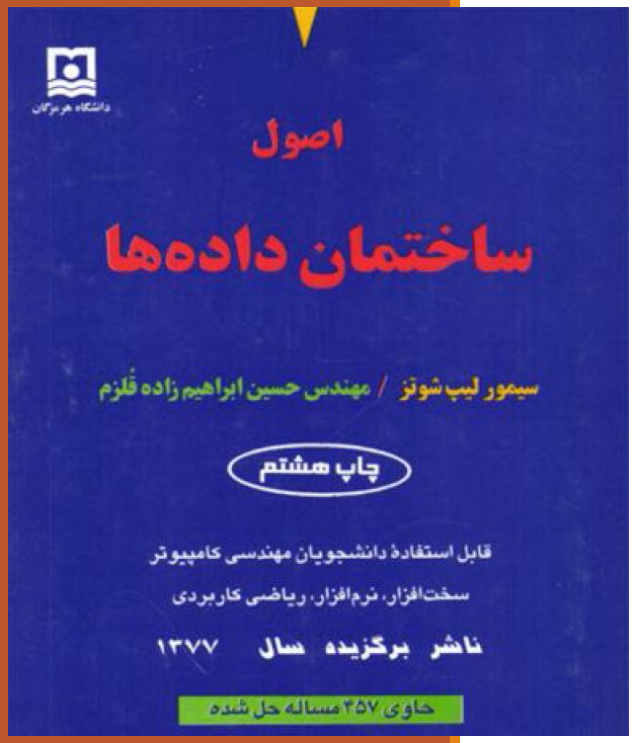
ساختمان داده ها
و الگوریتم
در C++

مدرس: یکماز

مرجع



- عنوان منبع:

ساختمان داده ها در C++

- مترجم:

حسین ابراهیم زاده ی قلزم

- انتشارات:

سیمای دانش

- منبع اصلی:

data Structure in C++

Horowitz Ellis

بارم بندی

- پایان ترم : ۱۵ نمره
- حل تمرین و فعالیت کلاسی: ۵ نمره
- حضور در کلاس: ۲- تا ۲ نمره

سرفصل آموزشی

- تحلیل الگوریتم ها
- آرایه
- صف و پشته
- لیست های پیوندی
- درخت
- گراف
- مرتب سازی

فصل اول:
تحلیل الگوریتم ها

- مفاهیم اساسی تولید نرم افزار
- آشنایی با الگوریتم ها
- برنامه های بازگشتی
- محاسبه پیچیدگی زمانی و مرتبه اجرایی

فصل اول : مفاهیم اساسی

اهداف

- آشنایی با سیکل زندگی نرم افزار
- آشنایی با الگوریتم

۱–۱ سیکل زندگی نرم افزار

مراحل مختلفی که در چرخه نرم افزار هستند

- نیازمندی‌ها
- تجزیه و تحلیل
- طراحی
- تعریف الگوریتم و برنامه سازی آن
- تست برنامه

۱–۱ سیکل زندگی نرم افزار – نیازمندی ها

نیازمندی ها

- تمام پروژه های بزرگ برنامه نویسی با مجموعه ای از مشخصات و خصوصیاتی که اهداف پروژه را مشخص می کند، شروع می شود.
- این نیازمندیها اطلاعاتی را به برنامه نویسان می دهند(ورودی) و نیز نتایجی را که باید ایجاد گردد(خروجی) تعیین می کنند

۱–۱ سیکل زندگی نرم افزار – تحلیل

تحلیل

- در این مرحله مساله را به بخشهای قابل کنترل تقسیم می کنند.
- در تحلیل یک سیستم دو شیوه وجود دارد :

۱- شیوه از بالا به پایین: با هدف طراحی و تولید طرحی سطح بالا شروع میشود که در آن برنامه به بخشهای قابل مدیریت تقسیم میشود.

۲- شیوه از پایین به بالا: روش قدیمی و غیرساخت یافته است که تاکید آن بر مطالعه نکات حساس برنامه نویسی استوار است

۱–۱ سیکل زندگی نرم افزار – طراحی

طراحی

- این مرحله ادامه کاری است که در مرحله تحلیل انجام می شود.
- طراح سیستم را از دو نقطه نظر بررسی می کند:
  - از نظرداده های مقصود (data objects) مورد نیاز برنامه – ایجاد نوع داده مجرد
  - از نظر اعمالی که بر روی آنها انجام می شود. این دیدگاه به مشخصات الگوریتم ها و فرضیات خط مشی ها ی طراحی الگوریتم نیاز دارد.

۱–۱ سیکل زندگی نرم افزار - تعریف الگوریتم و کدنویسی، بازبینی

تعریف الگوریتم و کدنویسی

- در این مرحله، نمایشی برای داده های مقصد خود انتخاب می شود و برای هر عملی که بر روی آنها انجام می شود، الگوریتم نوشته می شود.

