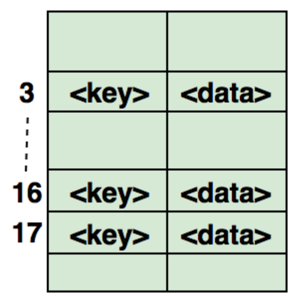
- ❖آرایه Array
- ❖پشته Stack
- ❖صف Queue
- ❖لیست پیوندی Linked List
- ❖درخت Tree
- ❖گراف Graph
- ❖جدول درهم سازی Hash Table



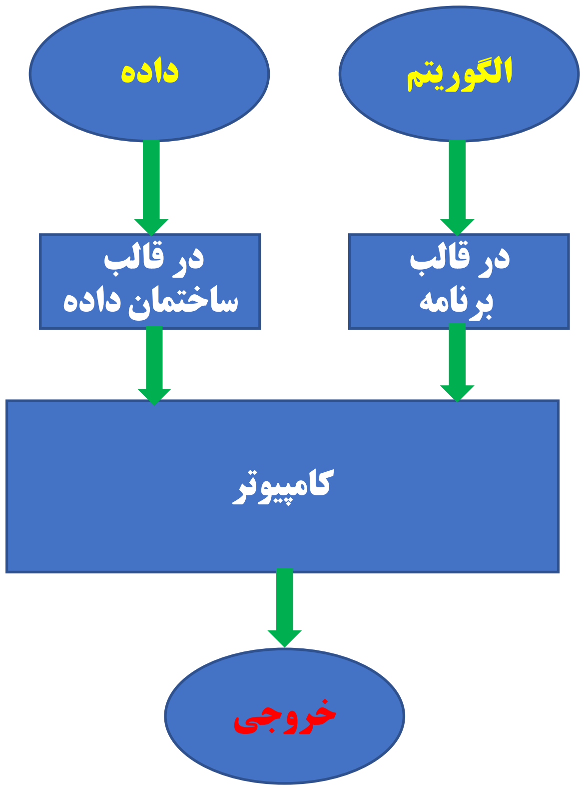








عناصر اساسی در کامپیوتر



- برای انجام هر عملی در کامپیوتر به دو عنصر مهم نیاز داریم:

1) الگوریتم : که باید مناسب و کارا باشد

2) ساختمان داده مناسب : تا بتوان بر اساس آن داده ها را به صورت درستی سازماندهی نمود.

الگوریتم

- الگوریتم یک توالی صریح، دقیق، بدون ابهام و قابل اجرا به لحاظ مکانیکی از دستورات اولیه است که معمولا برای انجام کار و هدف خاصی، تعبیه شده‌اند.
- الگوریتم روال یا فرمولی برای حل یک مسئله بر مبنای انجام یک توالی از فعالیت‌ها است.

معیار برتری یک الگوریتم (نسبت به الگوریتم دیگر)

- برای انجام هر مساله ای، الگوریتم های متفاوتی وجود دارد.
- بایستی کاراترین راه حل برای حل مساله را پیدا کنیم.
- معیار کارایی یک الگوریتم به دو فاکتور بستگی دارد:
  - زمان اجرای الگوریتم
  - میزان حافظه مصرفی

معیار برتری یک الگوریتم

- معیار زمان اجرا وابسته به ماشینی است که الگوریتم بر روی آن اجرا می شود.
- معیار حافظه مصرفی نیز به روش مدیریت حافظه توسط سیستم عامل بستگی دارد.
- بنابراین برای مقایسه دو الگوریتم بایستی شرایط کاملا یکسانی برای آنها فراهم نمود.
- برای فراهم آوردن شرایط یکسان، بایستی معیارهای فوق را به گونه ای تعریف نمود که مستقل از ماشین و سیستم عامل باشد.
- برای این منظور دو معیار زیر تعریف می شوند:
  - مرتبه زمانی اجرای الگوریتم
  - مرتبه مکانی اجرای الگوریتم
- **در این درس معیار مرتبه زمانی عامل مهمتری است و فقط به آن پرداخته می شود.**

پیچیدگی زمانی

- به شاخصی نیاز داریم که مستقل از کامپیوتر و زبان برنامه نویسی باشد.
- به طور کلی زمان اجرای یک الگوریتم با افزایش اندازه ورودی(n) زیاد می شود و زمان اجراء با تعداد دفعاتی که عملیات اصلی انجام می شود تناسب دارد.
- بنابراین بازدهی الگوریتم را با تعیین تعداد دفعاتی که یک عمل اصلی انجام می شود، به عنوان تابعی از ورودی تحلیل می کنیم.
- معمولا پیچیدگی زمانی الگوریتم را با T(n) نشان می دهند.

