ساختمان داده‌ها

فصل اول

E-mail: Hadi.khademi@gmail.com

مهر ماه ۹۴- دانشگاه علم و فرهنگ

- **Fundamental of data Structure in C++**

  Ellis Horowitz, Sartaj Sahni, Dinesh Mehta
- **Introduction to Algorithms**

  Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein

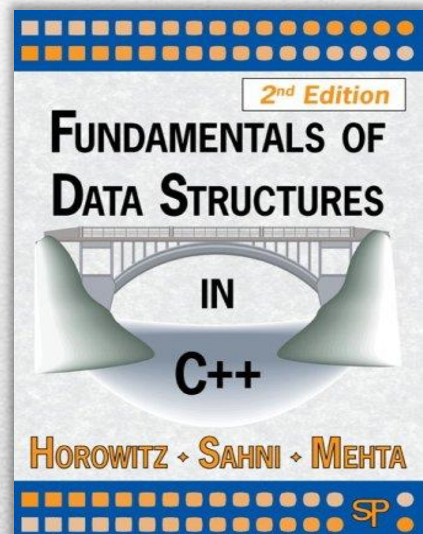
- ساختمان داده ها در C++

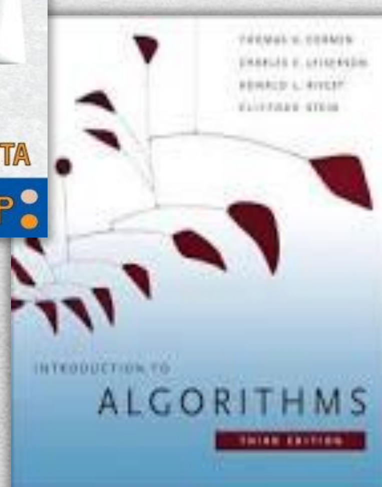
  حسین ابراهیم زاده قلزم

  انتشارات سیمای دانش
- ساختمان داده ها به زبان C

  امیر علیخانزاده

  انتشارات باغانی





مرجع

نحوه ارزیابی

- ارائه کلاسی ۱ نمره
- امتحان میان ترم ۴ نمره
- امتحان پایان ترم ۱۲نمره
- تمرین و پروژه ها ۳ نمره
- توجه تمرین و پروژه ها تنها در صورت تحویل در موعد ذکر شده نمره دارند

E-mail: Hadi.khademi@gmail.com

سرفصل مطالب

- مرتبه اجرایی (پیچیدگی اجرایی)
- توابع بازگشتی
- آرایه
- صف و پشته
- لیست پیوندی
- درخت
- گراف
- مرتب سازی
- درهم سازی

E-mail: Hadi.khademi@gmail.com

خصوصیات الگوریتم

- **ورودی:** یک الگوریتم می تواند هیچ یا چندین کمیت ورودی داشته باشد
- **خروجی:** الگوریتم بایستی حداقل یک کمیت به عنوان خروجی داشته باشد.
- **قطعیت (عدم ابهام):** هر دستورالعمل باید واضح و بدون ابهام باشد.
- **کارایی (انجام پذیر بودن):** هر دستورالعمل باید به قدر کافی ساده و ابتدایی باشد به گونه ای که با استفاده از قلم و کاغذ بتوان آن را با دست نیز اجرا نمود.
- **محدودیت (پایان پذیر بودن):** برای تمام حالات ، الگوریتم باید پس از طی مراحل محدودی خاتمه یابد.

