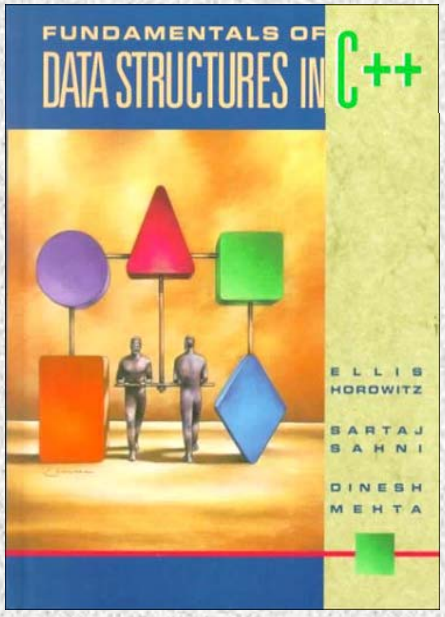
ساختمان داده ها در C++

Data Structure in C++

حقیقت دوست

شناسنامه منبع



- **عنوان منبع:**

  ساختمان داده ها در C++

- **مترجم:**

  حسین ابراهیم زاده ی قلزم

- **انتشارات:**

  سیمای دانش

- **منبع اصلی:**

  data Structure in C++

  Horowitz Ellis

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:

۱. نیازمندی‌ها
۲. تجزیه و تحلیل
۳. طراحی
۴. تعریف الگوریتم و برنامه سازی آن
۵. تست برنامه

۱-۱ سیکل زندگی نرم افزار-نیازمندی ها

نیازمندیها

- 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 غیرساختیافته 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:

۱. از نظرداده های (data objects) مورد نیاز برنامه ⟸ ایجاد نوع داده مجرد

۲. 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

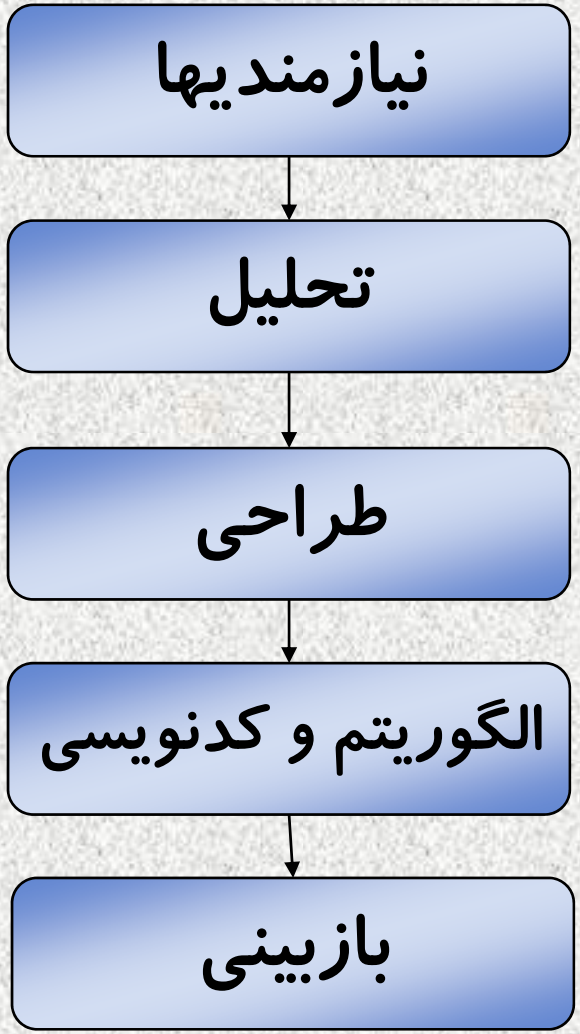
- **تعریف الگوریتم و کدنویسی:** در این مرحله، نمایشی برای داده های مقصد خود انتخاب می شود و برای هر عملی که بر روی آنها انجام می شود، الگوریتم نوشته می شود.

- **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 نمودار سیکل زندگی نرم افزار



۲-۱ تعریف الگوریتم

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

۲-۱ شرایط الگوریتم

- **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 مثالی از الگوریتم: الگوریتم 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 بازگشتی

تابع چیزی است که توسط تابع دیگر فراخوانده می شود .
توابع می توانند خودشان را صدا بزنند (بازگشت مستقیم)
یا
می توانند توابعی که تابع فراخواننده
را صدا میزنند(بازگشتی غیرمستقیم) را احضار نمایند.