مثال : پیچیدگی زمانی

❑ در تابع زیر (به زبان سی، سی پلاس پلاس) که جمع عناصر یک آرایه را حساب می کند، دستور اصلی و پیچیدگی زمانی را به دست آورید.

```
float sum (float list[ ], int n)
{
    float s=0; int i;
    for (i = 0; i<n; i++)
        s=s + list[i];
    return s;
}
```

- در این برنامه اندازه ورودی (اندازه آرایه) : برابر n است.
- عمل اصلی هم **s=s + list[i];** می باشد: که n بار تکرار می شود.
- پیچیدگی زمانی: $T(n) = n$

مثال : محاسبه کل مراحل یک تابع

- در تابع زیر (به زبان سی، سی پلاس پلاس) که جمع عناصر یک آرایه را حساب می کند، تعداد کل مراحل را به دست آورید.
- فرض می شود که زمان اجرای هر دستور ساده = ۱ واحد زمانی
- دستورات int i, { , } دستوراتی نیستند که توسط CPU اجرا شوند.
- همانطور که گفته شد اگر عمل اصلی **s=s + list[i];** را در نظر بگیریم T(n) = n خواهد شد.
- نکته : خود حلقه یکی بیشتر از داخل آن اجرا می شود. (چرا؟)

دستور	دفعات اجرا	زمان اجرا	مجموع
float sum (float list[ ], int n)	1	0	0
{	1	0	0
float s=0;	1	1	1
int i;	1	0	0
for (i = 0; i<n; i++)	n+1	1	**n+1**
s=s + list[i];	n	1	**n**
return s;	1	1	1
}	1	0	0
			2n+3

شمارش گام ها یا مراحل یک برنامه

```
for (i=a ; i<=b ; i=i+k )
        s;
```

$$\frac{b-a+1}{k}$$

```
for (i=b ; i>=a ; i=i-k )
        s;
```

```
for ( i=1 ; i<=n ; i=i+1 )
        s;
```

$$\frac{n-1+1}{1}=n$$

```
for ( i=3 ;i<=n ; i=i+2  )
        s;
```

$$\frac{n-3+1}{2}=\frac{n}{2}-1$$

```
for ( i=9 ;i<3n+4; i=i+5)
        s;
```

$$\frac{3n+4-9}{5}=\frac{3n}{5}-1$$

مثال

- تعداد تکرار برای دو حلقه تو در توی مستقل زیر را حساب کنید.
  - نکته : حلقه های i و j مستقل از هم هستند (ارتباطی بین آنها نیست)

دستور	دفعات اجرا
For i:=1 to m do	m+1
For j:=1 to n do	m(n+1)
s := s + 1	mn
	2mn + 2m + 1

- پیچیدگی زمانی :
  - برای دستور اصلی که s=s+1 است ← mn

مثال

- تعداد تکرار برای دو حلقه تو دو توی وابسته زیر را حساب کنید.

دستور (پاسکال)	دستور (سی)
For j:=1 to n do	for(j=1; i<=n; j++)
For i:=1 to j do	for(i=1; j<=j; i++)
x := x + 1	x = x+1 ;

• نکته : حلقه های i و j به هم وابسته هستند.

○ دفعات اجرای حلقه داخلی وابسته به حلقه خارجی است.

j	تغییرات i	تعداد اجرا شدن دستور اصلی
1	1	1 بار
2	1,2	2 بار
3	1,2,3	3 بار
	......	
n	1,2,3,...,n	n بار

$x:=x+1;$ تعداد اجرا شدن دستور $=1+2+3+\ldots+n=\frac{n(n+1)}{2}$

- پیچیدگی زمانی :
- برای دستور اصلی که x=x+1 است ← n(n+1)/2

مثال

- تعداد تکرار برای اجرا شدن دستور اصلی ($x:=x+1;$)را حساب کنید.

دستور (پاسکال)	دستور (سی)
i := n;	i = n;
while (i>1) do	while (i>1)
begin	{
x := x + 1;	x = x + 1;
i:= i div 2;	i = i / 2;
end;	}

- نکته : دفعات اجرا بصورت لگاریتمی کاهش می یابد
- اگر n=16 باشد:

i	شرط i >1	تعداد اجرا شدن دستور اصلی
16	درست	۱ بار
8	درست	۱ بار
4	درست	۱ بار
2	درست	۱ بار
1	غلط	–
		جمعاً ٤ بار

مثال

دستور (پاسکال)	دستور (سی)
i := n;	i = n;
while (i>1) do	while (i>1)
begin	{
x := x + 1;	x = x + 1;
i:= i div 2;	i = i / 2;
end;	}

- تعداد تکرار برای اجرا شدن دستور اصلی (x:=x+1;)را حساب کنید.

- نکته : دفعات اجرا بصورت لگاریتمی کاهش می یابد
- اگر n=14 باشد:

i	شرط i >1	تعداد اجرا شدن دستور اصلی
14	درست	۱ بار
7	درست	۱ بار
3	درست	۱ بار
1	غلط	–
		جمعاً ۳ بار

در حالت کلی دستور اصلی به تعداد $\lfloor \log_2 n \rfloor$ بار اجرا می شود.