برنامه

E-mail: Hadi.khademi@gmail.com

ارزیابی یک برنامه

معیارها

- 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؟

ساختمان داده

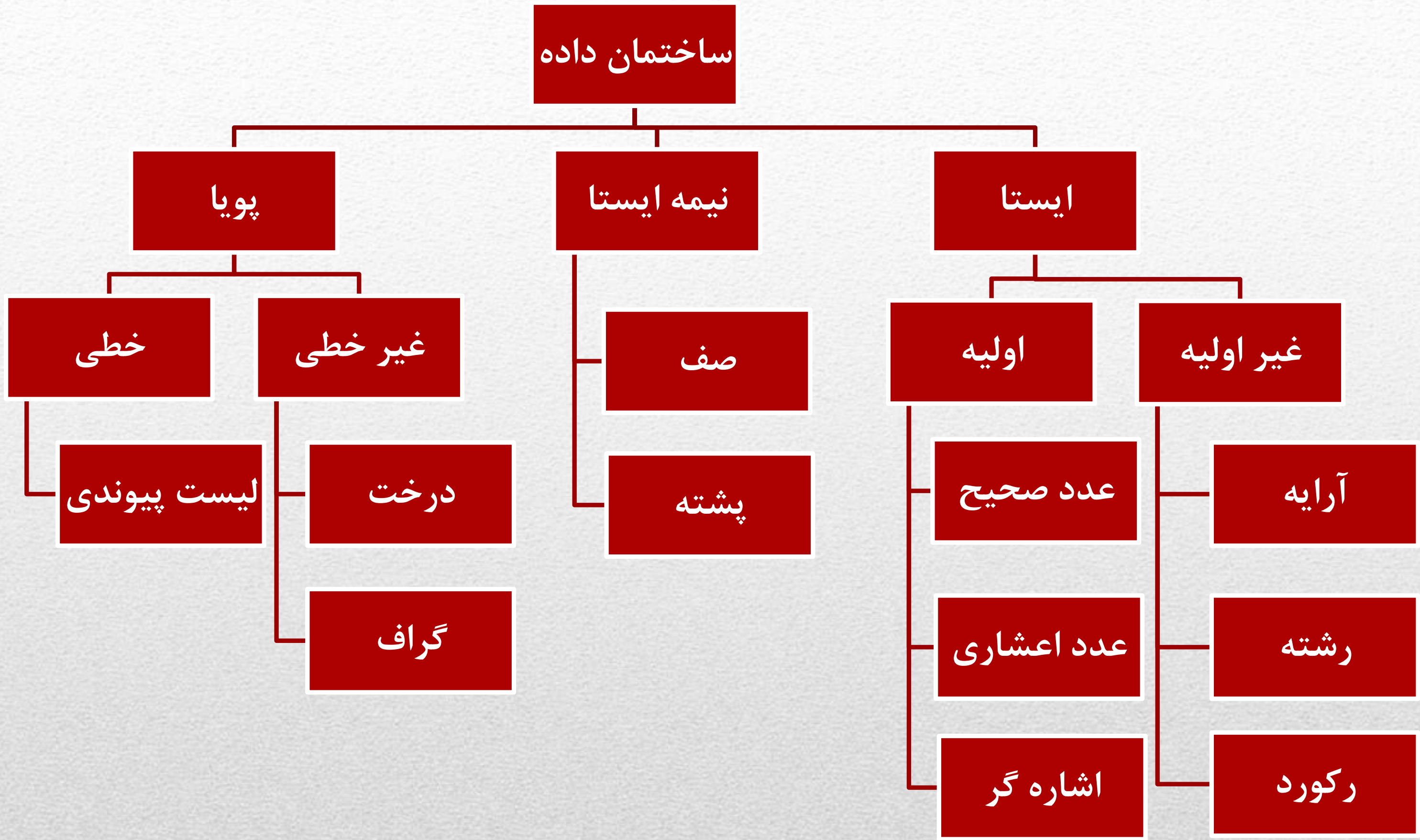
E-mail: Hadi.khademi@gmail.com

ساختمان داده

- چگونه برنامه های خوب و بهینه بنویسیم
- چگونه از حافظه سیستم به نحو مطلوب استفاده نماییم
- چگونه زمان اجرای برنامه ها را پایین بیاوریم و سرعت اجرای آنها را بالا ببریم

- **تعریف:**
  - ساختمان داده ها درسی است که هدف نهایی آن حل مساله سرعت اجرا و حافظه مصرفی الگوریتم ها است

E-mail: Hadi.khademi@gmail.com



انواع ساختمان داده

E-mail: Hadi.khademi@gmail.com

پیچیدگی یک برنامه

- **پیچیدگی زمانی و پیچیدگی حافظه**
- **پیچیدگی فضای یک برنامه** مقدارحافظه مورد نیازبرای اجرای کامل یک برنامه است.
- **پیچیدگی زمان یک برنامه** مقدار زمان کامپیوتر است که برای اجرای کامل برنامه لازم است.

E-mail: Hadi.khademi@gmail.com

پیچیدگی حافظه

نیازمندیهای فضای متغیر

فضای مورد نیاز که اندازه آن بستگی به نمونه I از مساله ای که حل می شود، دارد مانند حافظه مورد نیاز پشته بازگشتی و حافظه مورد نیاز برای متغیرهای ارجاعی

$$S(P) = c + S_p(I)$$

نیازمندیهای فضای کل

نیازمندیهای فضای ثابت

فضای مورد نیازی که به تعداد و اندازه ورودی و خروجی بستگی ندارد
مانند حافظه مورد نیاز دستورها، ثابت ها، متغیرهای با طول ثابت و ...

E-mail: Hadi.khademi@gmail.com

```
float sum ( float *a,  const int n)
{
    float s=0;
    For (int  i=0; i< n ; i++)
        S+=a[i];
    Return s;
}
```

- مشخصه موردی: n تعداد عضوهایی که با هم جمع می شوند

$$S_{sum}() = 0$$

پیچیدگی حافظه

E-mail: Hadi.khademi@gmail.com

```
float rs ( float *a,  const int n)
{
    if (n<=0) return 0;
    else return ( rs( a,n-1)+a[n-1] )
}
```

- مشخصه موردی: n تعداد عضوهایی که با هم جمع می شوند
- عمق بازگشتی n+1
- هر احضار تابع بازگشتی دست کم ۴ کلمه از حافظه
  - حافظه مقادیر a و n و مقدار برگشتی و ادرس برگشتی

$$S_{sum}() = 4(n+1)$$

پیچیدگی حافظه

E-mail: Hadi.khademi@gmail.com

زمان اجرای برنامه

$$T(P) = c + T_p(I)$$

نیازمندیهای زمان برنامه

زمان کامپایل

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

$$T_p(n) = c_a ADD(n) + c_s SUB(n) + c_l LDA(n) + c_{st} STA(n)$$

پیچیدگی زمانی

E-mail: Hadi.khademi@gmail.com

بسیاری عوامل در زمان اجرا دخیل هستند تخمینی از زمان اجرا

تنها تعداد مراحل داخل برنامه را می شماریم

یک مرحله برنامه ، قسمت با معنی برنامه است که زمان اجرای آن مستقل از خصیصه های نمونه باشد

return a+b+c+(a+b-c)/(a+b)+4.0

پیچیدگی زمانی

E-mail: Hadi.khademi@gmail.com

- توضیحات comments، تعاریف زیر برنامه و توابع، { }، begin، end
  - تعداد مراحل اجرایی صفر
- دستورهای تعیین نوع
  - تعداد مراحل اجرایی صفر مگر آنکه برای آنها مقدار دهی اولیه صورت گیرد در اینصورت یک