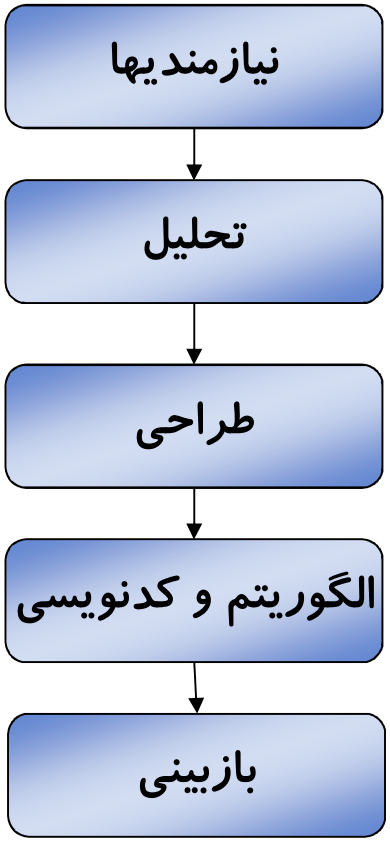
بازبینی

- در این مرحله درستی برنامه ها اثبات می شود و برنامه ها با انواع داده های ورودی مختلف تست و خطاهای برنامه رفع می شود.

جنبه های مهم بازبینی

اثبات درستی | تست | اشکال زدایی

۱–۱ سیکل زندگی نرم افزار– نمودار



۲–۱ الگوریتم – تعریف

مجموعه ای از **دستورالعمل** ها که اگر دنبال شوند، موجب **انجام کار** خاصی می گردد

۲-۱ الگوریتم – شرایط

ورودی: یک الگوریتم می تواند هیچ یا چندین کمیت ورودی داشته باشدکه از محیط خارج تامین می شود.

خروجی: الگوریتم بایستی حداقل یک کمیت به عنوان خروجی داشته باشد.

قطعیت: هر دستورالعمل باید واضح و بدون ابهام باشد.

محدودیت: اگر دستورالعمل های یک الگوریتم را دنبال کنیم ،برای تمام حالات ، الگوریتم باید پس از طی مراحل محدودی خاتمه یابد.

کارایی: هر دستورالعمل باید به گونه ای باشد که با استفاده از قلم و کاغذ بتوان آن را با دست نیز اجرا نمود.

۲–۱ الگوریتم – مثال: مرتب سازی

for (i=0 ; i<n ;i++)

{

Examine list [i] to list [n-1] and suppose that the smallest integer is at list [min];

Interchange list [i] and list [min];

{

۲-۱ الگوریتم – تابع

تعریف: مجموعه ای از دستورات مختلف که عملی منطقی را انجام می دهند

- نام تابع و پارامترهای آن بعنوان دستوری جدید درنظر گرفته می شوند
- با تعیین مشخصات ورودی-خروجی یک تابع می توانیم به تابع دسترسی پیدا کنیم از این رو لازم نیست بدانیم که تابع چگونه کارش را انجام می دهد

۲–۱ الگوریتم – تابع

- با داشتن چنین تصویری از تابع، ابتدا تابع فراخوانی می شود، آنگاه اجرا شده و کنترل اجرا به مکان مربوطه در تابع فراخواننده باز می گردد.
- توابع می توانند قبل از اتمام کارشان خودشان را فراخوانی کنند (توانمندی لازم برای الگوریتم های بازگشتی)

۲–۱ الگوریتم – بازگشتی

تابع می تواند خودش را فراخوانی کند (بازگشتی مستقیم)

یا

می توانند توابعی که تابع فراخواننده را صدا می زنند (بازگشتی غیرمستقیم) را احضار نمایند.

۲–۱ الگوریتم – بازگشتی

ویژگی های توابع بازگشتی

- فراخوانی تابع توسط خودش
- اغلب با کاهش مقدار ارسالی به تابع
- شرط اتمام فراخوانی

استفاده از حافظه پشته

۲–۱ الگوریتم – بازگشتی

بیان معمولی

$$\binom{n}{m} = \frac{n!}{m!(n-m)!}$$

بیان بازگشتی

$$\binom{n}{m} = \begin{cases} 1 & m = 0 \\ \binom{n-1}{m} + \binom{n-1}{m-1} & otherwise \end{cases}$$

۲–۱ الگوریتم – مثال: جستجوی دودویی(حلقه تکرار)

```
int binsearch ( int  list [ ]  ،int  x  ،int  n )
{
for(int left=0,right=n-1 ; left<=right ; )
   {
        int middle = ( left + right ) / 2 ;
        switch  ( COMPARE (x , list [ middle ]))
        {
        case '>' :  left = middle+1;  break;
        case '<' : right = middle-1;  break;
        case '=' :  return middle;  break;
        }
    }
    return  -1 ;
}
```

۲-۱ الگوریتم – مثال: جستجوی دودویی(بازگشتی)

```
int binsearch ( int  list [ ]  ،int  x ،int  left  ،int  right )
{
if (left <= right )
    {
       int middle = ( left + right ) / 2 ;
       switch  ( COMPARE (x  , list [ middle ]))
       {
       case '>' :
          return binsearch ( list  ،x  ،middle +1  ،right ) ;
       case '=' :
          return  middle ;
       case '<' :
          return binsearch ( list  ،x  ،left  ،middle -1 ) ;
       }
    }
    return  -1 ;
}
```