شمارش گام ها یا مراحل یک برنامه

مرتبه های لگاریتمی

```
for ( i=a ; i<=b ; i=i*k )
        s;
```

$$\log_k^b - \log_k^a + 1$$

```
for ( i=b ; i>=a ; i=i/k )
        s;
```

```
for ( i=1 ; i<=8 ; i=i*2 )
        s;
```

$$\Rightarrow \log_2^8 - \log_2^1 + 1 = 4$$

```
for ( i=27 ; i<=n ; i=i*3  )
        s;
```

$$\Rightarrow \log_3^n - \log_3^{27} + 1 = \log_3^n - 2$$

شمارش گام ها یا مراحل یک برنامه

حلقه های تودرتو

```
for ( i=1 ; i<=n ;  i++ )
     for ( j=1 ; j<=n ;  j++)
          s;
```

➡ n^2

```
for ( i=2 ; i<=n ; i=i+4 )
     for ( j=n ; j>3 ; j=j-2 )
          s;
```

➡ $\frac{n-2+1}{4} \times \frac{n-3}{2}$

```
for ( i=1 ; i<=n ;  i=i*2 )
     for ( j=1 ; j<=n ; j++ )
          s;
```

➡ $(\lg n + 1) \times n \Rightarrow ($

مرتبه اجرایی الگوریتم (O)

- در قسمت قبلی به طور دقیق محاسبه کردیم که یک دستور اصلی دقیقا چند بار اجرا می شود و یا اینکه تعداد کل مراحل برنامه چند گام است.
- در عمل، محاسبات دقیق فوق اغلب مشکل بوده و از طرف دیگر این محاسبه دقیق مورد نیاز نیست.
- در کاربردهای واقعی که سرعت کامپیوترها و نوع کامپایلر می تواند سرعت اجرای برنامه ها را چندین مرتبه کاهش یا افزایش دهد، بحث بر سر اینکه فلان دستورالعمل ۱۰۰۰ بار اجرا می شود یا ۹۹۹ بار، لازم نیست.
- به جای محاسبات دقیق ما به ابزاری نیاز داریم که زمان اجرای الگوریتم ها را به صورت **حدودی و طبقه بندی شده** نشان میدهد.

مرتبه اجرایی الگوریتم (O)

- این بحث تا حدی شبیه بحث هم ارزی ها در مسائل حل ریاضیات است.
- مثلا در ریاضیات خوانده اید که $\lim 5n^2 + 4n - 3$ هم ارز با عبارت $5n^2$ است
- یعنی هنگامی که n زیاد می شود عبارت ۳- $4n$ در مقابل $5n^2$ قابل صرف نظر بوده و می توانیم فقط $5n^2$ را در نظر بگیریم.

مرتبه اجرایی الگوریتم (O)

- مرتبه اجرایی یک الگوریتم نیز شبیه هم ارزی فوق است.
- مثلا الگوریتمی که پیچیدگی زمانی آن ۴ - $T(n) = 5n$ میباشد از مرتبه n است که آن را با $O(n)$ نمایش می دهیم.
- یا مثلا الگوریتمی که پیچیدگی زمانی آن $6n^2-3n+2$ میباشد از مرتبه $O(n^2)$ است.
- در توابع چند جمله ای، مرتبه اجرایی معادل بزرگترین توان است

قضیه اصلی : اگر $f(n) = a_m n^m + ... + a_1 n + a_0$ باشد آنگاه $f(n) = O(n^m)$ خواهد بود.

مرتبه اجرایی الگوریتم (O)

❑ مرتبه اجرایی برنامه های زیر را به دست آورید:

- نکته : O(1) زمان محاسبه ثابتی را نشان می دهد که تابعی از n نیست.

الف) $x := x + 1 ; \quad \Rightarrow \quad O(1)$

ب) for i: = 1 to n do

x: = x+1; $\quad \Rightarrow \quad O(n)$

ج) for i: = 1 to n do

for j: = 1 to n do $\quad \Rightarrow \quad O(n^2)$

x: = x + 1 ;

مرتبه اجرایی الگوریتم (O)

❑تعداد دقیق چاپ پیام OK و مرتبه اجرایی برنامه زیر را به دست آورید:

پاسکال	C

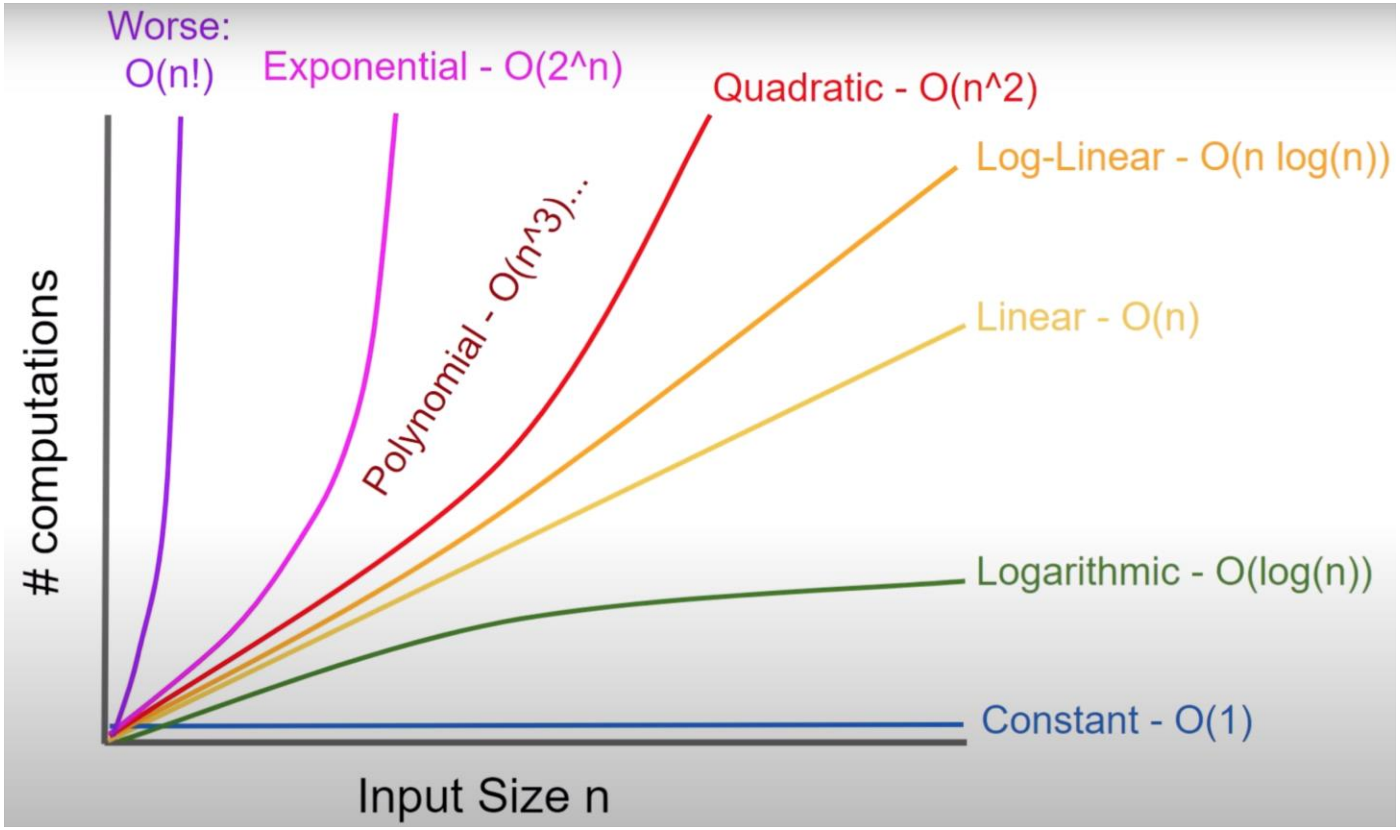
for i:=1 to n do
for j:=i to n do
write('OK');
	for (i = 1; i <= n; i ++)
	for(j = i ; j <= n; j++)
	printf("OK");

حل : حلقه‌های فوق وابسته به یکدیگر بوده و همانطور که قبلاً دیدید تعداد دقیق اجـرا شـدن دسـتور write برابر جمع تصاعد عددی از 1 تا n می‌باشد یعنی $\frac{n(n+1)}{2}$ و بنابر قضیه اصـلی مرتبـه اجرایـی آن $O(n^2)$ می‌باشد.

توابع رشد

- مقایسه مرتبه اجرایی چند تابع معروف به ترتیب زیر می باشند:

$$O(1) < O(\log n) < O(n) < O(n\log n) < O(n^2) < O(n^3) < \cdots < O(2^n) < O(3^n) < O(n!) < O(n^n)$$