Procedure f(…) ; void f(…);	0 0

- دستور اجرایی
  - به ازای هر بار اجرا دارای گام ۱

int   x;
int   x=3;
float a,b=5;
	0
	1
	1

تعداد مراحل

y=x*y+z;	1
write (y)	1
return p;	1

- دستور شرطی if

عبارت شرط ۱ گام و گام کل دستور وابسته به درست و غلط بودن شرط

If (x<y)	1
S=2;	1
Else	
S=5;	1

۲ ۱+۱ شرط درست

۲ ۱+۱ شرط نا درست

If (x<y)	1
S=S+1;	1
Else	
t=t+1;	1
r=r+1	1

۲ ۱+۱ شرط درست

۳ ۱+۲ شرط نا درست

تعداد مراحل

- تعداد گام در حلقه
- حلقه به تعداد "تکرار +۱" گام و جملات تکرار شونده داخل حلقه به تعداد "تکرار" گام اختیار میکنند

```
For (i=2; i<n; i++)
     s=s+1
                              +
```

```
int f( int x)
{
int i, j=0;
for ( i=2; i<=n; i++)
j=j+i;
return i;
}
                              +
```

تعداد مراحل

E-mail: Hadi.khademi@gmail.com

- تعداد گام در حلقه های تو در تو
- از بیرونی ترین حلقه شروع کرده و تعداد تکرار هر حلقه را برای تمام حلقه ها و دستورات تکرار شونده پایین آن در "تعداد تکرار+۱" را برای خود حلقه در نظر می گیریم

procedure add(var a,b,c: matrix; m,n: integer);	0
var i,j : integer;	0
begin	0
for i:=1 to m do	m+1
for j:=1 to n do	m(n+1)
c[i,j]:= a[i,j]+ b[i,j];	mn
end	0
+	
	2mn+2m+1

اگر m>n باشد بهتر است جای
دو دستور for را در برنامه
عوض کنیم تا شمار
مراحل 2mn+2n+1
شود

تعداد مراحل

E-mail: Hadi.khademi@gmail.com

void sum (int m, int n , float s[][])	0
{ int i,j	0
for (j=0;j<m; j++)	m+1
{ S[n-1][j]=0;	m
for (i=0;i<n-1;i++)	mn
S[n-1][j]+=S[i][j];	m(n-1)=mn-m
}	0
}	0
+	
	2mn+m+1

تعداد مراحل

E-mail: Hadi.khademi@gmail.com

float sum (float *a, const int n)	0
{	0
float s=0;	1
for (int i=0; i< n ; i++)	n+1
S+=a[i];	n
return s;	1
}	0
+	
	2n+3

تعداد مراحل

E-mail: Hadi.khademi@gmail.com

	n<=0	n>0
float rs (float *a, const int n)	0	0
{	0	0
if (n<=0) return 0;	2	1
else return (rs(a,n-1)+a[n-1])	0	1+t(n-1)
}	0	0
+		
	2	2+t(n-1)

$$t(n)=\begin{cases} 2 & \text{if } n<=0 \\ 2+t(n-1) & \text{if } n>0 \end{cases} \Longrightarrow$$

$$\begin{aligned} t(n) &= 2+t(n-1) \\ &=2+2+t(n-2) \\ &=2+2+2+t(n-3) \\ &=2n+t(0) \end{aligned}$$

$$t(n)=2n+2$$

تعداد مراحل

E-mail: Hadi.khademi@gmail.com

تعداد مراحل

float rs (float *a, const int n)
{
    if (n<=0)
        return 0;
    else return (rs(a,n-1)+a[n-1])
}
+
	
	n+1
	1
	n
	
	2n+2	

E-mail: Hadi.khademi@gmail.com

▪ انگیزه ما برای تعیین شمار مراحل توانایی مقایسه پیچیدگی زمانی دو برنامه است که یک عمل را انجام می دهند و نیز پیش بینی رشد زمان اجرا با تغییر مشخصه موردی است.

علامت گذاری مجانبیO، Ω، Θ

E-mail: Hadi.khademi@gmail.com

- تعریف O (Big "oh"):
  - $f(n)=O(g(n))$ اگر و فقط اگر ثابتهای مثبتی مانند C و N وجود داشته باشند به طوری که به ازای تمامی مقادیر n و $n>=N$، $f(n)<=cg(n)$ باشد.
  - وقتی n به سمت بینهایت میل می کند رفتار $f(n)$ حداکثر (کوچکتر یا مساوی) $g(n)$ خواهد بود.
  - وقتی می نویسیم $O(1)$ منظور این است که زمان اجرا ثابت است، $O(n)$ یعنی زمان اجرا خطی است، $O(n^2)$ یعنی زمان اجرا از درجه دوم و ...