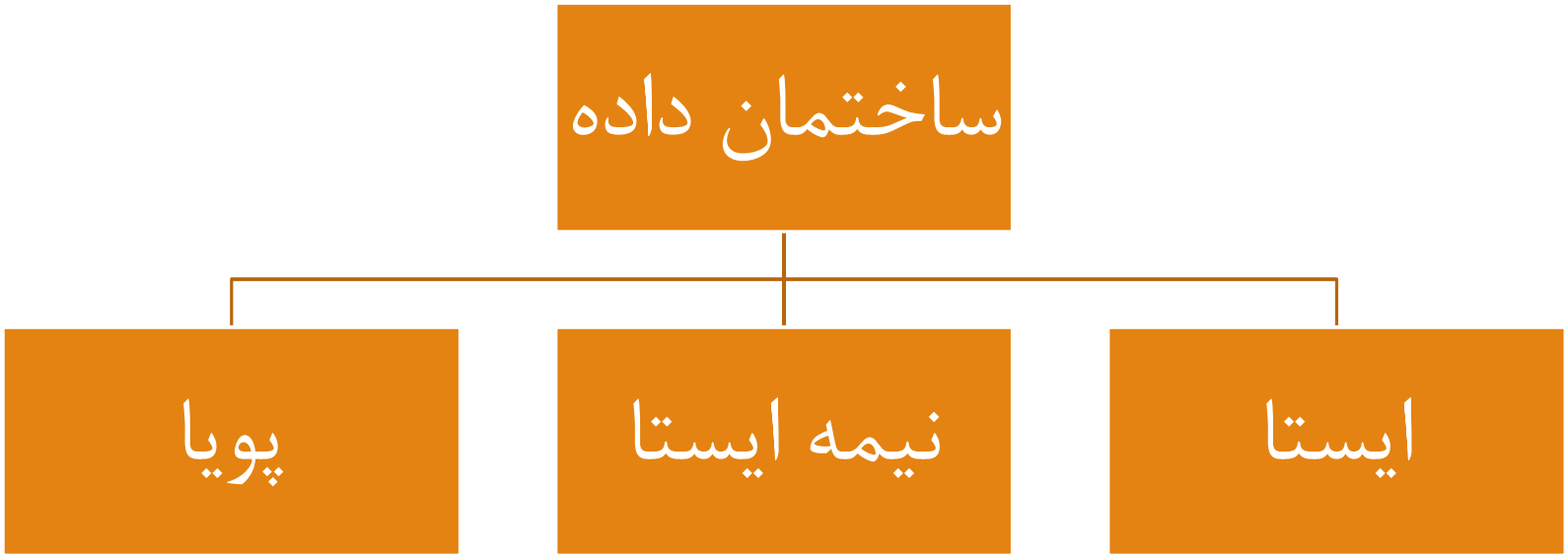
۲–۱ الگوریتم – مثال: فاکتوریل

```
int factorial (int  n)
{
if (n > 1 )
    {
       return n * factorial ( n – 1 ) ;
else
       return 1 ;
}
```

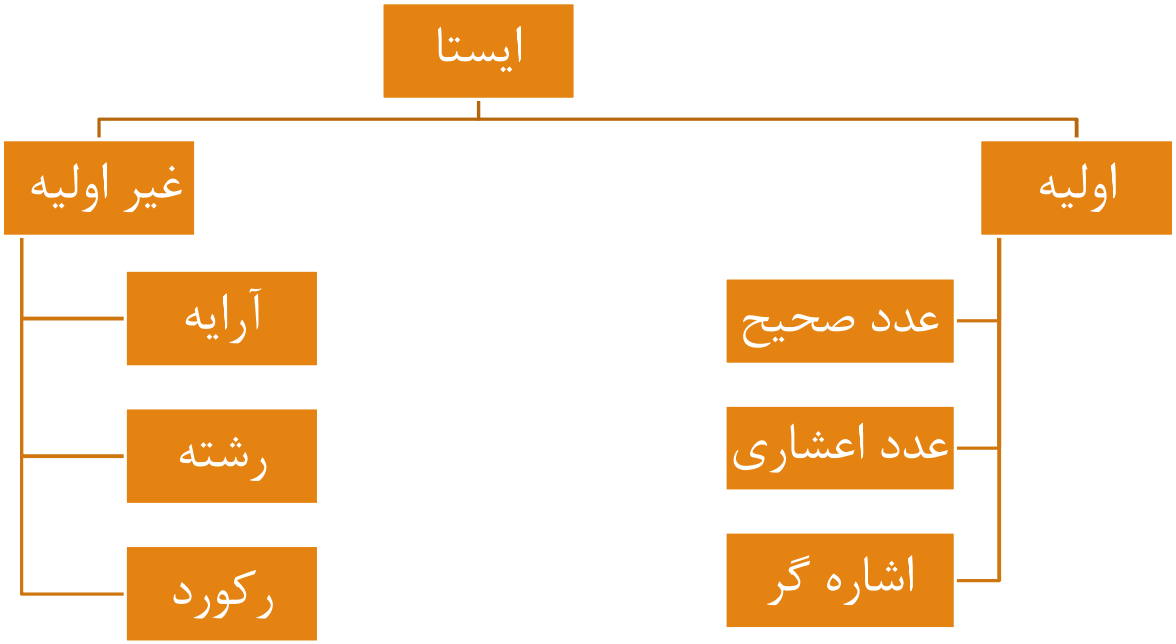
۳–۱ ساختمان داده

ساختارهایی که جهت دریافت داده های خام به شکل مناسب توسط کامپیوتر و پیاده سازی و اجرای الگوریتم های مختلف روی آنها مورد استفاده قرار می گیرند.