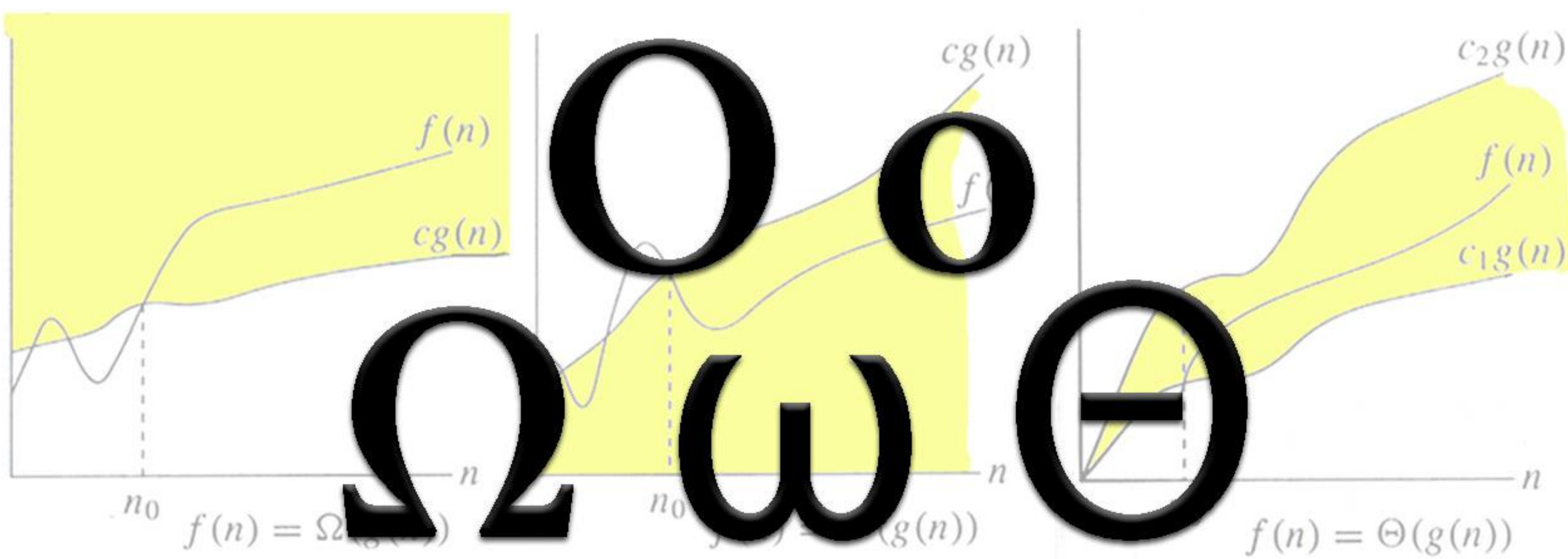
توابع رشد

- مقایسه مرتبه اجرایی چند تابع معروف به ترتیب زیر می باشند:

$$O(1) < O(\log n) < O(n) < O(n\log n) < O(n^2) < O(n^3) < \cdots < O(2^n) < O(3^n) < O(n!) < O(n^n)$$

Function	10	100	1,000	10,000	100,000	1,000,000
1	1	1	1	1	1	1
$\log_2 n$	3	6	9	13	16	19
n	10	10^2	10^3	10^4	10^5	10^6
$n * \log_2 n$	30	664	9,965	10^5	10^6	10^7
n^2	10^2	10^4	10^6	10^8	10^{10}	10^{12}
n^3	10^3	10^6	10^9	10^{12}	10^{15}	10^{18}
2^n	10^3	10^{30}	10^{301}	$10^{3,010}$	$10^{30,103}$	$10^{301,030}$

نمادهای مجانبی



نمادهای مجانبی

- علایمی که ما از آن‌ها برای نشان دادن کران‌های مجانبی استفاده می‌کنیم، معرف توابعی هستند که به آنها توابع رشد می‌گوییم و دامنه آن‌ها مجموعه‌ای از اعداد طبیعی است. این علایم عبارت‌اند از:

  - O – کران بالای مجانبی
  - Ω – کران پایین مجانبی
  - θ – کران دو طرف مجانبی

نماد O

❑تعریف ریاضیاتی این نماد به شکل زیر می‌باشد:

$$\boxed{O(g(n)) = \{f(n) \mid \exists c, n_0 > 0 \;,\; \forall n \geq n_0 : f(n) \leq c \cdot g(n)\}}$$