۲-۱ مثال الگوریتم جستجوی دودویی تکراری

الگوریتم جستجوي دودويي را بنويسيد.

```
int binsearch ( int  list [ ]  ،int  x  ،int  n )
{
/* search sorted array list[0]… list [n-1] for x */
   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 ;
}
```

۱-۲ مثال الگوریتم جستجوی دودویی بازگشتی

```
int binsearch ( int  list [ ]  ،int  x ،int  left  ،int  right )
{
/* search  list [0] <=  list [1] <=  … <=list [ n-1 ]  for  x  Return  its
position  if  found . Otherwise  return  -1  */
    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 بازگشتی

- مجموعه ای از دستورات مختلف که عملی منطقی را انجام میدهند میتوانند در کنار هم قرارگرفته و تابع را تشکیل دهند
- نام تابع و پارامترهای آن بعنوان دستوری جدید درنظر گرفته میشوند
- با تعیین مشخصات ورودی-خروجی یک تابع میتوانیم به تابع دسترسی پیدا کنیم از اینرو لازم نیست بدانیم که تابع چگونه کارش را انجام میدهد
- با داشتن چنین تصویری از تابع، ابتدا تابع احضار میشود آنگاه اجرا شده و کنترل اجرا به مکان مربوطه در تابع احضار کننده باز میگردد.
- توابع میتوانند قبل از اتمام کارشان خودشان را احضار کنند (توانمندی لازم برای الگوریتم های بازگشتی)

مثال الگوریتم های بازگشتی

بیان معمولی

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

بیان بازگشتی

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

مثال: برنامه مولد جایگشت را بصورت بازگشتی بنویسید