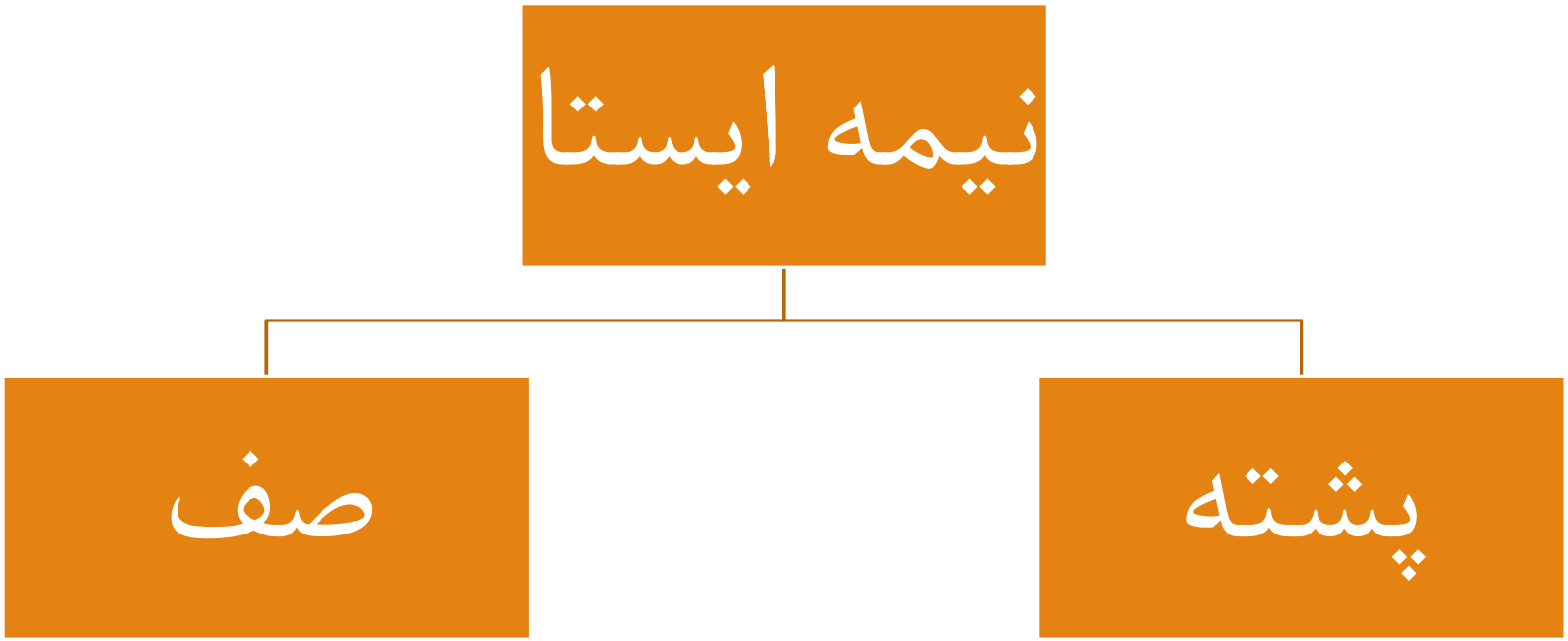
۳–۱ ساختمان داده



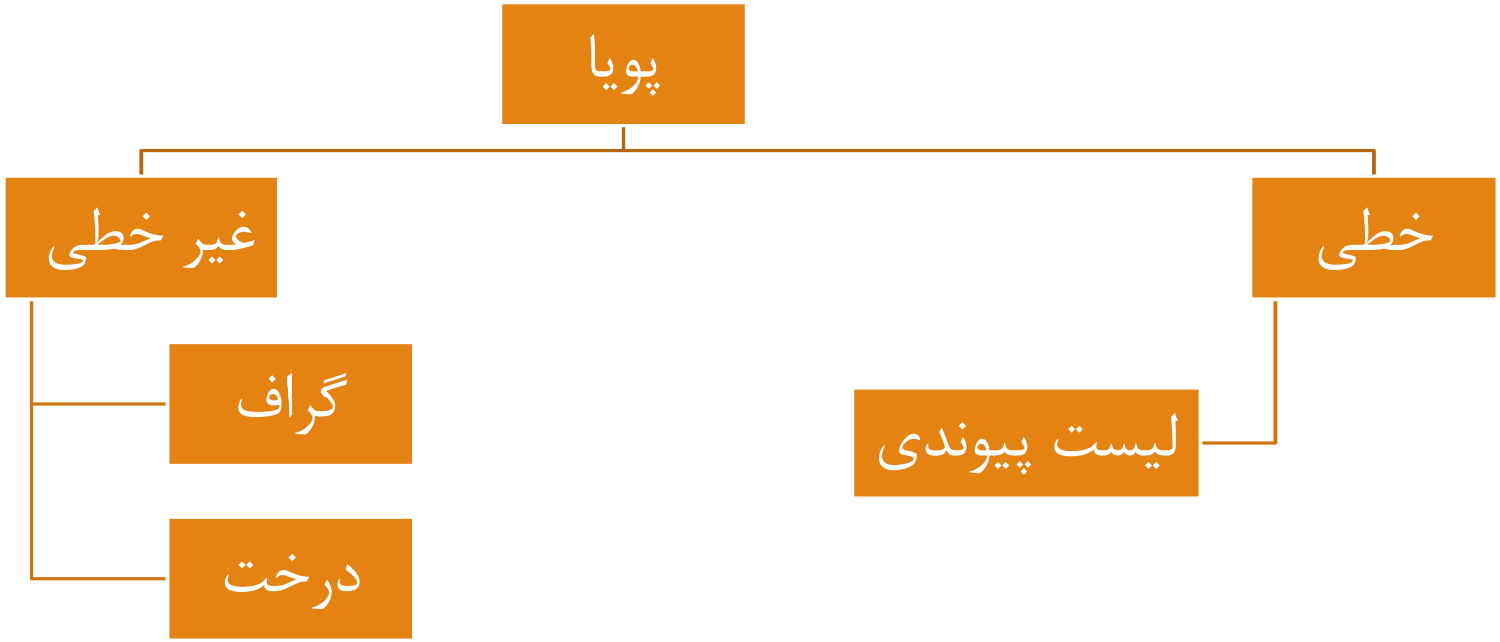
۳–۱ ساختمان داده – ایستا



۳–۱ ساختمان داده – نیمه ایستا



۳–۱ ساختمان داده – ایستا



۳–۱ ساختمان داده

آرایه: مجموعه ای از عناصر از یک نوع داده می باشد

ساختار: مجموعه ای از عناصر است که لزومی ندارد داده های آن یکسان باشد

نوع داده: مجموعه ای از انواع داده (object) و عملکردهایی است که بر روی این نوع داده ها عمل می کنند

۳–۱ ساختمان داده – نوع داده انتزاعی

نوع داده انتزاعی(ADT) که شامل مشخصات داده ها و اعمالی که بر روی آنها انجام می شود.

جهت جداسازی پیاده سازی و نمایش داده ها از یکدیگر از ADT استفاده می شود

۴–۱ تحلیل الگوریتم

معیارهای محک زدن یک برنامه

- آیا برنامه اهداف اصلی کاری را که می خواهیم، انجام می دهد؟
- آیا برنامه درست کار می کند؟
- آیا برنامه مستند سازی شده است تا نحوه استفاده و طرز کار با آن مشخص شود؟
- آیا برنامه برای ایجاد واحدهای منطقی ، به طور موثر از توابع استفاده می کند؟
- آیا کد گذاری خوانا است؟
- آیا برنامه از حافظه اصلی و کمکی به طور موثری استفاده می کند؟
- آیا زمان اجرای برنامه برای هدف شما قابل قبول است؟

۴-۱ تحلیل الگوریتم

- بازدهی الگوریتم ها به عوامل زیادی بستگی دارد
- به طور کلی با افزایش اندازه ورودی، زمان اجرا افزایش می یابد.
- زمان اجرا متناسب با تعداد دفعات اجرای **عملیات اصلی** است
- بنابر این بازدهی الگوریتم را با تعیین تعداد دفعات اجرای عمل اصلی به عنوان تابعی از ورودی تحلیل می کنیم.

۴-۱ تحلیل الگوریتم – پیچیدگی فضایی و زمانی

میزان حافظه یا **پیچیدگی فضایی** یک برنامه **مقدارحافظه مورد نیاز برای اجرای کامل** یک برنامه است.