مثال

$f(n) = 3n+2$

$3n + 2 <= 4n$, for all $n >= 2$, $\therefore 3n + 2 = O(n)$

$f(n) = 10n^2+4n+2$

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

علامت گذاری مجانبی O، Ω، Θ

E-mail: Hadi.khademi@gmail.com

- تعریف o:

  - $f(n)=o(g(n))$ اگر و فقط اگر برای هر ثابت حقیقی مثبتی C یک عدد N وجود داشته باشند به طوری که به ازای تمامی مقادیر n و $n>=N$، $f(n)<cg(n)$ باشد.
  - وقتی n به سمت بینهایت میل می کند رفتار f(n) کوچکتر از g(n) خواهد بود.

علامت گذاری مجانبی O، Ω، Θ

E-mail: Hadi.khademi@gmail.com

- تعریف امگا Ω:
  - $f(n)=\Omega(g(n))$ می باشد اگر و فقط اگر به ازای مقادیر ثابت مثبت c و n0 ، برای تمام مقادیر n به شرطی که $n \geq n0$ باشد داشته باشیم $f(n) \geq cg(n)$
  - وقتی n به سمت بینهایت میل می کند رفتار f(n) حداقل (بزرگتر یا مساوی) g(n) خواهد بود.

مثال

$f(n) = 3n+2$

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

$f(n) = 10n2+4n+2$

$10n2+4n+2 >= n2$, for all $n >= 1$, $\therefore$ $10n2+4n+2 = \Omega (n2)$

علامت گذاری مجانبی O، Ω، Θ

E-mail: Hadi.khademi@gmail.com

- تعریف تعریف امگای کوچک ω:
  - برای تابع پیچیدگی $g(n)$، $\omega(g(n))$ شامل مجموعه ای از توابع پیچیدگی $f(n)$ می باشد که برای آنها برای هر ثابت حقیقی مثبتی C یک عدد صحیح مثبت $n0$ وجود دارد به قسمی که برای تمام مقادیر n که $n \geq n0$ باشد داشته باشیم $f(n)>cg(n)$
  - وقتی n به سمت بینهایت میل می کند رفتار $f(n)$ بزرگتر از $g(n)$ خواهد بود.

علامت گذاری مجانبی O، Ω، Θ

E-mail: Hadi.khademi@gmail.com

- تعریف تتا [Theta]
  - $f(n) = \theta(g(n))$ می باشد اگر و فقط اگر به ازای مقادیر ثابت c1 و c2 و n0، برای تمام مقادیر $n>=n0$ داشته باشیم $c1g(n)<= f(n) <= c2g(n)$.
  - وقتی n به سمت بینهایت میل می کند رفتار f(n) برابر از g(n) خواهد بود.

مثال

f(n) = 3n+2

3n <= 3n + 2 <= 4n, for all n >= 2, $\therefore$ 3n + 2 = Θ (n)

f(n) = 10n2+4n+2

n2 <= 10n2+4n+2 <= 11n2, for all n >= 5, $\therefore$ 10n2+4n+2 = Θ (n2)

علامت گذاری مجانبی O، Ω، Θ

E-mail: Hadi.khademi@gmail.com

- نشانه گذاری تتا از دو نشانه گذاری ذکر شده O 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.

علامت گذاری مجانبیO، Ω، Θ

E-mail: Hadi.khademi@gmail.com

- قضیه

$$a_m > 0$$

$$f(n) = a_m n^m + \ldots + a_1 n + n_0 \quad \Rightarrow \quad f(n) = O(n^m)$$

$$f(n) = a_m n^m + \ldots + a_1 n + a_0 \quad \Rightarrow \quad f(n) = \Omega(n^m)$$

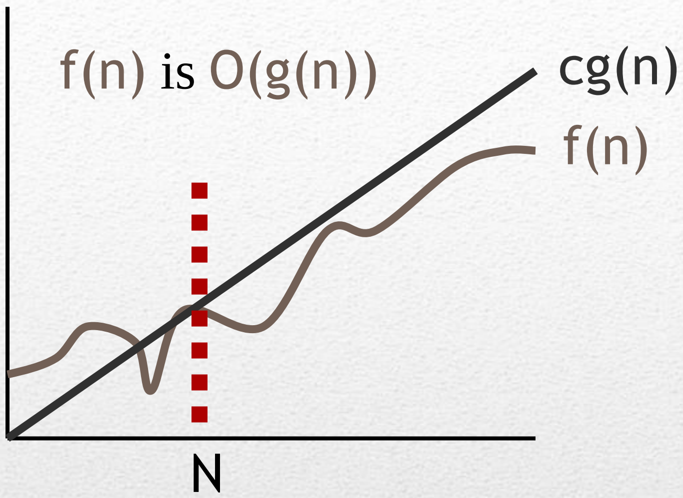
$$f(n) = a_m n^m + \ldots + a_1 n + a_0 \quad \Rightarrow \quad f(n) = \Theta(n^m)$$