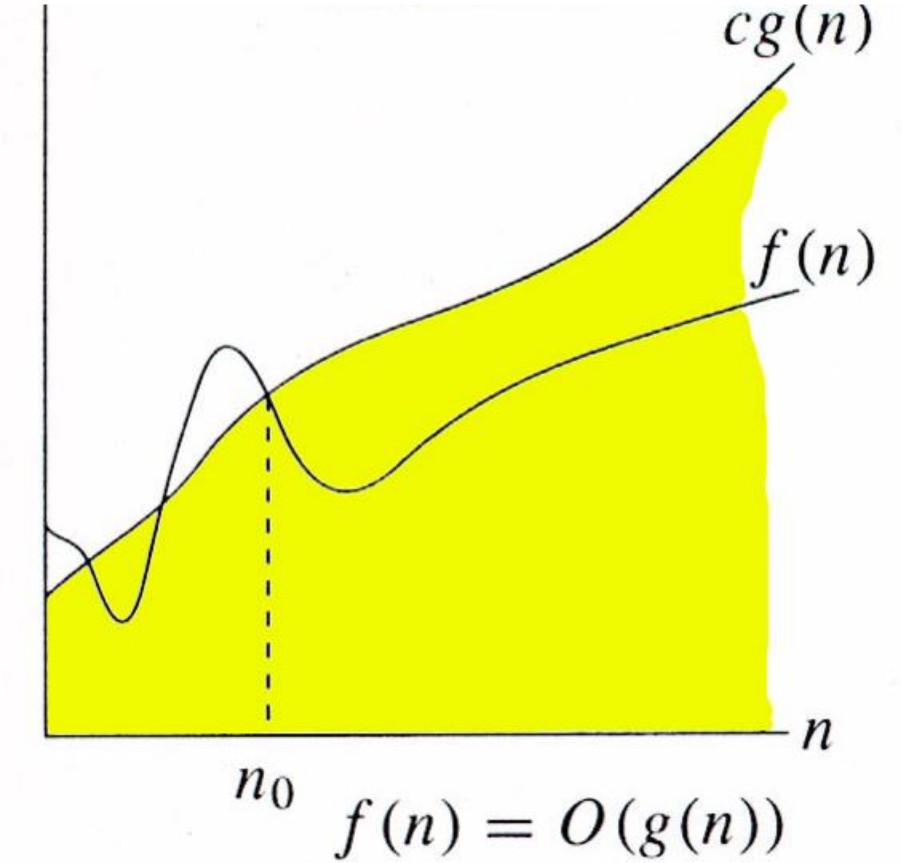
❑یعنی آهنگ رشد تابع g(n) برای مقادیر بزرگ n، بیشتر یا مساوی آهنگ رشد تابع f(n) است.

❑در این صورت می‌گوییم g(n) کران بالای مجانبی برای f(n) است.

❑بطور مثال:

- $nlogn \in O(n^2)$
- $3n + 3 \in O(n^2)$
- $3n^2 + n \in O(n^2)$

نماد O



- از یک نقطه ای به بعد (n_0) تابع f(n) همواره کوچکتر یا مساوی ضریبی از تابع g(n) است.

$$O(g(n)) = \{f(n) \mid \exists c, n_0 > 0 \ , \ \forall n \geq n_0 : f(n) \leq c \cdot g(n)\}$$

نماد Ω

- تعریف ریاضیاتی این علامت به شکل زیر می‌باشد که در آن تابع g(n) کران پایین مجانبی برای تابع f(n) است.

$$\boxed{\Omega(g(n)) = \{f(n) \mid \exists c, n_{\circ} > \circ \;,\; \forall n \geq n_{\circ} : f(n) \geq c \cdot g(n)\}}$$

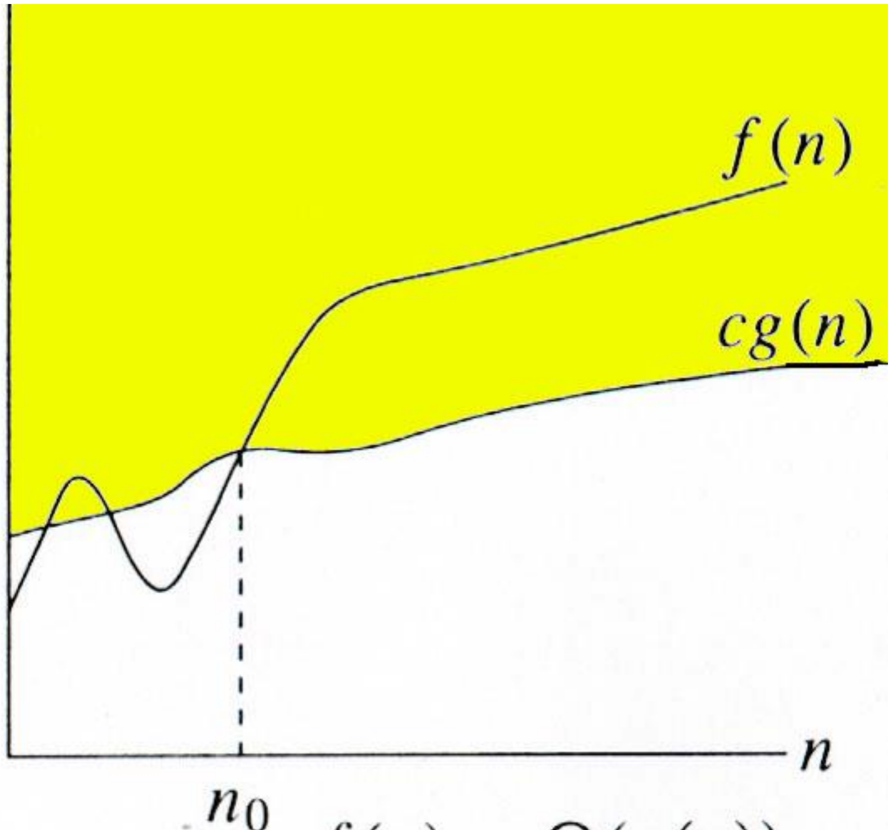
- یعنی آهنگ رشد تابع g(n) برای مقادیر بزرگ n، کمتر یا مساوی آهنگ رشد تابع f(n) است.