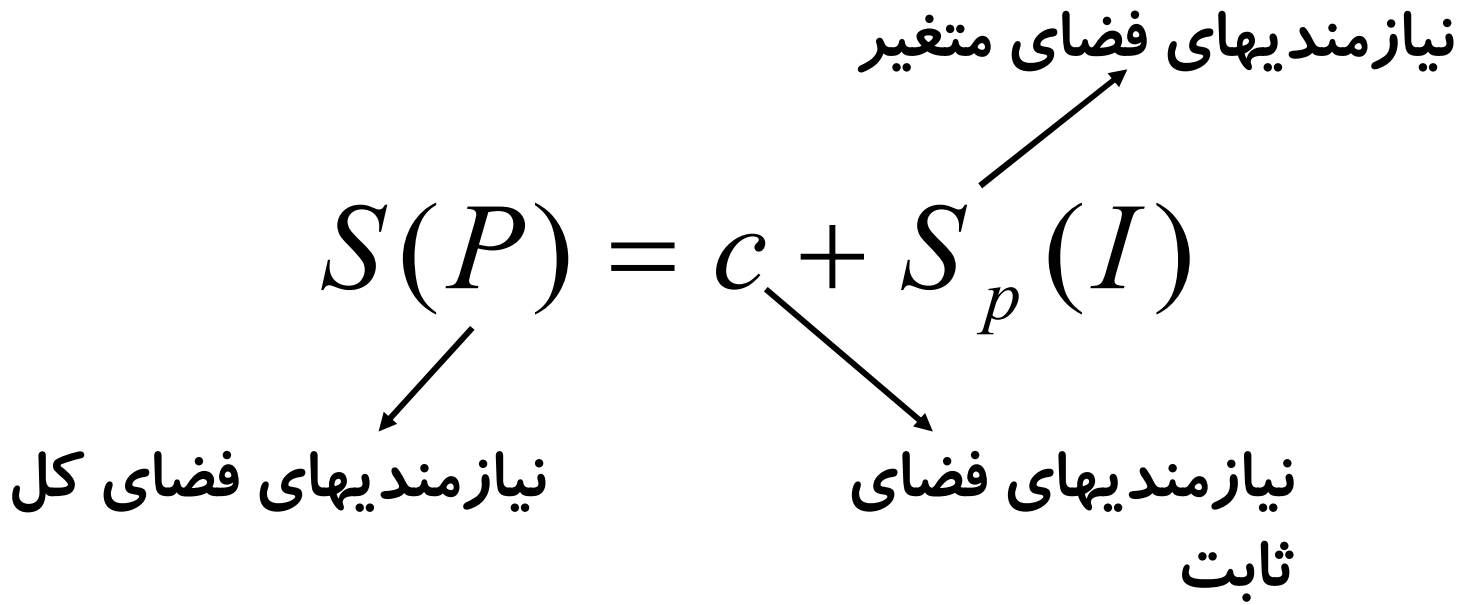
میزان زمان یا **پیچیدگی زمانی** یک برنامه **مقدار زمان کامپیوتر برای اجرای کامل** برنامه است.

۴–۱ تحلیل الگوریتم – پیچیدگی فضایی

فضای مورد نیاز یک برنامه شامل موارد زیر است :

- **نیازمندی های فضای ثابت**
- این مطلب به فضای مورد نیازی که به تعداد و اندازه ورودی و خروجی بستگی ندارد، اشاره دارد.
- **نیازمندی های فضای متغیر**
- این مورد شامل فضای مورد نیاز متغیرهای ساخت یافته ای است که اندازه آن بستگی به نمونه I از مساله ای که حل می شود، دارد.

۴–۱ تحلیل الگوریتم – پیچیدگی فضایی



۴–۱ تحلیل الگوریتم – پیچیدگی زمانی

پیچیدگی زمانی یا T(P) عبارت است ازمجموع زمان کامپایل و زمان اجرای برنامه.

زمان کامپایل مشابه اجزای فضای ثابت است زیرا به خصیصه های نمونه بستگی ندارد.

۴-۱ تحلیل الگوریتم – پیچیدگی زمانی

برای محاسبه پیچیدگی زمانی عبارتی به شکل زیر برای محاسبه Tp(n) بدست می‌آید

$$T_p(n) = c_a ADD(n) + c_s SUB(n) + c_m MUL(n) + \dots$$

تعداد عملیات جمع

تعداد عملیات تفریق

تعداد عملیات ضرب

در عبارت فوق n تعداد مشخصه های موردی (مشخصه نمونه) است

۴–۱ تحلیل الگوریتم – مراحل برنامه

به جای شمارش تعداد عملیات جمع و تفریق و ضرب برای حل مساله با n مشخصه موردی تنها **تعداد مراحل برنامه** را می شماریم.

یک مرحله برنامه، قسمت با معنی برنامه است (از لحاظ معنایی یا نحوی) که زمان اجرای آن مستقل از خصیصه های نمونه باشد.مثلا:

$$return\ \ a+b*b+c+(a+b+c)/(a+b)+4.0;$$

۴–۱ تحلیل الگوریتم – مثال

برنامه تکراری جمع عناصر یک لیست

Statement	s/e	Frequency	Total steps
float sum(float list[ ], int n)	0	0	0
{	0	0	0
float tempsum = 0;	1	1	1
int i;	0	0	0
for(i=0; i <n; i++)	1	n+1	n+1
tempsum += list[i];	1	n	n
return tempsum;	1	1	1
}	0	0	0
Total			2n+3

۵–۱ نمادهای مجانبی(Asymptotic) (Θ،Ω،O)

انگیزه برای تعیین تعداد مراحل، قابلیت مقایسه پیچیدگی دو برنامه که یک تابع را اجرا می کنند و پیش بینی رشد و افزایش زمان اجرا با تغییر مشخصه های موردی می باشد.

نمادهای مجانبی به ما کمک می کنند تا الگوریتم ها را با هم مقایسه کنیم.