```
void perm(char *a,const int k, const int n)
{
/* Generate all the permutations of a[k],...,a[ n-1 ] */
   if (k == n-1 )
   {
            for(int i=0 ; i< n ; i++)
                     cout<<a[i]<<" ";
            cout<<endl;
            return;
   }
   for(int i=k; i < n ; i++)
   {
            //interchange a[k] , a[i]
            Interchange(a,k,i);
            perm(a,k+1,n);
            //interchange a[k] , a[i] again to return the original configuration
            Interchange(a,k,i);
   }
}
```

۳-۱ آرایه ، ساختار و نوع 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 نوع داده ای مجرد

- نوع داده مجرد یا 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 میشود

۱-۴ تحلیل نحوه اجرای یک برنامه

- 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 میزان حافظه یا پیچیدگی فضای یک برنامه

- میزان حافظه یا پیچیدگی فضای یک برنامه مقدارحافظه مورد نیاز برای اجرای کامل یک برنامه است.

- میزان یا پیچیدگی زمان یک برنامه مقدار زمان کامپیوتر است که برای اجرای کامل برنامه لازم است.

۴-۱ میزان حافظه

فضای مورد نیاز یک برنامه شامل موارد زیر است :

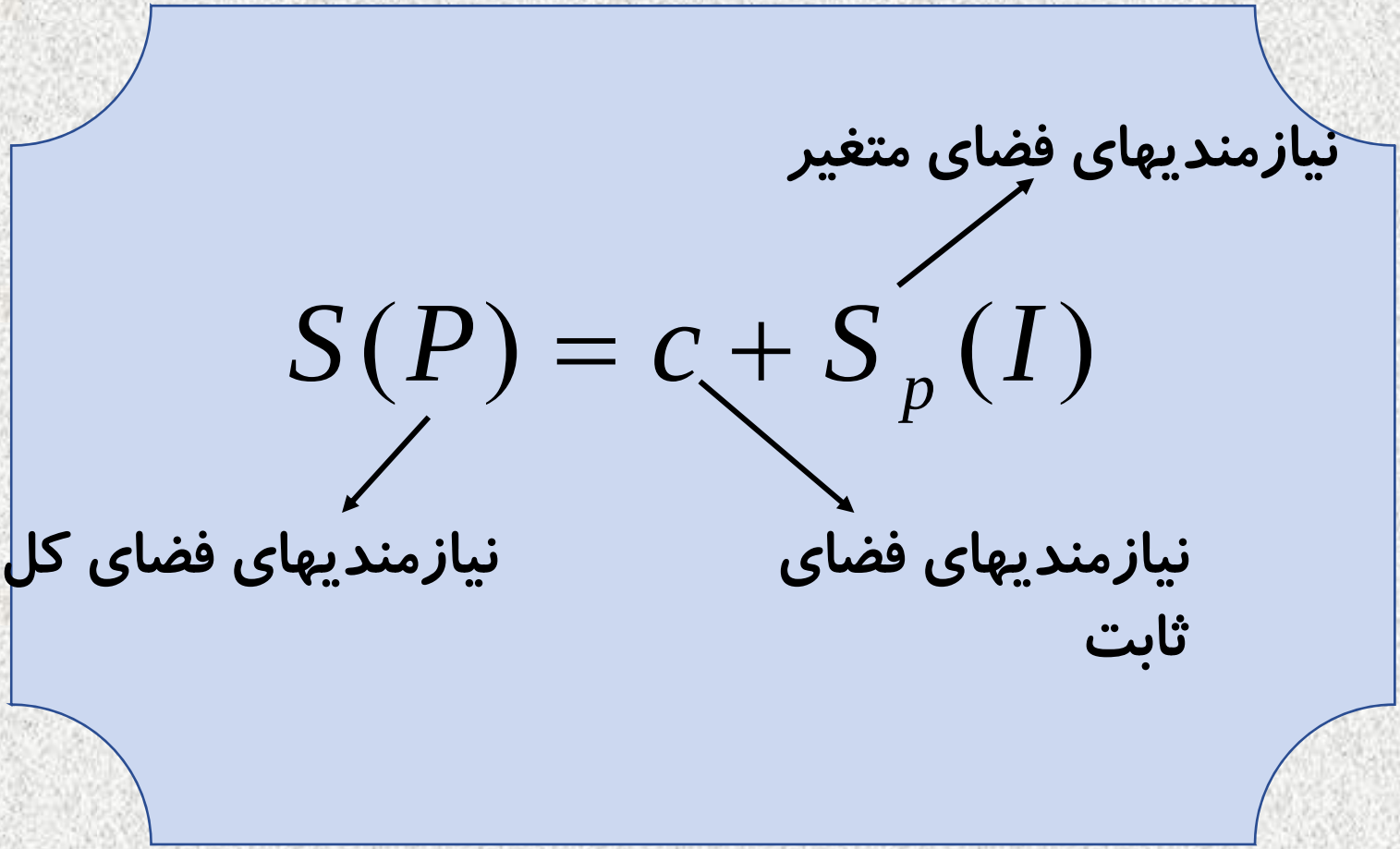
نیازمندیهای فضای ثابت

این مطلب به فضای مورد نیازی که به تعداد و اندازه ورودی و خروجی بستگی ندارد، اشاره دارد.

نیازمندیهای 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 میزان حافظه(ادامه)



۴-۱ زمان T(P) برنامه

زمان T(P) برنامه عبارتست ازمجموع زمان کامپایل و
زمان اجرای برنامه.
زمان کامپایل مشابه اجزای فضای ثابت است زیرابه
خصیصه های نمونه بستگی ندارد.

۴-۱ زمان T(P) برنامه

- برای محاسبه پیچیدگی زمانی عبارتی به شکل زیر برای محاسبه $T_p(n)$ 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 مرحله برنامه

بجای شمارش تعداد عملیات جمع و تفریق و ضرب برای حل مساله با **n** مشخصه موردی تنها تعداد مراحل برنامه را میشماریم

۴-۱ مرحله برنامه

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

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

تمامی دستور فوق یک مرحله در نظر گرفته میشود

Performance Analysis

Iterative function to sum a list of numbers

۴-۱علامت گذاری مجانبی(O)، Ω، Θ)(Asymptotic)

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.

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

f(n)=O(g(n)) (**f(n)** از اُردر **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

۴-۱علامت گذاری مجانبی(O)، Ω، Θ)(Asymptotic)