- در این صورت می گوییم g(n) کران پایین مجانبی برای f(n) است.

- مثال:

- $\sqrt{n} \in O(\log n)$
- $n^3 \in O(n^2)$
- $3n^2 + n + 5 \in O(n)$
- $3n^2 + n + 4 \notin O(n^3)$

نمادΩ



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

- از یک نقطه ای به بعد (n_0) تابع f(n) همواره بزرگتر یا مساوی ضریبی از تابع g(n) است.

$$O(g(n)) = \{f(n) \mid \exists c, n_\circ > \circ \ , \ \forall n \geq n_\circ : f(n) \leq c \cdot g(n)\}$$

نماد θ

❑بیانگر کران دو طرف مجانبی است که به صورت زیر تعریف می‌شود:

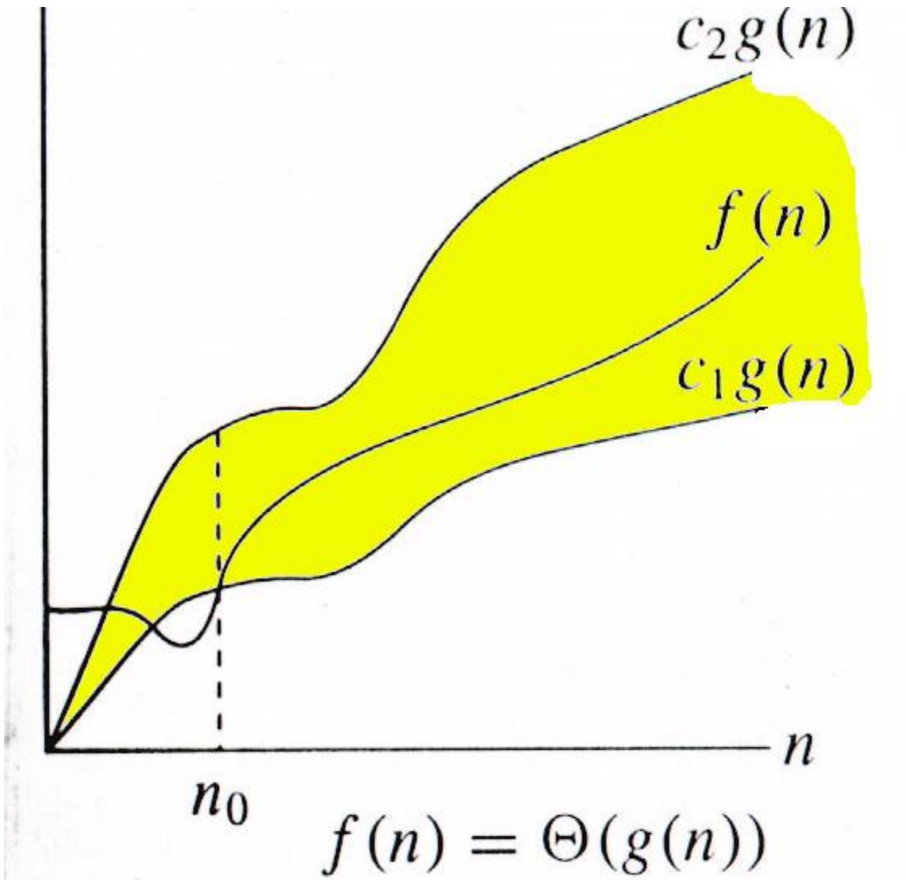
$$\theta(g(n)) = \{f(n) \mid \exists c_۱, c_۲, n_۰ > ۰ \;,\; \forall n \geq n_۰ : c_۱ \cdot g(n) \leq f(n) \leq c_۲ \cdot g(n)\}$$

❑این رابطه بدین معنی است که آهنگ رشد f و g برای مقادیر بزرگ nیکسان است و هیچ یک از این دو تابع از دیگری جلو نمی‌زنند.