به عبارت ساده تر، این نمادها مشابه نمادهای >، <، = در عبارات مقایسه ای ریاضی هستند.

۵–۱ نماد Big O (O)

تعریف ریاضی:

$$f(n) = O(g(n)) \Leftrightarrow \exists c, n_0 > 0 \quad \forall n \geq n_0 \quad f(n) \leq cg(n)$$

f(n)=O(g(n)) (خوانده می شود f(n) از مرتبه g(n)) است اگر و فقط اگر به ازای مقادیر ثابت n_0 و c برای تمامی مقادیر $n \geq n0$ مقدار f(n) کمتر یا مساوی cg(n) باشد

۵–۱ نماد Big O (O)

$O(1)$: زمان محاسبه ثابتی را نشان می دهد

$O(n)$: یک تابع خطی نامیده می شود

$O(n^2)$: تابع درجه دو نامیده می شود

قضیه: اگر $f(n) = a_m n^m + \ldots + a_1 n + n_0$ باشد، بنابراین $f(n) = O(n^m)$ خواهد بود یعنی برای هر چند جمله ای درجه m (بالاترین توان n) داریم: $O(n^m)$

۵–۱ نماد Big O (O)

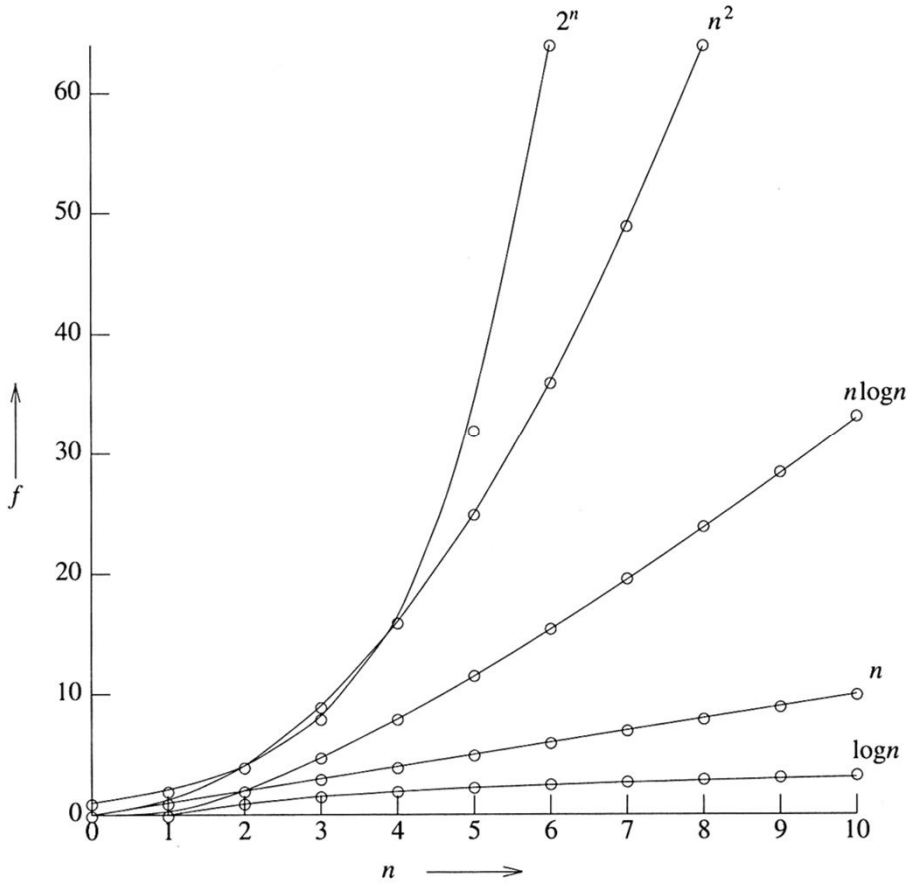
$$O(n^n) > \ldots O(2^n) > \ldots O(n^3) > O(n^2)$$
$$> O(n) > O(\log(n)) > O(1)$$

رابطه بالا نشان می دهد که در صورتی که برای یک مساله راه حلی با پیچیدگی O(n) داشته باشیم رشد آن بسیار کمتر از راه حلی است که پیچیدگی آن $O(n^2)$ است.

۵–۱ نمایش میزان رشد توابع با توجه به تغییر مشخصه ورودی

Time	Name	1	2	4	8	16	32
1	Constant	1	1	1	1	1	1
$\log n$	Logarithmic	0	1	2	3	4	5
n	Linear	1	2	4	8	16	32
$n \log n$	Log linear	0	2	8	24	64	160
n^2	Quadratic	1	4	16	64	256	1024
n^3	Cubic	1	8	64	512	4096	32768
2^n	Exponential	2	4	16	256	65536	4294967296
$n!$	Factorial	1	2	24	40326	20922789888000	26313×10^{33}