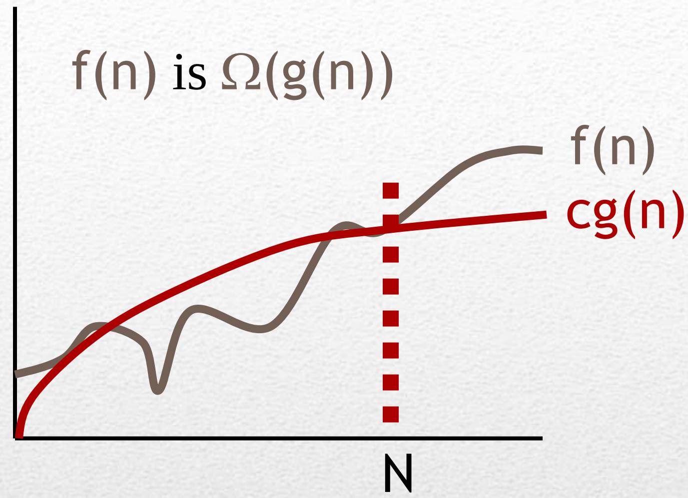
علامت گذاری مجانبی O، Ω، Θ

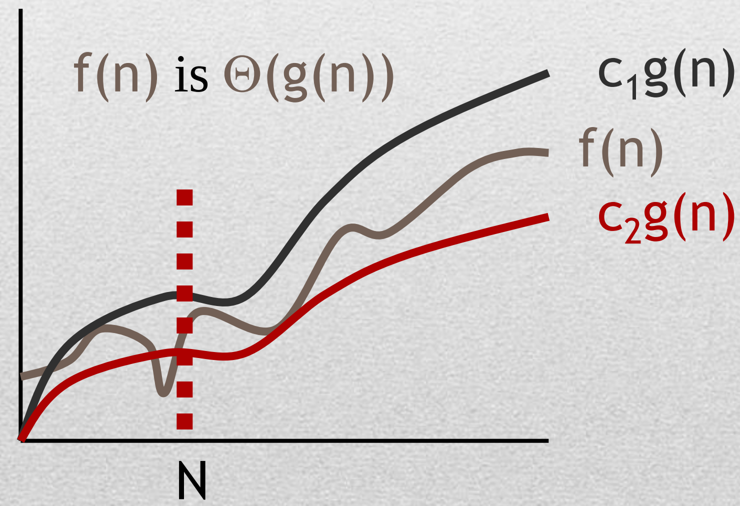
نمونه هایی از توابع رشد

- $O(1)$: constant
- $O(\log_2 N)$: logarithm
- $O(N)$: linear
- $O(N \log_2 N)$
- $O(N^2)$: quadratic
- $O(N^3)$: cubic
- $O(2^N)$: exponential
- $O(N!)$: factorial
- $O(N^k)$

E-mail: Hadi.khademi@gmail.com







مقایسه O، Ω، Θ

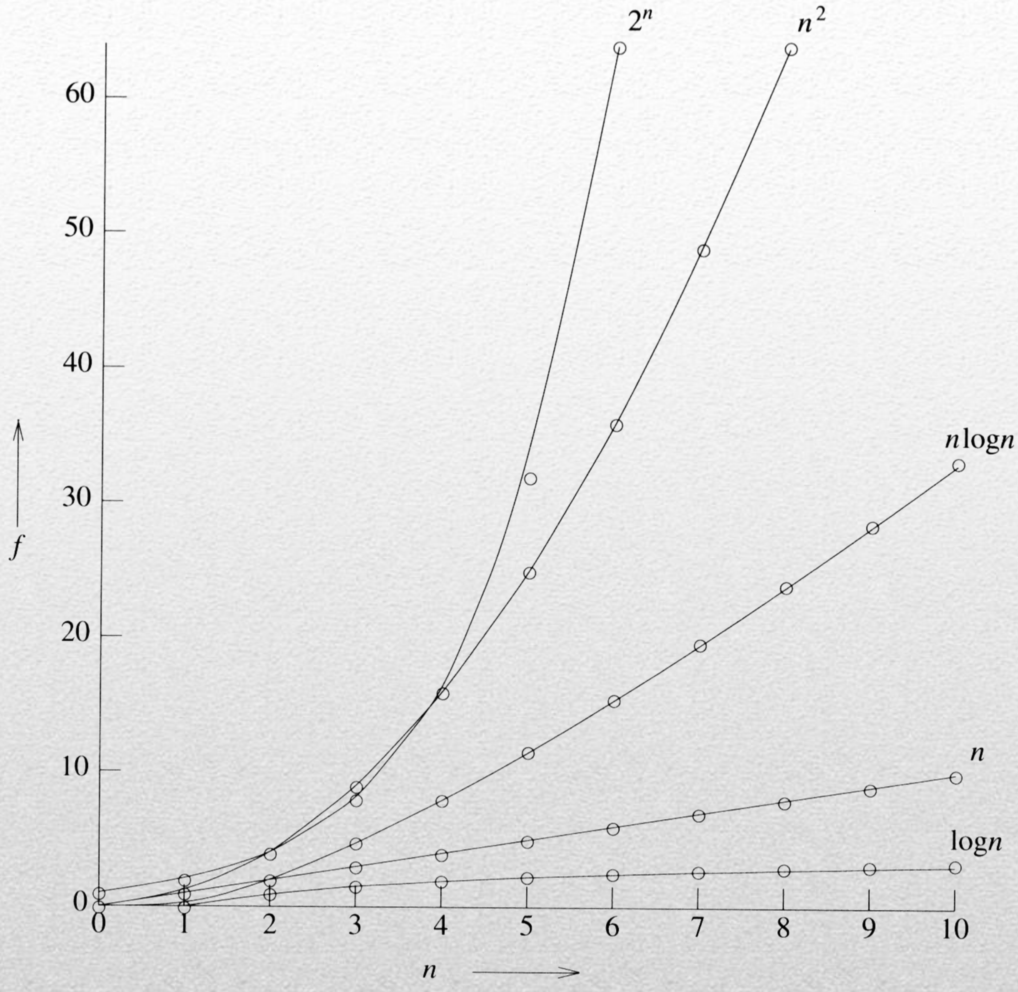
E-mail: Hadi.khademi@gmail.com

	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	

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

مقایسه توابع رشد

E-mail: Hadi.khademi@gmail.com



مقایسه توابع رشد

E-mail: Hadi.khademi@gmail.com

$f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$,
$f(n) = O(g(n))$ and $g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$,
$f(n) = \Omega(g(n))$ and $g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$,
$f(n) = o(g(n))$ and $g(n) = o(h(n)) \Rightarrow f(n) = o(h(n))$,
$f(n) = \omega(g(n))$ and $g(n) = \omega(h(n)) \Rightarrow f(n) = \omega(h(n))$.