- 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

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

رابطه بالا نشان میدهد 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.

۴-۱ علامت گذاری مجانبی(O)، (Asymptotic)Ω، Θ

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

تعریف:[امگا]: f(n)=Ω(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)≥cg(n) باشد(برای تمام مقادیر n به شرطی که n≥n0 باشد)

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**

۱-۴ علامت گذاری مجانبی(O)، Ω، Θ(Asymptotic)

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

تعریف تتا[Theta] : $f(n) = \theta(g(n))$ **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 " big oh " و امگا 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.**

۴-۱ علامت گذاری مجانبی(O)، Ω، Θ(Asymptotic)

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**

۱-۴ 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)

Performance Analysis

- Theorem 1.2:
  - **If $f(n) = a_m n^m + \ldots + a_1 n + a_0$, then $f(n) = O(n^m)$.**
- Theorem 1.3:
  - **If $f(n) = a_m n^m + \ldots + a_1 n + a_0$ and $a_m > 0$, then $f(n) = \Omega(n^m)$.**
- Theorem 1.4:
  - **If $f(n) = a_m n^m + \ldots + a_1 n + a_0$ and $a_m > 0$, then $f(n) = \Theta(n^m)$.**

- Examples
  - $f(n) = 3n+2$
    - $3n + 2 <= 4n$, 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 $n >= 1$ ➔ $10n^2+4n+2 = \Omega(n^2)$

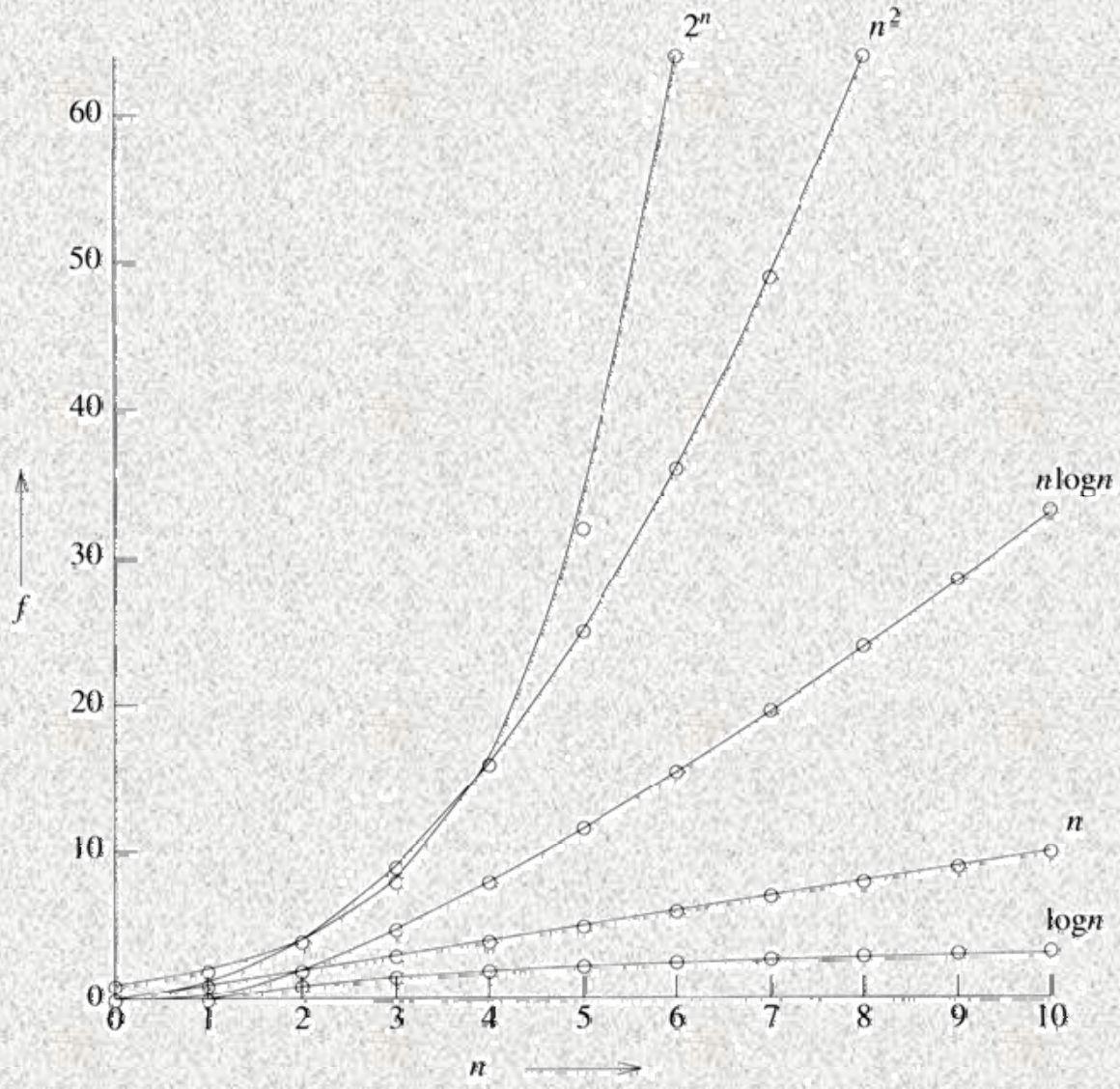
      $n^2 <= 10n^2+4n+2 <= 11n^2$, for all $n >= 5$ ➔ $10n^2+4n+2 = \Theta(n^2)$
  - $100n+6=O(n)$ /* $100n+6 \leq 101n$ for $n \geq 10$ */
  - $10n^2+4n+2=O(n^2)$ /* $10n^2+4n+2 \leq 11n^2$ for $n \geq 5$ */
  - $6*2^n+n^2=O(2^n)$ /* $6*2^n+n^2 \leq 7*2^n$ for $n \geq 4$ */

نمایش میزان رشد توابع با توجه به تغییر مشخصه ورودی

Instance characteristic n							
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}

Figure 1.7 Function values

Performance Analysis



1.9 Times on a billion instruction per second computer

	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	115.7r	$3.17*10^{25}$yr	
100,000	100.00μs	1.66ms	10sec	11.57d	3171yr	$3.17*10^{35}$yr	
1,000,000	1.00ms	19.92ms	16.67min	31.71yr	$3.17*10^{7}$yr	$3.17*10^{45}$yr	

μs = microsecond = 10^{-6} seconds
ms = millisecond = 10^{-3} seconds
sec = seconds
min = minutes
hr = hours
d = days
yr = years