۵–۱ نمایش میزان رشد توابع با توجه به تغییر مشخصه ورودی



۵–۱ نمایش میزان رشد توابع با توجه به تغییر مشخصه ورودی

	Time for $f(n)$ instructions on a 10^9 instr/sec computer						
n	$f(n)=n$	$f(n)=\log_2 n$	$f(n)=n^2$	$f(n)=n^3$	$f(n)=n^4$	$f(n)=n^{10}$	$f(n)=2^n$
10	.01μs	.03μs	.1μs	1μs	10μs	10sec	1μs
20	.02μs	.09μs	.4μs	8μs	160μs	2.84hr	1ms
30	.03μs	.15μs	.9μs	27μs	810μs	6.83d	1sec
40	.04μs	.21μs	1.6μs	64μs	2.56ms	121.36d	18.3min
50	.05μs	.28μs	2.5μs	125μs	6.25ms	3.1yr	13d
100	.10μs	.66μs	10μs	1ms	100ms	3171yr	$4*10^{13}$yr
1,000	1.00μs	9.96μs	1ms	1sec	16.67min	$3.17*10^{13}$yr	$32*10^{283}$yr
10,000	10.00μs	130.03μs	100ms	16.67min	115.7d	$3.17*10^{23}$yr	
100,000	100.00μs	1.66ms	10sec	11.57d	3171yr	$3.17*10^{33}$yr	
1,000,000	1.00ms	19.92ms	16.67min	31.71yr	$3.17*10^{7}$yr	$3.17*10^{43}$yr	

۵–۱ نماد Ω

تعریف ریاضی:

$$f(n) = \Omega(g(n)) \Leftrightarrow \exists c, n_0 > 0 \quad \forall n \geq n_0 \quad f(n) \geq cg(n)$$

$f(n) = \Omega(g(n))$ می باشد(خوانده می شود $f(n)$ امگای $g(n)$) اگر و فقط اگر به ازای مقادیر ثابت مثبت c و $n0$ ، $f(n) \geq cg(n)$ باشد(برای تمام مقادیر n به شرطی که $n \geq n0$ باشد)

۵–۱ نماد Θ

تعریف ریاضی:

$$f(n) = \Theta(g(n)) \Leftrightarrow \exists c_1, c_2, n_0 > 0 \quad \forall n \geq n_0 \quad c_1 g(n) \leq f(n) \leq c_2 g(n)$$

$f(n) = \theta(g(n))$ می باشد (خوانده می شود f(n) تتای g(n)) اگر و فقط اگر به ازای مقادیر ثابت c_1 و c_2 و n_0، $c_1 g(n) <= f(n) <= c_2 g(n)$ وجود داشته باشد (برای تمام مقادیر $n >= n_0$)

نشانه گذاری تتا از دو نشانه گذاری ذکر شده O و Ω دقیق تر می باشد . $f(n) = \Theta(g(n))$ می باشد اگر و فقط اگر g(n) هم به عنوان کرانه بالا و هم به عنوان کرانه پایین در f(n) باشد.

۵–۱ نمادهای Θ و Ω

قضیه: اگر $f(n) = a_m n^m + \ldots + a_1 n + a_0$ باشد، و
$a_m > 0$ بنابراین $f(n) = \Omega(n^m)$ خواهد بود

در نتیجه طبق توضیح صفحه قبل

قضیه: اگر $f(n) = a_m n^m + \ldots + a_1 n + a_0$ باشد، و
$a_m > 0$ بنابراین $f(n) = \Theta(n^m)$ خواهد بود

۵–۱ مثال– پیچیدگی جمع ماتریس ها

Statement	Asymptotic complexity
Void add (int a []MAX-SIZE...)	0
{	0
int i،j;	0
for(i=0;i<rows ; i++)	Θ(rows)
for(j=0 ; j<cols ; j++)	Θ(rows.cols)
C[i][j]=a[i][j]+b[i][j];	Θ(rows.cols)
}	0
Total	Θ(rows.cols)

۵– ۱ مثال

- $f(n) = 3n+2$
  - $3n + 2 <= 4n$, for all $n >= 2$ ➔ $3n + 2 = O(n)$

    $3n + 2 >= 3n$, for all $n >= 1$ ➔ $3n + 2 = \Omega(n)$

    $3n <= 3n + 2 <= 4n$, for all $n >= 2$ ➔ $3n + 2 = \Theta(n)$
- $f(n) = 10n^2+4n+2$
  - $10n^2+4n+2 <= 11n^2$, for all $n >= 5$ ➔ $10n^2+4n+2 = O(n^2)$

    $10n^2+4n+2 >= n^2$, for all $n >= 1$ ➔ $10n^2+4n+2 = \Omega(n^2)$

    $n^2 <= 10n^2+4n+2 <= 11n^2$, for all $n >= 5$ ➔ $10n^2+4n+2 = \Theta(n^2)$

۵– ۱ مثال

- $100n+6=O(n)$ /* $100n+6 \leq 101n$ for $n \geq 10$ */
- $10n^2+4n+2=O(n^2)$ /* $10n^2+4n+2 \leq 11n^2$ for $n \geq 5$ */
- $6*2^n+n^2=O(2^n)$ /* $6*2^n+n^2 \leq 7*2^n$ for $n \geq 4$ */