$f(n)=\Theta(f(n)), f(n)=O(f(n)), f(n)=\Omega(f(n))$.
$f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$.
$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$,
$f(n) = o(g(n)) \Leftrightarrow g(n) = \omega(f(n))$.

روابط بین نمادهای مختلف

$$f(n) = O(g(n)) \approx a \leq b,$$

$$f(n) = \Omega(g(n)) \approx a \geq b,$$

$$f(n) = \Theta(g(n)) \approx a = b,$$

$$f(n) = o(g(n)) \approx a < b,$$

$$f(n) = \omega(g(n)) \approx a > b.$$

تعبیر عددی نمادهای معرفی شده

E-mail: Hadi.khademi@gmail.com

- فرض کنید می خواهیم عضوهای آرایه ی a با n عضو را برای پیدا کردن x جستجو کنیم می خواهیم بدانیم چگونه زمان اجرا با تغییر n تغییر می کند پارامتر n نامناسب و ناکافی است، برای یک مقدار n شمار مراحل برپایه ی موقعیت x در آرایه ی a تغییر می کند، برای رهایی از این مشکل سه نوع شمار مراحل تعریف شده است:
  - بهترین حالت
  - بدترین حالت
  - حالت میانگین

پیچیدگی الگوریتم ها

E-mail: Hadi.khademi@gmail.com

Useful Summation Formulas

$$\sum_{i=m}^{n} c = c\left(\sum_{i=m}^{n} 1\right) = c.(n-m+1)$$

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

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

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

$$\sum_{i=m}^{n} x^i = \frac{x^{n+1} - x^m}{x-1} \quad , x \neq 1$$

$$\sum_{i=0}^{n} x^i = \frac{x^{n+1} - 1}{x-1} \quad , x \neq 1$$

A special case of the above is: $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$

$$\sum_{k=1}^{n} \frac{1}{k} = 1 + \frac{1}{2} + \ldots + \frac{1}{n} \approx \ln n$$

$$\sum_{k=1}^{n} \lg k \approx n \lg n$$

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

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

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

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

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

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

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

$$\sum_{k=1}^{n} (2k-1) = n^2$$

- **تابع بازگشتی (recursive)**

تابعی است که حاوی حداقل یک دستور باشد که خود تابع را صدا بزند. این تابع به تعداد مراحل محدودی اجرا می شود و پس از آن متوقف می شود.

- مثال هایی از توابع بازگشتی:
  - فاکتوریل
  - مجموع اعداد ۱ تا n
  - توان
  - ترکیب
  - خارج قسمت تقسیم صحیح
  - آکرمان
  - هانوی
  - فیبوناتچی
  - زاد و ولد خرگوش ها

توابع و برنامه های بازگشتی

E-mail: Hadi.khademi@gmail.com

تابع فاکتوریل

- تابع زیر فاکتوریل n را محاسبه می کند:

$$f(n)=\begin{cases}1 & n=1\\ n\times f(n-1) & n>1\end{cases}$$

```
Fact(n){
        if(n==1)
                return 1;
        else
                return n*fact(n-1);
}
```

E-mail: Hadi.khademi@gmail.com

- تابع زیر مجموع اعداد ۱ تا n را محاسبه می کند:

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

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

تابع مجموع اعداد 1 تا n

E-mail: Hadi.khademi@gmail.com

▪ **تابع زیر مقدار n^m را محاسبه میکند:** (ورودی ها صحیح و مثبت هستند)

$$f(n,m)=\begin{cases} n & m=1 \\ n\times f(n,m-1) & m>1 \end{cases}$$

```
power(n){
        if(m==1)
                return n;
        else
                return n*power(n,m-1);
}
```

تابع توان

E-mail: Hadi.khademi@gmail.com

- تابع زیر ترکیب m از n را محاسبه می کند:

$$\binom{n}{m} = 1 \quad \text{if } m = 0 \text{ or } m = n$$

$$\binom{n}{m} = \binom{n-1}{m} + \binom{n-1}{m-1} \quad \text{if } 0 < m < n$$

```
Comb(n,m){
        if( (n==m || (m==0) )
                return 1;
        else
                return Comb(n-1,m) + Comb(n-1,m-1);
}
```

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

تابع ترکیب

E-mail: Hadi.khademi@gmail.com

- تابع زیر خارج قسمت تقسیم صحیح a بر b:

$$f(a,b)=\begin{cases}0 & a<b\\ f(a-b,b)+1 & a>=b\end{cases}$$

تابع خارج قسمت تقسیم صحیح

E-mail: Hadi.khademi@gmail.com

- تابع زیر معروف به آکرمان است:

$$f(a,b)=\begin{cases} b+1 & a=0 \\ f(a-1,1) & b=0 \\ f(a-1,f(a,b-1)) & a>0, b>0 \end{cases}$$