❑مثال:

- $4n^3 + 3n + 10 \in \theta(n^2)$
- $3n^2 + n + 5 \in \theta(n^2)$
- $3n^2 + n + 4 \notin \theta(n^3)$

نماد Ω

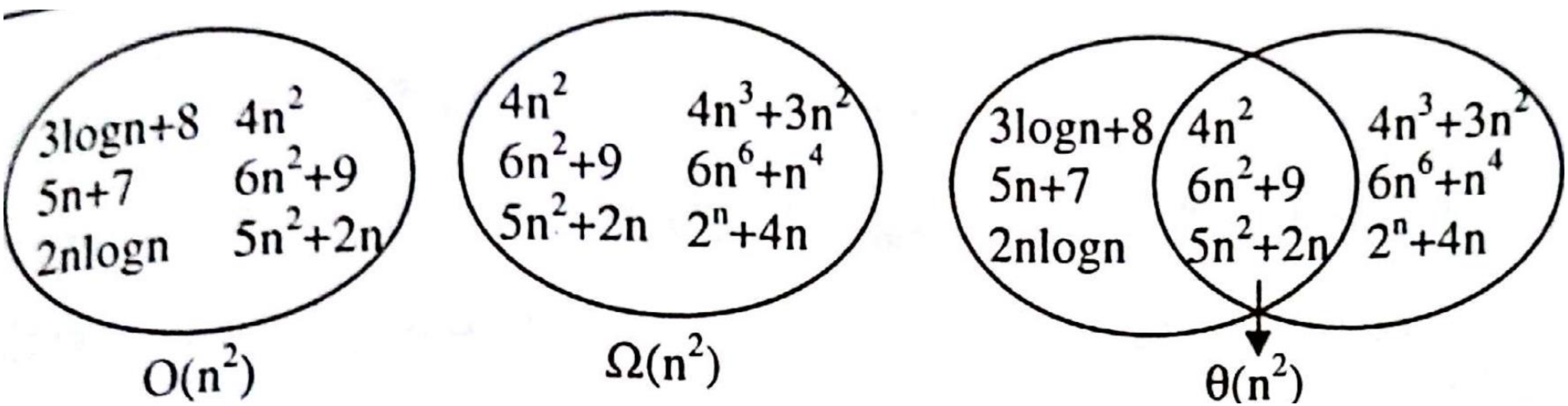


- از یک نقطه ای به بعد (n_0) تابع $f(n)$ همواره بین ضریبی از تابع $g(n)$ است.

$$\theta(g(n)) = \{f(n) \mid \exists c_1, c_2, n_0 > 0 \ , \ \ \forall n \geq n_0 : c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)\}$$

نمادهای مجانبی

- مثال و ارتباط بین نمادهای مجانبی



توابعی که مرتبه اجرایی مساوی n^2 دارند

توابعی که مرتبه اجرایی بزرگتر یا مساوی n^2 دارند

توابعی که مرتبه اجرایی کوچکتر یا مساوی n^2 دارند

تمرینات سری اول

۱) کدام یک از روابط زیر نشان دهنده رابطه صحیح زمان محاسبه الگوریتم های مختلف است؟

1) $O(\log n) < O(n) < O(nlogn) < O(2^n) < O(n^2)$

2) $O(n) < \ O(\log n) < O(nlogn) < O(2^n) < O(n^2)$

3) $O(n) < \ O(\log n) < O(nlogn) < O(n^2) < O(2^n)$

4) $O(logn) < \ O(n) < O(nlogn) < O(n^2) < O(2^n)$

تمرینات سری اول

۲) مرتبه اجرای برنامه زیر کدام است ؟

```
i : = n;
 while (i > 1) do
   {
        i: = i/2;          j: = n;
         while (j > 1)  do
                j: = j/3;
   }
```

۱) $O(\log_2 n)$ ۲) $O(\log_3 n)$ ۳) $O(\log_6 n)$ ۴) $O(\log_2 n \times \log_3 n)$

تمرینات سری اول

۳) شمار فراوانی دستورات اجرایی در تابع زیر چیست؟

C++ , C	پاسکال

void add(int a[ ][n] , int b[ ][n] , int c[ ][n] , int m)
{
int i , j;
for (i = 0; i < m; i ++)
for (j = 0; j < n; j ++)
c[i] [j] = a[i] [j] + b[i] [j];
}
	type matrix = array [1 .. m, 1 .. n] of integer;
	procedure add (var a,b,c : matrix, m,n : integer);
	var i,j : integer;
	begin
	for i := 1 to m do
	for j := 1 to n do
	c [i, j] := a[i , j] + b[i , j];
	end;

۱) 2mn+m-n ۲) 2mn+n-m ۳) 2mn +2n+1 ٤) 2mn+2m+1

تمرینات سری اول

۴) مرتبه زمانی الگوریتم زیر چیست؟

```
    proc (Var   x : integer; n: integer);
Var  i, j, k: integer;
{
      for i: = 1 to n do
         for j : = 1 to i do
            for k : = 1 to j do
                 x : = x + 1;
}
```

۱) $O(n)$

۲) $O(n^2)$

۳) $O(n^3)$

٤) $O(n \log n)$

تمرینات سری اول

۵) کدام یک از تساوی های زیر درست است؟

۱) $5n^2 - 6n = O(n^3)$

۲) $5n^2 - 6n = \theta(n^3)$

۳) $5n^2 - 6n = \Omega(n^3)$

۴) $5n^2 - 6n = \theta(n)$