- در تابع آکرمان روابط زیر برقرار است: f(m,n)

$m \backslash n$	۰	۱	۲	۳	۴	n
۰	1	2	3	4	5	$n+1$
۱	2	3	4	5	6	$n+2=2+(n+3)-3$
۲	3	5	7	9	11	$2n+3=2\cdot(n+3)-3$
۳	5	13	29	61	125	$2^{(n+3)}-3$
۴	۱۳	۶۵۵۳۳	2^{65536} − ۳	$2^{2^{65536}}-3$	$2^{2^{2^{65536}}}-3$	$\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{n+3}-3$

تابع آکرمان

E-mail: Hadi.khademi@gmail.com

- مسئله برج هانوی چنین تعریف می‌شود:

۳ میله با تعدادی مهره و با اندازه‌های مختلف داریم. هدف این است که از وضعیت شروع که در آن همه مهره‌ها در میله اول و به ترتیب اندازه (کوچکترین در بالا) قرار دارند به وضعیت هدف برسیم که در آن همه مهره‌ها در میله سوم قرار دارند و باز هم به ترتیب اندازه روی هم چیده شده اند. اجازه داریم در هر حرکت فقط یک مهره را جابجا نماییم. مهره‌ای را حق دست زدن داریم که در روی آن مهره دیگری نباشد و همچنین حق گذاشتن آن مهره را در میله‌ای داریم که مهره کوچکتر از آن در آن میله نباشد.

برج هانوی

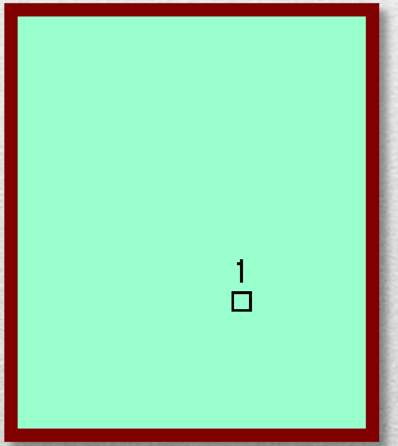
E-mail: Hadi.khademi@gmail.com

تابع فیبوناتچی

- تابع زیر محاسبه جمله nام سری فیبوناتچی را نشان می دهد:

$$f(n)=\begin{cases}0 & n=0\\ 1 & n=1\\ f(n-1)+f(n-2) & n\geq 2\end{cases}$$

```
fib(n){
        if( (n==0) || (n==1) )
                return n;
        else
                return fib(n-1) + fib(n-2);
}
```



E-mail: Hadi.khademi@gmail.com

- **معمای زاد و ولد خرگوش**

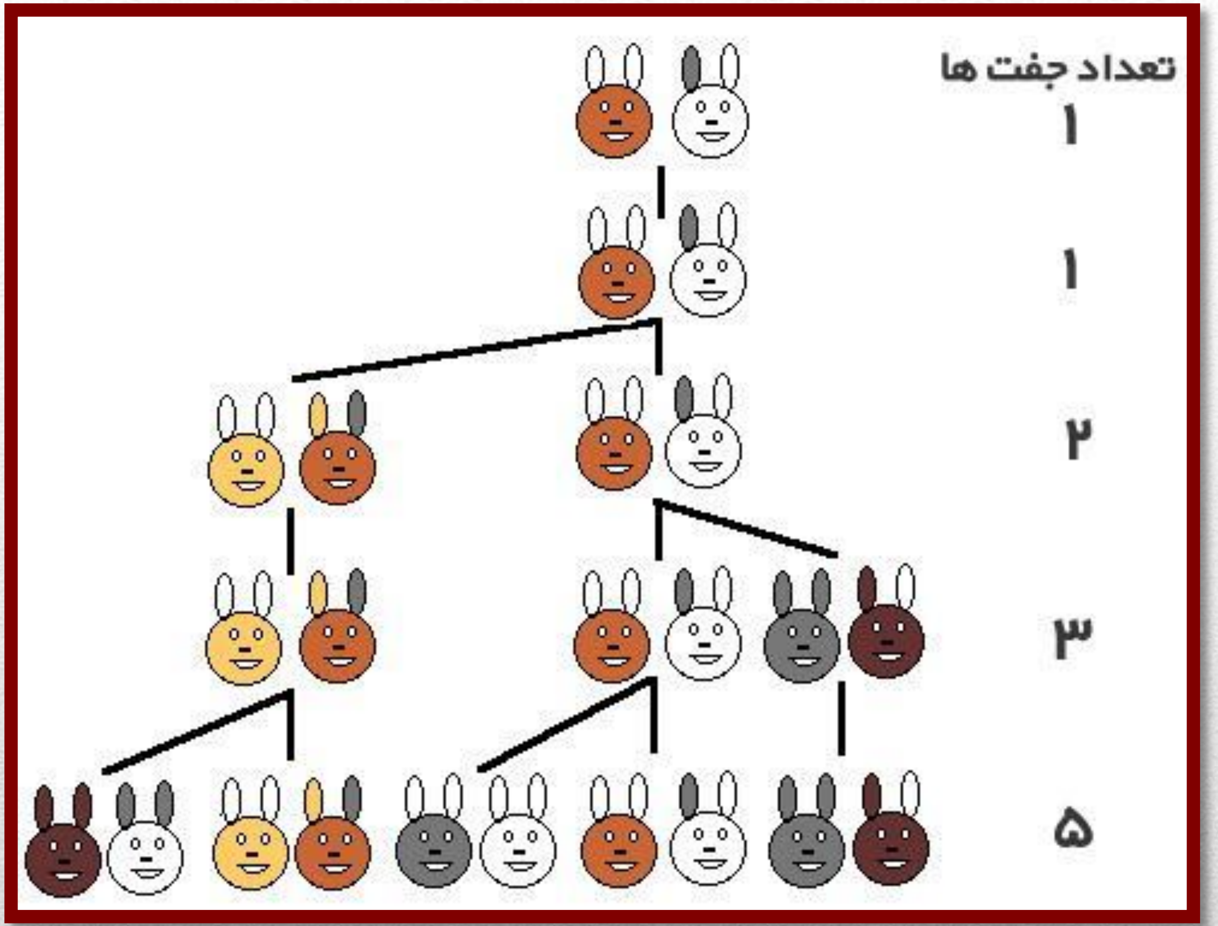
می خواهیم بدانیم اگر یک جفت خرگوش نر و ماده داشته باشیم و رفتاری برای زاد و ولد آنها تعریف کنیم در نهایت نتیجه چگونه خواهد شد. فرضیات اینگونه است:

  - شما یک جفت خرگوش نر و ماده دارید که همین الآن بدنیا آمده اند.
  - خرگوشها پس از یک ماه بالغ می شوند.
  - دوران بارداری خرگوشها یک ماه است.
  - هنگامی که خرگوش ماده به سن بلوغ می رسد حتما باردار می شود.
  - در هر بار بارداری خرگوش ماده یک خرگوش نر و یک ماده بدنیا می آورد.
  - خرگوش ها هرگز نمی میرند.

- رابطه بازگشتی بنویسید که تعداد خرگوش ها را در شروع ماه nام نشان دهد؟

زاد و ولد خرگوش ها

E-mail: Hadi.khademi@gmail.com



زاد و ولد خرگوش ها

E-mail: Hadi.khademi@gmail.com

- **رابطه های بازگشتی را می توان به روش های زیر حل کرد**
  - حدس
  - تکرار با جایگذاری
  - قضیه اصلی
  - درخت بازگشت
  - روش های حل رابطه های بازگشتی همگن و ناهمگن

روش های حل رابطه بازگشتی

E-mail: Hadi.khademi@gmail.com

- **روش درختی:** این روش زمانی استفاده می شود که تابع بازگشتی به صورت ضابطه ریاضی داده شود یا این که از مقدار بازگشتی در خطوط بعدی برنامه استفاده نشود.
- **روش مرحله به مرحله:** زمانی استفاده می شود که مقدار بازگشتی در خطوط بعدی زیربرنامه یا تابع بازگشتی به کار برده می شود (معمولا مربوط به زیر برنامه های بازگشتی)

روش حل تابع و زیر برنامه های بازگشتی

E-mail: Hadi.khademi@gmail.com

مثال الگوریتم جستجوی دودویی

```
int binsearch ( int  list [ ]  ،int  searchnum  ،int  n)
{
/* search  list [0] <=  list [1] <=  … <=list [ n-1 ]  for  searchnum  Return  its  position
if  found . Otherwise  return  -1  */
        for  (int left=0, right=n-1;    left<=right;   )  {
                int middle = ( left + right ) / 2 ;
                switch  ( COMPARE ( list [ middle ]  ،searchnum ))  {
                        case -1 :  left= middle +1; break;
                        case 0 :  return  middle ;
                        case 1 :  right=middle -1; break;
                }
        }
        return  -1 ; //not found
}
```

E-mail: Hadi.khademi@gmail.com

```
int binsearch ( int  list [ ]  ،int  searchnum  ،int  left  ،int  right )
{
/* search  list [0] <=  list [1] <=  … <=list [ n-1 ]  for  searchnum  Return  its  position
if  found . Otherwise  return  -1  */
            int  middle ;
            if (left <= right )  {
                        middle = ( left + right ) / 2 ;
                        switch  ( COMPARE ( list [ middle ]  ،searchnum )) {
                                    case -1 :  return binsearch ( list  ،searchnum  ،
                                                middle +1  ،right );
                                    case 0 :  return  middle ;
                                    case 1 :  return binsearch ( list  ،searchnum  ،left  ،
                                                middle -1 ) ;
                        }
            }
            return  -1 ;
}
```

مثال الگوریتم جستجوی دودویی

E-mail: Hadi.khademi@gmail.com

- فرض کنید که تابع پیچیدگی t(n) یک تابع غیر نزولی به صورت زیر است

$$\begin{cases} t(n) = a\, t(\frac{n}{b}) + cn^{K} \\ t(1) = d \end{cases}$$

- که n>1 و n توانی از b و b>=2
- k>=0 جزء اعداد صحیح و a>0 و c>0 و d>=0 باشد

$$t(n) = \begin{cases} \theta(n^{k}) & a < b^{k} \\ \theta(n^{k} \log n) & a = b^{k} \\ \theta(n^{\log_b^a}) & a > b^{k} \end{cases}$$

قضیه

E-mail: Hadi.khademi@gmail.com